CentraleSupélec

# Conception d’une Caméra Thermique

Alexandre LOEBLEIN HEINEN

Clyvian RIBEIRO BORGES

Série A

Groupe 6

Binôme 66

Encadrant : José PICHERAL

Gif-sur-Yvette, le 12 juin 2015

# Table des Matières

# 1. Introduction

Ce projet a pour objectif l'analyse et mise en fonctionnement d'une caméra thermique. Un corps émit toujours un rayonnement infrarouge dont la longueur d'onde dépend de la température de ce dernier. En vue de cela, la caméra thermique consiste d'un système capable de mesurer l'intensité des différentes longueurs d'onde dans le spectre infrarouge émises par un objet.

La bande spectrale infrarouge s'étend de 0.8 à 15 μm, cependant la bande de l'infrarouge moyen est définie par l'ISO 20473:2007 comme la bande des longueurs d'onde entre 3 μm et 6 μm [1] et ce sera à cette dernière bande que nous nous y intéresserons.

La caméra thermique ci-utilisée est munie d'un dispositif optico-mécanique qui est composé par un capteur MLX90620, un tableau de capteurs optiques infrarouges muni d'un système d'amplification, filtrage et de conversion analogique-numérique [2], et qui par le biais de deux servomoteurs permet le balayage d'une scène suivant des axes verticaux et horizontaux. Dans ce cas, le même capteur analyse chaque zone de la scène thermique avec un très léger décalage temporel.

D'abord on décrira les composants qui constituent le système : la mécanique, le capteur, l'acquisition des données, leur transmission, traitement et affichage. La suite de ce rapport est donc dédiée à détailler les procédures adoptées et les connaissances développées. Enfin, on présente les résultats obtenus bien comme les conclusions qu'on en a tiré.

# 2. Description et composants

Cette section a pour objectif décrire les caractéristiques et mode d'emploi des principaux composants du projet.

## 2.1. MLX90620

Le capteur MLX90620 est constitué par deux sous composants : un tableau de 16 x 4 capteurs infrarouges, la puce MLX90670, et la puce 24AA02, une EEPROM de 256 octets. D'ailleurs, le MLX90620 un capteur PTAT (acronyme en anglais de « *proportional to absolute temperature* ») pour mesurer la température ambiante.

Tous les capteurs de température (les infrarouges ainsi que le PTAT) sont déjà calibrés, c'est-à-dire que le fabriquant fournit dans l'EEPROM tous les paramètres nécessaires pour les calculs postérieurs, y compris :

- Calcul de la température ambiante ;
- Annulation des *offsets* ;
- Compensation de la sensibilité des pixels ;
- Compensation de l'émissivité de l'objet ;
- Calcul des températures des pixels ;

En plus, les mémoires du capteur : l'EEPROM pour les paramètres et la mémoire RAM qui porte les données mesurées, une interface par protocole I2C est mise en place. En effet, cette interface reçoit des commandes externes et renvoie les données requises par le bus I2C. Les différentes commandes existantes bien comme les procédures pour les calculs sont présentées dans la Section 0.

La Figure 1 présente la structure du capteur. De même que la présente quelques caractéristiques concertantes la suite du projet.

Tableau 1 Paramètres du MLX90620

| **Paramètre** | **MLX90620** |
|---|---|
| Température ambiante | -40 à 85 ºC |
| Température des objets | -50 à 300 ºC |
| Tension d'alimentation | 2,6 V |
| Courant | 5 à 9 mA |
| Fréquence d'opération | 0,5 à 521 Hz |

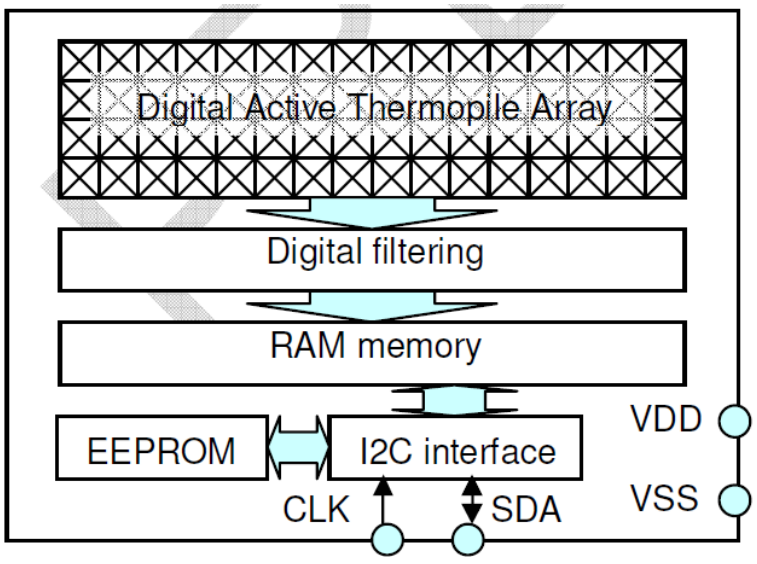


Figure 1 Les différent composants du capteur MLX90620

## 2.2. Arduino UNO

La carte Arduino Uno est une carte à microcontrôleur basée sur l'ATmega328 [3].

Elle dispose :

- de 14 broches numériques d'entrées/sorties (dont 6 peuvent être utilisées en sorties PWM (largeur d'impulsion modulée)),
- de 6 entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques),
- d'un quartz 16Mhz,
- d'une connexion USB,
- d'un connecteur d'alimentation jack,
- d'un connecteur ICSP (programmation "in-circuit"),
- et d'un bouton de réinitialisation (reset).

Elle contient tout ce qui est nécessaire pour le fonctionnement du microcontrôleur ; Pour pouvoir l'utiliser et se lancer, il suffit simplement de la connecter à un ordinateur à l'aide d'un câble USB (ou de l'alimenter avec un adaptateur secteur ou une pile, mais ceci n'est pas indispensable, l'alimentation étant fournie par le port USB). Son brochage est présenté sur la Figure 2.

Synthèse des caractéristiques :

Tableau 2 Caractéristiques de l'Arduino Uno

| | |
|---|---|
| **Microcontrôleur** | ATmega328 |
| **Tension de fonctionnement** | 5V |
| **Tension d'alimentation (recommandée)** | 7-12V |
| **Tension d'alimentation (limites)** | 6-20V |
| **Broches E/S numériques** | 14 (dont 6 disposent d'une sortie PWM) |
| **Broches d'entrées analogiques** | 6 (utilisables en broches E/S numériques) |
| **Intensité maxi disponible par broche E/S (5V)** | 40 mA (ATTENTION : 200mA cumulé pour l'ensemble des broches E/S) |
| **Intensité maxi disponible pour la sortie 3.3V** | 50 mA |
| **Intensité maxi disponible pour la sortie 5V** | Fonction de l'alimentation utilisée - 500 mA max si port USB utilisé seul |
| **Mémoire Programme Flash** | 32 KB (ATmega328) dont **0.5 KB** sont utilisés par le bootloader |

| | |
|---|---|
| **Mémoire SRAM (mémoire volatile)** | 2 KB (ATmega328) |
| **Mémoire EEPROM (mémoire non volatile)** | 1 KB (ATmega328) |
| **Vitesse d'horloge** | 16 MHz |

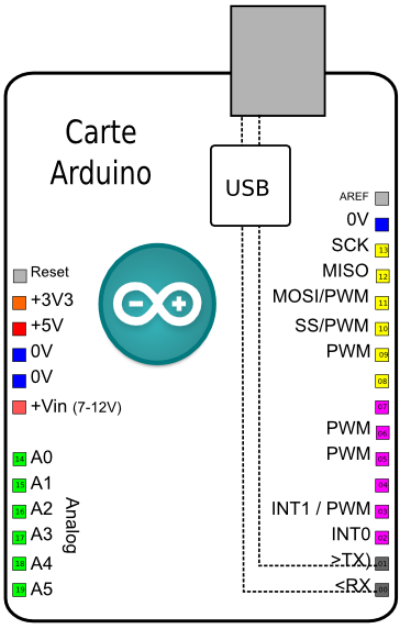


Figure 2 Plaque Arduino UNO

## 2.3. Servo moteurs

Un servomoteur associe :

- une motorisation de puissance qui va nécessiter une intensité de l'ordre de 100mA à plusieurs centaines de mA
- une électronique de commande qui positionne l'axe du servomoteur en fonction de la largeur de l'impulsion PWM reçue sur une entrée numérique qui ne va nécessiter que 1mA.

La tension d'alimentation d'un servomoteur standard doit être comprise entre 4V et 6V continus (variable selon les modèles), régulé ou non. Cet ordre de grandeur de tension est compatible avec les circuits numériques tels que la carte Arduino qui fonctionne en 5V régulé [3].

Ci-dessous, un schème de connexion avec la carte Arduino :

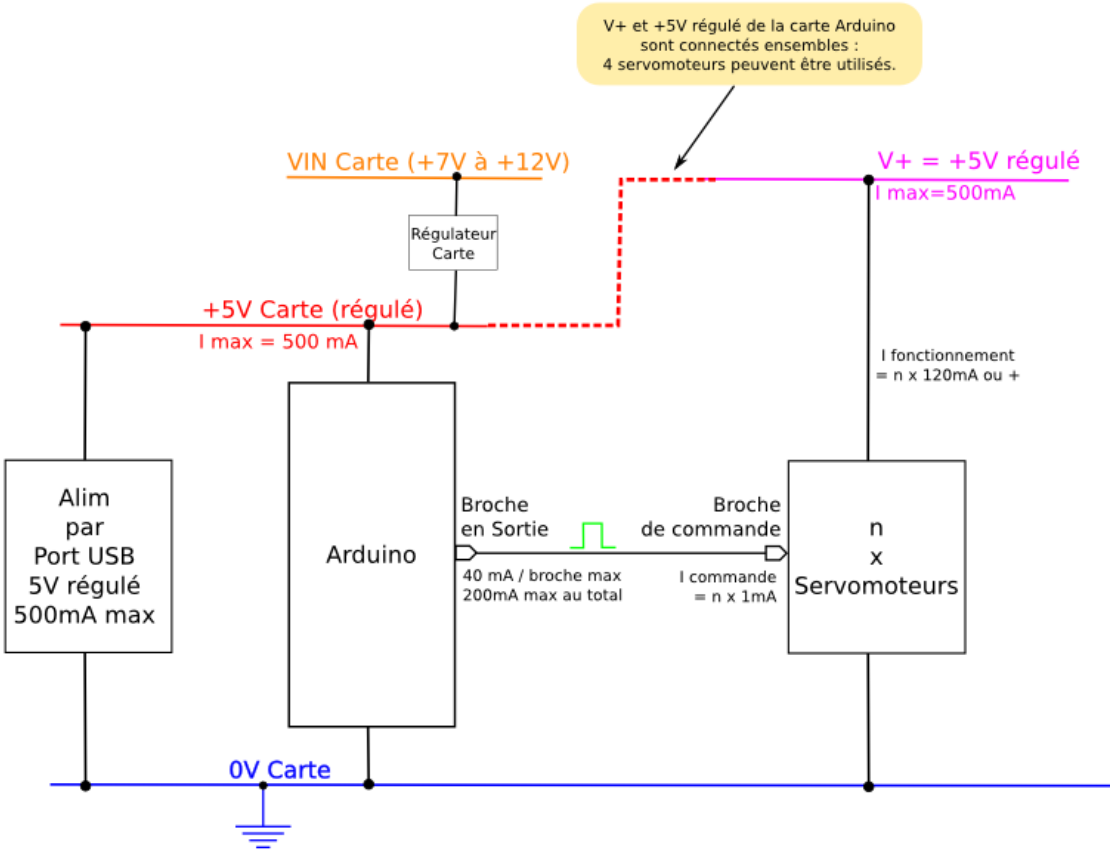


Figure 3 Servomoteur alimenté par une carte Arduino

## 2.4. Assemblage des composants

La Figure 4 représente le schéma du montage : les broches de la plaque Arduino utilisés pour la commande des servos et pour la communication avec le capteur.

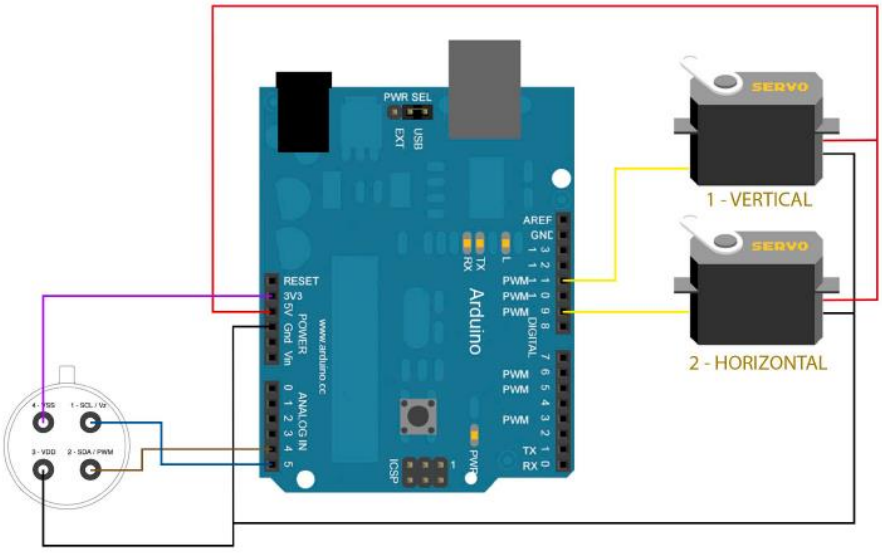


Figure 4 Schéma du montage ; Source : Solano, Sampère, 2014.

La maquette sur laquelle on a mène le projet et les tests est celle de la Figure 5. Sur cette figure, on peut aussi identifier les composants du projet et la façon comme le montage a été fait.

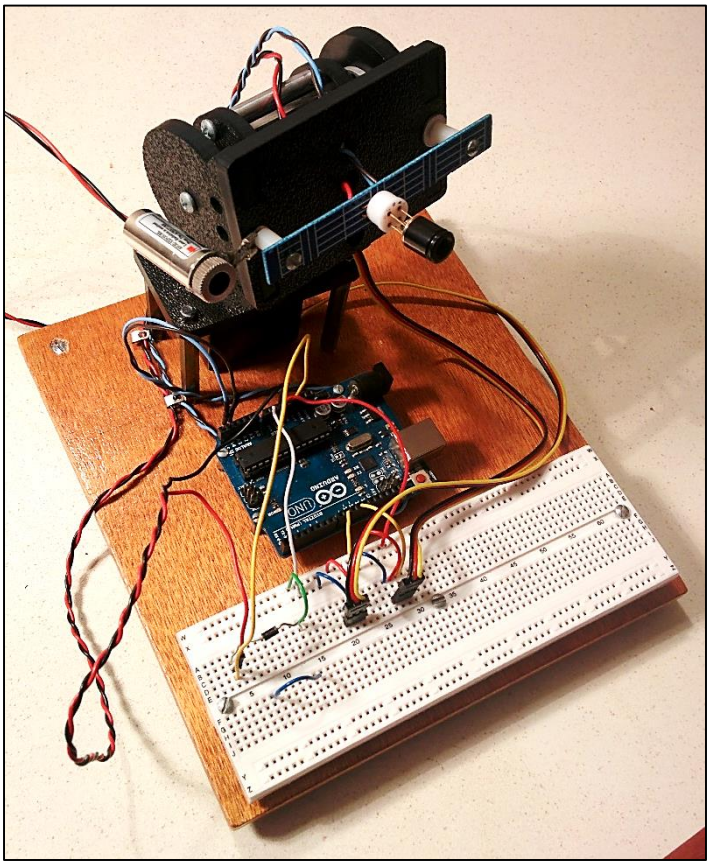

Figure 5 Photo de la maquette

**Remarque :** L'Arduino fournit une source de tension de 3,3 V. Cependant, la tension d'alimentation idéale pour le capteur c'est 2,6 V. Pour adapter ces valeurs, on a ajouté en série de la source de l'Arduino une diode 1N4007 qui cause une chute de tension d'environ 0,6 V.

# 3. Travail effectué

Cette section a pour objectif détailler les activités déroulées pendant le projet ainsi que les outils y utilisés et développés.

D'abord on s'intéresse à la mise en fonctionnement du capteur MLX90620. Dans cette perspective on a conçu la bibliothèque « MLX90620.h » qui contient les fonctions élémentaires pour cette tâche.

Ensuite, cette bibliothèque a donc été utilisée pour la mise en œuvre de la communication I2C entre le capteur et le microcontrôleur Arduino, étant ce dernier aussi

responsable pour les calculs des grandeurs physiques à partir des mesures du capteur, bien comme pour le lien par communication série avec l'ordinateur.

Une fois ces données envoyées à l'ordinateur, on dispose du logiciel MATLAB qui enfin organise, traite et affiche ces informations.

Le système a été d'abord conçu pour une mesure « statique » du capteur, c'est-à-dire qu'on ne s'est intéressé qu'à l'image formé par le capteur. Après, on propose un système de balayage où on actionne les servos moteurs afin qu'on obtienne une image de plus grande amplitude, en produisant plusieurs sous-images à différentes orientations qui sont assemblées dans une même figure.

Les programmes développés pour l'Arduino et sur MATLAB sont classés en deux catégories : ceux pour l'analyse en temps réel (une image) et pour l'analyse avec balayage.

## 3.1. La bibliothèque « MLX90620 »

Afin de permettre une utilisation future simplifiée du capteur MLX90620 pour la suite des travaux, une bibliothèque spécifique pour ce capteur a été développée. Cette bibliothèque appelée « MLX90620.h » contient les fonctions qui réalisent les opérations nécessaires sur le capteur, telles les lectures, écritures, calculs et configurations [4].

Cette sous-section a pour but décrire les fonctions y comprises et les mécanismes du capteur y relatifs.

Selon le *datasheet* du dispositif, la première action après la mise en fonctionnement du système c'est l'initialisation de ce dernier. Cela comprend quatre étapes :

I. Lecture de l'EEPROM afin de récupérer les données de la calibration du capteur. Cela pourrait être fait à chaque fois qu'on en aurait besoin, une réduction importante de la performance en résulterait.
II. Stocker ces données dans la mémoire du microcontrôleur.
III. Ecrire la valeur de la fréquence d'oscillation (extraite de l'EEPROM) dans son registre.
IV. Ecrire les paramètres de la configuration du capteur. Des valeurs sont disposées par l'EEPROM par défaut. Cependant, l'on peut les changer si cela convient.

### 3.1.1. *Lecture*

Il faut noter que le capteur répond par l'I2C en fonction de la commande que lui est envoyée. C'est-à-dire que chaque paramètre va être obtenu en envoyant par l'I2C la commande correspondante.

Le pseudo-algorithme de lecture de données par l'I2C c'est le suivant :

- Initialisation de la communication I2C en envoyant l'adresse de l'esclave
- Envoi de la (des) commande(s) qui indique les données d'intérêt
  - Ca échéant, envoi des paramètres de la lecture
- Lecture de la « réponse du capteur »
- Fin de la communication

Attendu que d'abord il faut lire les données de l'EEPROM, on dispose de la fonction

```
read_EEPROM_MLX90620(byte EEPROM_DATA[256])
```

où les 256 octets de l'EEPROM sont mis dans le vecteur EEPROM_DATA.

La commande pour la lecture de l'EEPROM c'est 0x00. Ensuite le capteur renvoie les 256 octets stockés. Vous pouvez trouver sur l'Annexe A la liste des données que chaque région de l'EEPROM contient.

### 3.1.2. *Registres*

Plusieurs registres sont requis pour les calculs et pour la configuration. Leurs acquisitions sont donc faites à partir de la commande de lecture du capteur – 0x02 – qui demande aussi trois paramètres respectifs : l'adresse initiale de la lecture, le pas de lecture et le nombre de lectures.

Premièrement, l'on peut lire directement le registre de la configuration (étape IV de la configuration), la réquisition c'est d'une seule lecture, celle de l'adresse 0x92. La fonction qui le fait c'est

```
read_Config_Reg_MLX90620()
```

Deuxième, les registres PTAT et CPIX, respectivement la sortie du capteur PTAT et la valeur du pixel de compensation du gradient de température (les calculs des températures les tiendra en compte dans les Sections 3.1.4 et 3.1.5). C'est aussi une lecture unique des adresses 0x90 et 0x91 dont les fonctions

```
read_PTAT_Reg_MLX90620()
```

et

```
read_Config_Reg_MLX90620()
```

Enfin, les derniers registres d'intérêt sont effectivement les données de la caméra thermique, c'est-à-dire la matrice 16x4 qui contient la lecture de la caméra. Pour cela, la lecture débute sur l'adresse 0x00 et les paramètres suivants nous permettent de choisir les pixels qui seront lits. Dans cette problématique on ne s'intéresse qu'à une lecture complète de la matrice, c'est-à-dire, 64 lectures avec un pas d'une adresse. Cette séquence d'instructions peut être trouvée dans la fonction

```
read_IR(int IRDATA[64])
```

qui place les 64 valeurs dans le vecteur IRDATA.

**Remarque :** voyons que l'adresse d'esclave de l'EEPROM et de la mémoire des registres (RAM) ne sont pas pareils. Ce sont donc deux dispositifs différents dont l'adresse est définie dans l'initialisation de la communication I2C.

### 3.1.3. *Configuration et ecriture*

Après que le système soit lancé, il n'y qu'un paramètre qu'on s'intéressera à changer séparaient : la fréquence de l'oscillateur. Pour cela l'on dispose de la fonction

```
config_MLX90620_Hz(int Hz)
```

dont le paramètre « Hz » définit la fréquence elle-même. Les fréquences acceptées sont 32, 16, 8, 4, 2, 1 ou 0 Hz. A défaut la fréquence choisie c'est 1 Hz.

Pour l'écriture on n'a besoin que d'une fonction de caractère auxiliaire pour l'étape III de l'initialisation :

```
write_trimming_value(byte val)
```

La procédure est décrite dans le *datasheet* du capteur et n'est pas de grand intérêt.

### 3.1.4. *Calculs de la température ambiante*

La fonction

```
calculate_TA (float k_t1, float k_t2, int v_th, int PTAT)
```

calcule la température ambiante à partir de ses paramètres. Le *datasheet* du composant définit ces calculs comme

$$T_a = \frac{-K_{T1} + \sqrt{K_{T1}^2 - 4\,K_{T2}[V_{TH} - PTAT]}}{2\,K_{T2}} + 25\,^{\circ}C$$

La Figure 10 présente donc les adresse où l'on trouve les paramètres $K_{T1}$, $K_{T2}$ et $V_{TH}$. PTAT c'est alors la valeur do capteur PTAT.

### 3.1.5. *Calculs des températures des pixels*

Pour chaque valeur de sortie $V_{ij}$ du tableau, on doit calculer la valeur en température équivalente $T_{ij}$. A cet effet, le calcul est constitué de deux grande étapes. La fonction

`calculate_TO (int a_cp, int b_cp, int a_ij[64], int b_ij[64], float alpha_ij[64], float emissivity, float ta, int tgc, int CPIX, int b_i_scale, int IRDATA[64], float v_ir_tgc_comp[64], float temperatures[64])`

utilise tous ses paramètres pour faire les calculs qui suivent et stocker les 64 valeurs trouvées dans le vecteur « temperatures ».

*3.1.5.1. Les compensations*

a. Compensation d'*offset* :

$$V_{ij}^{O} = V_{ij} - \left( A_{ij} + \frac{B_{ij}}{2^{B_{i_{scale}}}} (T_a - 25\,^{\circ}C) \right)$$

b. Compensation du gradient thermique :

$$V_{ij}^{T} = V_{ij}^{O} - \frac{TGC}{32}\, V(CPIX)$$

**Remarque :** CPIX c'est la valeur du « pixel de compensation du gradient de température » et donc V(CPIX) c'est sa valeur compensée en *offset*, où $A_{ij}$, $B_{ij}$ et $\alpha_{ij}$ sont remplacés par $A_{CP}$, $B_{CP}$ et $\alpha_{CP}$.

c. Compensation d'émissivité :

$$V_{ij}^{C} = \frac{V_{ij}^{T}}{\varepsilon}$$

Après toutes les compensations, on a un signal $V_{ij}^{C}$ dit « sans parasites » ou simplement « compensé ».

*3.1.5.2. Le calcul de $T_{ij}$*

Enfin, chaque pixel doit être pondéré par une sensibilité $\alpha_{ij}$ à lui attribuée et la température de ce pixel est donc calculée comme

$$T_{ij} = \sqrt[4]{\frac{V_{ij}^{C}}{\alpha_{ij}} + (T_a + 273{,}15)^4} - 273{,}15$$

Cette valeur est donnée en degrés Celsuis (°C).

**Remarque :** Tous les paramètres ci-employés peuvent être trouvés sur l'EEPROM. Leur disposition sur cette dernière est présentée sur la Figure 11.

### 3.1.6. *Envoie des informations*

Une fois qu'on a calculé les température il faut les envoyer par communication sérial à l'ordinateur. Cette tâche est accomplie par la fonction

```
transmit_temperatures (float temperatures[64])
```

qui transmet les 64 entrées du vecteur « temperature ».

## 3.2. Arduino

A partir de la bibliothèque développée, on met en place le système à partir des commandes envoyées par l'Arduino. L'Arduino aura donc comme tâches :

- initialiser tous les périphériques ;
- balayer les servos afin de construire l'image ;
- acquérir les données du capteur ;
- envoyer l'image capturée et toute autre information dont MATLAB aura besoin pour ses calculs par communication sérial ;

Ces tâches sont donc mises en forme de l'organigramme de programmation présenté sur la Figure 6.

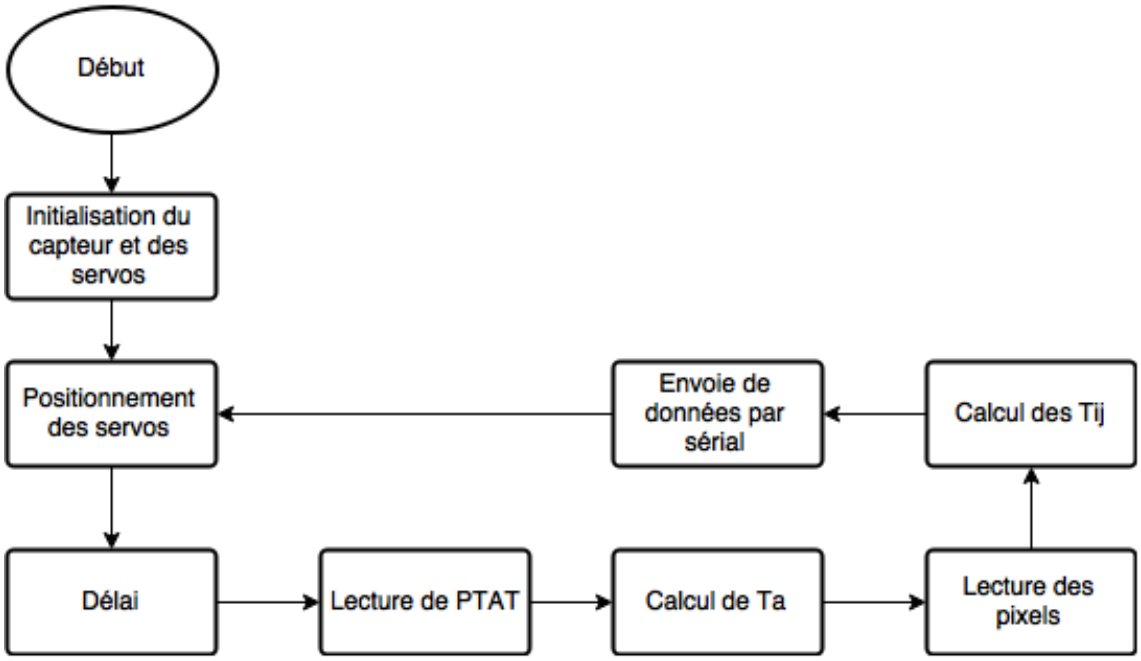


Figure 6 L'organigramme de programmation du code sur l'Arduino

Dans un premier temps, on considère que le système se déroule dans une « boucle » infinie où la fin d'une image s'enchaine directement sur le début de la prochaine.

### 3.2.1. *L'initialisation des périphériques*

Les bibliothèques « Serial.h » et « I2CMaster.h » nous fournissent des fonctions pour l'initialisation directe de la communication serial Arduino – Ordinateur et la communication I2C entre l'Arduino et le capteur.

L'initialisation du capteur suit donc les étapes décrites dans la Section 0 avec les fonctions de la bibliothèque « MLX90620.h » et l'on attache les servos à leurs broches correspondantes à l'aide de la bibliothèque « Servo.h ».

Il faut juste tenir en compte qu'entre la mise en fonctionnement do système et l'initialisation du capteur, il faut impérativement 5ms minimum.

### 3.2.2. *L'initialisation des variables*

Une fois qu'on reçoit les données de l'EEPROM du capteur, on a en effet tous les paramètres nécessaires pour les calculs. Cependant, pour rendre le code plus airé, on a décidé de créer des variables séparées pour chaque paramètre. Dans cette perspective, à partir des informations de l'Annexe A on désigne à chaque variable l'entrée correspondante du vecteur de données de l'EEPROM. Sur le code Arduino, la fonction responsable pour cela c'est

```
varInitialization ()
```

Remarque : Etant donné que les valeurs sont stockées dans l'EEPROM comme octets, les paramètres dont la valeur est un entier y sont stockés en deux entrées différentes : le premier c'est l'octet de poids faible et le second celui de poids fort. Il faut donc les composer dans un entier comme, par exemple,

```
Var = Var_H << 8 + Var_L
```

### 3.2.3. *Le positionnement des servos*

La simple utilisation de la fonction « Servo.write(position) » dans la bibliothèque règle cette problématique. Le calcul des positions des servos à partir de compteurs qui identifient les coordonnées $(i, j)$ de l'image actuelle dans la « matrice d'images » est évidant :

$$\begin{cases} p_1 = p_{10} + \Delta p_1 \, i \\ p_2 = p_{20} + \Delta p_2 \, j \end{cases}$$

où $p_{k0}$ est la position initiale du servo et $\Delta p_k$ est le pas de balayage.

### 3.2.4. *L'envoie des données*

Il faut d'abord remarquer que, soit une matrice $M_{ij}$ de températures générée aux coordonnées $(i, j)$, il faut que cette information « spatiale » soit envoyée avec les données de l'image elle-même. La solution proposée est celle qui suit :

Avant chaque image de 16x4 pixels, on envoie une valeur qui identifie la position des servos. Pour que ce ne soit pas confondu avec une température qui compose une image, il faut une valeur « absurde » de température, disons moins grand que -300.

On pose que le « code de début d'image » est donc

$$c = -(300 + \mathrm{i} + n\,j)$$

où $n$ est le nombre de colonnes de la matrice de images.

Avec ce code, le programme MATLAB ne tiendra les données envoyées comme des températures qu'après avoir reçu une valeur moins grande que -300. Aussi, si l'on pose la valeur de $n$ à priori, c'est facile d'y retrouver les coordonnées.

$$\begin{cases} C &= -(c + 300) \\ i &= C\ (mod\ n) \\ j &= \left\lfloor \frac{C}{n} \right\rfloor \end{cases}$$

## 3.3. MATLAB

Etant l'Arduino responsable pour l'interface avec le capteur et les calculs des températures, il envoie donc ces valeurs calculées par communication série à l'ordinateur, qui sont acquises, traitées et affichées par MATLAB.

Le rôle de MATLAB est donc :

- Recevoir les données envoies par l'Arduino ;
- Mettre ces informations dans une matrice équivalente au format 16x4 du capteur ;
- Faire le traitement sur cette matrice pour réduire le bruit et lisser l'image ;
- Afficher les résultats dans une fenêtre.

Le traitement et l'affichage seront meilleur détailles dans la section suivant.

### 3.3.1. *Traitement des données*

Etant reçu les valeurs numériques de la température, on les met dans une matrice de seize lignes par quatre colonnes (16x4) ou seize lignes par seize colonnes (16x16) dans la mesure avec balayage.

En observant les premiers données, on aperçoit que le bruit qui apparait sur les images est plutôt d'une nature « sel-et-poivre » (en anglais : « salt-and-pepper noise »). Dans ce

genre de bruit est caractérisé par des valeurs dits « aberrants », des valeurs qui ne suivent pas la tendance de l'image.

Le filtre médian permet alors la réduction abrupte de cette famille de bruit. En outre, le filtre médian conserve le contraste et la luminosité de l'image, caractéristique désiré dans cet abordage.

Le filtre médian consiste à remplacer la valeur d'un pixel donnée par la médiane des valeurs de ses voisins, c'est-à-dire, soit $T = (t_{ij})$ la matrice de températures

$$F_m(t_{ij}) = Med\big(\{t_{i+k\ j+l} : -m < k, l < m\}\big)$$

où $a \in \mathbb{N}$ est le paramètre du filtre.

Cependant, on observe aussi que l'application de tel filtre sur notre image de 16x4 pixels n'est pas efficace à cause des dimensions réduites de ce dernier. On propose donc d'auparavant ce filtre, effectuer une expansion de la matrice, où chaque entrée $t_{ij}$ de la matrice est remplacée par une matrice carrée de dimensions $n$ dont les entrées ont toutes la valeur $t_{ij}$. On peut exprimer cette expansion à l'aide du produit de Kronecker [5] comme

$$E_n(T) = T \otimes I_n$$

où $I_n$ c'est la matrice identité de dimension $n$.

En résumé, le traitement de la matrice de températures $T$ est

$$T' = F_m\big(E_n(T)\big)$$

Il faut alors déterminer les valeurs de $m$ et $n$ qui donneront des bons résultats.

### 3.3.2. *Les interfaces graphiques*

Pour commander le début des lectures, pour afficher les résultats et pour choisir les paramètres du traitement des images, on a mis en place des interfaces graphiques sur MATLAB [6].

Voici sur la Figure 7, l'exemple de l'interface pour l'analyse en temps réel. Le bouton « Marche » démarre la lecture, et logiquement le bouton « Arrêt » l'arrête. Les deux versions de l'image sont affichées, celle de la gauche c'est la matrice de températures originale et celle de la droite l'image filtré dont les paramètres « Facteur d'Expansion » $n$ et le « Rayon du Médian » $m$ peuvent être choisis juste en dessous des images.

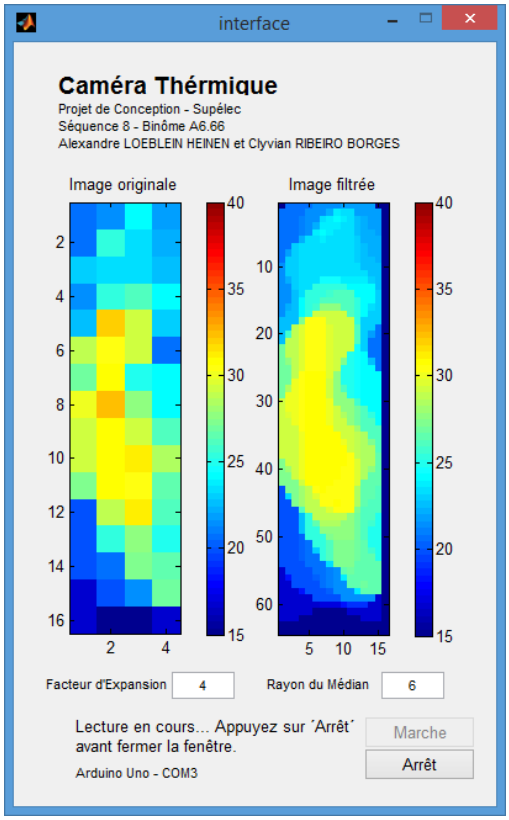


Figure 7 L'interface graphique pour l'analyse en temps réel.

# 4. Résultats

Dans cette section, les principaux résultats obtenus pendant la conception sont présentés.

Premièrement, on présente les résultats concernant l'analyse en temps réel, où a pu observer des applications plus intéressantes. Ensuite, on aborde l'analyse avec balayage.

## 4.1. Temps réel

D'abord, on visualise déjà sur la Figure 7 une comparaison entre l'image obtenue directement des calculs sur l'Arduino et l'image traitée. L'image en question représente une main humaine dans une vue latérale et les paramètres appliqués sont $m = 6$ et $n = 4$.

On voit bien que supprime effectivement les extrêmes présents sur l'image (dans ce cas les pixels oranges) et en plus, que le contours de la main sont beaucoup plus claires que dans l'image à 64 pixels.

En outre, la Figure 8 montre l'évolution dans un court intervalle de temps de la vision de la palme d'une main qui balançait latéralement. Dans cette figure on peut observer l'importance du lissage apporté par le filtre médian car cette évolution ne serait jamais aussi visible dans une image pure.

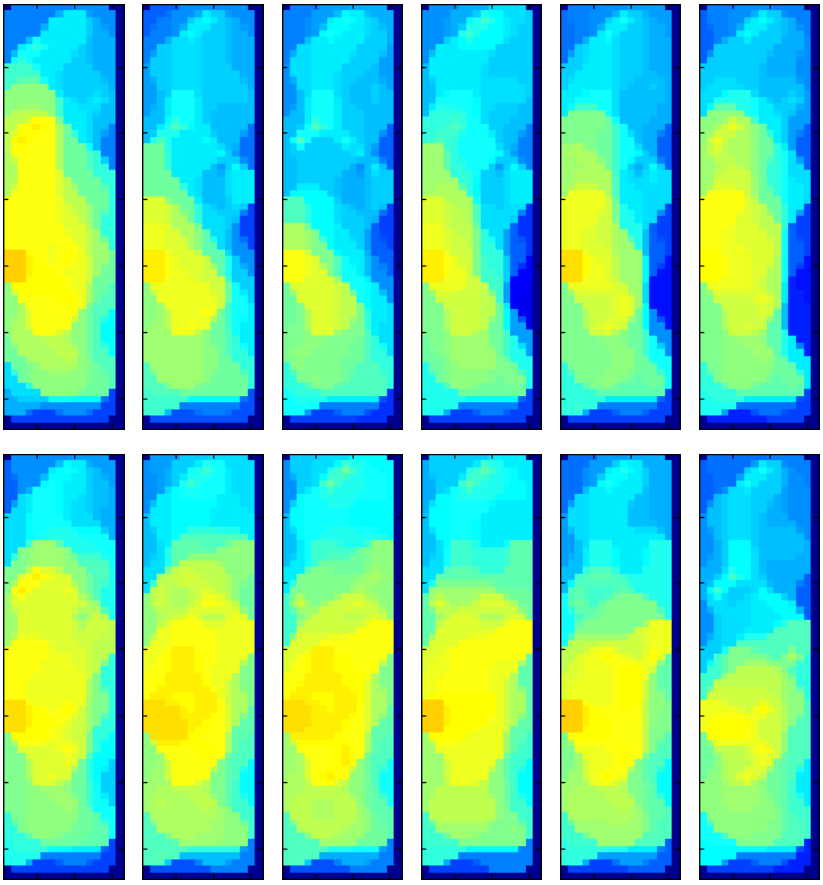

Figure 8 Evolution temporelle d'une image filtrée.

### 4.2. Balayage

Les applications du capteur MLX90620 et les difficultés ci-trouvées sont discutées dans la section suivante, mais pour le balayage il faut observer que ce capteur est plutôt projeté pour un emploi « en temps réel », c'est-à-dire, une analyse d'une seule image qu'il produit.

Dans cette perspective, on observe le résultat montré sur la Figure 9, où on ne peut distinguer une image claire, même si l'objet face à la caméra était un étudiant.

D'ailleurs on peut visualiser sur cette image l'interface graphique développé pour cette analyse avec balayage et encore une fois la grande réduction d'extrêmes apportée par le filtre.

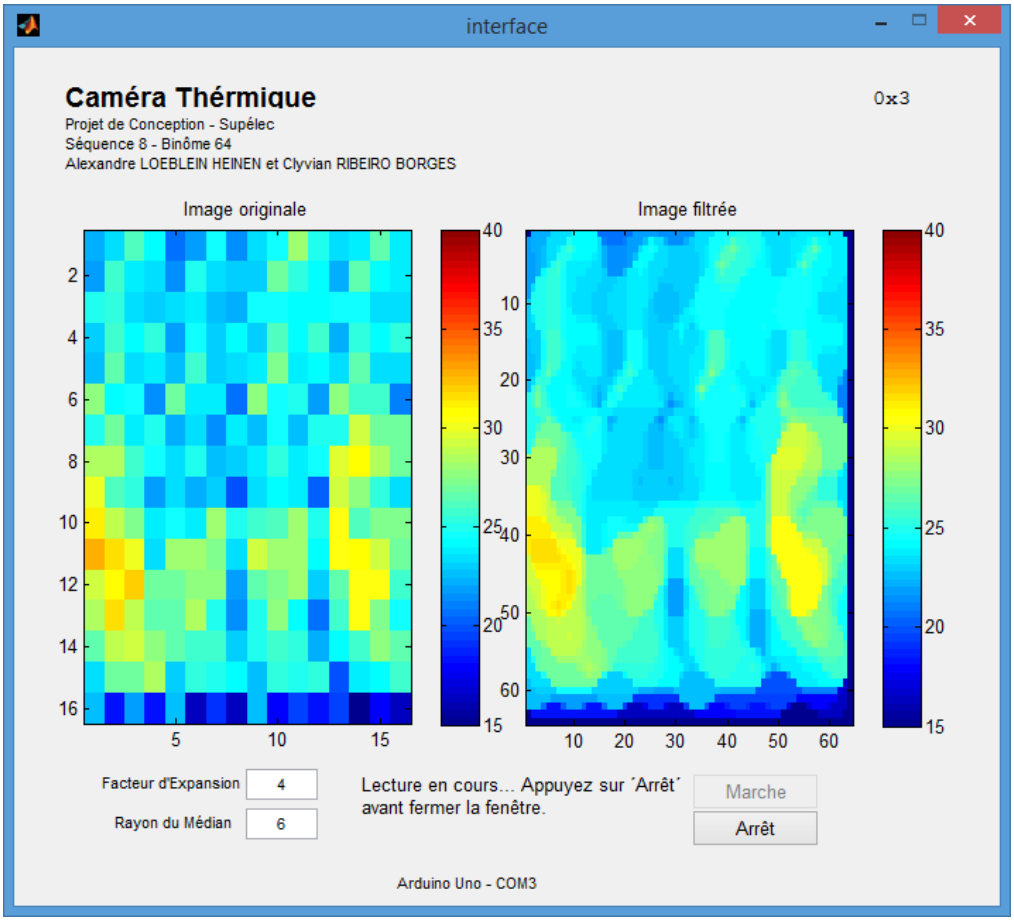


Figure 9 L'interface graphique pour l'analyse avec balayage.

## 5. Conclusions et perspective

Le premier objectif de ce projet c'était de rendre l'utilisation du capteur MLX90620 simple pour les groupes qui mèneront des projets sur ce même sujet. Dans cette intention, la bibliothèque créée a proportionné une versatilité pour l'emploi de ce capteur. Pour les projets futurs il suffira d'appeler les fonctions fournies par la bibliothèque selon les besoins. Les seules opérations qui concernent le capteur et qui restent « hors bibliothèque » c'est l'identification des différentes variables téléchargées de l'EEPROM.

En ce qui concerne les images extraites du capteur, on observe une tendance des températures dans certaines régions de matrice. Cela doit être sans doute cause par de gradient de température y existants. Il faut toujours comprendre que la précision est très sensitive aux conditions d'équilibre thermique et d'isotherme.

Ce gradient de température peut être causé (entre autre motifs) par l'échauffement des composants électroniques intérieurs au capteur ou même par les servos moteurs.

Spécialement dans la région tout en bas des images on aperçoit une « ligne morte » de pixels qui est, dans la plupart du temps, en bleu, c'est-à-dire qu'on en détecte des basses températures

Quelques autres facteurs qui peuvent apporter des erreurs inconnus aux mesures :

- Contamination du chemin optique du capteur ;
- L'absence de dissipation thermique dans le système, notamment dans le cas du balayage, le frottement des moteurs peut causer une augmentation locale de la température et créer des gradients ;
- L'alimentation de l'Arduino n'est pas très stable et, en plus, il n'y a pas de condensateurs de découplage de l'alimentation, ce qui peut introduire énormément de bruit sur les mesures ;

Pour nos analyses le réglage préféré pour les paramètres du filtre pendant la conception c'était un expansion de la matrice de dimension $n = 4$ et un filtre médian d'un rayon $m = 4$.

### 5.1. Pour les travaux futurs

En premier lieu, aucun calcul concernant les caractéristiques physiques du capteur n'a été fait. Ces calculs seraient intéressants surtout pour l'acquisition en balayage.

Le capteur utilisé a un champ de vision de 60º dans le sens le plus large, ce qui résulte dans un champ de vision de 16,4º dans l'autre sens. Pour le balayage il faudra donc trouver des pas de mouvement pour les servos moteurs qui permettra une juxtaposition ou une superposition (il faudra aussi tenir compte des positions des images dans leur assemblage afin de former une figure finale). Un balayage efficient est toujours manquant.

La question des calculs des températures est aussi toujours relevante. Les fabriquant indiquent les faire sur le microcontrôleur, cependant à cause de la relativement faible puissance de calcul de l'ATmega328, l'envoi auparavant des paramètres et l'exécution des calculs sur MATLAB peut aussi améliorer la performance du système.

## 6. Bibliographie

[1] I. 20473:2007, «Optique et photonique - Bandes spectrales,» 15 avril 2007. [En ligne]. Available: http://www.iso.org/iso/fr/home/store/catalogue_tc/catalogue_detail.htm?csnumber=39482. [Accès le 4 avril 2015].

[2] Melexis, “MLX90620: 16X4 Far InfraRed Array,” 20 septembre 2012. [Online]. Available: http://www.melexis.com/Infrared-Thermometer-Sensors/Infrared-Thermometer-Sensors/MLX90620-776.aspx.

[3] X. Hinault, “Mon Club Elec,” mai 2011. [Online]. Available: http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php.

[4] Arduino, “MLX90620 / MLX90621 - 16X4 pixel IR thermal array,” 8 10 2012. [Online]. Available: http://forum.arduino.cc/index.php?topic=126244.0.

[5] E. W. Weisstein, “Kronecker Product,” [Online]. Available: http://mathworld.wolfram.com/KroneckerProduct.html.

[6] MATLAB, “Interfaces graphiques MATLAB,” [Online]. Available: http://fr.mathworks.com/discovery/matlab-gui.html. [Acesso em 2015 avril 5].

# Annexes

# A. Division de l'EEPROM

| EEPROM address | Cell name | Stored as | Parameter |
|---|---|---|---|
| 0xDA | $V_{TH}$_L | 2's complement | $V_{TH0}$ of absolute temperature sensor |
| 0xDB | $V_{TH}$_H | | |
| 0xDC | $K_{T1}$_L | 2's complement | $K_{T1}$ of absolute temperature sensor |
| 0xDD | $K_{T1}$_H | | |
| 0xDE | $K_{T2}$_L | 2's complement | $K_{T2}$ of absolute temperature sensor |
| 0xDF | $K_{T2}$_H | | |

Figure 10 Paramètres de l'EEPROM pour le calcul de $T_a$

| EEPROM address | Cell name | Stored as | Parameter |
|---|---|---|---|
| 0x00…0x3F | $A_{i(i,j)}$ | 2's complement | IR pixel individual offset coefficient |
| 0x40…0x7F | $B_{i(i,j)}$ | 2's complement | Individual Ta dependence (slope) of IR pixels offset |
| 0x80…0xBF | $\Delta\alpha_{(i,j)}$ | unsigned | Individual sensitivity coefficient |
| 0xD4 | $A_{CP}$ | 2's complement | Compensation pixel individual offset coefficients |
| 0xD5 | $B_{CP}$ | 2's complement | Individual Ta dependence (slope) of the compensation pixel offset |
| 0xD6 | $\alpha_{CP\_L}$ | unsigned | Sensitivity coefficient of the compensation pixel |
| 0xD7 | $\alpha_{CP\_H}$ | | |
| 0xD8 | $TGC$ | 2's complement | Thermal gradient coefficient |
| 0xD9 | $B_{i\_SCALE}$ | unsigned | Scaling coefficient for slope of IR pixels offset |
| 0xE0 | $\alpha_{0\_L}$ | unsigned | Common sensitivity coefficient of IR pixels |
| 0xE1 | $\alpha_{0\_H}$ | | |
| 0xE2 | $\alpha_{0\_SCALE}$ | unsigned | Scaling coefficient for common sensitivity |
| 0xE3 | $\Delta\alpha_{SCALE}$ | unsigned | Scaling coefficient for individual sensitivity |
| 0xE4 | $\varepsilon\_L$ | unsigned | Emissivity |
| 0xE5 | $\varepsilon\_H$ | | |

Figure 11 Paramètres de l'EEPROM pour le calcul de $T_{ij}$