

UNIVERSITÉ PAUL SABATIER

RAPPORT TECHNIQUE CODE

PROJET M2 ISTR

DÉTECTION ACOUSTIQUE D'UN FRELON

Auteurs :
Ilyas YAHIA AISSA

7 mars 2022

1	Rapport technique	1
1.1	Analyse Code	1
1.1.1	Structure tuple	1
1.1.2	Remplir paramètres tuple	2
1.1.3	Ajouter tous les tuples dans un tableau	2
1.1.4	Vérifier tuple	2
1.1.5	Vérifier tous les tuples	4
1.1.6	Exécution séquentielle	4
1.2	Code Complet	5

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.

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

Nous avons déclaré le typedef pour nous faciliter l'écriture des programmes, et augmenter la lisibilité, nous pourrions 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.

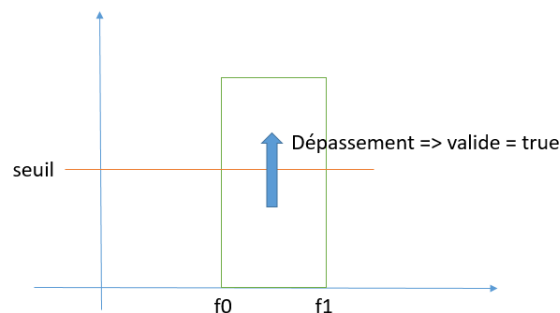


FIGURE 1.1 – Paramètres d'un tuple

1.1.2 Remplir paramètres tuple

Pour illustrer 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 :

```
tuple1.f0 = 230;
tuple1.f1 = 240;
tuple1.seuil = 40000;
tuple1.valide = false;

tuple2.f0 = 160;
tuple2.f1 = 170;
tuple2.seuil = 30000;
tuple2.valide = false;

tuple3.f0 = 130;
tuple3.f1 = 140;
tuple3.seuil = 50000;
tuple3.valide = 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.

1.1.3 Ajouter tous les tuples dans un tableau

La fonction qui vérifie la totalité des tuples aura comme paramètres un tableau qui contient tous les tuples.

```
tuple* tuple_arr [NB_TUPLE] = {tuple1 , tuple2 , tuple3 };
```

Le nombre de tuples est défini en précompilation

```
#define NB_TUPLE 3
```

1.1.4 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) {
            chrono2 = millis();
            if (Nb_chrono == 0) {
                chrono1 = chrono2 ;
            }
            unsigned long interval;
            interval = chrono2 - chrono1 ;

            if (interval < 5000 ) {
                chrono1 = chrono2;
                chrono2 = millis();
                Nb_chrono++;
                delay(100); // attente
            }
            else { Nb_chrono = 0;}
        }
    }
}
```

```

        if ( Nb_chrono > 4 ) {
            Nb_chrono=0;
            tuple_f.valide = true ;
        }
    }
}

```

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 librairie `ArduinoFFT` (voir figure en dessous), on doit arrondir à l'aide de la fonction **round** car `j` est un entier.

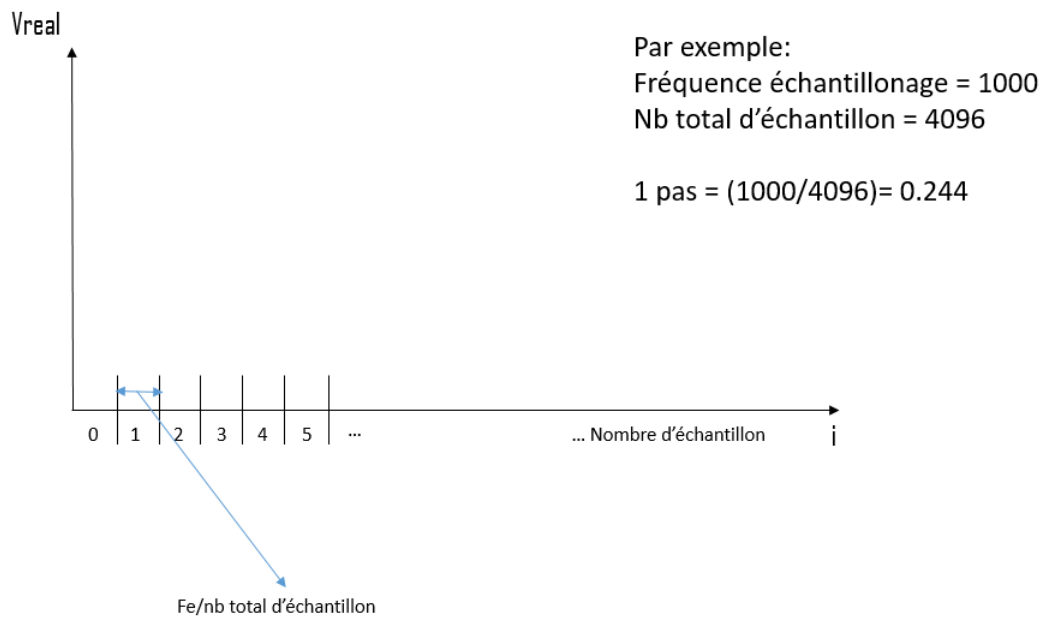


FIGURE 1.2 – Conversion fréquence vers indice

```

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)

```

Pour confirmer que la fréquence est détectée pendant une période :

```

if (interval < 5000 ) {
    chrono1 = chrono2;
    chrono2 = millis();
    Nb_chrono++;
    delay(100); // attente
}

```

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.

1.1.5 Vérifier tous les tuples

```
void check_all_tuple (tuple* tuple_array){
    int cpt =0;

    for (int i=0; i<NB_TUPLE; i++) {
        check_tuple(tuple_array[i]);
    }

    for (int j=0; j<NB_TUPLE; j++) {
        if(tuple_array[j].valide == true) {
            cpt++;
            tuple_array[j].valide = false;
        }
    }

    if (cpt > 2*NB_TUPLE/3) {
        Serial.println("__ALERTE_FRELON__");
    }
}
```

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 où 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 où 2/3 des tuples ont été détecté.

1.1.6 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];

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

tuple tuple1, tuple2, tuple3;

tuple1.f0 = 220;
tuple1.f1 = 240;
tuple1.seuil = 40000;
tuple1.valide = false;

tuple2.f0 = 220;
tuple2.f1 = 240;
tuple2.seuil = 40000;
tuple2.valide = false;

tuple3.f0 = 220;
tuple3.f1 = 240;
tuple3.seuil = 40000;
tuple3.valide = false;

tuple* tuple_arr [NB_TUPLE] = {tuple1, tuple2, tuple3};

unsigned long newTime, chrono1, chrono2;
int compteur_tour =0;
int Nb_chrono =0;

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){
    int cpt =0;

    for (int i=0; i<NB_TUPLE; i++) {
        check_tuple(tuple_array[i]);
    }

    for (int j=0; j<NB_TUPLE; j++) {
        if(tuple_array[i].valide == true) {
            cpt++;
            tuple_array[i].valide = false;
        }
    }

    if (cpt > 2*NB_TUPLE/3) {
        Serial.println("__ALERTE_FRELON__");
    }
}

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) {
            chrono2 = millis();
            if (Nb_chrono == 0) {
                chrono1 = chrono2 ;
            }
            unsigned long interval;
            interval = chrono2 - chrono1 ;

            if (interval < 5000 ) {
                chrono1 = chrono2;
                chrono2 = millis();
                Nb_chrono++;
                delay(100); // attente
            }
            else { Nb_chrono = 0;}
        }
    }
}

```



```

        if ( Nb_chrono > 4 ) {
            Nb_chrono=0;
            tuple_f.valide = true ;
        }
    }
}

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 */ }
    }
}

```