

Année Universitaire 2023 – 2024

Service d'accueil : XLIM-AXE Mathématiques et Sécurité
de l'Information

Encadrants : K. Tamine & P. Dusart

Le Rapport de Stage de Master 1 : Intelligence Artificielle en environnement constraint, Couleurs chaudes ou froides

Cleque Marlain MBOULOU
cleque.mboulou@etu.unilim.fr

Résumé :

Le stage que je vais effectuer a pour objectif la construction d'un détecteur de couleurs froides ou chaudes à l'aide de arduino, puis d'entrainer un petit réseau de neurones pour le rentre autonome. Ce rapport me permettra de mettre en évidence les connaissances et compétence que je prévois acquérir ainsi que les buts et importances de ce stage.

Tout d'abord, Je vais présenter l'environnement arduino ainsi que le capteur couleurs utilisé (TCS34725). Ensuite, j'expliquerai comment connecter ce capteur à une carte Arduino.

Le montage du capteur est également expliqué, fournissant les étapes nécessaires pour connecter les fils et les composants de manière appropriée.

Ensuite, j'aborderai l'instanciation du capteur, c'est-à-dire comment le configurer dans le code Arduino. Une fois le capteur instancié, il est important de le calibrer correctement pour obtenir des mesures précises.

Ensuite, j'explorerai la détection de couleurs chaudes ou froides à l'aide d'un seuil des composantes RVB. Des exemples de résultats obtenus sont présentés pour illustrer cette méthode.

Enfin, je présenterai l'utilisation d'un réseau de neurones pour la détection de couleurs chaudes ou froides. Les fonctions d'apprentissage nécessaires à la mise en place du réseau seront expliquées, ainsi que leur implémentation dans le code Arduino. De plus, il est possible d'utiliser un réseau de neurones pré-entraîné pour cette tâche.

Je finirai par la présentation des applications possibles de ces techniques de détection de couleurs, et une conclusion récapitulant les points importants abordés. Des remerciements sont également adressés à la fin du document.

Mots clés : Arduino, Capteur TCS34725, Réseau de neurones,

Sommaire

1	Introduction	3
2	Présentation d'Arduino	3
2.1	Qu'est ce que Arduino ?	3
2.2	Description en détails de la carte arduino	4
2.3	Logiciel de programmation d'un Arduino	5
2.4	Programmation Arduino	5
3	Capteur de couleurs et Montage	6
3.1	Description du capteur TCS34725	6
3.2	Comment connecter le capteur TCS34725 à Arduino ?	8
3.3	Montage	9
4	Instanciation et calibration du capteur	9
4.1	Instanciation du capteur	9
4.2	Calibration du capteur	9
5	Détection de couleurs chaude ou froide : Utilisation des caractéristique des couleurs chaudes et froides	10
6	Détection de couleurs chaudes ou froide : Utilisation d'un réseau de neurones	12
6.1	Les fonctions pour l'apprentissage	13
6.2	Implémentation	14
6.3	Détection de couleurs chaudes ou froide : Utilisation d'un réseau de neurones déjà entrainé	17
7	Applications	20
8	Conclusion	21

1 Introduction

Il est claire que les couleurs chaudes font pensées à la chaleur et les couleurs froides à la fraîcheur. C'est pourquoi, il est naturelle que sur le cercle chromatique ci-dessous l'orange représente la couleur la plus chaude et le bleu la couleur la plus froide. Mais comment faire parvenir cette information à notre détecteur de couleurs froides ou chaudes ? Peut-il être en mesure de prendre cette décision tout seul ?

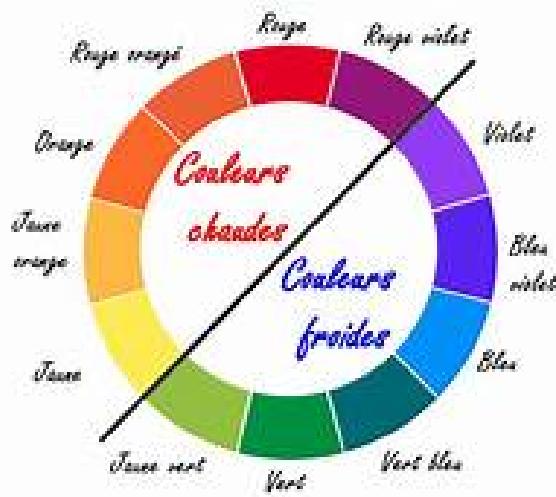


FIGURE 1 – Couleurs froides & chaudes

2 Présentation d'Arduino

N'ayant pas l'habitude d'utiliser arduino, j'ai jugé nécessaire de faire une brieve présentation.

2.1 Qu'est ce que Arduino ?

Un Arduino représente une carte électronique regroupant plusieurs composants électroniques afin de réaliser des objets électroniquement interactifs. Il peut être vu comme un ordinateur.

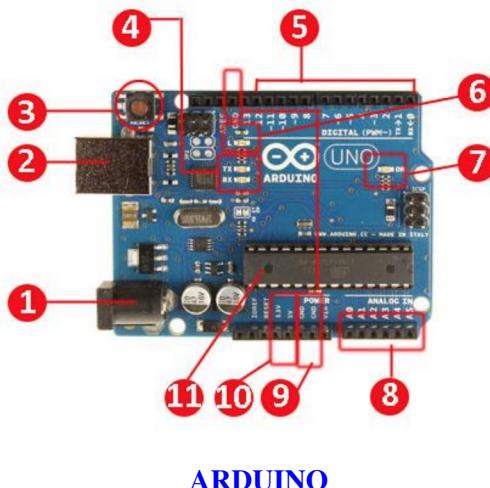
Il ressemble à ça :



ARDUINO_UNO_A06

Pour faire simple, l'Arduino est une carte qui se connecte sur l'ordinateur pour être programmée, et qui peut ensuite fonctionner seule si elle est alimentée en énergie. Elle permet de recevoir des informations et d'en transmettre depuis ou vers des matériels électroniques : diodes, potentiomètres, récepteurs, servomoteurs, moteurs, détecteurs... L'Arduino est donc capable de produire ou de capter ces signaux à notre demande grâce à la programmation et discuter avec un PC pour interagir avec l'environnement.

2.2 Description en détails de la carte arduino



1. Port d'alimentation de la carte entre 7V et 12V
2. Port USB pour brancher la carte à l'ordinateur et pouvoir téléverser le programme
3. Bouton Reset
4. Diode TX (**Transmission**) et RX (**Reception**) : clignote durant le téléchargement du programme et lors de la communication série
5. Pins 2-13 : Elles servent à piloter les composants digitaux ou analogiques (les pins avec PMW). Pour les utiliser on emploiera le code :

```
1 digitalRead();
```

Listing 1 – Code Arduino

Cette fonction prend en paramètre le numéro de la pin que vous souhaitez lire et renvoie la valeur de cette pin.

```
1 digitalWrite();
```

Listing 2 – Code Arduino

Cette fonction prend en paramètres le numéro de la pin que l'on souhaite contrôler et l'état (**HIGH** ou **LOW**) que l'on souhaite lui attribuer. Envoie ou écrit la valeur digitale du composant situé sur la pin entre parenthèse avec HIGH pour alimenter en électricité et LOW pour couper l'électricité

```
1 analogWrite();
```

Listing 3 – Code Arduino

Lit la valeur analogique du composant situé sur la pin entre parenthèse

6. LED de la pin 13 : LED interne associée à la pin 13
7. LED d'alimentation.
8. PIN A0-A5 : Pin analogique, permet de piloter des composants analogiques. Pour les utiliser on emploiera le code

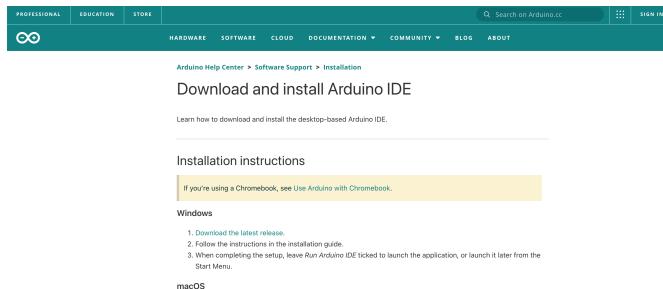
1 **analogRead()**

Envoie la valeur analogique du composant situé sur la pin entre parenthèse

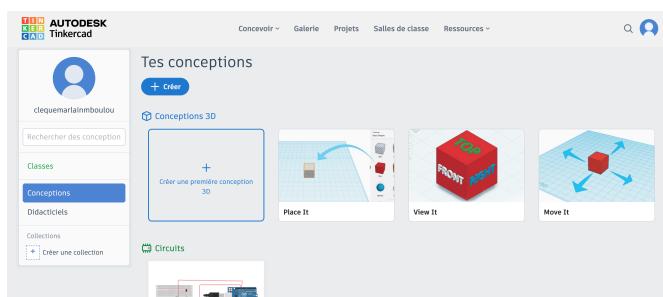
9. Pins GND : Permet d'alimenter les composants électroniques du circuit. Elle représente le négatif
10. Pin alimentation : Permet d'alimenter les composants électroniques du circuit. Elle représente le positif. L'une est à 5V et l'autre à 3,3V
11. Microcontrôleur

2.3 Logiciel de programmation d'un Arduino

Pour programmer un Arduino, nous avons besoin d'un éditeur de code fonctionnant sur un ordinateur et d'un programme permettant de téléverser le code vers la carte.



L'IDE (Integrated Development Environment) **Arduino Software** est un programme libre de droit qui nous permet d'écrire du code et de le téléverser dans la carte. C'est l'outil recommandé pour l'utilisation du matériel Arduino.



Il existe également un simulateur appelé **Tinkercad Circuits** qui nous permet de tester notre code et montages électriques de manière virtuelle. Cela nous permet d'expérimenter et de déboguer notre code sans avoir besoin d'un matériel physique.

En utilisant l'IDE Arduino Software et éventuellement le simulateur Tinkercad Circuits, nous disposons des outils nécessaires pour développer et tester notre code et notre montage.

2.4 Programmation Arduino

Le langage Arduino est basé sur les langages C et C++. Les programmes Arduino sont intégrés dans un schéma (*sketch*) **setup/loop**. Il s'agit de deux blocs de fonctions obligatoirement présents dans tous les

programmes Arduino.

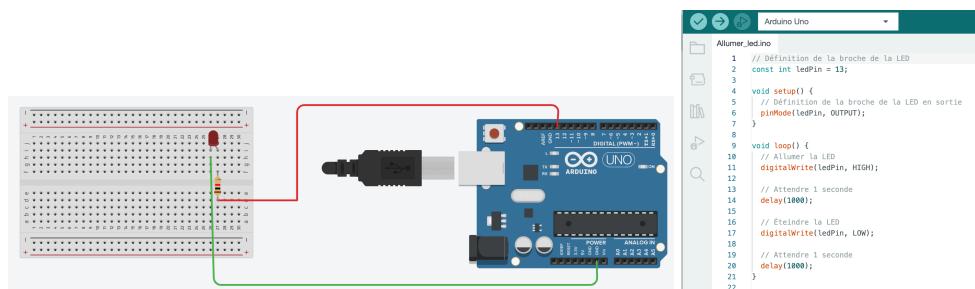
Un bloc commence par le symbole { et se termine par le symbole }. Le nom d'une fonction est toujours suivi des symboles () .

La fonction `setup()` est appelée systématiquement au démarrage de l'Arduino, une seule fois, après une réinitialisation ou une mise sous tension. Elle est utilisée pour initialiser des variables, démarrer des librairies, modifier le paramétrage des broches, etc.

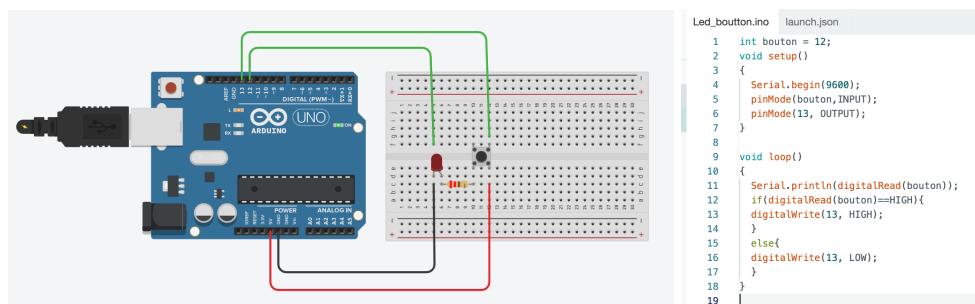
Après avoir utilisé la fonction `setup()`, la fonction `loop()` exécute de manière infinie le code à l'intérieur de ce bloc afin de répondre aux interactions demandées.

Afin de rendre plus intelligible le code écrit, il est possible d'écrire du texte qui ne sera pas interprété comme du code. Une ligne de commentaire commence par les symboles //, tandis qu'un bloc de commentaires sera encadré par les symboles /* et */.

Exemples :



allumer une LED 1 seconde et l'éteindre pendant 1/2 seconde



allumer une LED en appuyant sur un boutton

On peut également trouver d'autres fonction sur le [site officiel](#)

3 Capteur de couleurs et Montage

Cette partie vise à extraire les différentes valeurs RGB de la couleurs présentée et d'en déduire s'il s'agit d'une couleur froide ou chaude. Pour cela, afin de déterminer avec précision les couleurs RGB, il existe plusieurs capteurs de couleurs, dans notre cas, nous allons nous intéresser au capteur **TCS34725**.

3.1 Description du capteur TCS34725

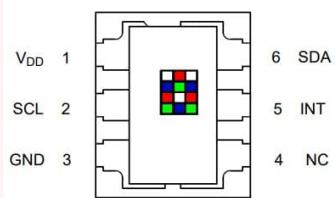
Un capteur RGB (Red, Green, Blue) est un dispositif électronique qui est utilisé pour détecter et mesurer les niveaux de couleur sur un objet. Il est capable de mesurer les intensités des composantes rouge, verte et bleue de la lumière, ce qui permet de déterminer avec précision la couleur dominante.



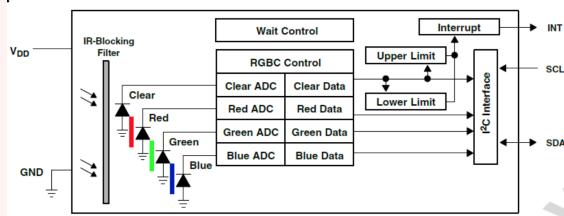
Capteur **TCS34725** de Adafruit

Le capteur TCS34725 ci-dessus est un capteur de couleur développé par Adafruit Industries. Il s'agit d'un capteur de couleur numérique basé sur la technologie des capteurs de lumière ambiante et de couleur RGB. Voici les principaux composants :

1. Photodiodes : Le capteur TCS34725 est équipé d'un réseau de 9 photodiodes qui capturent la lumière ambiante et la lumière réfléchie par les objets. Ces photodiodes sont sensibles aux différentes composantes de couleur, à savoir le rouge, le vert et le bleu.



2. Filtre IR (infrarouge) : Un filtre infrarouge est intégré, il permet de réduire les interférences indésirables provenant de la lumière infrarouge ambiante. Cela permet d'améliorer la précision des mesures de couleur en éliminant les composantes infrarouges.



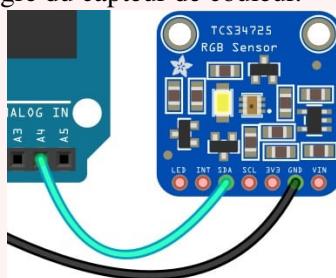
3. Convertisseur analogique-numérique CAN : Un CAN est intégré, convertissant les signaux électriques provenant des photodiodes en valeurs numériques correspondant à l'intensité de chaque composante de couleur (RVB). Le CAN offre une résolution de 16 bits, offrant ainsi une précision élevée dans la détection des couleurs.
4. Interface de communication : Une interface de communication I^2C permet d'échanger des données avec la carte Arduino ou le microcontrôleur. L' I^2C est un protocole de communication série qui permet de connecter facilement le capteur à d'autres périphériques.
5. Broches de connexion : Le capteur TCS34725 dispose aussi de broches de connexion qui permettent de le connecter à la carte Arduino ou au microcontrôleur.
 - (a) Un régulateur de tension est intégré ce qui permet d'alimenter en 5V (**VIN & GND**)
 - (b) Les données transitent avec le bus I^2C (**SCL & SDA**)
 - (c) Une LED(température 4150K) permet d'éclairer l'objet à mesurer

3.2 Comment connecter le capteur TCS34725 à Arduino ?

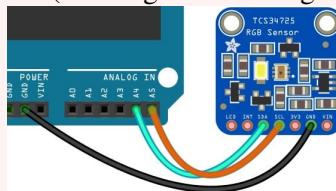
Etapa 1. Connectez la broche GND sur le module de capteur de couleur avec Arduino. Commencez toujours par la connexion à la terre afin que les deux cartes aient une référence commune avant d'effectuer d'autres connexions.



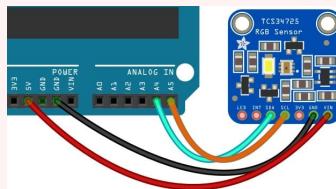
Etapa 2. Connectez ensuite la ligne de données I^2C . Connectez la broche A4 de l'Arduino à la broche *SDA* du circuit intégré du capteur de couleur.



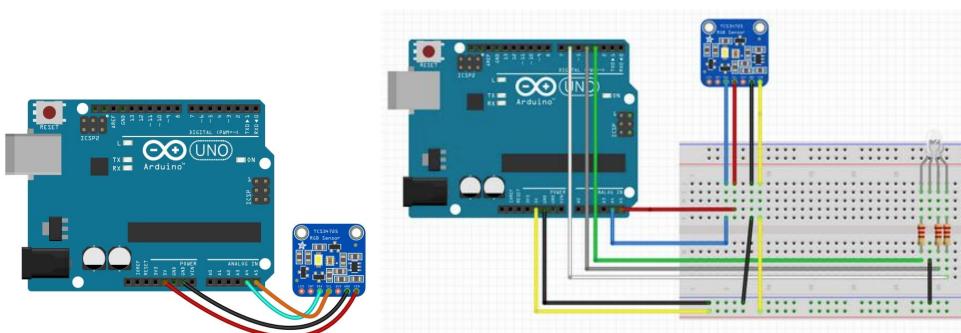
Etapa 3. Connectez la ligne d'horloge I^2C . Connectez la broche A5 sur l'Arduino à la broche *SCL* sur le capteur de couleur *IC* (fil orange dans l'image ci-dessous)



Etapa 4. connectez la broche d'alimentation. Connectez la broche du capteur étiquetée *VIN* à la broche *5V* de l'Arduino.



On pourra rajouter une LED nous permettant de couleur (chaude ou froide) qui a été mesurée.



3.3 Montage

4 Instanciation et calibration du capteur

Afin d'exploiter les capacités du capteur TCS34725, la société Adafruit à développée une librairie qui lui est dédié. Il suffit donc de télécharger cette librairie sur **Arduino IDE** et ensuite l'importer avec :

```
1 #include "Adafruit_TCS34725.h"
```

Listing 4 – Code Arduino

4.1 Instanciation du capteur

On instancie le capteur de la manière suivante :

```
1 Adafruit_TCS34725 tcs = Adafruit_TCS34725
2 (TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X)
```

Listing 5 – Code Arduino

Elle crée une instance de la classe **Adafruit_TCS34725** nommée **tcs** en utilisant le constructeur de la classe.

Le constructeur **Adafruit_TCS34725** prend en paramètres deux arguments :

TCS34725_INTEGRATIONTIME_50MS et **TCS34725_GAIN_4X**. Ces arguments définissent le temps d'intégration du signal et de gain du capteur de couleur TCS34725.

1. **TCS34725_INTEGRATIONTIME_50MS** indique une intégration de temps de 50 millisecondes, ce qui détermine la durée pendant laquelle le capteur collecte la lumière pour mesurer la couleur. Un temps d'intégration plus long peut améliorer la sensibilité du capteur aux couleurs faibles, mais il nécessite également plus de temps pour effectuer une mesure. Ainsi dans **TCS34725_INTEGRATIONTIME_50MS**, nous pouvons changer 50 par n'importe de quelles autres valeurs (2, 4, 100, 700, ...)
2. **TCS34725_GAIN_4X** spécifie un gain de $4x$ pour amplifier le signal du capteur. Le gain peut être ajusté pour améliorer la précision des mesures en fonction des conditions d'éclairage.

En utilisant ces paramètres d'intégration et de gain lors de la création de l'objet **tcs**, nous pouvons contrôler les caractéristiques de fonctionnement du capteur TCS34725. Cela nous permet d'adapter le comportement du capteur en fonction de nos besoins spécifiques.

Une fois que l'objet **tcs** est créé, nous pouvons utiliser les fonctions de la bibliothèque **Adafruit_TCS34725** pour interagir avec le capteur, comme la lecture des valeurs de couleur.

Ensuite avec **tcs.begin()**, nous initialisons le capteur. Cette fonction renvoie **true** si le capteur est bien présent.

4.2 Calibration du capteur

Il est important de calibrer un capteur TCS34725 pour garantir des mesures de couleur précises et cohérentes. La calibration consiste à ajuster les paramètres du capteur pour corriger les éventuelles erreurs ou variations dans les mesures. Voici quelques raisons importantes pour lesquelles la calibration est nécessaire : Pour garantir la cohérence et la précision du capteur TCS34725, il est important de le calibrer,

c'est-à-dire ajuster les paramètres du capteurs pour corriger les éventuelles erreurs ou variations de mesures. Cette étape est nécessaire car elle permet de :

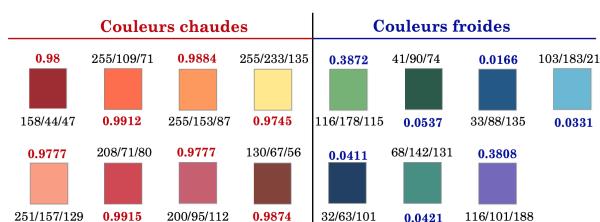
1. Eliminer les écarts de fabrication
2. Compenser les variation de l'environnement, comme les conditions d'éclairage...
3. Corriger les influences externes, tels que la température, l'humidité ou la pollution lumineuse.
4. S'adapter aux besoins spécifiques.

En calibrant le capteur TCS34725, nous pouvons améliorer considérablement la précision et la fiabilité des mesures de couleur. La calibration doit être effectuée initialement lors de la mise en service du capteur, dans la fonction `setup()`. Dans notre cas, cela consiste à mesurer les valeurs **RGB** du blanc et du noir où les valeur **RGB** du noir seront les valeurs minimales ("luminosité nulle") et celles du blanc maximal("forte luminosité") . Pour cela, deux choix se présente à nous, soit on extrait ces valeur avec la fonction :

`tcs.getRGB(&rouge,&vert,&bleu)` ou avec la fonction `tcs.getRawData(&rouge,&vert,&bleu,&clear)`, clear pour dire sans filtre **RGB**.

5 Détection de couleurs chaude ou froide : Utilisation des caractéristique des couleurs chaudes et froides

En général, les couleurs chaudes sont caractérisées par une intensité élevée de la composante rouge, qui est toujours supérieure aux composantes verte et bleue. En revanche, les couleurs froides présentent une intensité de rouge inférieure à celles des composantes verte et bleue. Pour mieux illustrer cette distinction, voici un exemple :



J'ai alors eu l'idée de construire dans un premier temps, un détecteur de couleurs chaudes ou froides grâce à cette information. Dans une fonction `estFroid(int rouge, int vert, int bleu)`, on prend en paramètre les valeurs RGB de la couleur de l'objet présenté et en renvoie **true** si la couleur est froide et **false** sinon (est chaude). Dans la fonction `loop()`, exécutée à l'infinie, on allumera la LED correspondante.

voici le code complet :

```

1 #include <Wire.h>
2 #include "Adafruit_TCS34725.h"
3
4 // #include<NeuralNetwork.h>
5
6 const int Led_froid = 12;
7 const int Led_chaud = 11;
8

```

```
9 Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
10   TCS34725_GAIN_4X);
11 float rmin = 13;
12 float rmax = 1953;
13 float gmin = 12;
14 float gmax = 2561;
15 float bmin = 5;
16 float bmax = 2152;
17
18 void setup() {
19   // put your setup code here, to run once:
20   Serial.begin(9600);
21   tcs.begin();
22
23   pinMode(Led_froid, OUTPUT);
24   pinMode(Led_chaud, OUTPUT);
25 }
26
27 bool estFroid(int rouge, int vert, int bleu) {
28   // Verification des composantes RVB
29   if (rouge <2 && bleu<2 && vert<2) {
30     // Couleur chaude detectee
31     return 1;
32   }
33   if (rouge > bleu && rouge > vert) {
34     // Couleur chaude detectee
35     return false;
36   } else {
37     // Couleur froide detectee
38     return true;
39   }
40 }
41
42 }
43
44 void loop() {
45   // Mesure des couleurs
46   uint16_t r, g, b, c;
47   //float r, g, b, c;
48   //tcs.getRGB(&r, &g, &b); // Extraction des valeurs RGB
49   tcs.getRawData(&r, &g, &b, &c);
50   int rmap = map(r, rmin, rmax, 0, 255);
51   int gmap = map(g, gmin, gmax, 0, 255);
52   int bmap = map(b, bmin, bmax, 0, 255);
53
54   int R = constrain(rmap, 0, 255);
55   int G = constrain(gmap, 0, 255);
56   int B = constrain(bmap, 0, 255);
57
58   if (estFroid(R, G, B) == true) {
59     Serial.println("Froid");
60     Serial.println(R);
61     Serial.println(G);
62     Serial.println(B);
63     digitalWrite(Led_chaud, LOW);
64     digitalWrite(Led_froid, HIGH);
65     delay(300);
66 }
```

```

66 } else {
67   Serial.println("Chaud");
68   Serial.println(R);
69   Serial.println(G);
70   Serial.println(B);
71   digitalWrite(Led_chaud, HIGH);
72   digitalWrite(Led_froid, LOW);
73 }
74 }
```

Listing 6 – Code Arduino

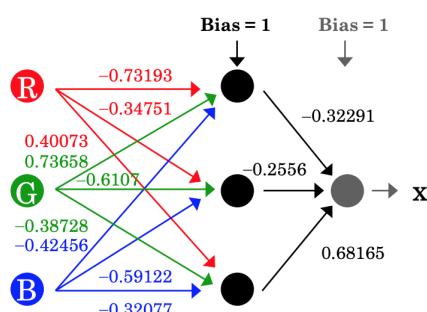
6 Détection de couleurs chaudes ou froide : Utilisation d'un réseau de neurones

Il existe sur arduino une librairie **NeuralNetwork** qui nous permet de construire et entraîner un réseau de neurones à fin de lui faire prédire les classes chaude ou froide d'une couleur considérée.

Dans un réseau de neurones, chaque neurone est connecté à d'autres neurones par des connexions pondérées. Dans notre cas, chaque couche à l'exception de la couche de sortie aura 3 neurones, correspondant aux valeurs RGB. Chaque neurone sera connecté à tous les neurones de la couche suivante. Les poids représentent l'importance de chaque connexion entre les neurones et déterminent comment l'entrée d'un neurone est pondérée avant d'être transmise à d'autres neurones de la couche suivante.

Le biais, quant à lui, est une valeur constante ajoutée à l'entrée pondérée d'un neurone. Le biais permet de déplacer la fonction d'activation du neurone. La librairie **NeuralNetwork** initialise aléatoirement les poids et biais de tout le réseau.

Voici comment fonctionne le processus de calcul dans un réseau de neurones avec poids et biais :



1. Les neurones de la couche d'entrée reçoivent les valeurs RGB.
2. Chaque neurone de la couche d'entrée transmet les données pondérées aux neurones de la couche suivante en utilisant les poids correspondants.
3. Les neurones de la couche suivante effectuent une somme pondérée des entrées reçues, en utilisant les poids et le biais associés.
4. Les neurones appliquent ensuite une fonction d'activation à la somme pondérée pour introduire une non-linéarité dans le modèle.
5. Le résultat est propagé à travers les couches cachées du réseau jusqu'à atteindre la couche de sortie.
6. La couche de sortie effectue une dernière transformation en utilisant des poids et des biais spécifiques à la couche de sortie.
7. La sortie finale est générée, qui représente la prédiction ou le résultat du réseau de neurones.

Pendant l'apprentissage, les poids et les biais sont ajustés itérativement à l'aide d'algorithmes d'optimisation, tels que la rétropropagation du gradient, pour minimiser l'erreur entre les prédictions du réseau et les valeurs réelles des données d'entraînement. L'apprentissage consiste à trouver les valeurs de poids et de biais qui permettent au réseau de produire les résultats les plus précis possibles.

6.1 Les fonctions pour l'apprentissage

- **NeuralNetwork Nom(couches, n)** : Elle permet de définir un réseau de neurone avec un certain "Nom". n correspond au nombre des couches (couche d'entrée, couches cachées et couche de sortie). La variable "couches" est un tableau correspondant au nombre de neurones par couche. Cette fonction est mise avant le **setup()** afin qu'elle soit connue de tout le programme.
- Il faut ensuite définir une base d'apprentissage :
 1. Un tableau 2D des entrées, **const float entrees[X₁][X₂]** où X_1 correspond au nombre d'exemples et X_2 au nombre de neurones en entrée, c'est-à-dire 3 (Red, Green et Blue)
 2. Un tableau 2D des sorties attendues, **const float sortie[X₁][X₃]**, où X_1 représente le nombre d'exemples et X_3 le nombre de neurones en sortie (ici un seul car nous avons un problème de classification)
- **Nom.FeedForward (entrees)**, permet de calculer la sortie pour un exemple de la base d'apprentissage
- **Nom.BackProp (sortiesAttendues)**, permet de corriger les poids et biais en fonction de l'erreur sur la sortie en utilisant l'algorithme de rétropropagation du gradient (Back Propagation). L'algorithme de rétropropagation du gradient fonctionne en calculant l'erreur entre les prédictions du réseau et les valeurs réelles, puis en propageant cette erreur en arrière à travers les couches du réseau. Les poids et les biais sont ajustés en fonction de l'erreur calculée afin de minimiser cette dernière. Cela se fait en utilisant la dérivée partielle de la fonction de perte par rapport à chaque poids et biais.
- Il faut utiliser successivement les deux fonctions précédentes sur toute la base d'apprentissage et répéter cette opération un grand nombre de fois.

On peut également trouver d'autres fonctions de cette bibliothèque sur [github GiorgosXou/NeuralNetworks](https://github.com/GiorgosXou/NeuralNetworks)

6.2 Implémentation

En dehors des fonctions **setup()** et **loop()**, nous incluons les bibliothèques nécessaires, importons les données d'apprentissage de notre réseau de neurones (entrées et sorties attendues), définissons les couches du réseau, les variables de calibration du capteur et les broches des LED de sortie (chaud ou froid).

```

1 #include <NeuralNetwork.h>
2 #include <Wire.h>
3 #include "Adafruit_TCS34725.h"
4 #define ITER 4000
5 const int Led_froid = 12;
6 const int Led_chaud = 11;
7 Adafruit_TCS34725 tcs =
8     Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X);
9 const float entrees[19][3] = {...};
10 const float sortiesAttendues[19][1] = {...};
11 const float teste [6][3]= {...};
12 unsigned int couches[] = { 3, 3, 1 };
13 float *sorties;
14 uint16_t rmin, rmax, gmin, gmax, bmin, bmax, cmin, cmax;
15 NeuralNetwork NN(couches, 3);

```

Listing 7 – Code Arduino

Dans la fonction **setup()**, nous prenons les valeurs brutes RVB fournies par le capteur et effectuons une calibration. Ensuite, nous entraînons notre réseau de neurones (**NN**) sur les données d'entraînement. Après cela, nous effectuons un test sur d'autres données dont nous connaissons la sortie (chaud ou froid), et nous sommes satisfaits des résultats obtenus.

```

1 void setup() {
2     Serial.begin(9600);
3     tcs.begin();
4     // Phase calibration
5     Serial.println("Phase de calibration: Identification du blanc. Entrer
6         'y' quand c'est OK");
7     ...
8     delay(4000);
9     Serial.println("Phase de calibration: Identification du noir. Entrer 'z'
10        quand c'est OK");
11    ...
12    delay(4000);
13    // Entrainement du reseau de neurone NN
14    ...
15    // Exemples
16    ...
17    pinMode(Led_froid, OUTPUT);
18    pinMode(Led_chaud, OUTPUT);
19 }

```

Listing 8 – Code Arduino

Dans la fonction **loop()**, nous récupérons les valeurs RVB brutes de la couleur détectée par le capteur et les normalisons pour qu'elles restent dans l'intervalle [0, 1]. Ensuite, nous fournissons ces valeurs RVB au réseau de neurones pour obtenir la sortie attendue (chaud ou froid). Si la valeur renvoyée par le réseau de neurones se situe dans l'intervalle [0, 0.5], la couleur est considérée comme froide et une LED verte s'allume pour le signaler. Dans le cas contraire, si la valeur est supérieure à 0.5, la couleur est considérée comme chaude et une LED rouge s'allume.

Le code complet est ci-dessous :

```
1 #include <NeuralNetwork.h>
2 #include <Wire.h>
3 #include "Adafruit_TCS34725.h"
4 #define ITER 4000
5
6 const int boutton = 13;
7 const int Led_froid = 12;
8 const int Led_chaud = 11;
9
10 Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
11     TCS34725_GAIN_4X);
12 const float entrees[19][3] = {
13     { 1.00, 0.00, 0.00 }, //rouge
14     { 0.00, 1.00, 0.00 }, //vert
15     { 0.00, 0.00, 1.00 }, //bleu
16     { 1.00, 1.00, 0.00 }, //cyan
17     { 1.00, 0.00, 1.00 }, // Magenta
18     { 255.0/255.0, 165/255.0, 0/255.0}, //Orange vif
19     { 255.0/255.0, 105/255.0, 180/255.0}, //Rose chaud
20     { 135.0/255.0, 206/255.0, 250/255.0}, //Bleu ciel
21     { 152.0/255.0, 251/255.0, 152/255.0}, //Vert menthe
22     { 128/255.0, 0.0/255.0, 128/255.0}, //Violet
23     { 211/255.0, 211/255.0, 211/255.0}, //Gris clair
24     { 139/255.0, 0/255.0, 0/255.0}, //Rouge fonce
25     { 255/255.0, 140/255.0, 0/255.0}, //Orange fonce
26     { 176/255.0, 224/255.0, 230/255.0}, //Bleu ciel clair
27     { 0/255.0, 71/255.0, 171/255.0}, //Bleu cobalt
28     { 255/255.0, 215/255.0, 0/255.0}, //Jaune dore
29     { 255/255.0, 20/255.0, 147/255.0}, //Rose vif
30     { 128/255.0, 0/255.0, 0/255.0}, //Bordeaux
31     { 112/255.0, 128/255.0, 144/255.0}, //Gris bleu
32 };
33 const float teste [6][3]= {
34     { 208.0/255.0, 71/255.0, 80/255.0}, //
35     { 41.0/255.0, 90/255.0, 74/255.0}, //
36     { 103.0/255.0, 183/255.0, 214/255.0}, //
37     { 158.0/255.0, 44/255.0, 47/255.0}, //
38     { 130/255.0, 67.0/255.0, 56/255.0}, //
39     { 116/255.0, 101/255.0, 188/255.0}, //
40 };
41 const float sortiesAttendues[19][1] = {
42     { 1 }, // chaud
43     { 0 }, // froid
44     { 0 }, // froid
45     { 1 }, //chaud
46     { 0 }, //froid
47     { 1 }, //chaud
48     { 1 }, // chaud
49     { 0 }, //froid
50     { 0 }, //froid
51     { 0 }, //froid
52     { 0 }, //froid
53     { 1 }, // chaud
54     { 1 }, // chaud
55     { 0 }, //froid
56     { 0 }, //froid
```

```

57 { 1 }, // chaud
58 { 1 }, // chaud
59 { 1 }, // chaud
60 { 0 } //froid
61 };
62
63 unsigned int couches[] = { 3, 3, 1 };
64 float *sorties;
65 uint16_t rmin, rmax, gmin, gmax, bmin, bmax, cmin, cmax;
66 NeuralNetwork NN(couches, 3);
67
68 void setup() {
69   Serial.begin(9600);
70   tcs.begin();
71
72   // Phase calibration
73   Serial.println("Phase de calibration: Identification du blanc. Entrer 'y' quand c'est OK");
74   while (Serial.read() != 'y') {
75     tcs.getRawData(&rmax, &gmax, &bmax, &cmax);
76     tcs.setInterrupt(false);
77     Serial.print("R : ");
78     Serial.print(rmax);
79     Serial.print(" ");
80     Serial.print("G : ");
81     Serial.print(gmax);
82     Serial.print(" ");
83     Serial.print("B : ");
84     Serial.print(bmax);
85     delay(4000);
86   }
87
88   Serial.println("Phase de calibration: Identification du noir. Entrer 'z' quand c'est OK");
89   while (Serial.read() != 'z') {
90     tcs.getRawData(&rmin, &gmin, &bmin, &cmin);
91     Serial.print("R : ");
92     Serial.print(rmin);
93     Serial.print(" ");
94     Serial.print("G : ");
95     Serial.print(gmin);
96     Serial.print(" ");
97     Serial.print("B : ");
98     Serial.print(bmin);
99     Serial.println(" ");
100    delay(4000);
101  }
102
103 // Entrainement du reseau de neurone NN
104 size_t nbCouleurs = sizeof(entrees)/sizeof(entrees[0]);
105 unsigned long startTime = millis(); // Temps de depart
106 for (int i = 0; i < ITER; i++) {
107
108   for (int j = 0; j < nbCouleurs; j++) {
109     NN.FeedForward(entrees[j]);
110     NN.BackProp(sortiesAttendues[j]);
111   }
112 }
```

```

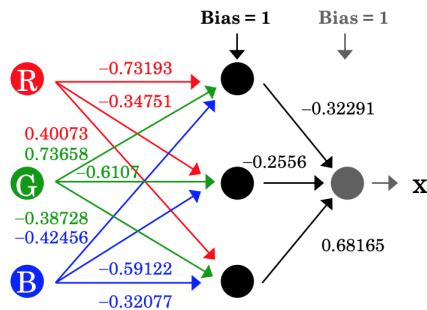
113     unsigned long endTime = millis(); // Temps d'arrêt
114     unsigned long executionTime = endTime - startTime; // Temps d'exécution
115     en millisecondes
116
116     Serial.print("Temps d'exécution de l'entraînement : ");
117     Serial.print(executionTime/1000.0,4);
118     Serial.println(" s");
119     NN.print();
120
121     for (int i = 0; i < 6; i++) {
122         sorties = NN.FeedForward(teste[i]);
123         Serial.print(i);
124         Serial.print(":");
125         Serial.println(sorties[0], 7);
126     }
127     pinMode(Led_froid, OUTPUT);
128     pinMode(Led_chaud, OUTPUT);
129 }
130
131 void loop() {
132
133     float couleurMesuree[3];
134     float chaud_Ou_froid;
135     uint16_t r, g, b, c;
136     // Extraction des valeurs RGB
137     tcs.getRawData(&r, &g, &b, &c);
138     int rmap = map(r, rmin, rmax, 0, 255);
139     int gmap = map(g, gmin, gmax, 0, 255);
140     int bmap = map(b, bmin, bmax, 0, 255);
141
142     // On constraint les valeurs à être dans l'intervalle [0,1]
143     couleurMesuree[0] = constrain(rmap, 0, 255) / 255.0;
144     couleurMesuree[1] = constrain(gmap, 0, 255) / 255.0;
145     couleurMesuree[2] = constrain(bmap, 0, 255) / 255.0;
146     chaud_Ou_froid = NN.FeedForward(couleurMesuree)[0];
147
148     if (chaud_Ou_froid > 0.5) {
149         Serial.println("Chaud");
150         digitalWrite(Led_chaud, HIGH);
151         digitalWrite(Led_froid, LOW);
152         delay(1200);
153     } if (chaud_Ou_froid < 0.5) {
154         Serial.println("Froid");
155         digitalWrite(Led_chaud, LOW);
156         digitalWrite(Led_froid, HIGH);
157         delay(1200);
158     }
159 }
160 }
```

Listing 9 – Code Arduino

6.3 Détection de couleurs chaudes ou froide : Utilisation d'un réseau de neurones déjà entraîné

L'entraînement d'un réseau de neurone peut parfois durer longtemps en raison du nombre d'itérations mais aussi la taille des données d'entraînement. Quand nous voulons des résultats presque instantanés

cela peut causer problème. L'idée est d'utiliser un réseau de neurone déjà entraîné, mais comment faire avec arduino ?



Dans l'apprentissage supervisé, les réseaux de neurones sont formés à partir de données d'entrée et de sorties correspondantes. Les poids et les biais des neurones sont ajustés de manière itérative pour minimiser l'erreur entre les sorties prédites par le réseau et les sorties réelles. La prédition des bons poids et biais est cruciale pour obtenir des résultats précis. En réutilisant la classe **NeuralNetwork Nom(couches, poids, biais, 3)** avec les poids et biais récupérer après apprentissage du réseau, à l'aide de la fonction **NN.print()**, nous pouvons prédire si la couleur mesurée est chaude ou froide presque instantané.

Voici le code complet de cette opération avec des poids et biais satisfaisant.

```

1 #include <NeuralNetwork.h>
2 #include <Wire.h>
3 #include "Adafruit_TCS34725.h"
4 #define ITER 5000
5
6 const int Led_froid = 12;
7 const int Led_chaud = 11;
8
9 Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
10     TCS34725_GAIN_4X);
11 const float biais[] = {1, 0.96};
12
13 // 3*3 + 3*1 [poids entrainnes]
14 const float poids[] = {
15     -8.0463991, -0.0822454, 8.0792446,
16     1.7588860, -0.9593605, -1.5000342,
17     2.7917187, -0.7965161, -3.6734311,
18
19     -12.7535037, 2.4088299, 4.6125631
20 };
21 const float teste [6][3] = {
22     {208.0/255.0, 71/255.0, 80/255.0}, //
23     {41.0/255.0, 90/255.0, 74/255.0}, //
24     {103.0/255.0, 183/255.0, 214/255.0}, //
25     {158.0/255.0, 44/255.0, 47/255.0}, //
26     {130/255.0, 67.0/255.0, 56/255.0}, //
27     {116/255.0, 101/255.0, 188/255.0}, //
28 };
29 unsigned int couches[] = { 3, 3, 1 };
30 float *sorties;
31 float rmin = 13;
32 float rmax = 1953;
  
```

```
33 float gmin = 12;
34 float gmax = 2561;
35 float bmin = 5;
36 float bmax = 2152;
37 NeuralNetwork NN(couches, poids, biais, 3); // Creation d'un reseau de
neurone a partir de donnees pre-entraînées
38
39 void setup() {
40   Serial.begin(9600);
41   tcs.begin();
42
43
44   for (int i = 0; i < 6; i++) {
45     sorties = NN.FeedForward(teste[i]);
46     Serial.print(i);
47     Serial.print(":");
48     Serial.println(sorties[0], 7);
49   }
50   NN.print();
51   pinMode(Led_froid, OUTPUT);
52   pinMode(Led_chaud, OUTPUT);
53 }
54
55 void loop() {
56
57   float couleurMesuree[3];
58   float chaud_Ou_froid;
59   uint16_t r, g, b, c;
60   //float r, g, b, c;
61   //tcs.getRGB(&r, &g, &b); // Extraction des valeurs RGB
62   tcs.getRawData(&r, &g, &b, &c);
63   int rmap = map(r, rmin, rmax, 0, 255);
64   int gmap = map(g, gmin, gmax, 0, 255);
65   int bmap = map(b, bmin, bmax, 0, 255);
66
67   // On constraint les valeurs a etre dans l'intervalle [0,1]
68   couleurMesuree[0] = constrain(rmap, 0, 255) / 255.0;
69   couleurMesuree[1] = constrain(gmap, 0, 255) / 255.0;
70   couleurMesuree[2] = constrain(bmap, 0, 255) / 255.0;
71   chaud_Ou_froid = NN.FeedForward(couleurMesuree)[0];
72
73   if (chaud_Ou_froid > 0.5) {
74     Serial.println("Chaud");
75     digitalWrite(Led_chaud, HIGH);
76     digitalWrite(Led_froid, LOW);
77     delay(100);
78   } if (chaud_Ou_froid < 0.5) {
79     Serial.println("Froid");
80     digitalWrite(Led_chaud, LOW);
81     digitalWrite(Led_froid, HIGH);
82     delay(100);
83   }
84 }
```

Listing 10 – Code Arduino

7 Applications

Un détecteur de couleurs chaudes ou froides peut être utilisé dans de nombreux domaines et applications. Voici quelques exemples où un tel détecteur peut être appliqué :

1. Contrôle de la température des objets : Un détecteur de couleurs chaudes ou froides peut être utilisé pour mesurer la température des objets sans contact. Par exemple, dans l'industrie alimentaire, il peut être utilisé pour vérifier la température des aliments pendant la cuisson ou la réfrigération.



Adafruit Color Sensors

2. Caméras thermiques : Les caméras thermiques utilisent des détecteurs de couleurs chaudes ou froides pour capturer des images thermiques. Elles sont utilisées dans divers domaines tels que la maintenance préventive, la surveillance de la température corporelle, la recherche et le sauvetage, etc



Caméra thermique

3. Contrôle climatique : Dans les systèmes de climatisation, un détecteur de couleurs chaudes ou froides peut être utilisé pour mesurer la température ambiante et ajuster le fonctionnement du système en conséquence.

4. Contrôle de qualité des produits : Dans l'industrie de la fabrication, un détecteur de couleurs chaudes ou froides peut être utilisé pour vérifier la couleur des produits afin de s'assurer qu'ils répondent aux normes de qualité. Par exemple, dans l'industrie textile, il peut être utilisé pour détecter les variations de couleur dans les tissus.
5. Reconnaissance des émotions : Dans les domaines de la psychologie et des sciences cognitives, la couleur peut être utilisée comme indicateur des émotions. Un détecteur de couleurs chaudes ou froides peut être utilisé pour analyser les expressions faciales et déterminer l'état émotionnel d'une personne.
6. Contrôle de la sécurité incendie : Les détecteurs de couleurs chaudes peuvent être utilisés pour détecter les sources de chaleur ou d'incendie potentielles dans les bâtiments. Ils peuvent être utilisés pour surveiller les zones sensibles et déclencher des alarmes en cas de dépassement des seuils de température.
7. Surveillance environnementale : Un détecteur de couleurs chaudes ou froides peut être utilisé dans des études environnementales pour surveiller les variations de température dans les écosystèmes, les cours d'eau ou les zones côtières

8 Conclusion

En conclusion, ce stage m'a offert une occasion précieuse d'approfondir mes connaissances techniques en utilisant Arduino et les réseaux de neurones, tout en explorant les possibilités de détection des couleurs chaudes et froides. Je suis reconnaissant d'avoir pu acquérir ces compétences et je suis enthousiaste à l'idée de les appliquer dans des projets futurs et à ceux qui voudrait en savoir plus.

XLIM-AXE MATHÉMATIQUES ET SÉCURITÉ DE L'INFORMATION
ENCADRANTS: K. TAMINE & P. DUSART
Stagiaire: Cleque Marlain MBOULOU

INTELLIGENCE ARTIFICIELLE EN ENVIRONNEMENT CONTRAINTE

CHAUD OU FROID?

Context

Une bonne compréhension de ce projet nécessite une connaissance de Arduino.
La carte arduino ci-dessus représente un mini ordinateur nous permettant d'interagir avec l'environnement extérieur grâce à des capteurs comme le TCS34725. Ceci est rendu possible par une suite d'instructions programmable (code) en langage arduino (proche de C, C++)

Ceci est rendu possible par une suite d'instructions programmable (code) en langage arduino (proche de C, C++)

Objectifs

L'objectif est de construire un détecteur de couleurs froides ou chaudes à l'aide de arduino, d'entrainer un petit réseau de neurones pour le rentrer autonome, et d'utiliser la version entrainée du réseau de neurone pour prédire (chaud ou froid)

Outils

Le capteur TCS34725 ci-contre est un capteur de couleur développé par Adafruit Industries. Il s'agit d'un capteur de couleur numérique basé sur la technologie des capteurs de lumière ambiante et de couleur RGB.

Applications

Bias = 1 Bias = 1

1. int keyton = 22;
2. void setup() {
3. Serial.begin(56000);
4. analogReference(INTERNAL);
5. pinMode(12, OUTPUT);
6. }
7.
8. void loop() {
9. if (analogRead(keyton) < 10000) {
10. if (digitalRead(keyton) == HIGH){
11. digitalWrite(12, HIGH);
12. } else{
13. digitalWrite(13, LOW);
14. }
15. }
16. }

Biases

Code Arduino

XLIM ACSYON ul

Références

[Intelligence artificielle avec Arduino, Pascal Masson \[en ligne\]](#)

[Tutoriel-1-presentation-de-larduino.pdf \[en ligne\]](#)

[Présentation et principe de l'Arduino \[en ligne\]](#)

[Interfaçage d'un capteur de couleur RVB TCS34725 avec Arduino - Un guide complet \[en ligne\]](#)