Matériel & Méthodes

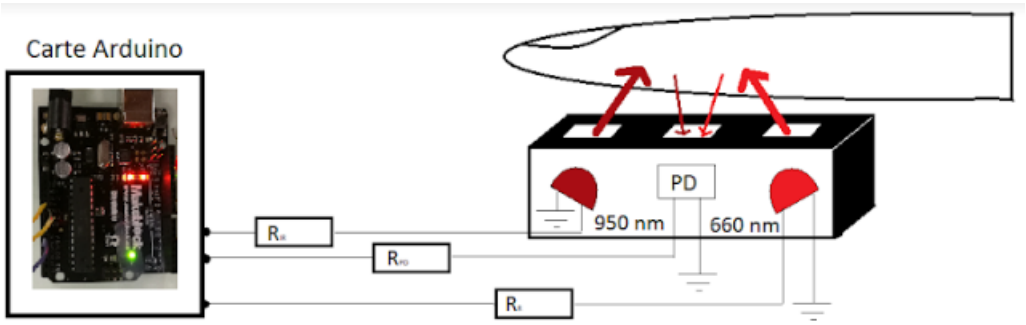
Hardware

- Arduino Uno de chez Makeblock
- Diodes $660\ nm$
- Diodes $950\ nm$ (lien)
- Résistances (cf. plus bas)
- Photodiode (lien)
- Câbles

Software

- Logiciel Arduino (lien pour le téléchargement)
- Matlab R2017a
- Microsoft Excel
- Autodesk Fusion 360 (lien pour le téléchargement)

Figure 1. Schéma fonctionnel du montage expérimental



Les valeurs des résistances sont les suivantes : $R_{IR} = R_R = 220\ \Omega$ et $R_{PD} = 470\ k\Omega$.

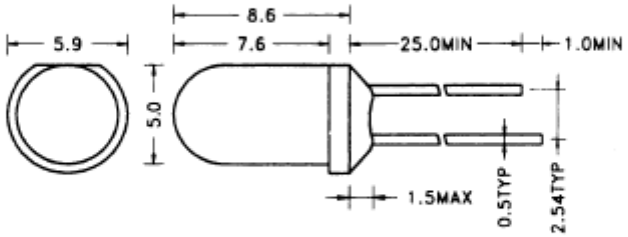
Les diodes fixées dans le support dessiné sur Autodesk sont reliées à deux sorties numériques (les 9 et 12 ici) et la photodiode est reliée à une entrée analogique (A1 ici). Un code Arduino (cf. Annexe 1) est utilisé à la fois pour piloter les diodes et pour récolter les données. Celles-ci sont exportées sous forme de fichier .txt que l'on importe ensuite dans Excel. Ces données sont ensuite exportées d'Excel vers Matlab. Le programme Matlab présenté en annexe 2 permet d'importer les données et de faire tous les traitements nécessaires pour obtenir le bpm et la SpO_2. Les sous-programmes sont en annexes eux aussi. Ils ont tous été réalisés par nos soins sauf le programme peakdet. Le programme `diagramme_poincare` permet de tracer les diagrammes de Poincaré (durée d'un battement en fonction de celle du battement précédent).

Le programme Arduino heartbeat_FFT permet d'afficher en temps réel la transformée de Fourier du signal reçu par l'Arduino.

Annexe 1. Fiche technique de la diode rouge (660 nm)

LED lamp R-660-530/C

Color	Type	Technology	Case
Red	R-660-530/C	AlGaAs	5 mm plastic lens, water clear



Maximum Ratings at T_a = 25°C

Parameter	Test conditions	Symbol	Value	Unit
Forward current		I_F	30	mA
Peak forward current	Duty 1/10, f≤10 kHz	I_{FP}	150	mA
Power dissipation		P_D	75	mW
Reverse voltage	I_R=10 µA	V_R	5	V
Reverse current	U_R=5 V	I_R	10	µA
Electrostatic discharge	Human body model	ESD	2000	V
Operating temperature		T_{opr}	-40 to +85	°C
Storage temperature		T_{stg}	-40 to +100	°C
Soldering temperature	5 sec max, 4 mm from body base	T_{sol}	260	°C

Optical and Electrical Characteristics at T_a = 25°C

Parameter	Test conditions	Symbol	Min.	Typ.	Max.	Unit
Forward voltage	I_F = 20 mA	V_F		1.85	2.5	V
Luminous intensity	I_F = 20 mA	I_V	3500	4000	4500	mcd
Luminous flux	I_F = 20 mA	Φ_V		240		mlm
Radiant power	I_F = 20 mA	Φ_e		4.1		mW
Peak wavelength	I_F = 20 mA	λ_p		660		nm
Dominant wavelength	I_F = 20 mA	λ_d		660		nm
Spectral halfwidth	I_F = 20 mA	$\Delta\lambda$		20		nm
Viewing angle	I_F = 20 mA	φ		30		deg.

rev.09/10
Power Light Systems GmbH, Kinzerallee 20, D-12555 Berlin, Germany, tel.: +49-30-6501-7868, fax: +49-30-6501-7869
www.led-eshop.de, email: sales@led-eshop.de

Annexe 2. Code Arduino pour l'acquisition des données

```
unsigned long time;
int pinInfra = 9;
int pinRed = 12;
int pinPhoto = A1;
int valeurInfra;
int valeurRed;

void setup() {
  pinMode (pinInfra,OUTPUT);
  pinMode (pinRed,OUTPUT);
  pinMode (pinPhoto,INPUT);
  Serial.begin(9600);
  Serial.println("CLEARDATA");
  Serial.println("LABEL,TimeI,Infra,TimeR,Red");

}

void loop() {
    //allumage de la diode infra
    Serial.print("DATA,");
    digitalWrite (pinInfra,HIGH);
    delay(1);
    valeurInfra = analogRead (pinPhoto);
    time=micros();
    //Serial.print(time);
    //Serial.print(" ");
    Serial.print(valeurInfra); //La première valeur affichée correspond à l'infrarouge
    Serial.print(" ");
    digitalWrite (pinInfra1,LOW);

    //allumage de la diode rouge
    digitalWrite (pinRed,HIGH);
    delay(1);
    valeurRed = analogRead (pinPhoto); //La deuxième valeur correspond au rouge
    time=micros();
    //Serial.print(time);
    //Serial.print(" ");
    Serial.println(valeurRed);
    digitalWrite (pinRed1,LOW);

}
```

Annexe 2. Script Matlab

```
clc
close all
clearvars

path='C:\Users\arthu\Desktop\Matlab\test_cyprien_09.03.18_
modulation_1.ods';    %chemin de l'echantillon

N = 1024; % Take the first N points of the signal
samp = 4;% Take 1/SAMP points in the signal

batt = xlsread(path);
batt2 = batt(N:samp:2*N,:);

fe = 1000000/(batt2(3,1)-batt2(2,1)); % Fréquence
d'échantillonage
fprintf("Taille du signal (nombre de points) = %d \n",N);
fprintf("Taux d'échantillonage = 1/%d \n",samp);
fprintf("Fréquence d'échantillonage = %d Hz \n",fe);

signal_infra = batt2(:,2);
time_infra = batt2(:,1)/1000000;
signal_red = batt2(:,4);
time_red = batt2(:,3)/1000000;

%% Détermination du BPM

bpmR = bpm(signal_red,time_red,fe);
bpmI = bpm(signal_infra,time_infra,fe);

MBpm = (bpmI + bpmR)/2;

if abs(bpmR - bpmI) > 10
    error('Pulsations trop différentes entre les deux
diodes. Revoir le montage')
else
   fprintf('Votre fréquence cardiaque est de %d battements
par minute \n',vpa(MBpm));
end


%% Calcul des parties variables des signaux

AC_infra = variation(signal_red,time_infra);
AC_red = variation(signal_infra,time_red);
```

```
%% Calcul du SPO2

DC_infra = mean(signal_infra);
DC_red = mean(signal_red);

R = (AC_infra/DC_infra)/(AC_red/DC_red);

A = .1;
B = 98.83;

SPO2 = A*R+B;

fprintf('Votre taux de saturation en dioxygène est de %f
%% \n',SPO2)

%% Estimation SNR
close all

noise_infra = noise(batt2(:,2));
noise_red = noise(batt2(:,4));

SNR_infra = AC_infra/noise_infra;
SNR_red = AC_red/noise_red;

if SNR_infra < 3 || SNR_red < 3
    error("Rapport signal sur bruit insuffisant. Refaire
la mesure")
end
```

Annexe 3. Fonction bpm

```
function [BPM] = bpm(signal,temps,fe)
%   bpm yields the main frequency between 0.5 and 2.5 Hz
in the computed
%   signal
%   INPUTS
%       signal : the signal in which the frequencies have
to be found
%       fe : the sampling frequency
%   OUTPUT
%       BPM : the main frequency between .5 and 2.5 Hz

four = fft(signal-mean(signal));
norm_four = abs(four);
L = length(temps);
freq = ((0:L-1)*fe/L)';

delta = 100; % Limite pour la détection de pic

[listmax1,~] = peakdet(norm_four,delta,freq);

temp1 = listmax1(:,1) < 2.5 & listmax1(:,1) > .5;
% On élimine tous les max au dessus de 2.5 Hz ou sous de
.5 Hz
temp2 = [listmax1(:,1) .* temp1, listmax1(:,2) .* temp1];
[~, freqMaxI] = max(temp2(:,2));
BPM = listmax1(freqMaxI,1);
BPM = 60*BPM;

end
```

Annexe 4. Fonction variation

```
function [AC] = variation(signal,time)
%   variation yields the height of the peaks
%   INPUTS
%       signal : the signal that as to be calculated
%       time : the array of time
%   OUTPUTS
%       AC = the height of the peaks of the signal

delta = 8; % Seuil de détection des pics

[maximum,minimum] = peakdet(signal,delta,time); %
Détection des min et max avec les indices

if minimum(1,1) < maximum(1,1)     % On vérifie que le
signal commence bien par un max et pas un min
    mini = minimum(2:end,:);
else
    mini = minimum;
end

taille = min(length(mini(:,1)),length(maximum(:,1)));
diff = maximum(1:taille,2)-mini(1:taille,2); % Liste des
différences max-min
AC = mean(diff);  % On moyenne

end
```

Annexe 5. Fonction `noise`

```
function [N] = noise(signal)
%   SNR yields the mean noise of the signal
%   Inputs
%       signal : the treated signal
%   Outputs
%       noise : return the noise of the signal
len = length(signal)-1;
diff = zeros(1,len);
for i=1:len
    diff(i) = abs(signal(i+1)-signal(i));
end
N = mean(diff);
end
```

Annexe 5. Fonction peakdet

```
function [maxtab,mintab] = peakdet(v,delta,x)
%PEAKDET Detect peaks in a vector
%        [MAXTAB, MINTAB] = PEAKDET(V, DELTA) finds the
local
%        maxima and minima ("peaks") in the vector V.
%        MAXTAB and MINTAB consists of two columns. Column
1
%        contains indices in V, and column 2 the found
values.
%
%        With [MAXTAB, MINTAB] = PEAKDET(V, DELTA, X) the
indices
%        in MAXTAB and MINTAB are replaced with the
corresponding
%        X-values.
%
%        A point is considered a maximum peak if it has
the maximal
%        value, and was preceded (to the left) by a value
lower by
%        DELTA.

maxtab = [];
mintab = [];

v = v(:); % Just in case this wasn't a proper vector

if nargin < 3
    x = (1:length(v))';
else
    x = x(:);
    if length(v) ~= length(x)
        error('Input vectors v and x must have same
length')
    end
end

if (length(delta(:)))>1
    error('Input argument DELTA must be a scalar');
end

if delta <= 0
    error('Input argument DELTA must be positive');
end
```

```
mn = Inf; mx = -Inf;
mnpos = NaN; mxpos = NaN;

lookformax = 1;

for i=1:length(v)
    this = v(i);
    if this > mx, mx = this; mxpos = x(i); end
    if this < mn, mn = this; mnpos = x(i); end

    if lookformax
        if this < mx-delta
            maxtab = [maxtab ; mxpos mx];
            mn = this; mnpos = x(i);
            lookformax = 0;
        end
    else
        if this > mn+delta
            mintab = [mintab ; mnpos mn];
            mx = this; mxpos = x(i);
            lookformax = 1;
        end
    end
end

end
```

Annexe 6. Fonction diagramme_poincare

```
function diagramme_poincare(path,delta,u)
% renvoie le digramme n,n+1 de la mesure effectuée et le
signal

batt = xlsread(path);
N = length(batt);   % Take the first N points of the
signal
samp = 1;   % Take 1/SAMP points in the signal
batt2 = batt(1:samp:N,:);
N2=length(batt2(:,2));

fe = 1000000/(batt2(3,1)-batt2(2,1));   %Fréquence
d'échantillonage

signal_infra = batt2(:,2);
time_infra = batt2(:,1)/1000000;
signal_red = batt2(:,4);
time_red = batt2(:,3)/1000000;


[maxima_infra,~] = peakdet(signal_infra,delta,time_infra);
[maxima_red,~] = peakdet(signal_red,delta,time_red);

N_infra=length(maxima_infra);
N_red=length(maxima_red);


diff_infra = maxima_infra(2:N_infra-1,1)-
maxima_infra(1:N_infra-2,1);
diff_red = maxima_red(2:N_red-1,1)-maxima_red(1:N_red-
2,1);

moy_infra = mean(diff_infra);
moy_red = mean(diff_red);


norm_infra= diff_infra;    %/moy_infra;
norm_red= diff_red;        %/moy_red;

norm_infra(norm_infra>1.5)=5;
norm_red(norm_red>1.5)=5;
norm_infra(norm_infra<0.4)=5;
norm_red(norm_red<0.4)=5;


one_infra=ones(N_infra-3,1);
```

```
one_red=ones(N_red-3,1);

ecart_xy_red=sum((2^0.5)*(abs(norm_red(1:N_red-3)-
norm_red(2:N_red-2))))/(N_red-3);
ecart_xy_infra=sum((2^0.5)*(abs(norm_infra(1:N_infra-3)-
norm_infra(2:N_infra-2))))/(N_infra-3);
ecart_11_red=sum(((norm_red(1:N_red-3)-
one_red).^2+(norm_red(2:N_red-2)-
one_red).^2).^0.5)/(N_red-3);
ecart_11_infra=sum(((norm_infra(1:N_infra-3)-
one_infra).^2+(norm_infra(2:N_infra-2)-
one_infra).^2).^0.5)/(N_infra-3);

disp(mean([ecart_xy_red,ecart_xy_infra]));
disp(mean([ecart_11_red,ecart_11_infra]));
disp('     ');


disp(moy_infra);
disp(moy_red);

figure
s=scatter(norm_infra(1:end-
1),norm_infra(2:end),500,'filled');
s.MarkerFaceAlpha=5/N_infra;
s.MarkerFaceColor=[0.5 0.5 0.5];

axis equal
grid
hold on
s=scatter(norm_red(1:end-1),norm_red(2:end),500,'filled');
s.MarkerFaceAlpha=5/N_red;
axis([0 2 0 2])
s.MarkerFaceColor=[0.5 0.5 0.5];
switch u
    case 1
        figure(1)
        subplot(1,3,1);
        s=scatter(norm_infra(1:end-
1),norm_infra(2:end),500,'filled');
        s.MarkerFaceAlpha=5/N_infra;
        s.MarkerFaceColor=[0.5 0.5 0.5];

        axis equal
        grid
        hold on
```

```
        s=scatter(norm_red(1:end-
1),norm_red(2:end),500,'filled');
        s.MarkerFaceAlpha=5/N_red;
        axis([0 2 0 2])
        s.MarkerFaceColor=[0.5 0.5 0.5];
        xlabel('période N-1 (s)');
        ylabel('période N (s)');
        title('Repos');
        text(0,0.1,'\fontsize{10} {\color{black}période
moyenne = 1.07 s}')
        text(1.1,1.9,'\fontsize{10}
{\color{black}sigma_1_1= 0.11 s}')
        text(1.1,1.8,'\fontsize{10}
{\color{black}sigma_x_y= 0.14 s}')


    case 2
        figure(1)
        subplot(1,3,2);
        s=scatter(norm_infra(1:end-
1),norm_infra(2:end),500,'filled');
        s.MarkerFaceAlpha=5/N_infra;
        s.MarkerFaceColor=[0.5 0.5 0.5];

        axis equal
        grid
        hold on
        s=scatter(norm_red(1:end-
1),norm_red(2:end),500,'filled');
        s.MarkerFaceAlpha=5/N_red;
        axis([0 2 0 2])
        s.MarkerFaceColor=[0.5 0.5 0.5];
        xlabel('période N-1 (s)');
        ylabel('période N (s)');
        title('Activité');
        text(0,0.1,'\fontsize{10} {\color{black}période
moyenne = 0.60 s}')
        text(1.1,1.9,'\fontsize{10}
{\color{black}sigma_1_1= 0.17 s}')
        text(1.1,1.8,'\fontsize{10}
{\color{black}sigma_x_y= 0.22 s}')


    case 3
        figure(1)
        subplot(1,3,3);
```

```
        s=scatter(norm_infra(1:end-
1),norm_infra(2:end),500,'filled');
        s.MarkerFaceAlpha=5/N_infra;
        s.MarkerFaceColor=[0.5 0.5 0.5];
        axis equal
        grid
        hold on
        s=scatter(norm_red(1:end-
1),norm_red(2:end),500,'filled');
        s.MarkerFaceAlpha=5/N_red;
        axis([0 2 0 2])
        s.MarkerFaceColor=[0.5 0.5 0.5];
        xlabel('période N-1 (s)');
        ylabel('période N (s)');
        title('Récupération');
        text(0,0.1,'\fontsize{10} {\color{black}période
moyenne = 0.92 s}')
        text(1.1,1.9,'\fontsize{10}
{\color{black}sigma_1_1= 0.18 s}')
        text(1.1,1.8,'\fontsize{10}
{\color{black}sigma_x_y= 0.16 s}')


end
```

Annexe 7. Programme Arduino heartbeat_fft pour afficher en direct le transformée de Fourier du signal

```
#include "arduinoFFT.h"
arduinoFFT FFT = arduinoFFT();
const uint16_t samples = 256; //This value MUST ALWAYS be a power of 2
const double signalFrequency = 1;
const double samplingFrequency = 10;
const uint8_t amplitude = 100;
double vReal[samples];
double vImag[samples];
#define SCL_INDEX 0x00
#define SCL_TIME 0x01
#define SCL_FREQUENCY 0x02
#define SCL_PLOT 0x03

//mesure
double value[samples];
double value_filter[samples];
unsigned long t;
int k;
double a=0.35;
double somme;
int w=1;

void setup (){
 pinMode(A0,INPUT);
 Serial.begin(115200);
 }

void loop () {
 if (w==1)
 for (int u=0;u<samples;u++){
  value[u]=analogRead(A0);
  delay(100);
 };
 w=0;
 //mesure
 for (k=0;k<samples;k++){
  value[k]=analogRead(A0);
  delay(100);
  somme=somme+value[k];
  int j=k%16;
  if (j==15)
   Fft(value);
 }

}
```

```
void Fft (double  value[64]){
 /* Build raw data */
 //double cycles = (((samples-1) * signalFrequency) / samplingFrequency); //Number of signal cycles that the sampling will read

 for (int i=0;i<samples;i++){
 vReal[i]= value[i];
 vImag[i]= 0.0; //Imaginary part must be zeroed in case of looping to avoid wrong calculations and overflows
 }
 /* Print the results of the simulated sampling according to time */
 //Serial.println("Data:");
 //PrintVector(vReal, samples, SCL_TIME);
 FFT.Windowing(vReal, samples, FFT_WIN_TYP_HAMMING, FFT_FORWARD);  /* Weigh data */
 //Serial.println("Weighed data:");
 //PrintVector(vReal, samples, SCL_TIME);
 FFT.Compute(vReal, vImag, samples, FFT_FORWARD); /* Compute FFT */
 //Serial.println("Computed Real values:");
 //PrintVector(vReal, samples, SCL_INDEX);
 //Serial.println("Computed Imaginary values:");
 //PrintVector(vImag, samples, SCL_INDEX);
 FFT.ComplexToMagnitude(vReal, vImag, samples); /* Compute magnitudes */
 //Serial.println("Computed magnitudes:");
 //PrintVector(vReal, (samples >> 1), SCL_FREQUENCY);
 double x = FFT.MajorPeak(vReal, samples, samplingFrequency);
 float batt= x*60;
 Serial.println(batt,1);
}

void PrintVector(double *vData, uint16_t bufferSize, uint8_t scaleType)

{
 for (uint16_t i = 0; i < bufferSize; i++)
 {
  double abscissa;
  /* Print abscissa value */
  switch (scaleType)
  {
   case SCL_INDEX:
    abscissa = (i * 1.0);
 break;
   case SCL_TIME:
    abscissa = ((i * 1.0) / samplingFrequency);
 break;
   case SCL_FREQUENCY:
    abscissa = ((i * 1.0 * samplingFrequency) / samples);
 break;
  }
  Serial.print(abscissa, 6);
```

```
  // if(scaleType==SCL_FREQUENCY)
  //   Serial.print("Hz");
    Serial.print(" ");
    Serial.println(vData[i], 4);
  }
  Serial.println();
}
```