Laboratoire 1

Oscilloscope et générateur de formes d'onde : Montage avec microcontrôleur

Équipier:

Nom: Loïc Olivier **Matricule:** 536 898 399
Nom: Xavier Isabel **Matricule:** 536 892 061

Partie 1 : Acquisition et génération de signaux avec l'Arduino Uno

1. La programmation de microcontrôleur

   ...

   1.2.

   void setup():
   Cette fonction est utilisée pour initialiser les variables, les composantes des librairies et les modes des ports de l'Arduino(*INPUT*, *OUTPUT* etc). Cette fonction peut être utilisée pour initialiser la communication sérielle avec l'Arduino pour communiquer avec celui-ci lors de son fonctionnement autonome pendant l'exécution du programme.

   void loop():
   Cette fonction agit comme la boucle principale infinie du programme. Ainsi, elle permet une réponse et un contrôle actif de l'Arduino. Cette fonction peut être utilisée pour échantillonner, calculer et afficher répétitivement la fréquence cardiaque d'un être vivant à l'aide d'électrodes.

2. Acquisition et transmission de signaux

   ...

   2.2.

   C'est ainsi puisque la fonction ***void setup()*** est la première fonction exécutée par l'Arduino et, exécutée qu'une seule fois dans le programme, elle est l'endroit propice pour configurer le port A0 comme entrée.

   ...

   2.7.

   La plage des valeurs numériques lues se situe entre 0 et 1023. On en déduit donc que le convertisseur analogique-numérique de l'Arduino a une résolution sur 10 bits.

2.8.

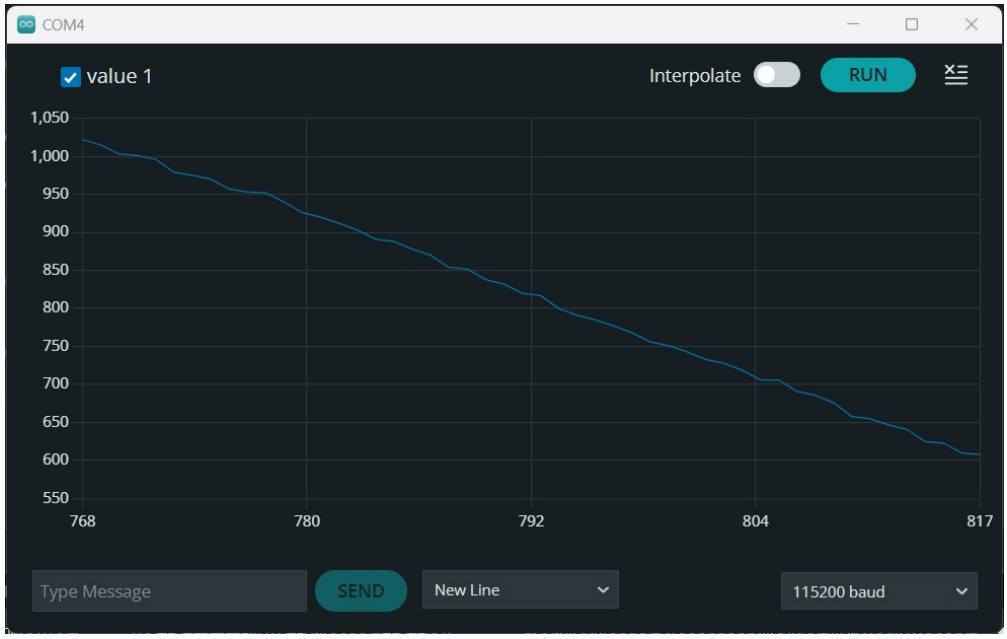

Figure 1: Graphique du *Serial Plotter* en variant le potentiomètre.

3. Générateur d'onde carrée

...

3.2.3.

La fréquence théorique attendue de l'onde carrée est de 166,7 Hz.

3.2.4.

La fréquence mesurée de l'onde carrée grâce à l'oscilloscope est de 131,6 Hz.

3.2.5.

La fréquence mesurée de l'onde carrée NE correspond PAS à la fréquence théorique. Cela est dû à la durée d'exécution non négligeable des fonctions utilisées dans la boucle. Cette durée s'ajoute aux pauses de la fonction "delay()" et génère une période de signal plus grande, d'où la fréquence expérimentale plus faible. **BONUS:** On pourrait corriger cela en utilisant la fonction "millis()"/ "micros()". Ces fonctions retournent le temps respectivement en millisecondes et en microsecondes depuis le début d'exécution du programme. Il suffirait donc de vérifier continuellement le temps du programme en appelant l'une de ces fonctions (dépendamment de la précision nécessaire) et de vérifier si la valeur est multiple de la moitié de la période de signal voulue pour changer l'état de la sortie. Cette méthode serait beaucoup plus robuste puisque la fréquence du signal ne serait pas affectée par l'exécution des autres fonctions du programme.

3.2.6.

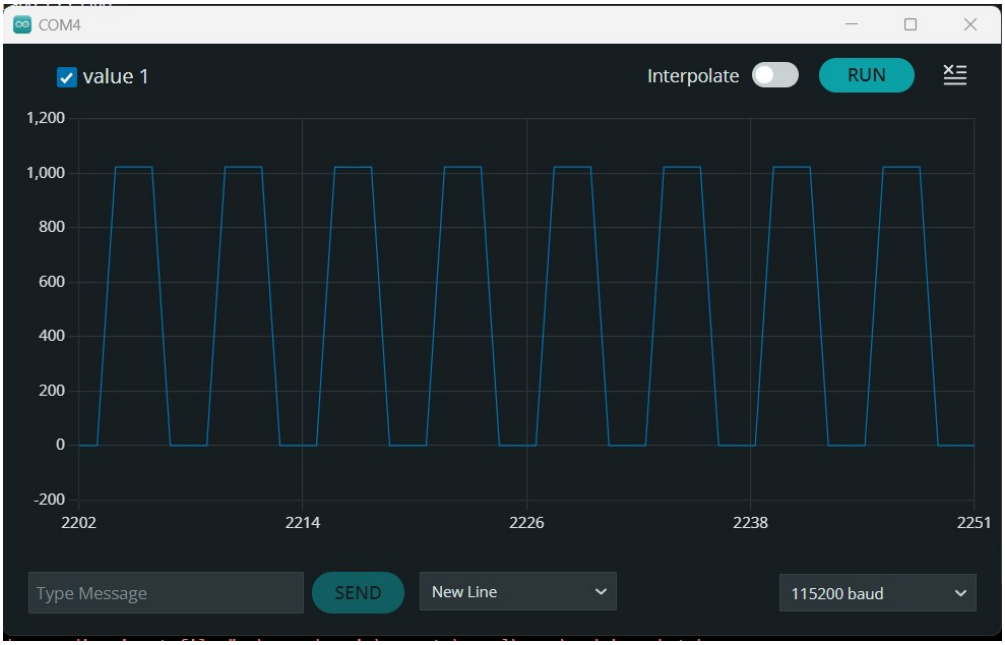

Figure 2: Graphique du *Serial Plotter* en connectant la sortie numérique à l'entrée analogique A0.

4. Générateur d'onde sinusoïdale

4.1.

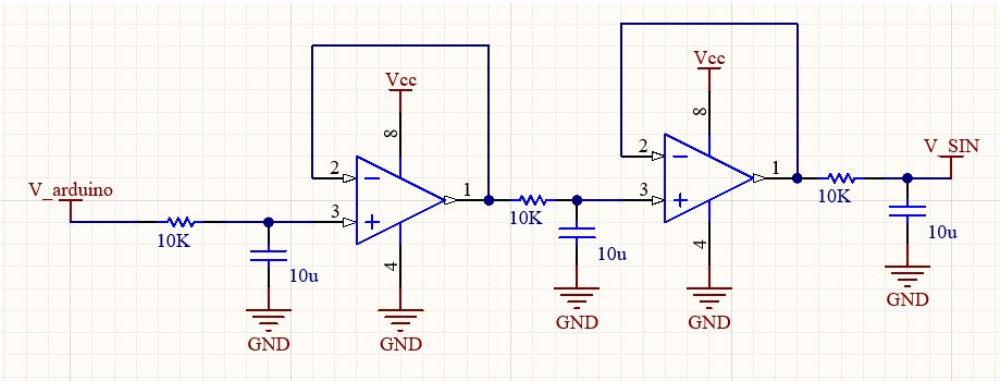

Figure 3: Schéma du circuit utilisé pour filtrer l'onde carrée

L'onde carré générée par l'*Arduino* est filtrée à 3 reprises par des filtres *RC*. Chacun des filtres est séparé par un ampli-op configuré en mode suiveur ce qui permet d'isoler chacun des filtres. L'isolation des étages de filtres est essentielle puisque sans elle, la configuration devient complètement différente. En effet, le second filtre se positionne alors en parallèle avec le condensateur du premier filtre et atténue ainsi le signal à ses bornes. Le même phénomène se produit de nouveau avec l'ajout du troisième filtre sans *buffer*. Sans l'utilisation des 2 *buffers* (ampli-op en mode suiveur), le signal de sortie serait grandement atténué ce qui est totalement indésirable. Autrement dit, les "buffers" permettent de multiplier les fonctions de transfert des filtres.

Pour faire le choix des valeurs de R et de C, on modélise la fonction de transfert du circuit selon

les impédances et on trouve la valeur de $x = RC$ à la fréquence de coupure $\omega = 130 \times 2\pi\ rad$:

$$H(jw) = \frac{1}{RCjw + 1}$$

$$|H(j260\pi)| = -3dB = 20log(\frac{1}{\sqrt{1 + (x260\pi)^2}})$$

$$x = 0,00122 \approx 0,001$$

On choisit donc R = 10 000 kΩ et C = 10 µF.

4.2.1.

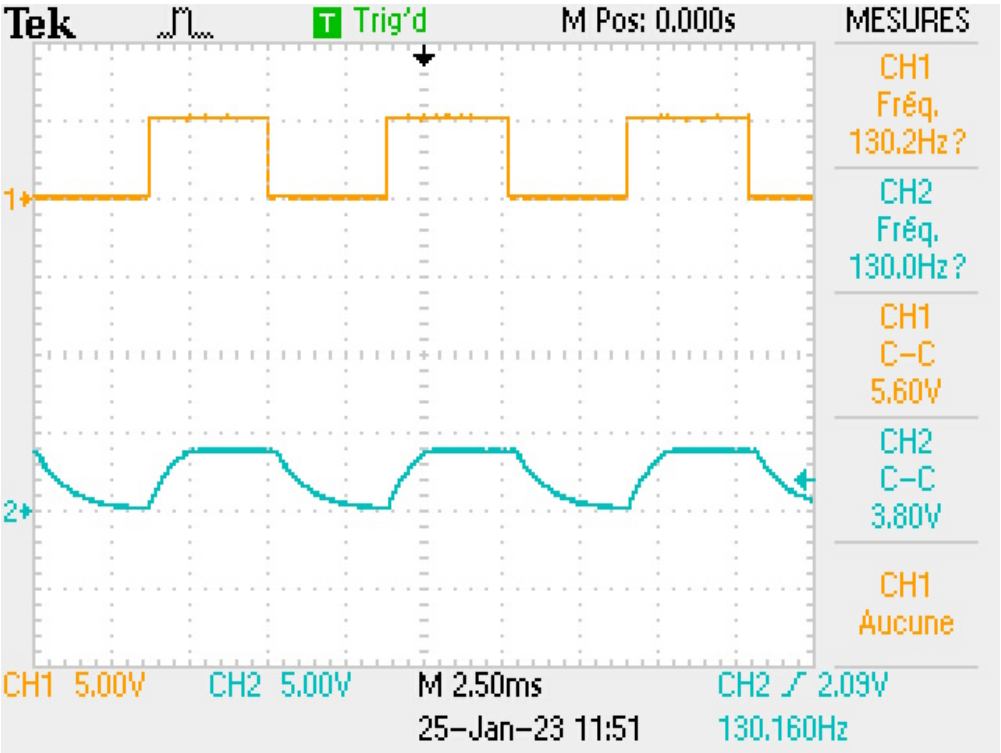

Figure 4: Onde carrée à l'oscilloscope avec effet du filtre (saturation).

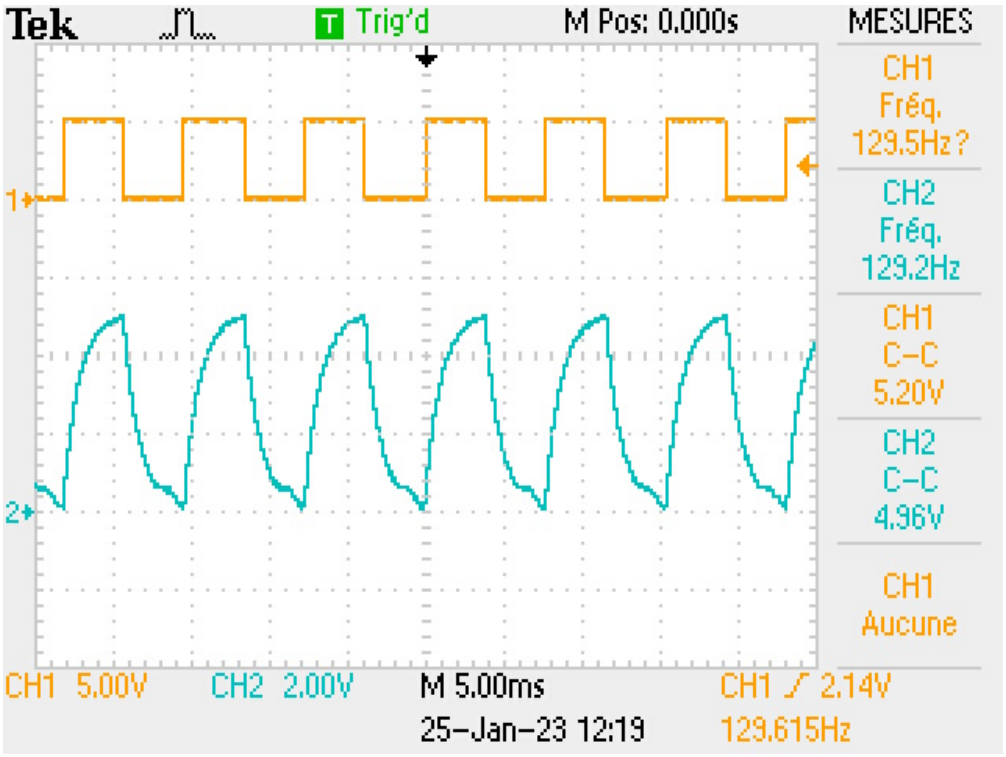

Figure 5: Onde carrée à l'oscilloscope avec effet du filtre(sans saturation).

Tout d'abord, on observe une saturation entre la figure 3 et la figure 4. Cela s'explique par le gain légèrement supérieur à l'unité de l'amplificateur. On corrige donc cette saturation en augmentant la tension d'alimentation de l'amplificateur. Pour cela, l'amplificateur est connecté à une source de tension plutôt qu'à l'Arduino, en prenant soin de connecter la référence de la source au *ground* de l'Arduino. Ainsi, grâce à notre choix distinct de filtre, on observe la réponse caractéristique d'une onde carrée à un filtre RC.

4.2.2.

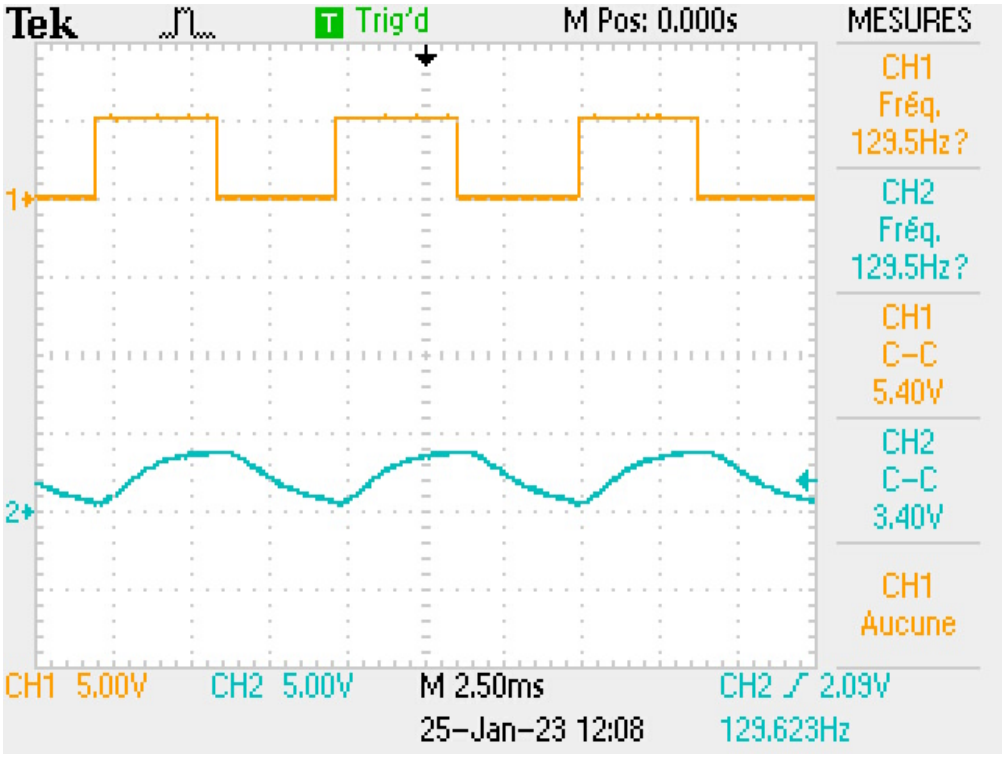

Figure 6: Onde carrée à l'oscilloscope avec deuxième étage.

À la figure 5, on remarque une atténuation de près de 40% de l'amplitude du signal. En revanche, le signal semble beaucoup plus triangulaire qu'à l'étape précédente. Cette forme triangulaire s'explique par la caractéristique exponentielle de charge et de décharge d'un filtre RC, soit par les équations:

$$V_{charge} = Ve^{\frac{-t}{RC}}$$

$$V_{decharge} = V(1 - e^{\frac{-t}{RC}})$$

.

4.2.3.

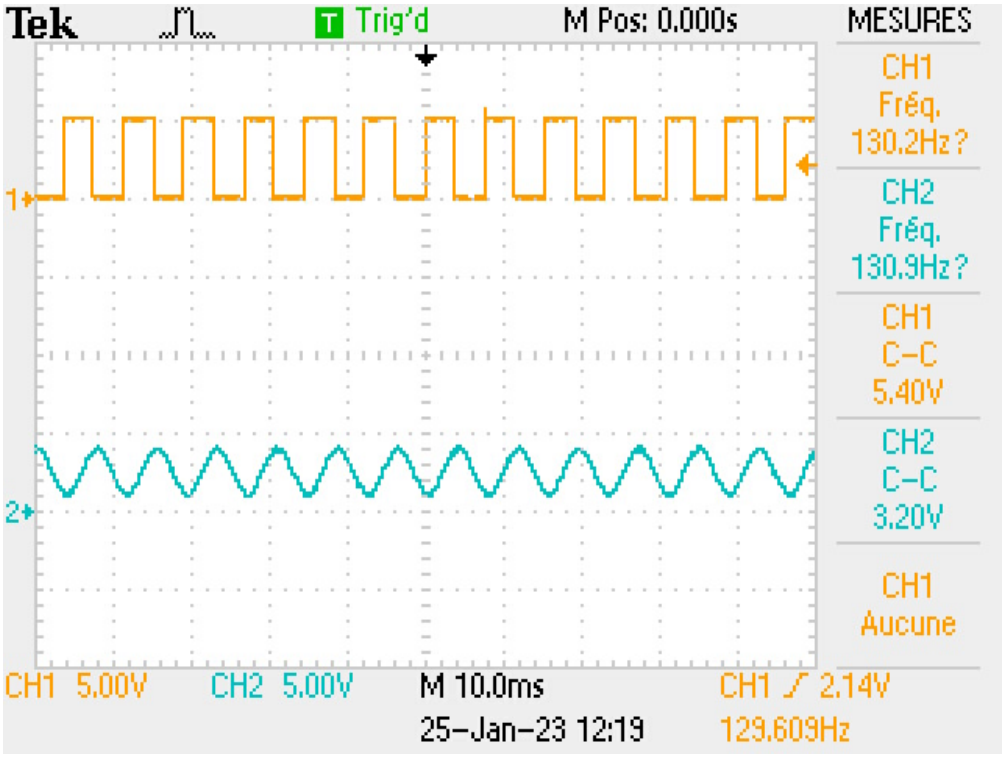

Figure 7: Onde carrée à l'oscilloscope avec troisième étage.

À la figure 6, on remarque que le signal est vraisemblablement sinusoïdal et que la valeur de tension crête-crête est environ à 60% de la valeur initiale d'entrée. La forme sinusoïdale du signal s'explique et se visualise par une convolution en temps de l'onde de filtre observée à la figure 4 par l'onde observée à la figure 5. Cette convolution en temps représente une multiplication en fréquence des deux ondes mentionnées ci-haut. On comprend que l'onde devient sinusoïdale puisqu'un signal convolué à l'infini en temps tends à devenir une gaussienne et que le 3e filtre représente la 3e convolution effectuée sur l'onde carrée initiale.

4.3.

Entre l'oscilloscpe et le *Serial Plotter* d'Arduino, on remarque que le signal à l'oscilloscope est à la fois plus lisse et plus sinusoïdal que celui à l'Arduino. Cela s'explique par la fréquence d'échantillonage de l'oscilloscope qui est énormément plus élevée que celle de l'ADC de l'Arduino. En effet, l'oscilloscope peut échantillonner des signaux allant jusqu'aux MHz.

5. Distorsion harmonique

5.1.

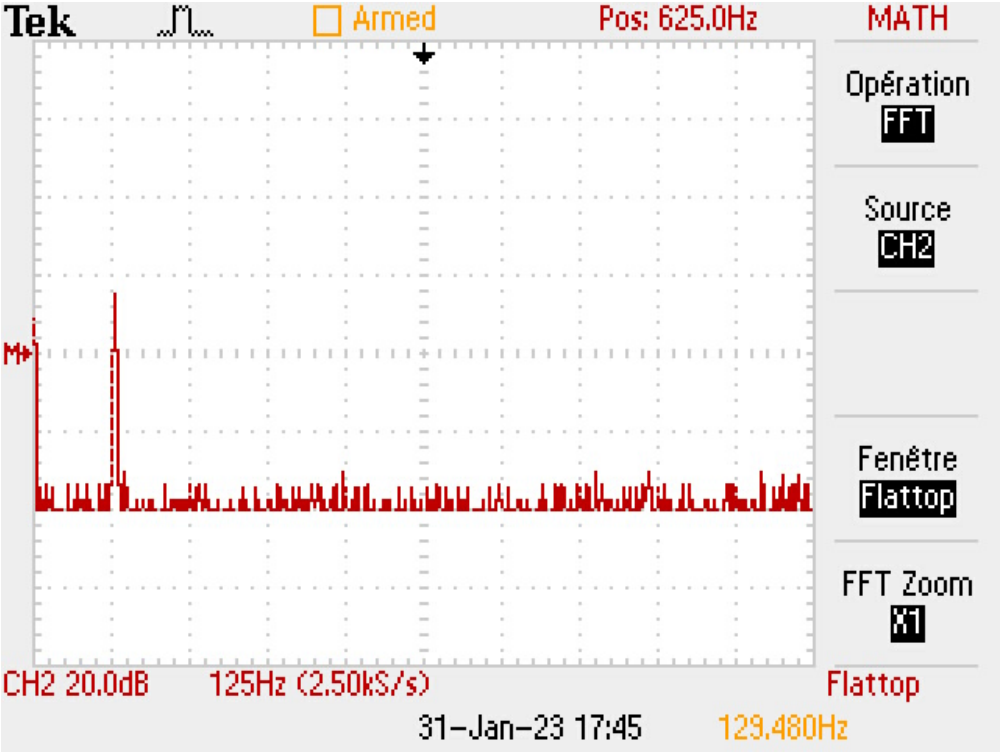

Figure 8: Contenue spectral de l'onde sinusoïdale du générateur de fonctions.

5.2.

Le contenu fréquentiel de notre onde sinusoïdale est très similaire à celle du générateur. En effet, dans les deux cas, un pic peut être observé à environ 130 Hz ainsi qu'un second à 0 Hz, associé à la composante DC du signal. Notre onde sinusoïdale est toutefois moins pure puisque deux autres pics non négligeables d'amplitude moindre sont présents aux fréquences suivantes: 390 et 650 Hz, des harmoniques impaires. Ce n'est pas le cas du signal du générateur de fonction qui n'a que les deux pics attendus (0 et 130 Hz).

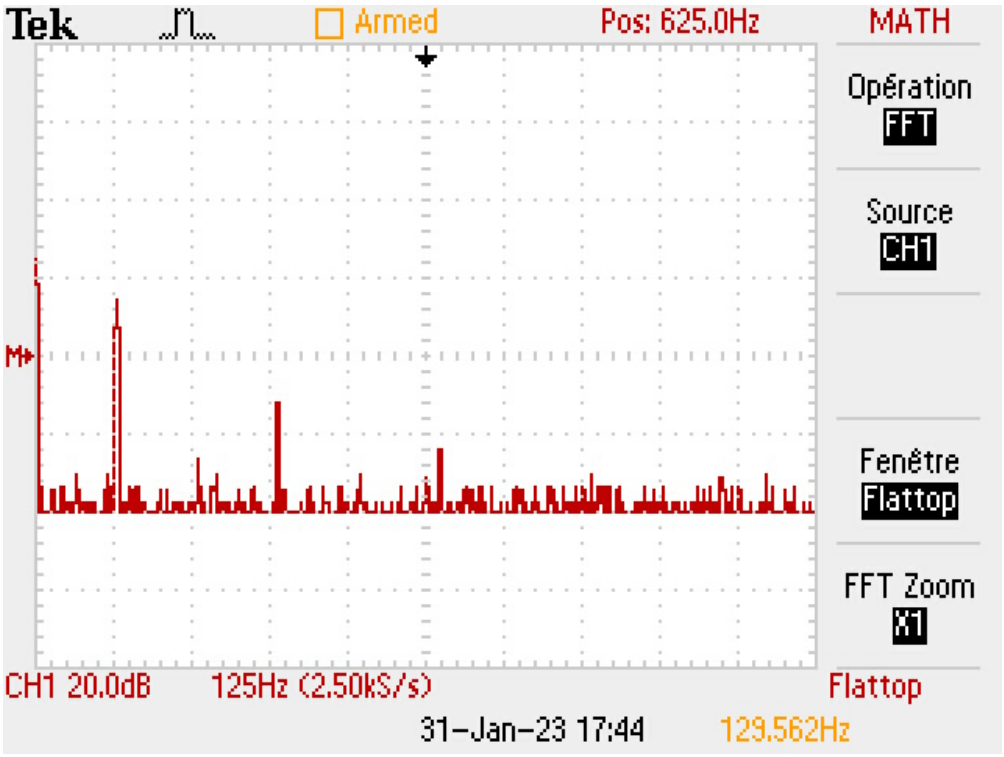

Figure 9: Contenue spectral de l'onde sinusoïdale(onde carrée filtrée) de l'Arduino.

5.3. Le signal du générateur, ne contient pas de pics non-négligeables, et donc:

$$THD_{gen} = \frac{\sqrt{V_{1REMS}^2 + V_{2REMS}^2 + V_{3REMS}^2 + \dots}}{V_{0REMS}} \approx 0$$

Le niveau dB des harmoniques peut être converti en V_{RMS} en utilisant l'équation suivante:

$$V_{RMS} = 10^{\frac{A}{20}}$$

$$A_0 = 56dB \longrightarrow V_{0RMS} = 10^{\frac{56}{20}} = 630,957$$
$$A_1 = 30dB \longrightarrow V_{1RMS} = 10^{\frac{30}{20}} = 31,6227$$
$$A_2 = 16dB \longrightarrow V_{2RMS} = 10^{\frac{16}{20}} = 6,3095$$

$$THD_{sys} = \frac{\sqrt{V_{1REMS}^2 + V_{2REMS}^2 + V_{3REMS}^2 + \dots}}{V_{0REMS}} = \frac{\sqrt{31,6227^2 + 6,3095^2}}{630,957} = 0,051$$

En comparant le taux de distorsion harmonique de notre signal sinusoïdale à celui du générateur, force est de constater que celui du générateur est plus pure puisque le calcul de son taux de distorsion donne une valeur nulle. En effet, le niveau dB des harmoniques étant non observable, le numérateur sera très faible alors que le dénominateur est élevé pour l'expression du taux de distorsion harmonique. Le taux de distorsion de notre signal à base d'onde carré est de 0,051, ce qui est tout de même très faible et témoigne de la qualité du signal obtenue avec 3 étages de filtre RC.

Partie 2 : Prise de mesures via MATLAB

6. Capture de l'Arduino via MATLAB

...

6.7.

```
    //Modes d'utilisations: D finir le mode d sir  uniquement et commenter les
    autres:
// ------------------------------------------------------------------------------

//#define SCOPE //D finition de la macro pour la question 2
//#define SQUARE_WAVE //D finition de la macro pour la question 3
//#define MATLAB_ACQUISITION //D finition de la macro pour la question 6

// ------------------------------------------------------------------------------


int x = 0;
int y = 0;


void setup() {
  // put your setup code here, to run once:
  //2)
  Serial.begin(115200); //Configuration de la communication serielle    une frequence
    de 115200 bits/seconde
  pinMode(A0, INPUT); //Configuration de la pin A0 comme entree
  //3)
  pinMode(3, OUTPUT);//Configuration de la pin 3 comme sortie
}

void loop() {
  // put your main code here, to run repeatedly:
  //2)
  #ifdef SCOPE
  int data = analogRead(A0); //Lecture analogique et conversion num rique de la
    tension lue sur A0
  Serial.println(data); //Affichage de data dans le moniteur de communication s rie
  delay(100); //D lai de 100ms ajout  pour  chantillonner    une fr quence de 10 Hz
  #endif

  //3)
  #ifdef SQUARE_WAVE
  int data = analogRead(A0); //Lecture analogique et conversion num rique de la
    tension lue sur A0
  Serial.println(data); //Affichage de data dans le moniteur de communication s rie
  if (x % 3 == 0) { //Condition pour  g n rer l'onde carr e
    x = 0;
    if (y) {
      digitalWrite(3, HIGH);
      y = -y + 1;
  } else {
      digitalWrite(3, LOW);
      y = -y + 1;}
  }
    x = x + 1;
    delay(1); //D lai de 1 ms ajout  pour  chantilloner    une fr quence d'environ
    136 Hz
  #endif

  //6)

  #ifdef MATLAB_ACQUISITION

  int data = analogRead(A0); //Lecture analogique et conversion num rique de la
```

```
    tension lue sur A0
55
  int mask1 = 0b0011111111; //Cr ation d'un masque
57 int mask2 = 0b1100000000; //Cr ation d'un masque
  byte LSB = (data & mask1); //Comparaison AND binaire pour g n r er le message des 8
    bits LSB
59 byte MSB = (data & mask2) >> 8; //Comparaison AND binaire pour g n r er le message
    des 2 bits MSB
  byte intro = 0x01; //Message de d but de message
61 byte outro = 0xFE; //Message de fin de message
  Serial.write(intro);       //Communication s rie sur Matlab
63 Serial.write(LSB);
  Serial.write(MSB);
65 Serial.write(outro);

67 #endif
  }
69
```

Figure 10: Code Arduino.

```
%% Nettoye le port et reset les variables Matlab
2 instr = instrfind;
if isempty(instr) == 0;
4     fclose(instr);
      delete(instr);
6     clear instr;
end
8 clear
close all
10 clc


12
%%   tablie   connection s rie
14
com_port = 'COM6'; % CHANGER LE PORT COM!
16 baud_rate = 115200;
databits = 8;
18 input_buffer = 8192;

20     try
           s = serial(com_port, 'Baudrate', baud_rate, 'DataBits',
       databits, 'InputBufferSize', input_buffer);
22         fopen(s);
           disp('The port is now opened');
24     catch M
           disp('Error opening the com port');
26         fclose(s);
           delete(s);
28         clear all;
           return;
30     end

32 %% Graphique temps r el
figure
34 title('real time plot');
```

```
graphCanal1 = animatedline('MaximumNumPoints', 5000, 'Color', 'r');
xlabel('sample')
ylabel('val')
legend('Signal')


%% Main  Loop
inc = 0.0;
size_of_window = 500;
y1=0;
y2=5;

lower_lim_xaxis = 0;
cycle_counter = 0;
upper_lim_xaxis = size_of_window;


while(1)
    %Appelle la fonction get_data
    data_out = get_data_lab1(s);
    % Converti les valeurs d'une plage de 0-1023    0-5V
    volt = data_out*(5/1023);

    %Affichage des donn es
    inc = inc(end)+(1:length(volt));
    addpoints(graphCanal1, inc, volt);
    drawnow

    if (inc(end)>=upper_lim_xaxis)
        lower_lim_xaxis = upper_lim_xaxis;
        upper_lim_xaxis = upper_lim_xaxis+size_of_window;
        axis([lower_lim_xaxis upper_lim_xaxis y1 y2])
    end

end
```

main_lab1.m

Figure 11: Code *main* Matlab.

```
function [data_out] = get_data_lab1(board)
%this function collects data from the serial port and returns it to
    the
%main-loop.
%The input is the serial port we are trying to reach.
number_of_bytes_to_read = 100;
while(board.BytesAvailable <= number_of_bytes_to_read)
end
v = fread(board, number_of_bytes_to_read);%reading the board
    if(all(v(1:4:end) == 1) && all(v(4:4:end) == 254))
        data_out = 256*double(v(3:4:end))+double(v(2:4:end));%returns
    a double
    else
        disp('Error format Paquet')
```

```
        data_out = 0;
        %read until tag equal to 1
        while(1)
            trash = fread(board,1);
            if (trash==254)
                break
            end
        end
    end
end
```

get_data_lab1.m

Figure 12: Code Matlab.

6.8.

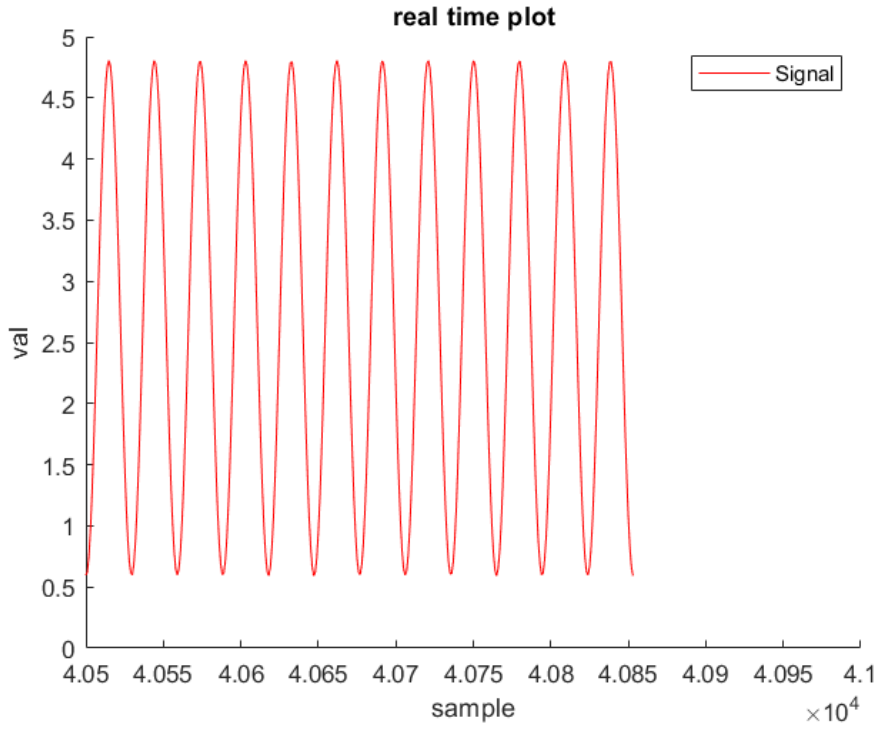

Figure 13: Signal sinusoïdal demandé.

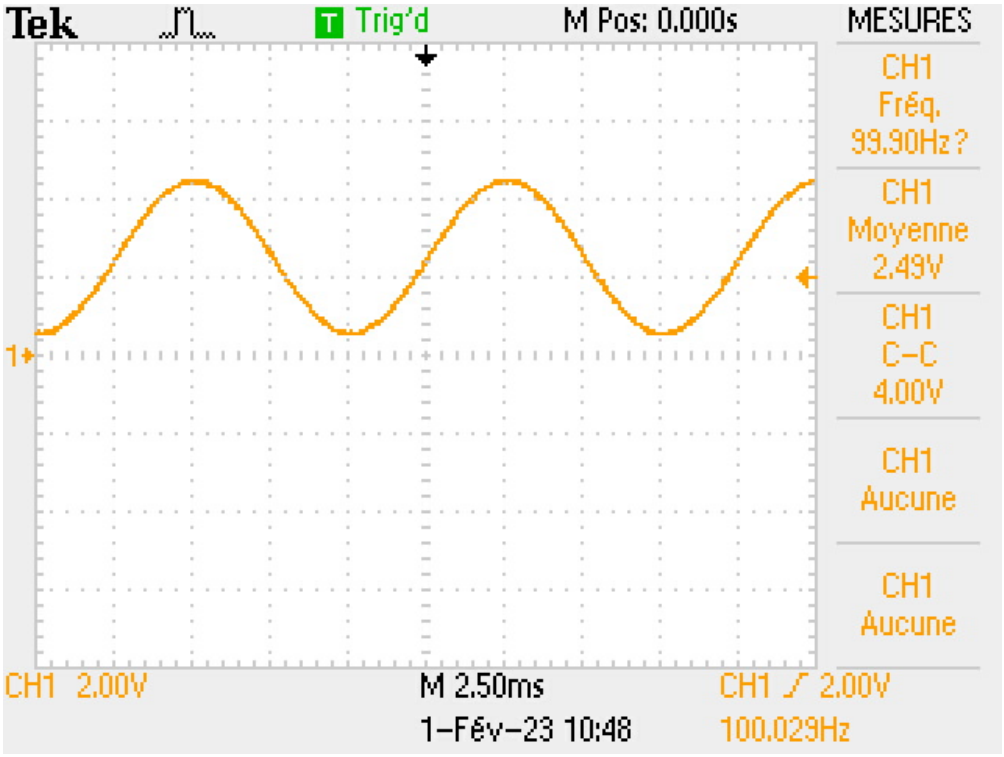

Figure 14: Signal sinusoïdal demandé à l'oscilloscope.

En comparant la Figure 12 à la Figure 13, on remarque très peu de différences. Effectivement, l'oscilloscope fait une mesure plus précise du signal, mais Matlab obtient tout de même une précision d'acquisition qui est bonne et surtout meilleure que le *Serial Plotter* d'Arduino. Aussi, on observe que l'affichage en temps réel de l'oscilloscope se fait à une fréquence beaucoup plus rapide que Matlab.

Questions Post-laboratoire

1. Quelles sont la résolution et la précision d'un ADC 10-bits avec une plage de 0-5 volts comme dans l'Arduino? Quel serait l'avantage d'utiliser un ADC 16-bits?

   La résolution d'un ADC 10 bits est de 10 bits donc 1024 valeurs numériques possibles. Pour une plage analogique de 0-5 volts, sa précision est d'environ 4.88 mV. Pour une même plage analogique, l'avantage d'utiliser un ADC 16 bits est que cela augmenterait la précision à 76,3 μV.

2. Donnez la représentation d'une valeur de 2.7 volts à chaque étape de la chaine d'acquisition (du voltage mesuré par l'ADC jusqu'à l'affichage sur le graphique Matlab). Indice : N'oubliez pas la conversion en binaire; il y a en gros 6 étapes.

   1) Tout d'abord, la valeur analogique de 2.7 V est captée par le port *Arduino* qui fait la conversion analogue-digitale de cette tension avec une résolution de 10 bits. 2) Sachant que la plage de lectures possibles est 0-5V, 1023 (0b1111111111) représente 5 V et 0 (0b0000000000) représente 0V. La valeur analogique de 2.7V est donc convertie de la façon suivante: $\frac{2.7}{5} \times 1023 = 552,42 \approx 552.552 = 2^9 + 2^5 + 2^3 = 0b1000101000$**. La valeur est désormais stockée sur 10 bits dans l'*Arduino*. 3) Cette valeur doit ensuite être envoyée sur le port de communication série qui transmet 1 octet (8 bits) à la fois. Décomposition**

de la valeur en 2 octets: Le premier octet représente les 8 bits les moins significatifs (LSB) et le second, les 2 bits les plus significatifs (MSB) décalés aux deux position les moins significatives (bit shift). $LSB = 0b00101000 \quad MSB = 0b00000010$**. 4) Les deux octets sont ensuite envoyés un à la suite de l'autre sur le port série, étant entouré d'un octet de début "0x01" et d'un octet de fin, "0xFE". 5) Les deux octets sont interceptés par le programme Matlab via le port de connexion série. Le programme recombine les deux octets pour ré-obtenir la valeur digitale initiale sur 10 bits. 6) La valeur est stockée dans une variable de résolution beaucoup plus grande que 10 bits et est reconverti en un voltage:** $data_{out} \times (5/(1023))$**. La valeur est envoyée dans un "buffer" pour finalement être affichée sur le graphique dynamique.**

3. Expliquez pourquoi le réseau de 3 filtres RC utilisé pour convertir une onde carrée en onde sinusoïdale dans l'étape 4 n'est pas une solution idéale et proposez une solution alternative pour générer une onde sinusoïdale à l'aide de l'Arduino.

Le réseau de 3 filtres RC utilisé n'est pas une solution idéale puisqu'il nécessite plusieurs composantes physiques et n'est pas optimisé en espace et peu versatile pour d'autres types d'ondes. Il demande également une source de tension externe à l'Arduino et supérieure à 5V pour éviter la saturation. Une solution alternative pour générer une onde sinusoïdale à l'aide de l'Arduino est de discrétiser de petits segments d'une onde sinusoïdale continue en un *array* de tensions discrètes pour ensuite itérer sur cet *array* pour générer, à l'aide d'une sortie PWM, une onde sinusoïdale. Tout cela se fait avec la source de tension de 5V de l'Arduino. Cette solution est idéale à condition que la charge ne soit pas particulièrement inductive. En effet, une charge inductive produirait des pics de tension en raison de f.é.m. aux hautes fréquences observées entre chaque segment discret.