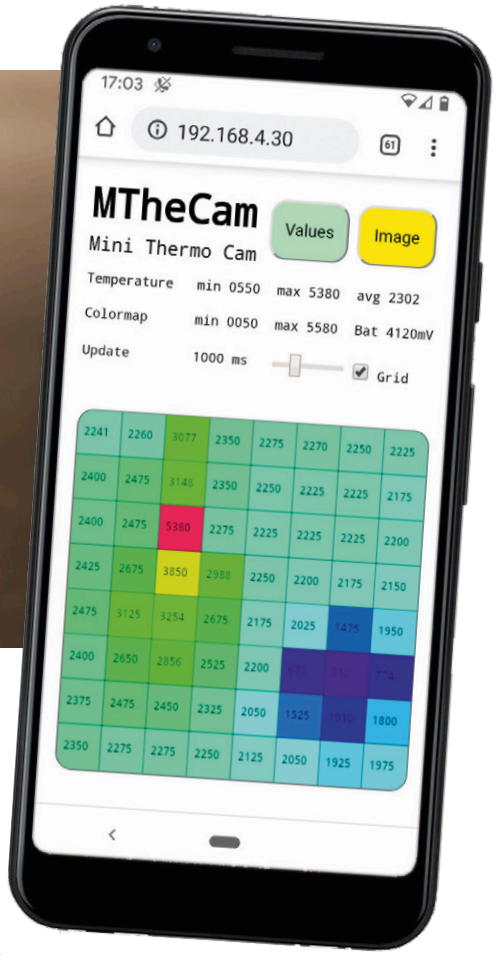
mini caméra thermique
MTheCam

Minicaméra thermique simple pour localisation de points chauds et froids

Olaf Mertens

Ai-je soufflé la bougie en partant ? Et la plaque chauffante, est-elle éteinte ? Qui n'a pas déjà quitté son logis avec la crainte d'avoir oublié un appareil susceptible de provoquer de gros dégâts hors surveillance ? Il est désormais possible d'interroger MTheCam par ordiphone. Bien entendu, le projet présenté ici, avec son capteur thermique à 8x8 pixels de *Panasonic*, se prête à bien d'autres applications.



Le mot *hot-spot* (= point chaud) nous fait plutôt penser aux points d'accès publics à l'internet, mais ici ce terme désigne des points dont la température dépasse sensiblement celle de leur environnement (et inversement pour *cold-spots*). Il peut s'agir de flammes, de points de surchauffe, de courts-circuits, ou inversement, de défauts d'isolation thermique. Ces points chauds, s'ils ne se manifestent pas par des flammes ou des braises incandescentes, restent invisibles à l'œil humain. Pour les détecter, il faut un capteur, comme MTheCam. La localisation de points chauds ne se limite certes pas à des choses comme une plaque chauffante oubliée. Dans les montages électroniques, un échauffement anormal de composants surchargés peut annoncer très tôt leur décès prochain. Sur les machines, les frottements de paliers usés ou de surfaces mal lubrifiées se traduise par un échauffement excessif ; une maintenance au bon moment prolongera leur durée de vie. Même les humains peuvent être reconnus comme points chauds, on

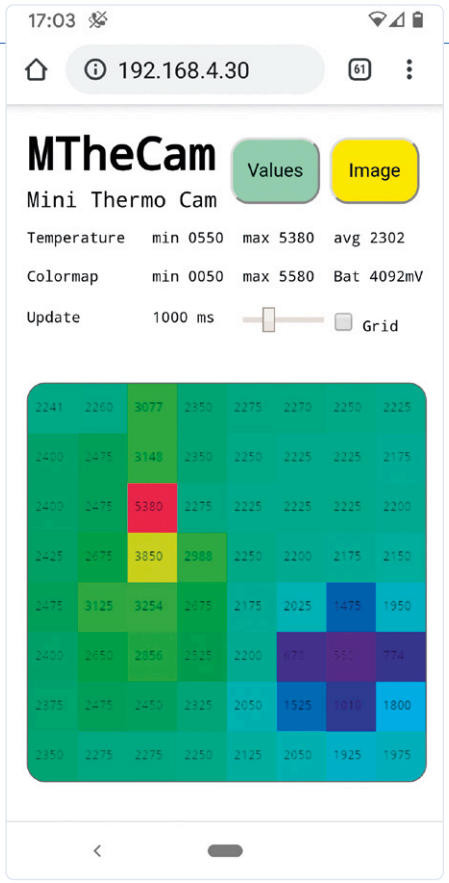


*Figure 1. Saisie d'écran des valeurs de température sur 8*8 pixels.*

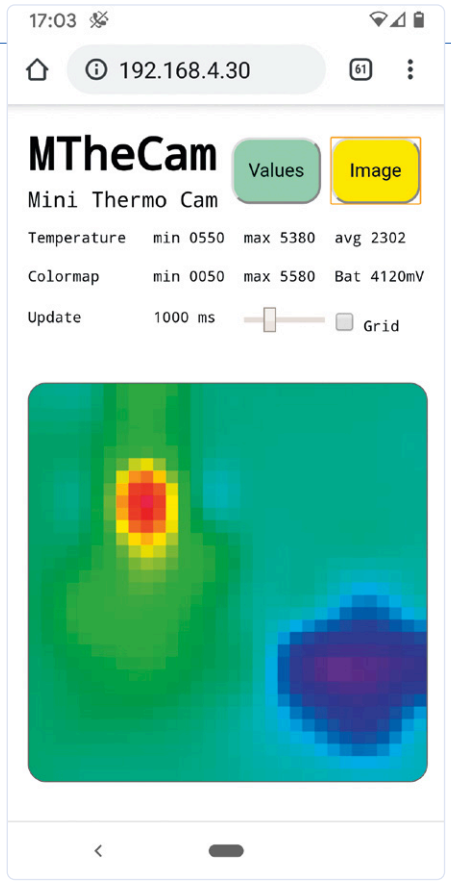


*Figure 2. Saisie d'écran des valeurs interpolées sur 32*32 pixels.*

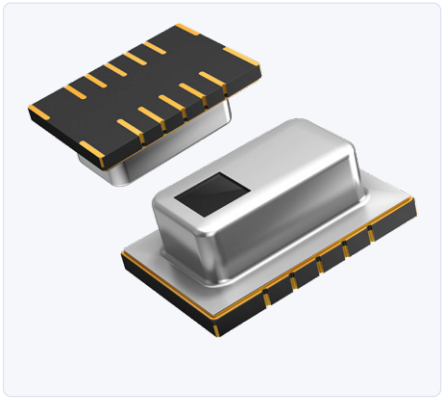


Figure 3. Le capteur AMG88xx de Panasonic.

peut ainsi les compter ou suivre leurs mouvements. Outre les appareils de mesure de température par contact direct, il existe des capteurs sans contact, sensibles au rayonnement infrarouge des objets et en déduisent la température moyenne du champ de mesure. Ils sont basés sur l'effet pyroélectrique qui modifie le potentiel électrique d'un cristal polarisé pris entre deux électrodes quand il est exposé à un rayonnement thermique. Cet effet est évidemment utilisable en électronique [1].

Projet avec MTheCam

MTheCam mesure simultanément 64 points jusqu'à cinq fois par seconde sur huit lignes et colonnes, disposées comme dans une caméra optique avec un angle de vue de 60 °. Chaque point indique la température de 0 à 80 °C (ou de –20 à 100 °C). Aux différentes valeurs de mesure sont attribuées des couleurs d'une palette, de sorte qu'on obtient une image à assez basse résolution, affichable par Wi-Fi sur un ordiphone. Cet amas de pixels colorés n'impressionnera sans doute pas les habitués de la TVHD, mais il affiche les points chauds ou froids au milieu d'un environnement de couleurs nettement différentes. Outre cette image en fausses couleurs, la valeur de mesure de chaque pixel est affichée en °C, ce qui permet une analyse plus précise (**fig. 1**). Les 64 valeurs de mesure sont lisibles sous forme d'un enregistrement de données JSON, utilisable comme source de données pour d'autres applications.

La matrice de 8x8 points est transformée en «fausse» image de 32x32 points de couleur par interpolation bicubique [2], plus agréable à regarder et où les objets sont plus faciles à identifier. Le spectre de l'échelle de couleurs est adapté au domaine des températures mesurées, du minimum au maximum avec un peu de jeu ajouté, ce qui produit un effet de loupe sur les valeurs de température (**fig. 2**).

Capteur de température AMG88xx

Pour la prise d'une image thermique, *Panasonic* commercialise un capteur performant (**fig. 3**) en technologie MEMS (MicroSystème ElectroMécanique) en deux variantes (deux domaines de mesure), qui réunit l'optique, le traducteur thermoélectrique, la mesure analogique et d'autres fonctions de préparation des données dans un boîtier relativement petit [3]. Les valeurs de mesure dans le domaine 0 à 80 °C (AMG8853) ou –20 à 100 °C (AMG8854), qui présentent une erreur maximum de +/–2,5 K (+/–3,0), sont lues via une interface I²C. Dans l'absolu, la précision n'est pas impressionnante, mais suffisante pour l'évaluation qualitative de valeurs relatives. Avec quelques lignes de code, on peut y ajouter un étalonnage qui augmente la précision et diminue le bruit de fond. Le boîtier CMS peu encombrant du capteur *Panasonic* supporte le soudage par refusion. Pour notre application, il trouve place sur une petite carte d'essai maison qui comprend quelques autres composants passifs (au format 0805 soudable à la main) nécessaires au découplage de la tension d'alimentation et au rappel des lignes du bus I²C. Outre la tension d'alimentation (+5 V) et la masse, trois lignes importantes (INT et le bus I²C) sont amenées sur des pastilles au bord de la carte pour une barrette à broches droites ou coudées. La **fig. 4** montre le schéma de cette carte et révèle quelques détails du circuit interne du capteur. Sur la page du projet [9] on peut télécharger gratuitement le tracé des deux faces de la carte et le plan d'implantation des composants. Pour plus de confort, on peut se procurer le module tout équipé chez l'auteur.

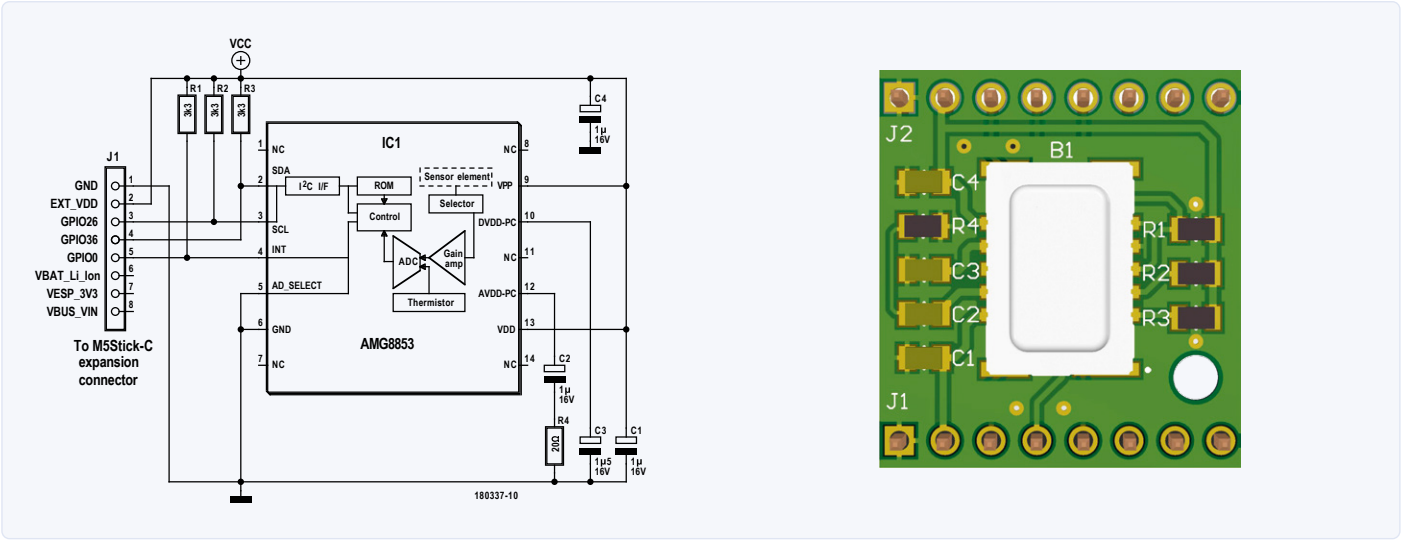


Figure 4. Schéma interne et périphérique du capteur AMG88xx sur la carte d'essai.

M5StickC, circuit à tout faire

Chaque ordiphone est compatible Wi-Fi et possède un écran tactile pour la lecture et les saisies ; c'est l'interface idéale pour notre MTheCam avec laquelle il peut établir une communication sans fil. Pour cela, le capteur doit être équipé d'un transmetteur de données intelligent : un système sur une puce (SoC) avec contrôleur et module Wi-Fi. Notre choix s'est porté sur l'ESP32 d'Espressif, doté de tout ce qui peut séduire un développeur, bon marché, peu gourmand et facile à programmer avec l'EDI Arduino. Outre le capteur et le module ESP, il ne manque que l'alimentation sous 5 V que peut fournir un port USB ou une batterie en opération autonome. Dans la boutique en ligne d'Elektor, mon attention a été attirée par un produit paré d'orange vif, le M5StickC qui offre en sus de l'ESP32 quelques fonctions intéressantes. Ce petit appareil qui ne mesure que 50x26x14 mm en a dans le coffre ! Outre l'ESP-32Pico avec Flash de 4 Mo et SRAM de 520 Ko, il possède un écran de 2,5 cm (0,96") et 80x160 pixels, un accéléromètre à six axes, une horloge en temps réel, une unité de gestion de l'alimentation, une LED rouge, une LED infrarouge pour la télécommande, un microphone MEMS, une batterie de 80 mAh, l'USB-C, un connecteur Grove (alimentation, masse et deux ports I²C) et une barrette à 8 broches femelles avec trois ports et l'alimentation. Voilà de quoi émoustiller l'électronicien le plus blasé !

Avec sa petite taille, son afficheur et son alimentation autonome, le M5StickC peut se faire passer pour une montre chic. Avec son support et son bracelet orange vif, on est certain de se faire remarquer (**fig. 5) !** Après ça, qui voudrait encore mettre à son poignet un quelconque bijou d'un producteur de pommes californien ? Sur lequel on ne peut même pas brancher quoi que ce soit ! Dans ce projet nous n'utiliserons que l'ESP32, l'écran et la batterie, mais les autres fonctions seront certainement les bienvenues dans de futures applications.

En dépit de sa petite taille, l'image sur l'écran du M5StickC n'en est pas moins nette et en couleurs. Pour une image thermique, c'est bien ; pour quelques lignes de valeurs de mesure, aussi. Le défi pour le concepteur sera de faire tenir davantage d'informations sur cette surface minuscule. La bibliothèque graphique fait une grande place aux jeux et aux animations colorées. Elle donne aux inventeurs de nombreuses possibilités d'exercer leur créativité.

Avec ses 80 mAh, la batterie Li-Ion peut alimenter le M5StickC à pleine charge pendant près d'une heure. Elle se recharge via une prise USB-C de 5 V / 500 mA, qui reçoit le câble fourni, à brancher d'autre part sur une banale fiche USB-A. Une pression sur un petit bouton latéral allume le M5StickC, une pression pendant six secondes l'éteint.

Coopération

Selon que la barrette utilisée aura les broches droites ou coudées, la position de la carte du capteur sera parallèle ou perpendiculaire (**fig. 6**) au plan du M5StickC. Avec une imprimante 3D, il est facile de créer de jolis boîtiers pour les deux variantes en fonction de l'axe de visée que vous préférez.

Le connecteur d'extension du M5StickC possède huit broches affectées à la sortie 5 V, à la masse, au signal d'interruption INT (GPIO36) et aux deux lignes I²C de signaux de données et d'horloge (SDA sur GPIO26 et SCL sur GPIO0). Les trois autres broches (BAT, 3V3 et l'entrée 5 V) restent libres. Remarquons que GPIO36 ne peut être utilisée qu'en entrée, ce qui évite bien du débogage.

Pour l'instant, la possibilité d'interruption n'est pas exploitée par le logiciel. Elle permettrait, par ex., de programmer la puce du capteur pour émettre un signal INT lorsqu'un seuil est franchi dans un sens ou dans l'autre, et ce pour chaque pixel individuel ! La puce pourrait alors surveiller toute seule une certaine partie de son champ de vision et ne réveiller l'ESP32 que lorsqu'un seuil est franchi, ce qui économiserait de l'énergie. Le processus de surveillance pourrait être géré ensuite par l'application sur l'ordiphone exploitant l'image entière.

Question de logiciel

Pour le développement du micrologiciel, au lieu de l'environnement propriétaire d'Espressif, on peut utiliser l'écosystème Arduino, très intéressant pour son support très efficace par une communauté mondiale. L'EDI Arduino est excellent pour démarrer mais rapidement insuffisant pour des applications exigeantes. C'est pourquoi l'auteur recommande l'usage de l'éditeur de code libre du Visual Studio de Microsoft, qui possède une extension supportant l'Arduino [6]. *M5Stick-C (esp32)* est ajouté par le gestionnaire des cartes.

L'auteur a utilisé ces outils pour écrire le micrologiciel en C++. Au

fichier source *MTheCam_LT.ino,* s'ajoutent quelques fichiers .h et .cpp (*Mxxx.cpp/h*), ainsi que la bibliothèque *ArduinoJson* hautement recommandée (au moins la V6, la V5 ne fonctionne pas ici !) pour le traitement JSON [7]. La bibliothèque *M5StickC* [8] gère les particularités matérielles sans problème. Elles doivent être ajoutées via le gestionnaire de bibliothèques (*Outils > Gérer les bibliothèques*).
Contrairement à la superbe documentation de l'*ArduinoJson-Lib,* la description de la bibliothèque *M5StickC* laisse beaucoup à désirer. Afin de l'utiliser correctement, une laborieuse lecture du code source s'impose. Cet épineux sujet mériterait un article séparé !
Voyons quelques fragments de code pour différentes fonctions dans l'ordre logique.

Lecture des capteurs
Les valeurs mesurées actuelles sont rafraîchies par le capteur dix fois par seconde dans des registres d'un octet, que le logiciel lit en permanence via l'interface I²C *Wire*. En raison de la résolution de 12 bits, il faut lire deux octets par pixel. C'est ce que fait *Wire* dans `MTC_readReg():`

```
#define BUF_LENRX 128
int reg = 0x80;
byte _rxBuf[BUF_LENRX];
. . .
Wire.beginTransmission(devAddr); // Chip-Address @
   datasheet
Wire.write(reg); // 0x80 -> read 128bytes of 64
   pixels @12bit
Wire.endTransmission();
if (Wire.readTransmission(devAddr, rxBuf, BUF_
   LENRX) == I2C_ERROR_OK) { success = true; }

Wire.endTransmission();
```

En cas d'échec de la lecture, `Wire.readTransmission()` peut éventuellement retourner des codes d'erreur utiles pour le débogage.
Pour la lecture du capteur et le calcul des valeurs de température, on se reportera au **listage 1**.
À partir de deux entrées du tableau `rxBuf[]`, et, moyennant quelques décalages de bits, on arrive à calculer la valeur de la température – ici sous forme d'un nombre entier en centièmes de degré Celsius : pour 21,35 °C on a donc la valeur 2135. Le capteur est initialisé de telle manière qu'il forme en interne la moyenne mobile de deux images et réduit ainsi un bruit un peu fort. Tous les registres sont lus en une seule fois, de sorte que toutes les valeurs sont garanties appartenir à une même fenêtre de mesure. Les autres calculs de la température des pixels et de la puce (non utilisée ici) diffèrent par un facteur (voir la fiche technique [3]). Pour plus de précision de calcul, on utilise la multiplication par 10000, divisée par 100 pour obtenir le 100ᵉ de degré.

Figure 5. Le M5StickC comme montre-bracelet orange vif (source: m5stack.com)

Figure 6. Droit ou coudé : deux manières de connecter le capteur au M5StickC.

Calcul des valeurs de couleur
Pour la représentation des couleurs, on utilise le diagramme HSL [10] (**fig. 7**). Pour cela, la collection M5Stick-C contient une bibliothèque TFT correspondante. Des paramètres teinte (Hue, H), saturation (Saturation, S) et luminosité (Lightness, L), seul le paramètre H est utilisé, S et L sont ignorés (constants). Les valeurs HSL sont converties en valeurs RGB pour un affichage ultérieur.
Comme nous résolvons la plage de 0 à 80 °C en 8000 valeurs mesurées (100ᵉ de Kelvin) pour l'affichage numérique, celles-ci doivent d'abord être normalisées au domaine des valeurs de teinte possibles entre 0 et 359. Comme nous avons l'habitude d'associer le bleu au froid et le rouge au chaud, le grand arc de cercle entre 255 degrés (t_{Min} = bleu-vert) et 315 degrés (t_{Max} = rouge vif) semble être optimal pour l'affichage. Comme nous n'avons pas l'habitude d'associer le violet à une sensation de température, les valeurs de violet sont ignorées. Ainsi, des 360 valeurs de couleur possibles, 300 sont utilisées et 60 ne le sont pas.
La fonction `map()` (**listage 2**) applique les valeurs de température dans l'angle de 300 degrés compris entre 315 et 15 degrés de l'espace des teintes dans le sens antihoraire. Puis elle tourne le tout de 60 ° vers la gauche (angle de –60 °) pour finalement atteindre les bornes 255 et

Listage 1. Lecture du capteur et calcul de la température.

```
. . .
 // read one frame with 64 temp-values à 12bit -> join 2 bytes to 1 int
 // result is int[] with calculated values -> _frame[]
 int  _frame[FRAMESIZE];          // filled by MTC_getFrame()
. . .
 int* MTC_getFrame()
  {
    byte _rxBuf[BUF_LENRX];
    byte dL[FRAME_SIZE];  // low byte
    byte dH[FRAME_SIZE];  // high byte
    int reg = 0x80;       // start with this reg for next 128 bytes to read
    . . .
    // sensor-read 128 bytes - low + high
    boolean ok = MTC_readReg(MTCADDRESS, reg, _rxBuf, 128);  // readings now in _rxBuf
    if (ok)
    {
        int k = 0;
        for (int i = 0; i < FRAME_SIZE; i++)
        {
            int index   = i;
            int d1 = _rxBuf[k++];                // low byte here
            int d2 = _rxBuf[k++];                // high byte next: includes sign!
            // sensor on PCB is upside down, start from the end
            index  = (FRAME_SIZE - i - 1);
            _frame[index] = MTC_calcTemp(d1, d2); // 2 bytes calc'd to one int
        }
     }
    return _frame;
 }
. . .
 // make one int from low/hight bytes d1/d2
 int MTC_calcTemp(uint8_t d1, uint8_t d2)        // return temp in deg Celsius * 100
  {
    int factor = (int) (0.25 * 10000);           // pixel-temp, for device-temp use 0.0625
    if (d2 & 0b00001000) d2 |= 0b11111000;       // handle sign-bit #4 in high-byte
    int valRaw = (d2 << 8) | d1;                 // merge to one int …
    int valCalc = (int)(valRaw * factor / 100);  // … with this factor
    return valCalc;
 }
. . .
```

Listage 2. Calcul des valeurs de couleur.

```
#define MC_COLORWHEEL_LOW  315     // degrees on colorwheel
#define MC_COLORWHEEL_HIGH  15
#define MC_DIFFCW           60     // diff in degrees high to low
. . .
int MC_getColorHSL(int value, int vmin, int vmax, int vavg)
{
    // do not use 255 - 315 deg = 60 deg -> it's not clearly 'felt' as cold or hot
    // vmin/vmax for minimum spread of spectrum -> less color flicker if no spots seen
    if((vavg - vmin) < MC_TMIN_SPREAD2AVG)  { vmin = vavg-MC_TMIN_SPREAD2AVG; }
    if((vmax - vavg) < MC_TMIN_SPREAD2AVG)  { vmax = vavg+MC_TMIN_SPREAD2AVG; }
    // map range vmin -> 315, vmax -> 15 degree 'left turn' - exclude 315...15 = 60 deg
    int colInd = map(value, vmin, vmax, MC_COLORWHEEL_LOW, MC_COLORWHEEL_HIGH); // 315/15
    colInd = colInd - MC_DIFFCW;   // turn - 60: icecold->DEEPBLUE=255, hot->DARKRED=315
    if (colInd < 0) colInd += 360; // no neg. values!
    return colInd;
}
```

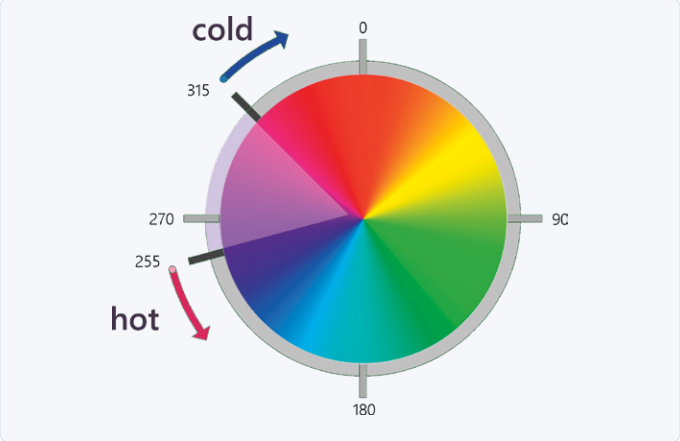


Figure 7. Modèle HSL: disque chromatique et domaine utilisé.

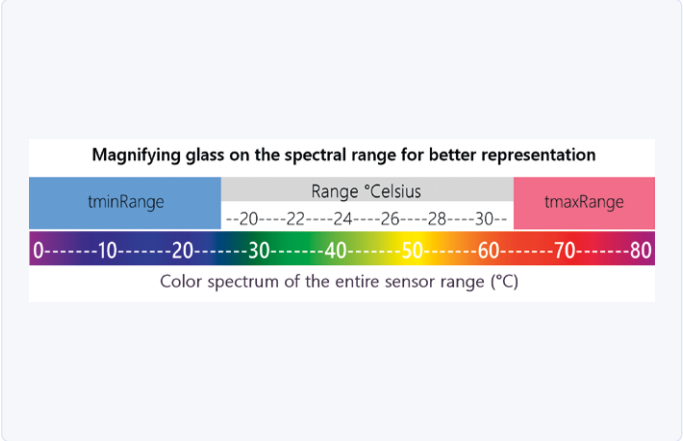


Figure 8. Le domaine de mesure est étendu à la partie souhaitée du spectre.

315. Pour réduire le bruit de fond relativement intense dans les zones de température uniforme sans point chaud ou froid, on introduit un intervalle minimum de 500 unités entre la température moyenne et tMin et TMax, pour que le spectre d'affichage (la «loupe») ait une largeur minimum de 1000 unités, correspondant à 10 °C, afin que chaque pixel bruité ne prenne pas une couleur complètement différente.
La barre de couleurs avec la zone de spectre utilisable est ainsi fixée : la température minimale correspond à 255 degrés du cercle des couleurs, la maximale à 315 degrés, parcourus dans le sens antihoraire.

Représentation sur l'écran

Pour réaliser une sorte de fonction de loupe qui applique le domaine d'affichage effectif dans cette plage spectrale possible, la fonction de détermination des couleurs utilise les valeurs `tminRange` et `tmaxRange` comme paramètres (**fig. 8**). Il s'agit là d'un moyen ultrasimple de créer un affichage couleur attrayant, qu'on pourrait améliorer par l'utilisation d'une palette de couleurs spécialement conçue, un calcul plus astucieux de la loupe de couleur, par ex. en déterminant les valeurs de la gamme de manière adaptative sur plusieurs mesures afin d'éviter les sauts de couleur, ou un traitement séparé de la dynamique du point chaud et de l'arrière-plan afin que ce point apparaisse plus clairement. Les experts en apprentissage machine auront plaisir à utiliser l'IA pour la reconnaissance des motifs, la réduction du bruit et l'analyse des données.
L'écran TFT du M5StickC montre (**fig. 9**) dans sa moitié supérieure l'image thermique originale avec le spectre de couleurs utilisé, et, en dessous, les valeurs mesurées minimum et maximum (tMin et tMax) pour l'ensemble des pixels de l'image, ainsi que la valeur moyenne calculée à partir de celles-ci (tAvg). Ces valeurs sont données en °C * 100, c'est-à-dire sans virgule.

Wi-Fi et serveur web

Jusqu'à présent, tout fonctionne localement sur le M5StickC et sans Wi-Fi. Mais si les valeurs mesurées doivent être affichées ou même traitées sur un appareil mobile distant, le M5StickC doit pouvoir se connecter par Wi-Fi et fonctionner comme serveur web capable de fournir une page web avec les valeurs mesurées.
Dans le logiciel Arduino, les bibliothèques `WiFi.h` et `WiFiClient.h` créent un tel serveur web, qu'il suffit de paramétrer avec les données

Figur 9. Le domaine de température du prototype.

d'accès à votre réseau (`ssid` et `password`). Après une réinitialisation ou une mise sous tension, l'écran affiche la tentative de connexion au réseau. En cas de succès, l'adresse IP reçue est affichée. Sinon : les données d'accès sont-elles correctes ? Un point d'accès est-il suffisamment proche ? D'ailleurs, la portée de ce petit appareil est étonnante, puisqu'elle peut facilement se comparer à celle des ordiphones.
Le serveur web écoute alors les requêtes `/`, `/index.html` et `/frame` sous son adresse IP locale attribuée par le routeur, par ex. `http://192.168.10.1/`.
Une fois la connexion Wi-Fi établie, le serveur web apprend ce qu'il doit faire lorsqu'il reçoit une requête spécifique du navigateur client. La fonction associée est alors appelée - par ex., à `/` (identique à `/index.html`) est associée la fonction `MW_index()`, qui délivre le document au client :

```
webServer.send_P(200, «text/html», _uidoc);
```

Listage 3. Wi-Fi et serveur web.

```
. . .
    const char *ssid     = "YourSSIDHere";      // change…
    const char *password = "YourPasswordHere";  // … to your WLAN
    WebServer webServer(80);                    // listening on port 80 = standard

    WiFi.mode(WIFI_STA);                        // switch to ESP station mode
    WiFi.begin(ssid, password);                 // look for known WLAN & try to connect
    while (WiFi.status() != WL_CONNECTED)       // wait for connection
    {
       delay(500);                              // 500ms delay
       Serial.print(".");                       // still alive
    }
    Serial.println("\n[Setup] WIFI connected!");
    // what to do if browser request is coming in
    webServer.on("/", MW_index);                // call MW_index() for / or /index.html
    webServer.on("/index.html", MW_index);
    webServer.on("/frame", MW_frameJS);         // /frame - output as JSON-Data
    webServer.onNotFound(MW_notFound);
    webServer.begin();                          // webserver starts here
. . .
// Webserver callback
void MW_index()     // send HTML-Doc from literally defined PROGMEM var
 {
    webServer.send_P(200, "text/html", _uidoc); // send_P: use PROGMEM-Var
 }
. . .
// main loop
void loop()
 {
                    // give webserver a chance to handle requests. No delay() in LOOP!
      webServer.handleClient();
     . . .
 }
. . .
```

`Send_P` est utilisé parce que le document HTML `_uidoc` est conservé en mémoire flash en tant que `PROGMEM` pour ne pas encombrer la mémoire RAM. Cela permet à MTheCam d'afficher des images thermiques en couleur sur n'importe quel ordiphone. Le **listage 3** montre et documente les parties de programme du Wi-Fi et du serveur web.

HTML et JSON

Les requêtes `/` et `/index.html` créent une page web avec un tableau de 8x8 cellules colorées avec indication numérique de la température. De plus, sont affichées la température minimale (tmin), maximale (tmax) et moyenne (tavg) d'une mesure. Chaque teinte correspond à une température, le gradient de couleur étant ajusté dynamiquement à la plage des valeurs MIN et MAX (`tminrange / tmaxrange`). Le bouton «*Image*» commute l'animation thermique sur une image «haute résolution» en fausses couleurs de valeurs interpolées, le bouton «*Values*» la fait revenir en arrière. Chaque seconde, une nouvelle mesure est affichée; l'intervalle d'affichage peut être modifié à l'aide d'un curseur. La case à cocher «*Grid*» superpose un quadrillage sur l'image.

La requête `/frame` retourne les valeurs mesurées sous forme d'un enregistrement de données JSON pour un traitement ultérieur par d'autres programmes, avec un délai à prévoir d'au moins 200 ms pour laisser le temps à la puce du capteur d'effectuer une nouvelle mesure, ce qui prend environ 100 ms. Un tel enregistrement pourrait être créé au besoin par manipulation de chaînes de caractères, mais la bibliothèque très pratique `ArduinoJson.h` permet d'alléger ce travail et peut même lire de telles données. Le résultat ressemble par ex. à ceci :

```
{"device": "MTheCam","ver": "1.0","rowsize":
   8,"batvolt": 3.39,
"tavg": 2317,"tmin": 2050,"tmax": 2750,"tminrange":
   1817,"tmaxrange": 2817,
"frame": [2075, 2050, 2100, 2275, 2600, 2475, 2100, 2050,
, ..... ]} // 64 values °C * 100 // + 2 arrays with
'HSL'-colors of 8*8 and 32*32 tiles.
```

Le **listage 4** montre les parties intéressantes du texte HTML. Le document avec les balises CSS et HTML est défini comme une chaîne fixe et envoyé au navigateur client. L'image thermique est construite à partir d'une balise `<canvas>` avec un tableau 8*8 (*values*) ou 32*32 (*image*).

Pilotée par une minuterie, une partie du script récupère alors les données du capteur (*/frame*) qui sont ensuite livrées par paquet JSON. Javascript et JSON s'entendent bien, de sorte que les données utilisateur peuvent être extraites du paquet très facilement : *jsonData.frame* ainsi que *jsonData.framecol* fournissent directement sous forme de tableaux les valeurs et les couleurs à partir desquelles la température originale et les valeurs interpolées sont évaluées, les cellules correspondantes colorées et pourvues de leur valeur numérique de

Listage 4. Créer un document HTML avec des valeurs mesurées.

```
. . .
/* Canvas-Area 400 px with 8*8/32*32 tiles à 50/12 px */
<div class="row">
      <div class="column">
        <canvas id="canvas"></canvas>    /* size see resizeCanvas() */
      </div>
</div>
. . .
/* resize canvas for value or image mode */
function resizeCanvas () {
 . . .
     canvasSize = rectSize * gridSize    /* gridSize: 8 or 32 @rectSize: 12 or 50 */
     canvas.width  = canvasSize
     canvas.height = canvasSize
. . .
/* update image,  rate = 200…3000 ms */
function refresh (rate = 1000) {
 . . .
refresher = setTimeout(function () {
        getData()
        }, rate)
. . .
/* request temperature-frame from MTC via /frame - call  */
function getData () {
     fetch(url)
        .then(response => response.json())
        .then(jsonData => {
          if (mode == 1) {
            // tempvalues, colors (interpolated) in framecolinter array
            paint(jsonData.frame, jsonData.framecolinter);
          } else {
            // tempvalues, colors (original) in framecol array
            paint(jsonData.frame, jsonData.framecol)
          }
          infoText(jsonData)             /* show min/max etc. values */
        }).catch(err => {
          errorText.innerHTML = 'Error: $'
      }).then(function () {
        refresh()                        /* start timer for next ride */
     })
}
. . .
/* paint tiles on canvas */
const ctx  = canvas.getContext("2d")

function paint(values, colors) {
  . . .
   ctx.clearRect(0, 0, canvas.width, canvas.height);
   colors.foreach(color, index) ….  {
     . . .   /* set x/y of next tile: x+=rectSize … */ . . .
     /* paint tile */
     ctx.beginPath()
     ctx.rect(x, y, rectSize, rectSize)           /* paint square */
     ctx.fillStyle = 'hsl($, 100%, 50%)'   /* set color of square */
     if (grid) { ctx.stroke() }                    /* show grid optional */
     ctx.fill()                                   /* do fill */
     ctx.closePath()
     if (mode == 0) {                             /* values as text over */
         // show tempvalues @mode=0 -> lowres 8*8 only
         drawText(values[index], x + 7, y + rectSize / 2)
       }  }
 . . .
```

température. L'intervalle de mise à jour de cet affichage peut être réglé de 200 ms à 3 s.

Bonne base pour de nouveaux projets

Dans une version enrichie du micrologiciel, il faut prévoir un point d'accès Wi-Fi privé, géré par ESP32, établissant son propre réseau local et permettant une saisie aisée des données d'accès à d'autres réseaux. Pour un affichage optimal, la palette et la loupe de couleurs seront adaptables, des algorithmes de filtrage réduiront le bruit et le scintillement. Une application Android peut simultanément gérer, configurer et afficher en direct plusieurs appareils MTheCam. Des états et des messages d'alarme peuvent être définis et déclenchés ; Node-Red via le serveur MQTT donnera plus de confort à l'exploitation des données. Cela vaut donc le coup de consulter de temps en temps la page du projet [9]. Les bases d'applications pratiques intelligentes étant jetées, ne manquent plus que les bonnes idées ! Nous remercions tout particulièrement Daniel Zelosko pour le développement du prototype.

180337-03

PRODUITS

> **M5StickC ESP32-PICO Mini – Carte de développement pour l'IdO**
www.elektor.fr/m5stack-m5stickc-esp32-pico-mini-iot-development-board

Votre avis, s'il vous plaît ?
Votre avis et vos commentaires sont bienvenus par courriel adressé en allemand ou en anglais à l'auteur (mtc@micom.de) ou en français à la rédaction (redaction@elektor.fr).

Ont contribué à cet article
Auteur : **Olaf Mertens** (Micom)
Traduction : **Helmut Müller**
Rédaction : **Rolf Gerstendorf**
Maquette : **Giel Dols**

LIENS

[1] **Pyromètre** : https://fr.wikipedia.org/wiki/Pyrom%C3%A8trerg/wiki/Pyrometer
[2] **Interpolation en photographie** : https://bit.ly/interpolation-photo
[3] **AMG88x** : https://eu.industrial.panasonic.com/products/sensors-optical-devices/sensors-automotive-and-industrial-applications/infrared-array
[4] **Documentation ESP** : www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
[5] **M5StickC** : www.elektor.fr/m5stack-m5stickc-esp32-pico-mini-iot-development-board
[6] **VisualStudio** : https://code.visualstudio.com/
[7] **Arduino-JSON** : https://arduinojson.org/
[8] **Bibliothèque M5StickC** : https://github.com/m5stack/M5StickC
[9] **Page de projet de l'auteur** : www.micom.de/lab/mthecam
[10] **Système de couleurs HSV** : https://fr.wikipedia.org/wiki/Teinte_saturation_lumi%C3%A8re