Université Paul Sabatier

RAPPORT TECHNIQUE CODE

PROJET M2 ISTR

Détection acoustique d'un frelon

Auteurs :
Ilyas YAHIA AISSA

9 mars 2022

1.1 Analyse Code

1.1.1 Structure tuple

Nous avons créé une structure nous permettant de manipuler des tuples, elle contient 4 paramètres :

f0 : Représente la borne inférieure de la plage de fréquence du tuple.

f1 : Représente la borne supérieure de la plage de fréquence du tuple.

Seuil : Représente le seuil de la plage de fréquence à dépasser.

Valide : Quand le seuil est dépassé, on passe ce paramètre booléen à true pour l'utiliser dans la vérification de tous les tuples.

```
struct tuple {
    int f0;
    int f1;
    int seuil;
    bool valide;
};
```

Nous avons déclaré le typedef pour nous faciliter l'écriture des programmes, et augmenter la lisibilité, nous pourrons alors déclarer directement les tuples.

Utilité bool valide : il nous permettra de comptabiliser à l'aide d'un compteur le nombre de seuil de tuple dépassé et prendre une décision sur la détection ou pas de frelon.

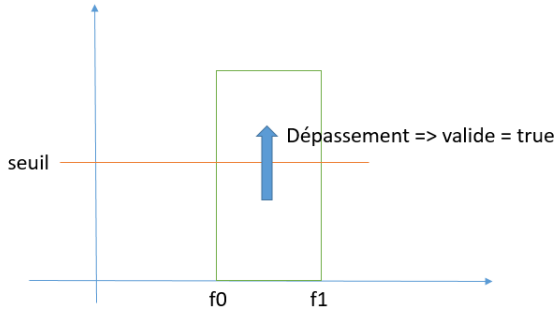

FIGURE 1.1 – Paramètres d'un tuple

1.1.2 Remplir paramètres tuple

Pour illuster le code, nous allons faire une détection sur 3 tuples, nous allons en premier lieu les déclarer

```
tuple tuple1, tuple2, tuple3;
```

Pour remplir les paramètres du tuples et les ajouter dans un tableau :

```
tuple tuple_arr [NB_TUPLE]= {{220,240,20000,false},{370,380,20000,false},
  {620,640,20000,false}};
```

Donc tous les paramètres seront ajustables et manipulables à notre guise, et le code pourra même être utilisé pour la détection d'autres fréquences.

Le nombre de tuples est défini en précompilation

```
#define NB_TUPLE 3
```

1.1.3 Vérifier tuple

```
void check_tuple(tuple* tuple_f) {

    for(int j = round(tuple_f->f0/0.244) ; j<= round(tuple_f->f1/0.244) ; j++) {
      if(vReal1[j] > tuple_f->seuil) {
            tuple_f->valide = true ;
            Serial.println("verifiee");
      }
    }
}
```

La fonction a comme paramètre un tuple_f,

La plage de fréquence sera donc entre **tuple_f.f0 * 0.244** et **tuple_f.f1 * 0.244**, la fréquence est multiplié à 0.244 vu les paramètres choisis et le fonctionnement de la libraire ArduinoFFT (voir figure en dessous), on doit arrondir à l'aide de la fonction **round** car j est un entier.

```
    for(int j = round(tuple_f.f0 * 0.244) ; j<= round(tuple_f.f1 * 0.244) ; j++)
    {
```

On vérifie le dépassement du seuil du tuple.

```
  if(vReal1[j] > tuple_f.seuil)
```

Dans le cas ou le seuil est dépassé pendant un intervalle

```
tuple_f.valide = true ;
```

On met l'attribut **valide** de la structure tuple associé à la page de fréquence à true.

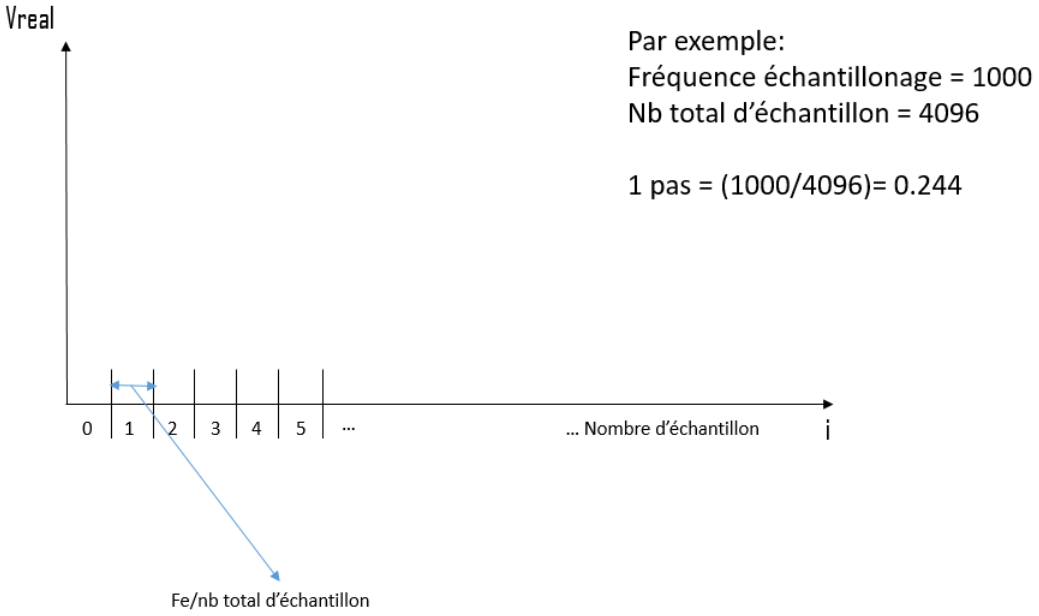

FIGURE 1.2 – Conversion fréquence vers indice

1.1.4 Vérifier tous les tuples

```
void check_all_tuple (tuple* tuple_array){
    static int cpt =0;
    for (int i=0; i<NB_TUPLE; i++) {
    check_tuple(&tuple_array[i]);
    }

    for (int j=0; j<NB_TUPLE; j++) {
        Serial.println("//");
        Serial.println("tuple");
        Serial.println(j);
        Serial.println("validit ");
        Serial.println(tuple_array[j].valide);
        if(tuple_array[j].valide == true) {
            cpt++;
            tuple_array[j].valide = false;

        }
    }
    Serial.println("compteur=");
    Serial.println(cpt);

    if (cpt > 2*NB_TUPLE/3) {
        Serial.println("  ALERTE FRELON ");
        cpt=0;
    }
    else {
      cpt = 0;
      }
}
```

Cette fonction aura comme paramètre un tableau contenant tous les tuples déclarés plus tôt.

Elle utilisera la fonction **check_tuple** sur tous les tuples présents dans le tableau, puis on vérifie la validité des tuples, en incrémentant un compteur dans le cas ou l'attribut valide est **true**, sans oublier de le remettre à **false** pour la prochaine itération.

Pour être tolérant aux fautes, On suppose qu'il y aura détection de frelons dans le cas ou 2/3 des tuples ont été détecté.

1.1.5 Exécution séquentielle

```
void loop(){

    Read_microphone();

    //   prise de son et cr ation de sa FFT

    FFT.DCRemoval();
    FFT.Windowing(vReal1, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
    FFT.Compute(vReal1, vImag1, SAMPLES, FFT_FORWARD);
    FFT.ComplexToMagnitude(vReal1, vImag1, SAMPLES);

    check_all_tuple(tuple_arr);
=
}
```

L'exécution se fera sous cet ordre :
- Lire le microphone.
- Faire la FFT.
- Faire le test sur tous les tuples.
- Annoncer détection.

1.2 Code Complet

```
#include "arduinoFFT.h"

arduinoFFT FFT = arduinoFFT();
#define SAMPLES 4096
#define SAMPLING_FREQUENCY 1000
#define amplitude 200
#define NB_TUPLE 3
unsigned int sampling_period_us;
double vReal1[SAMPLES];
double vImag1[SAMPLES];
unsigned long newTime;

struct tuple {
    int f0;
    int f1;
    int seuil;
    bool valide;
};


tuple tuple_arr [NB_TUPLE]= {{220,240,20000,false},{370,380,20000,false},
   {620,640,20000,false}};

void setup() {

    Serial.begin(115200);
    pinMode(15, INPUT);  // entr e son sur GPIO 0
    sampling_period_us = round(1000000 * (1.0 / SAMPLING_FREQUENCY));
}


void loop(){

    Read_microphone();

    //   prise de son et cr ation de sa FFT

    FFT.DCRemoval();
    FFT.Windowing(vReal1, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
    FFT.Compute(vReal1, vImag1, SAMPLES, FFT_FORWARD);
    FFT.ComplexToMagnitude(vReal1, vImag1, SAMPLES);


    check_all_tuple(tuple_arr);
```

```
}

void check_all_tuple (tuple* tuple_array){
    static int cpt =0;
    for (int i=0; i<NB_TUPLE; i++) {
    check_tuple(&tuple_array[i]);
    }

    for (int j=0; j<NB_TUPLE; j++) {
        Serial.println("//");
        Serial.println("tuple");
        Serial.println(j);
        Serial.println("validit ");
        Serial.println(tuple_array[j].valide);
        if(tuple_array[j].valide == true) {
            cpt++;
            tuple_array[j].valide = false;

        }
    }
    Serial.println("compteur=");
    Serial.println(cpt);

    if (cpt > 2*NB_TUPLE/3) {
        Serial.println("  ALERTE FRELON ");
        cpt=0;
    }
    else {
      cpt = 0;
      }

}


void check_tuple(tuple* tuple_f) {

    for(int j = round(tuple_f->f0/0.244) ; j<= round(tuple_f->f1/0.244) ; j++) {
      if(vReal1[j] > tuple_f->seuil) {
            tuple_f->valide = true ;
            Serial.println("verifiee");
      }
    }
}

void Read_microphone() {
     for (int i = 0; i < SAMPLES; i++) {
        newTime  = micros();          //  lecture du son via un micro mis en
   GPIO0
        vReal1[i] = analogRead(15);    //  Using pin number for ADC port ici
   GPIO0
        vImag1[i] = 0;
        while ((micros() - newTime) < sampling_period_us) { /* do nothing to
   wait */ }
    }
}
```