N° de candidat : 27520

T.I.P.E.

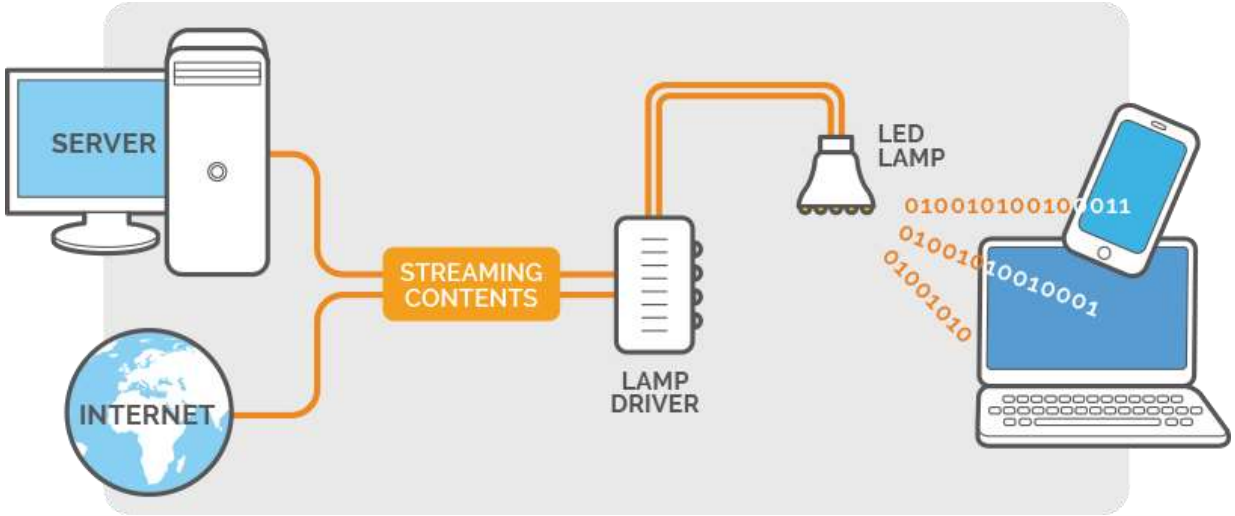
Thème : Transport

Transmission de données par modulation lumineuse

2017 - 2019

Transmission de données par modulation lumineuse

Introduction



Problématique :

Comment permettre une transmission fiable et efficace d'images par modulation lumineuse, à partir d'un système Li-Fi ?

Transmission de données par modulation lumineuse

Introduction

I] Elaboration du signal

1°) Compression

a. *Méthode naïve*

b. *Méthode des moyennes*

c. *Composantes connexes*

d. *Division d'images*

2°) Construction du signal

II] Transmission du signal

1°) Modèle

2°) Conception

a. *1er dispositif* : Transmission avec un fil

b. *2nd dispositif* : Transmission sans fil

3°) Transmission

III] Réception du signal

1°) Réorganisation

2°) Implémentation des codes correcteurs

3°) Résultats

Conclusion

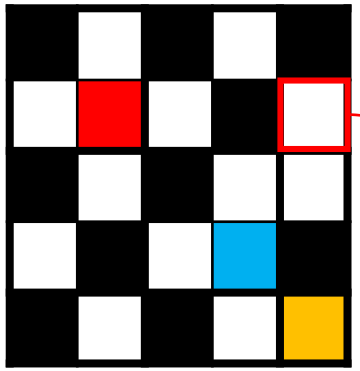
Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

a. *Méthode naïve*

Exemple :



Triplet RGB
(255,255,255)

Binaire
(11111111,11111111,11111111)

5x5	*1280x720 (720p)*	*1920x1080 (HD)*	*3840x2160 (UHD)*
75 octets	$2{,}8 \times 10^6$ octets 2,8 Mo	$6{,}2 \times 10^6$ bits 6,2 Mo	$2{,}5 \times 10^7$ bits 25 Mo

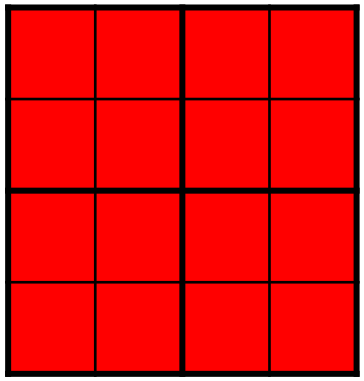
Transmission de données par modulation lumineuse

I] Elaboration du signal

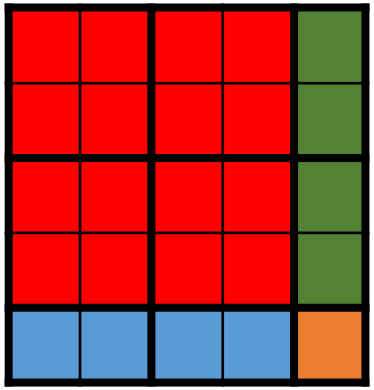
1°) Compression

b. *Méthode des moyennes*

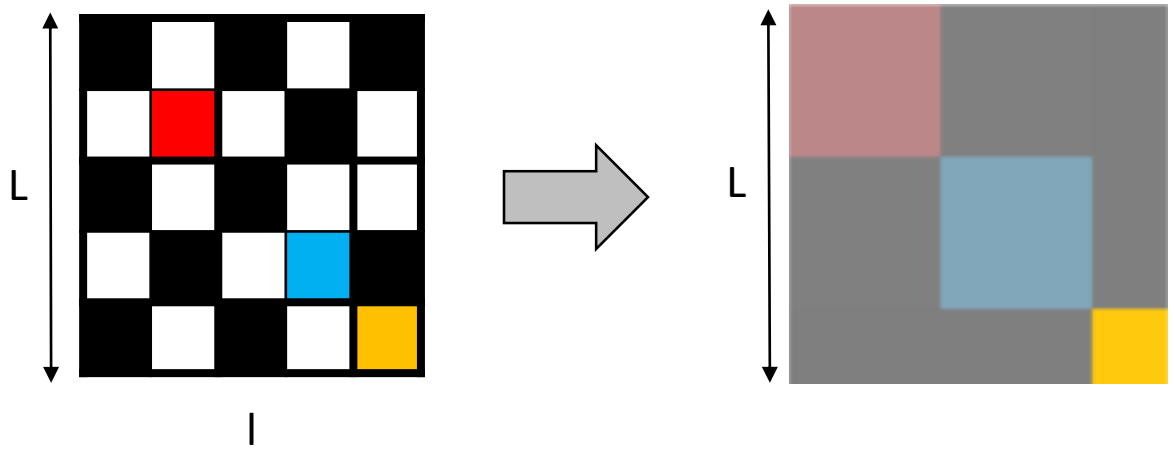
Cas n°1 : *dimensions paires*



Cas n°2 : *dimensions impaires*



Exemple :



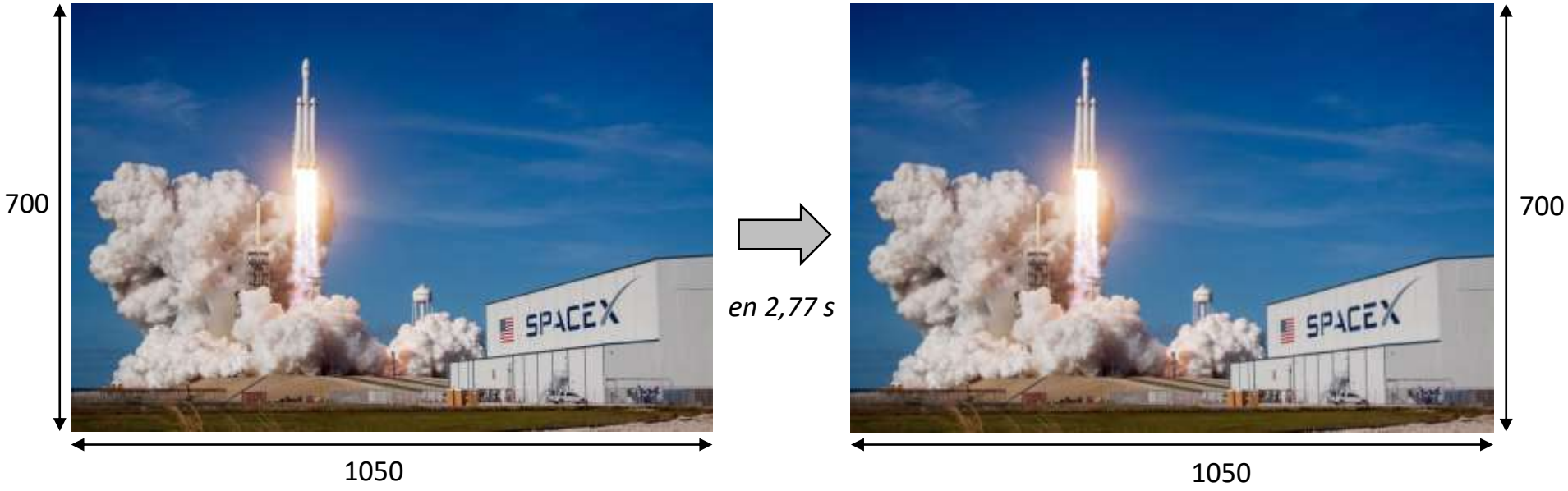
Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

b. *Méthode des moyennes*

Exemple :



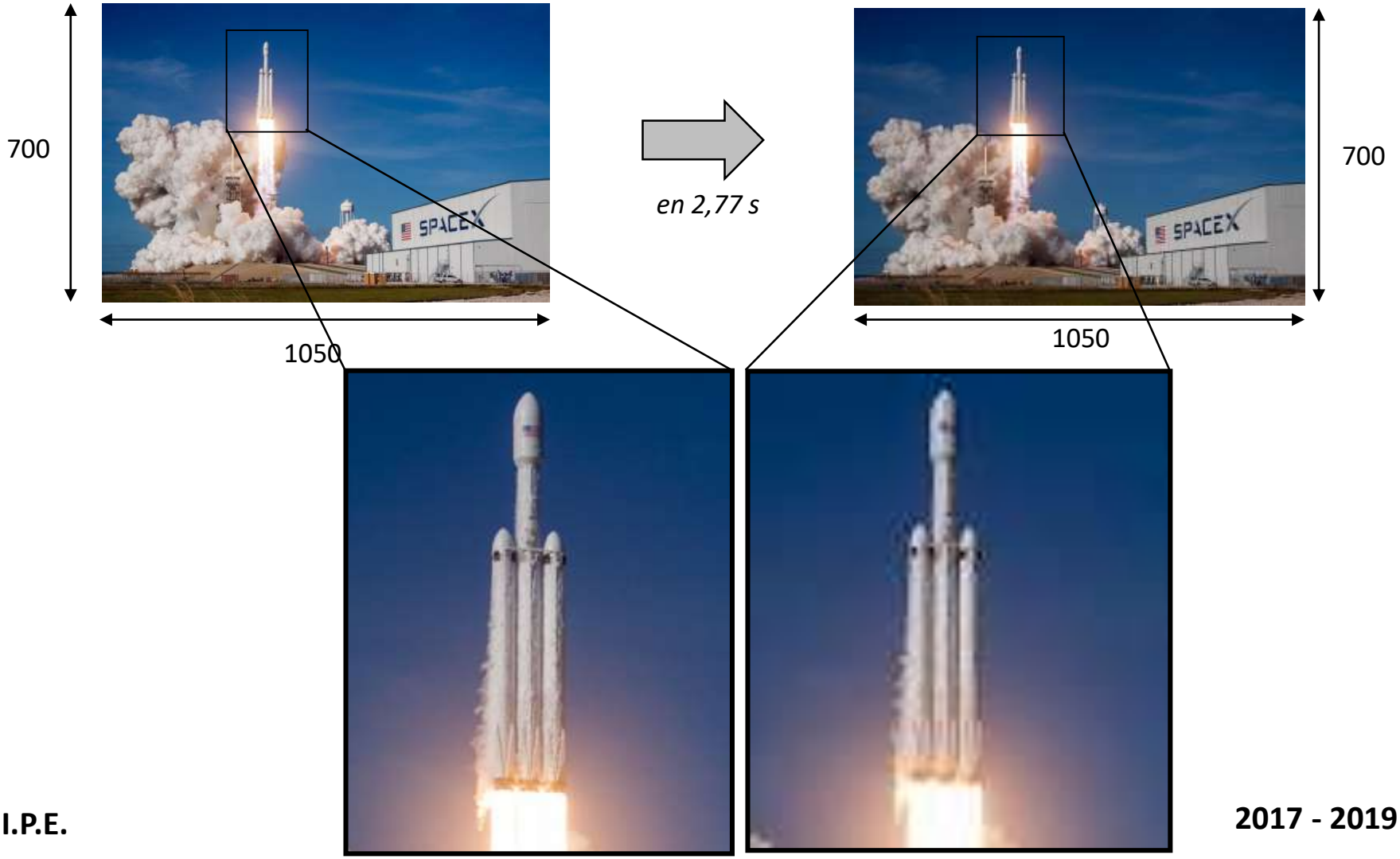
Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

b. *Méthode des moyennes*

Exemple :



Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

b. *Méthode des moyennes*

Exemple :

en 2,77 s

Avant compression :

$2{,}2 \times 10^6$ octets

2,2 Mo

Après compression :

$5{,}6 \times 10^5$ octets

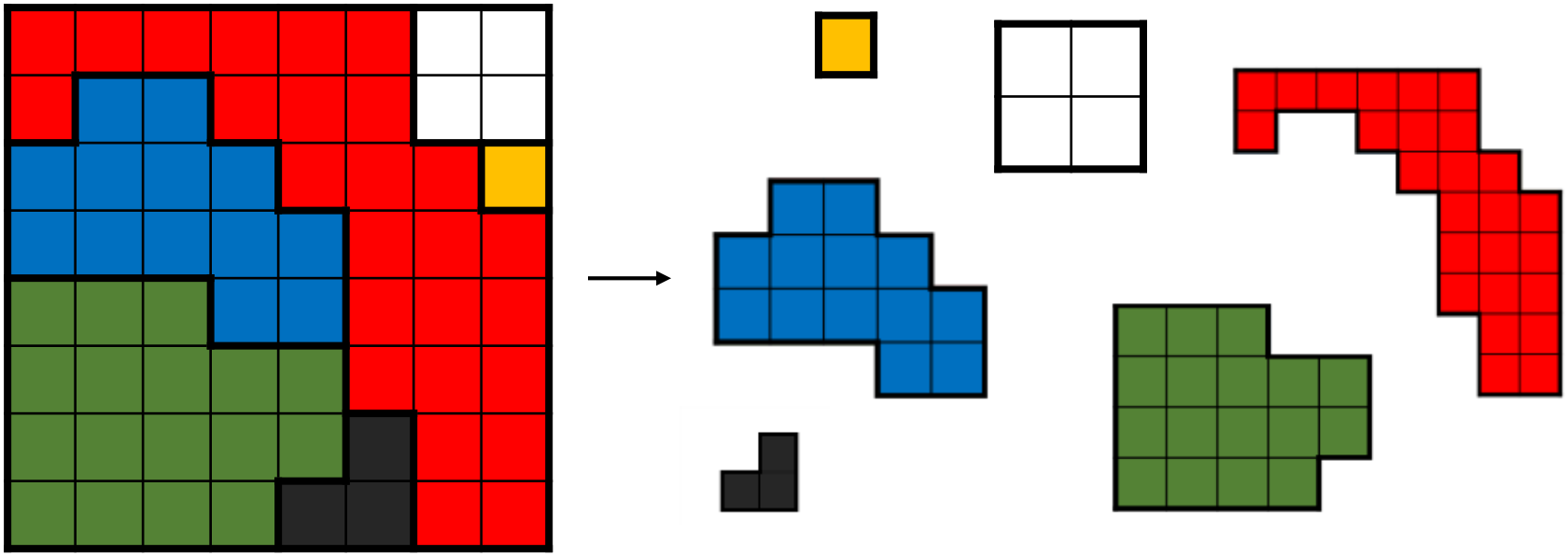
550 ko

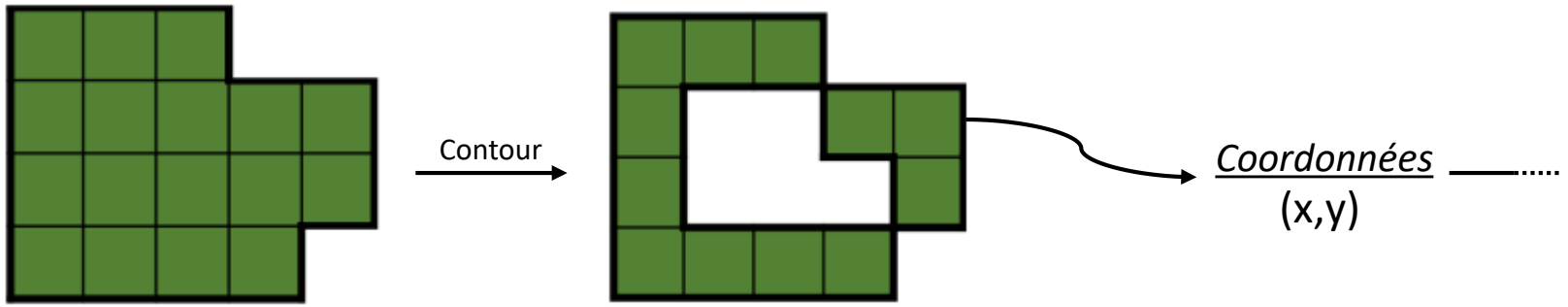
Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

c. *Composantes connexes*





Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

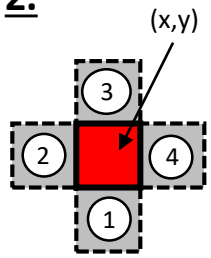
c. *Composantes connexes*

Programme : *Composantes connexes*

1. coord=[(0,0),(0,1),(0,2),(0,3),(0,4),…,(0,p-1),
(1,0),(1,1),(1,2),(1,3),(1,4),…,(1,p-1),
⋮
(n-1,0),(n-1,1),(n-1,2),(n-1,3),…,(n-1,p-1)]

➔ while coord!=[]

2.



Composante connexe :

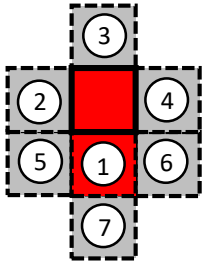
comp=[(x,y)]

Etude des voisins :

voisins=[(x1,y1),(x2,y2),(x3,y3),(x4,y4)]

➔ while voisins!=[]

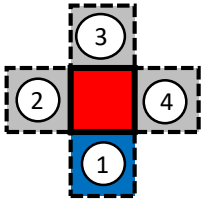
3. *1er cas :*



comp=[(x,y),(x1,y1)]

voisins=[(x2,y2),(x3,y3),(x4,y4),(x5,y5),(x6,y6),(x7,y7)]

2nd cas :



Composante connexe :

comp=[(x,y)]

Etude des voisins :

voisins=[(x2,y2),(x3,y3),(x4,y4)]

⋮

etc.

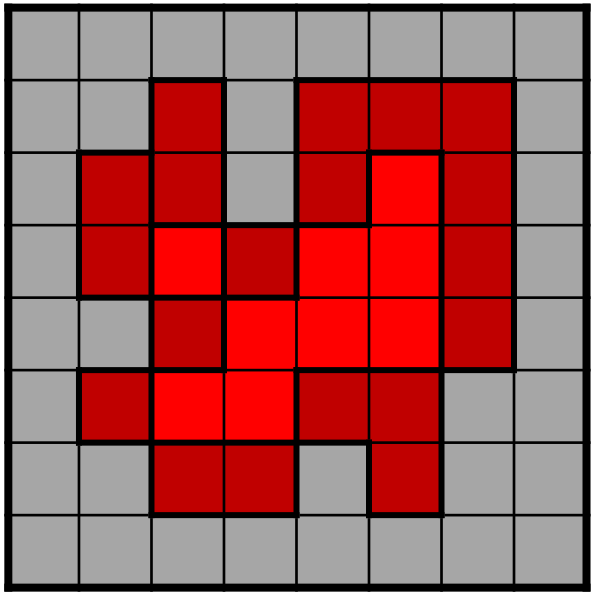
Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

c. *Composantes connexes*

Programme : *Contour*



appartient au contour :

admet au plus 3 voisins de même couleur.

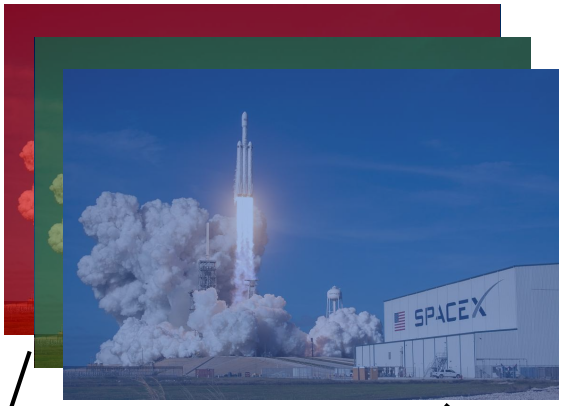
Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

c. *Composantes connexes*

Décomposition
R,*G*,*B*



R

G

B

Composantes connexes

Contours

Composantes connexes

Contours

Composantes connexes

Contours

Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

c. *Composantes connexes*

Image originale :

Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

c. *Composantes connexes*

Image compressée à 80 % (en 49,2 sec) :

Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

c. *Composantes connexes*

Image compressée à 90 % (en 39 sec) :

700

1050

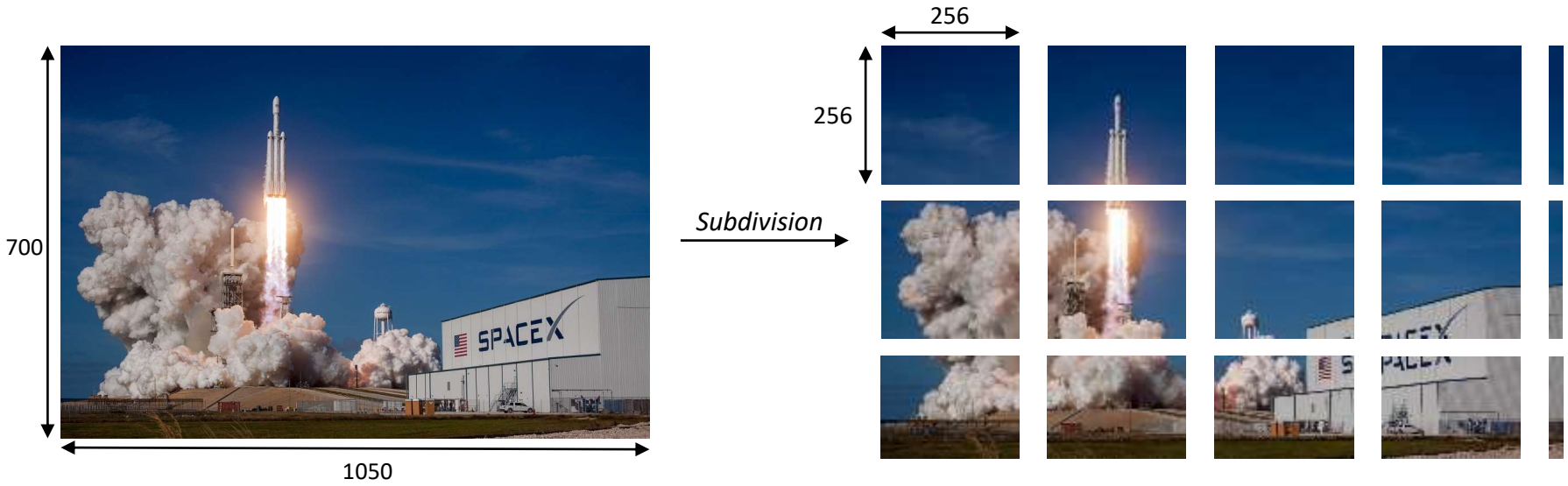
Image compressée à 95 % (en 32,6 sec) :

Transmission de données par modulation lumineuse

I] Elaboration du signal

1°) Compression

d. *Division d'images*



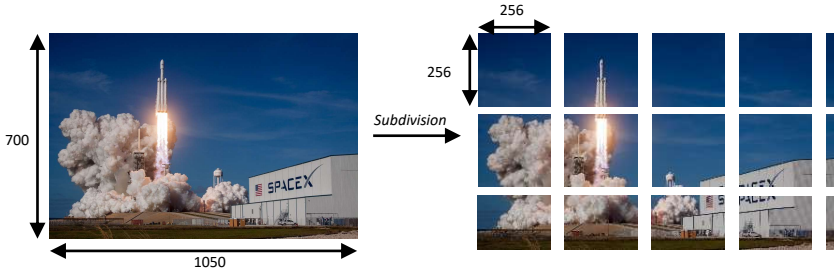
A chaque image 256*256 :

- Composantes connexes
- Contours des composantes connexes

I] Elaboration du signal

2°) Construction du signal

➔ Méthode de composantes connexes par division d'images



Organisation du signal :

- Dimensions de l'image totale
- Nombre d'images issue de la subdivision

-Pour chaque **image** issue de la subdivision :

- « 0 » si la dimension est de 256*256
- « 1 » suivi de la dimension sinon

-Pour chaque **composante R, G, B** de l'image issu de la subdivision :

- Nombre de couleurs

-Pour chaque **couleur** :

- Le numéro de la couleur
- Le nombre de composantes connexes associées à cette couleur

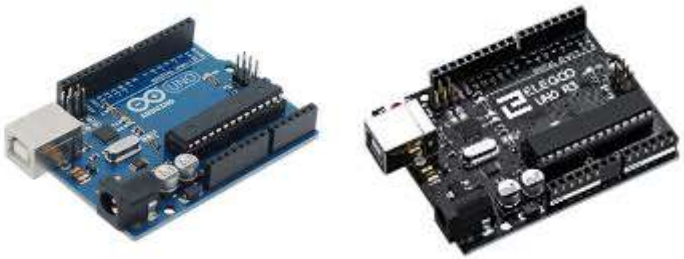
-Pour chaque **composante connexe** associée à cette couleur:

- Les contours de cette composante connexe

II] Transmition du signal

1°) Modèle

1er dispositif :



2 cartes Arduino

Laser rouge

Capteur

2nd dispositif :

\+

2nd Laser rouge

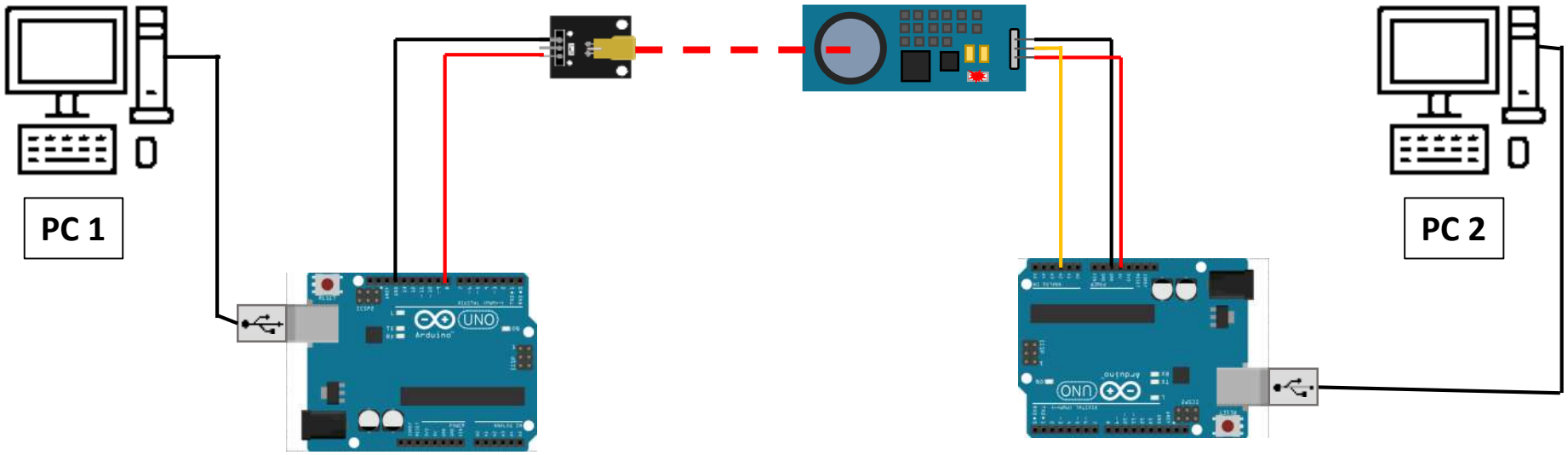
Résistance photosensible

Transmission de données par modulation lumineuse

II] Transmition du signal

1°) Modèle

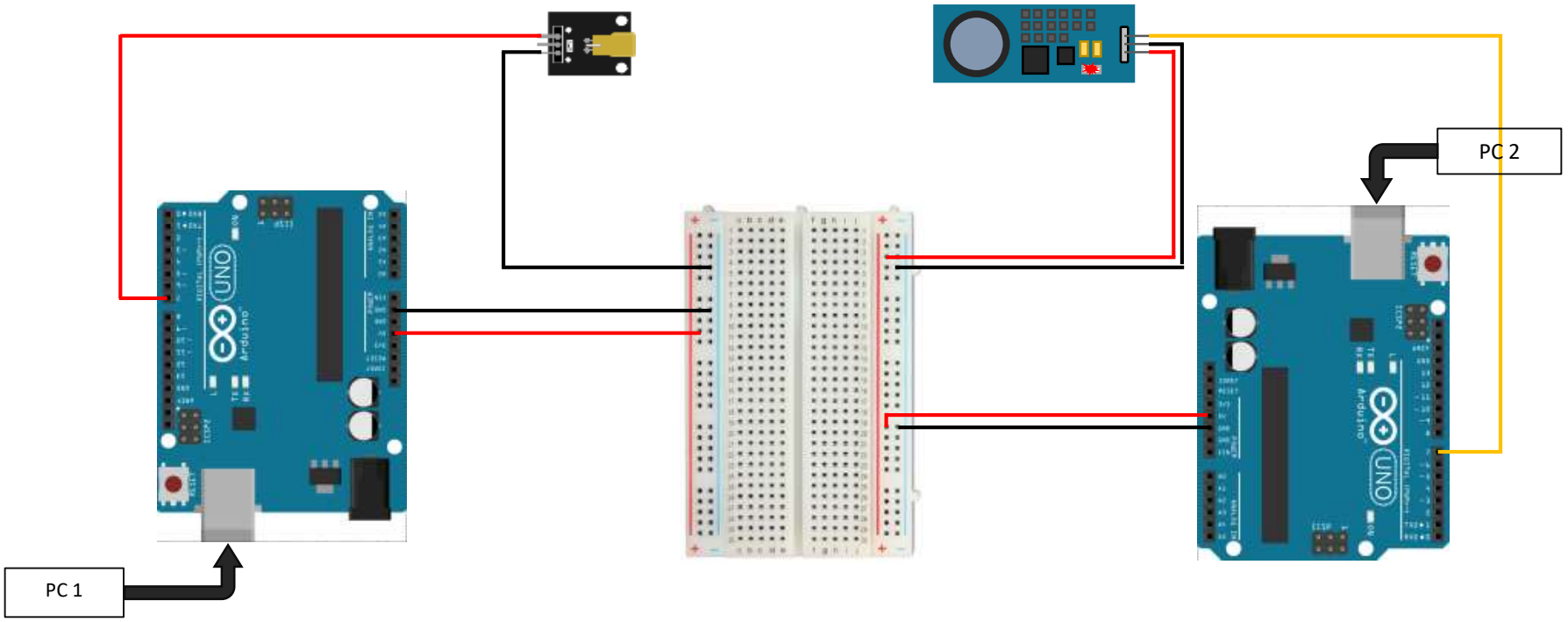
Laser : • **Allumé** : Transmission d'un 1
• **Eteint** : Transmission d'un 0



II] Transmission du signal

2°) Conception

a. *1er dispositif : transmission avec un fil*

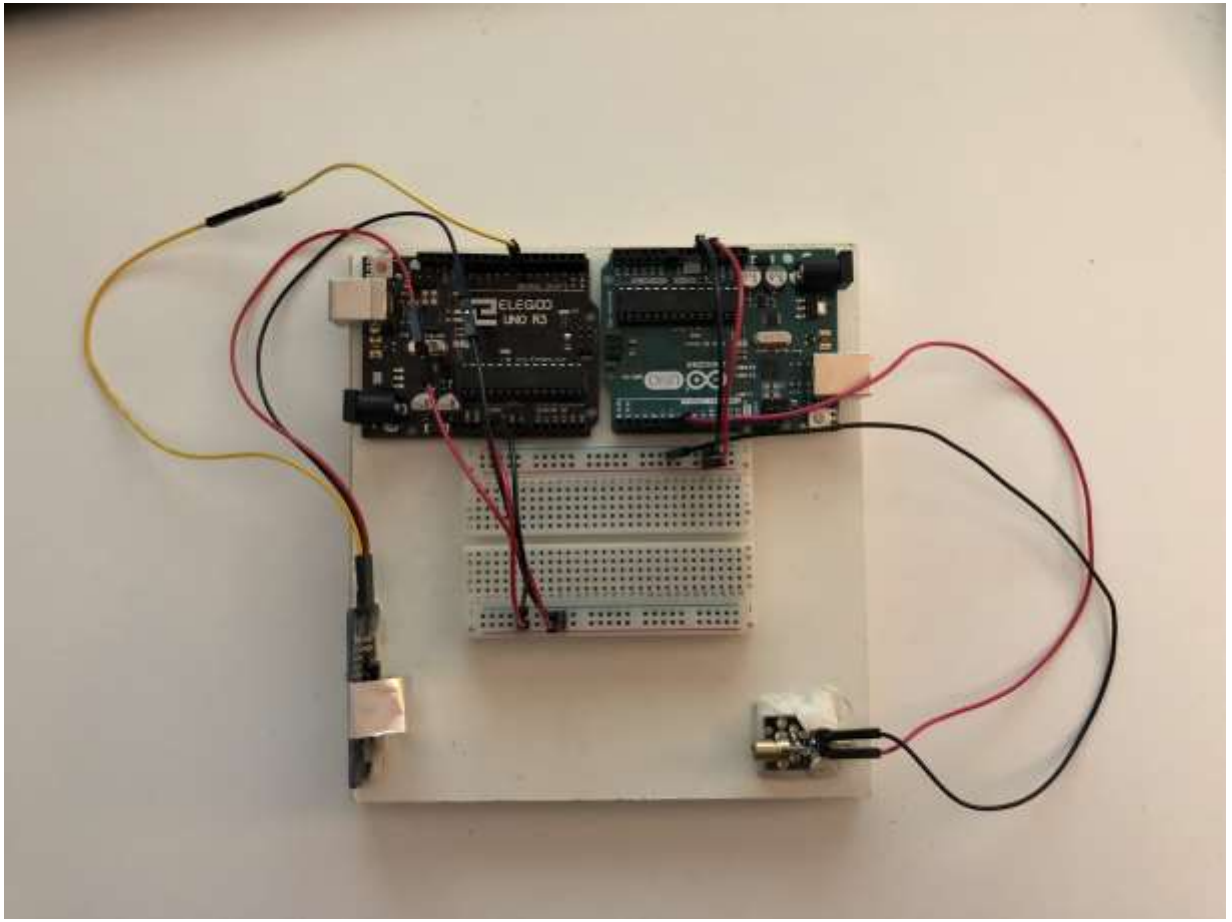


II] Transmission du signal

2°) Conception

a. *1er dispositif : transmission avec un fil*

Transmetteur Li-Fi 1 :



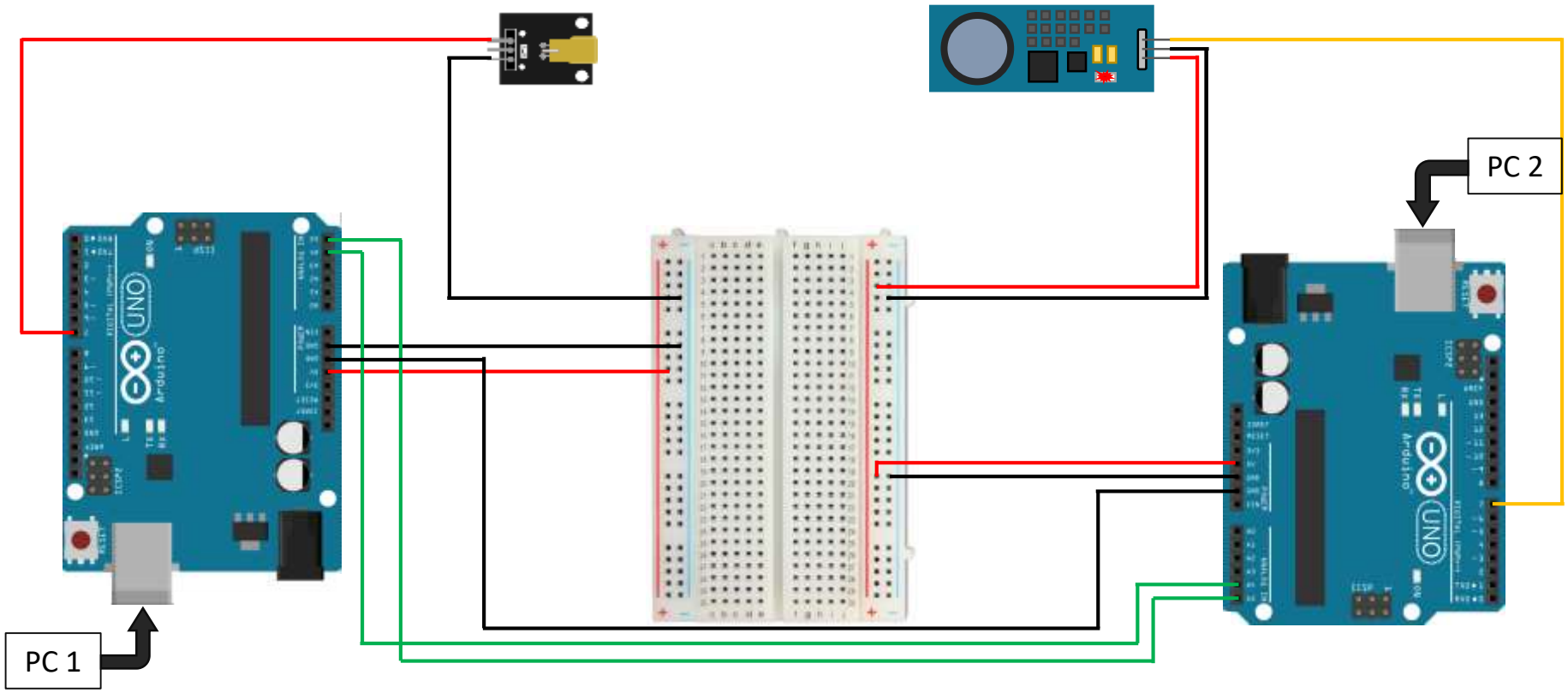
Transmission de données par modulation lumineuse

II] Transmission du signal

2°) Conception

a. *1er dispositif : transmission avec un fil*

Transmetteur Li-Fi 1 :



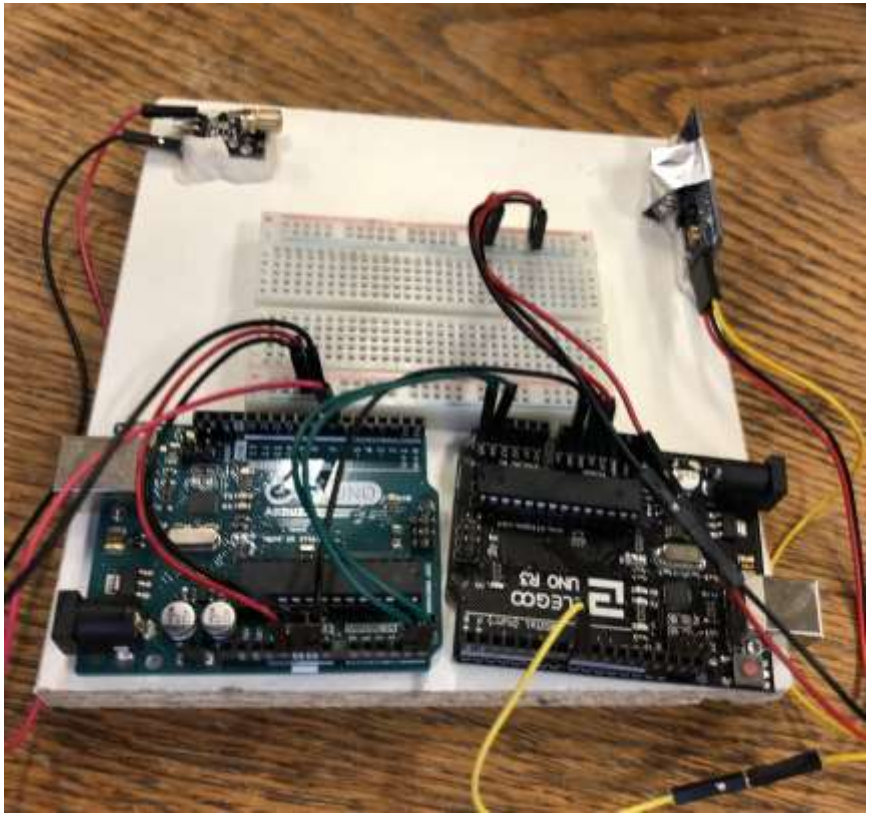
Relation Master/Reader entre les 2 cartes

II] Transmission du signal

2°) Conception

a. *1er dispositif : transmission avec un fil*

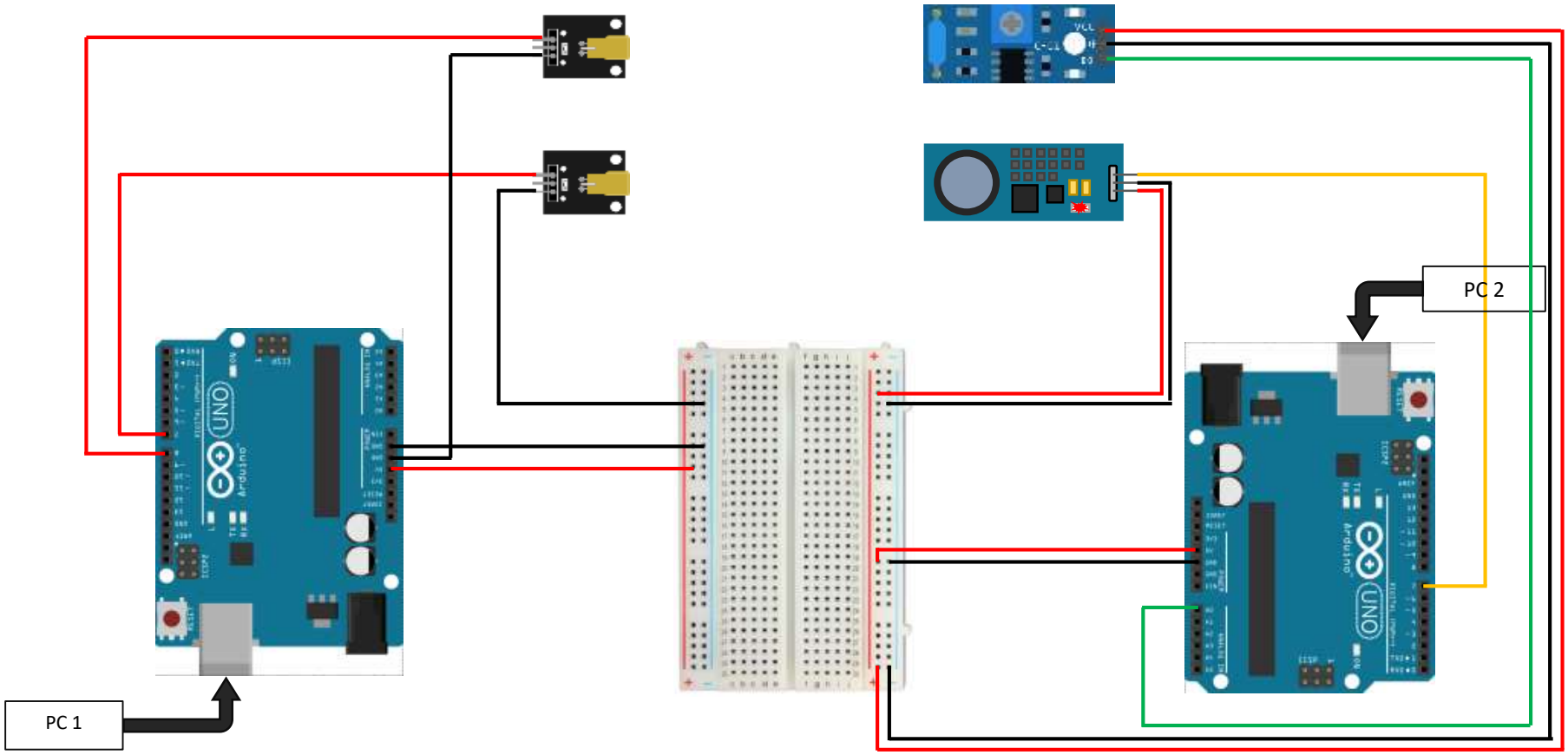
Transmetteur Li-Fi 1 :



II] Transmission du signal

2°) Conception

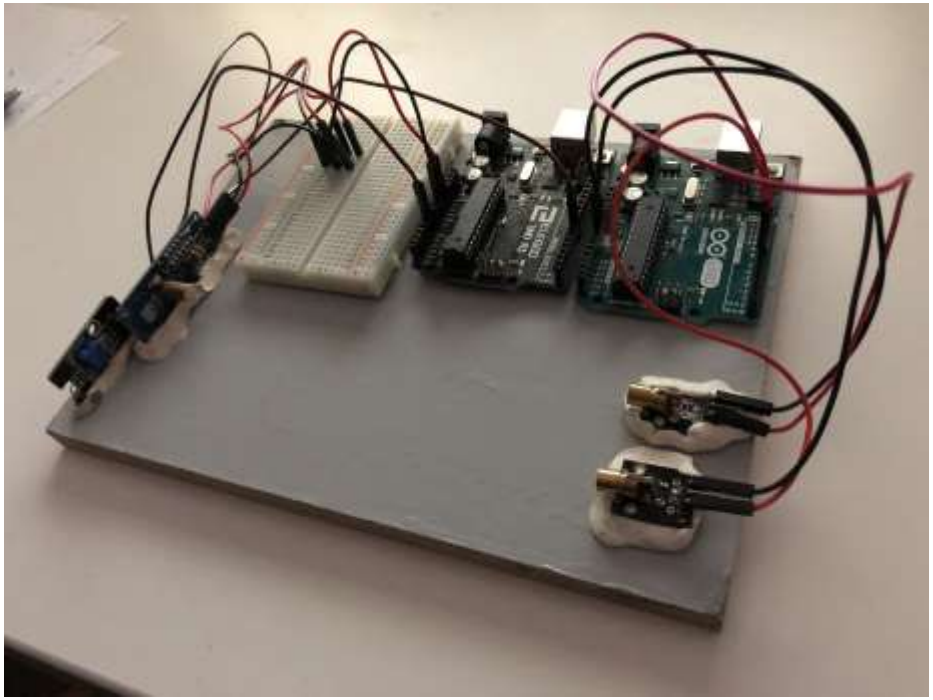
b. *2nd dispositif : transmission sans fil*

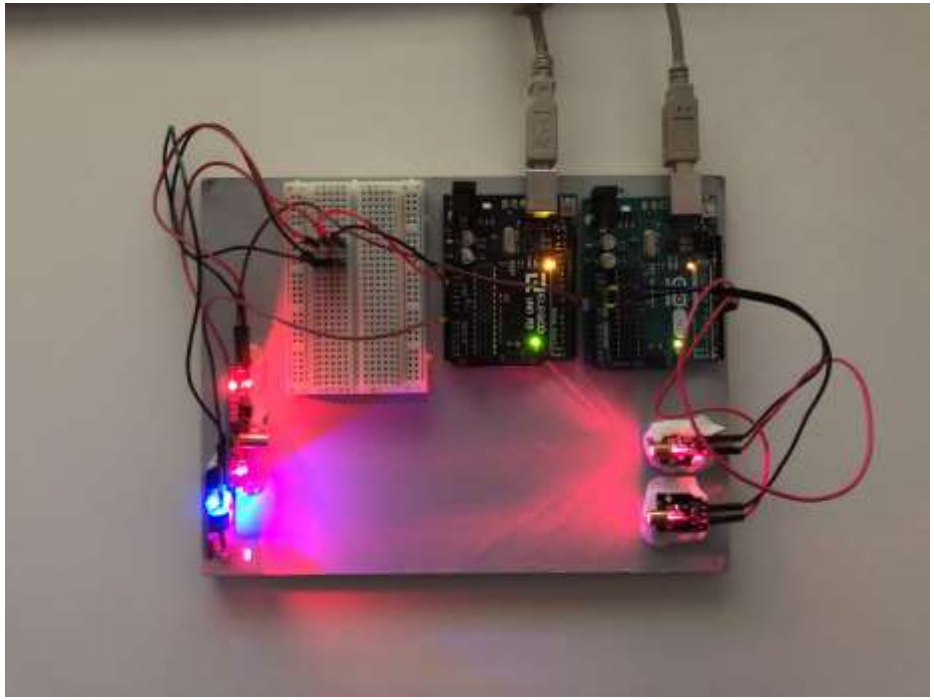


II] Transmission du signal

2°) Conception

b. *2nd dispositif : transmission sans fil*





II] Transmission du signal

2°) Conception

b. *2nd dispositif : transmission sans fil*

ARDUINO_-_LASER

```
char val;
int Laser1=7;
int Laser2=9;

void setup() {
  pinMode(Laser1, OUTPUT);
  pinMode(Laser2, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  if(Serial.available()>0){
    digitalWrite(Laser2,HIGH);
    int val=Serial.read();
    Serial.print(val);
    if (val=='1'){
    digitalWrite(Laser1, HIGH);
    delay(2);}
    if (val=='0') {
    digitalWrite(Laser1, LOW);
    delay(2);}
  else {
    digitalWrite(Laser2,LOW);
  }
  Serial.flush();
    }
  }
```

ARDUINO_-_DETECTEUR §

```
int Detecteur1 = 7;
int Detecteur2 = A0;
int data = 0;


void setup() {
  pinMode(Detecteur1, INPUT);
  pinMode(Detecteur2, INPUT);
  Serial.begin(9600);

  }

void loop() {
  while(analogRead(Detecteur2)<21){
      data = digitalRead(Detecteur1);
      Serial.write(data);
      delay(50);
      Serial.flush();
  }}
```

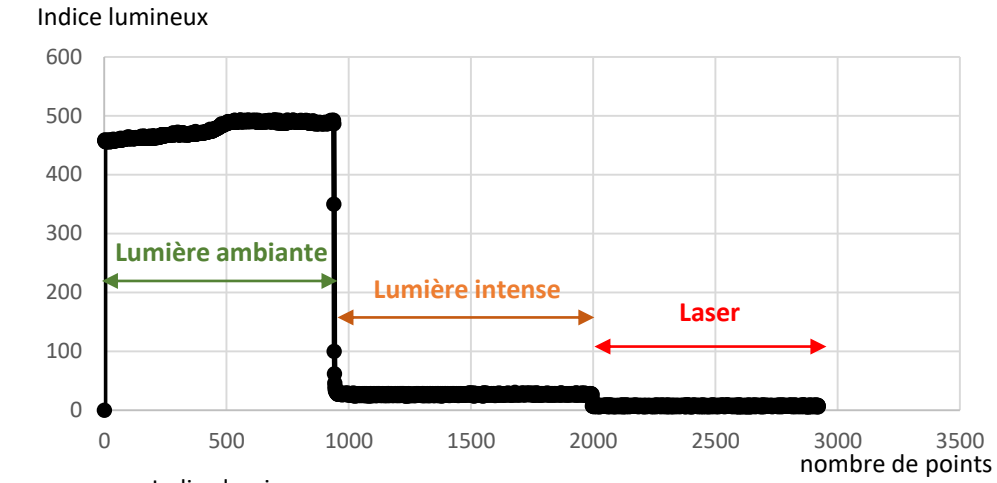
II] Transmission du signal

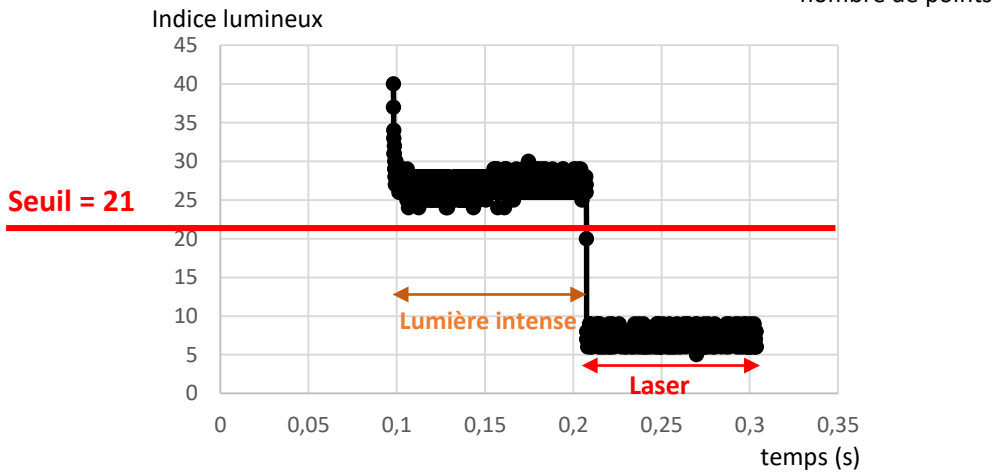
2°) Modèle

b. *2nd dispositif : transmission sans fil*

ARDUINO_-_DETECTEUR §

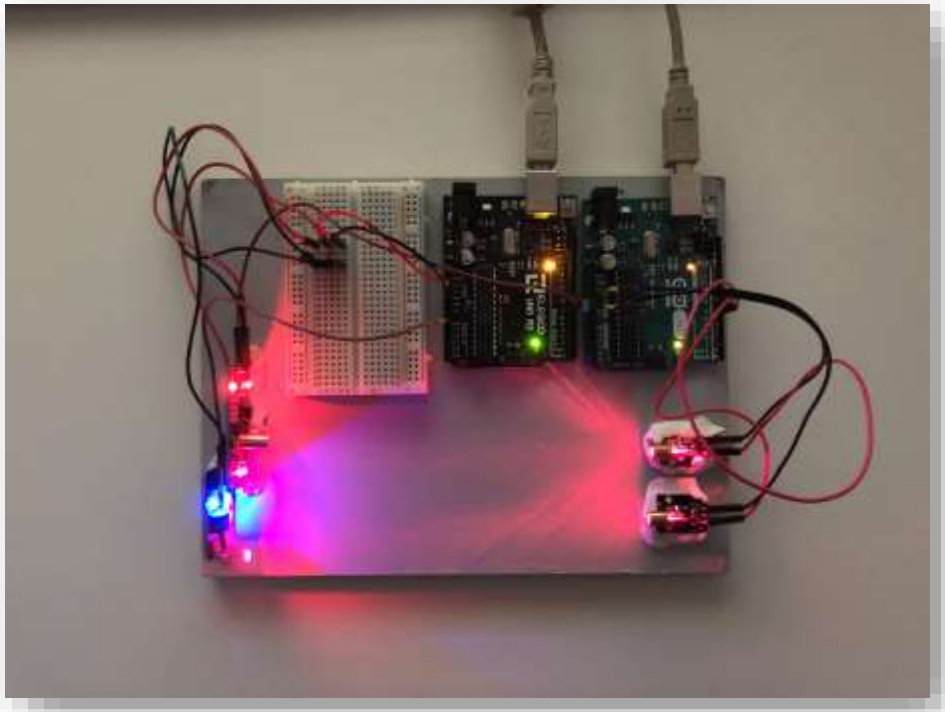
```
int Detecteur1 = 7;
int Detecteur2 = A0;
int data = 0;


void setup() {
  pinMode(Detecteur1, INPUT);
  pinMode(Detecteur2, INPUT);
  Serial.begin(9600);

  }

void loop() {
  while(analogRead(Detecteur2)<21){
      data = digitalRead(Detecteur1);
      Serial.write(data);
      delay(50);
      Serial.flush();
  }}
```





II] Transmission du signal

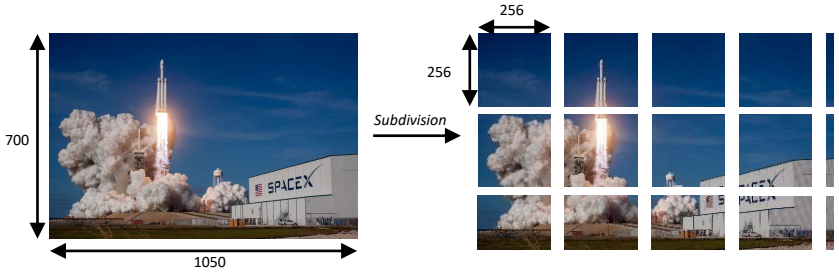
3°) Transmission



Succession d'état **« ouvert »** et **« fermé »** pour le laser, étant interprétés comme des **« 1 »** et des **« 0 »** par les cartes Arduino. La signal est ensuite transmis à l'ordinateur.

III] Réception du signal

1°) Réorganisation



Rappel :

Organisation du signal :

- Dimensions de l'image totale
- Nombre d'images issue de la subdivision

-Pour chaque **image** issue de la subdivision :

- « 0 » si la dimension est de 256*256
- « 1 » suivi de la dimension sinon

-Pour chaque **composante R, G, B** de l'image issu de la subdivision :

- Nombre de couleurs

-Pour chaque **couleur** :

- Le numéro de la couleur
- Le nombre de composantes connexes associées à cette couleur

-Pour chaque **composante connexe** associée à cette couleur:

- Les contours de cette composante connexe

III] Réception du signal

1°) Réorganisation

Réorganisation du signal :

- Dimensions de l'image totale
- Nombre d'images issue de la subdivision

-Pour chaque **image** issue de la subdivision :

- Dimension de l'image

-Pour chaque **composante R, G, B** de l'image issu de la subdivision :

-Pour chaque **couleur** :

- Le numéro de la couleur

-Pour chaque **composante connexe** associée à cette couleur:

- Les contours de cette composante connexe

Exemple :

[(10, 5), 1, [[10, 5],
[[**0**, [(0, 4), (0, 5), (1, 0), (1, 1), (1, 2), (1, 3), (1,6), (1, 7), (1, 8), (1, 9), (2, 0), (2, 1), (2, 2), (2, 3), (2, 6), (2, 7), (2,8), (2, 9), (3, 4), (3, 5), (4, 4), (4, 5)]], [**147**, [(4, 0), (4, 1), (4, 2), (4, 3)], [(4, 6), (4, 7), (4, 8), (4, 9)]], [**231**, [(0, 0), (0, 1), (0, 2), (0, 3)], [(0, 6), (0, 7), (0, 8), (0, 9)]], [**255**, [(3, 0), (3, 1), (3, 2), (3, 3)], [(3, 6), (3, 7), (3, 8), (3, 9)]]],

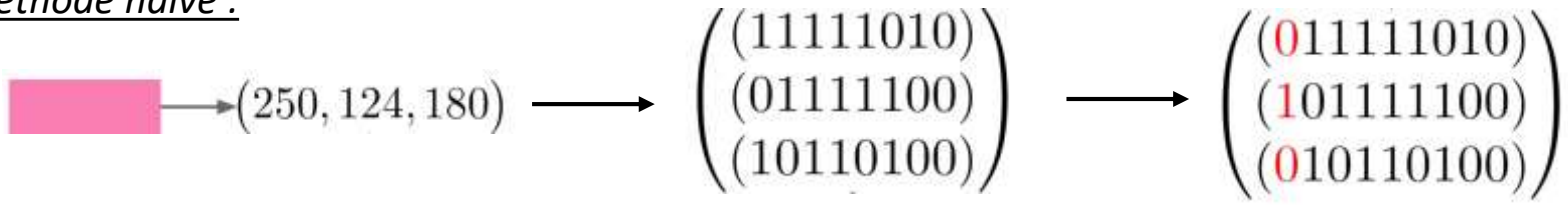
[[**0**, [(0, 4), (0, 5), (1, 4), (1, 5), (2, 0), (2, 1), (2, 2), (2, 3), (2, 6), (2, 7), (2, 8), (2, 9), (3, 4), (3, 5), (4, 4), (4, 5)]], [**21**, [(0, 0), (0, 1), (0, 2), (0, 3)], [(0, 6), (0, 7), (0, 8), (0, 9)]], [**63**, [(4, 0), (4, 1), (4, 2), (4, 3)], [(4, 6), (4, 7), (4, 8), (4, 9)]], [**126**, [(3, 0), (3, 1), (3, 2), (3, 3)], [(3, 6), (3, 7), (3, 8), (3, 9)]], [**147**, [(1, 0), (1, 1), (1, 2), (1, 3)], [(1, 6), (1, 7), (1, 8), (1, 9)]]],

[[**0**, [(0, 4), (0, 5), (1, 4), (1, 5), (2, 0), (2, 1), (2, 2), (2, 3), (2, 6), (2, 7), (2, 8), (2, 9), (3, 4), (3, 5), (4, 4), (4, 5)]], [**21**, [(0, 0), (0, 1), (0, 2), (0, 3)], [(0, 6), (0, 7), (0, 8), (0, 9)], [(3, 0), (3, 1), (3, 2), (3, 3)], [(3, 6), (3, 7), (3, 8), (3, 9)]], [**147**, [(4, 0), (4, 1),(4, 2), (4, 3)], [(4, 6), (4, 7), (4, 8), (4, 9)]], [**231**, [(1, 0), (1, 1), (1, 2), (1, 3)], [(1, 6), (1, 7), (1, 8), (1, 9)]]]]

III] Réception du signal

2°) Implémentation des codes correcteurs

- *Méthode naïve :*



- *Méthode de Hamming :*

Exemple :

$$155$$
$$(10011011)$$
$$((1001)\quad(1011))$$
$$((cb1a001)\quad(cb1a011))$$
$$((0011001)\quad(0110011))$$

Message émis

$$((0011001)\quad(0110011))$$

Message reçu

$$((0011101)\quad(0110011))$$

$$1+1+0+1=1 \qquad a=1$$
$$0+1+0+1=0 \qquad b=0$$
$$0+1+1+1=1 \qquad c=1$$
$$abc=101 \longrightarrow 5$$

Erreur à la 5e position

Conclusion

Transmission de données par modulation lumineuse

Annexes :

```
## Compression moyenne

def col_lign_plus_moy():
    J=[]#Dernière ligne en trop
    K=[]#Dernière colonne en trop
    P=[]#Pixel en Bas à Droite
    l,L=i.size
    L1,l1=L,l
    L_=L1-1
    l_=l1-1
    if L%2==1:
        L=L-1
        if l%2==1:
            l=l-1
    if l%2==1:
        l=l-1
        if L%2==1:
            L=L-1
    if L1-L==1:
        for n in range(0,l_):
            J.append(i.getpixel((n,L_)))
    if l1-l==1:
        for t in range(0,L_):
            K.append(i.getpixel((l_,t)))
    P.append(i.getpixel((l_,L_)))
    n_J=[]
    n_K=[]
    if J!=[] and l_%2==0:#Moyenne de la ligne en plus
        a=-2
        b=-1
        for k in range(int((len(J)/2))):
            I=[]
            a+=2
            b+=2
            I.append(int((J[a][0]+J[b][0])/2))
            I.append(int((J[a][1]+J[b][1])/2))
            I.append(int((J[a][2]+J[b][2])/2))
            n_J.append(I)
    elif J!=[] and l_%2!=0:
        a=-2
        b=-1
        derl_p=[]
        derl_p.append(J[l_-1][0])
        derl_p.append(J[l_-1][1])
        derl_p.append(J[l_-1][2])
        for k in range((len(J))//2):
            I=[]
            a+=2
            b+=2
            I.append(int((J[a][0]+J[b][0])/2))
            I.append(int((J[a][1]+J[b][1])/2))
            I.append(int((J[a][2]+J[b][2])/2))
            n_J.append(I)
    if K!=[0] and L_%2==0:#Moyenne de la colonne en plus
        c=-2
        d=-1
        for l in range((len(K))//2):
            Q=[]
            c+=2
            d+=2
            Q.append(int((K[c][0]+K[d][0])/2))
            Q.append(int((K[c][1]+K[d][1])/2))
            Q.append(int((K[c][2]+K[d][2])/2))
            n_K.append(Q)
    elif K!=[] and L_%2!=0:
        c=-2
        d=-1
        derc_p=[]
        derc_p.append(K[L_-1][0])
        derc_p.append(K[L_-1][1])
        derc_p.append(K[L_-1][2])
        for l in range((len(K))//2):
            Q=[]
            c+=2
            d+=2
            Q.append(int((K[c][0]+K[d][0])/2))
            Q.append(int((K[c][1]+K[d][1])/2))
            Q.append(int((K[c][2]+K[d][2])/2))
            n_K.append(Q)
        if l1%2==1 and l1%2==0:
            ML=[]
            ML.append((derl_p[0]+P[0][0])//2)
            ML.append((derl_p[1]+P[0][1])//2)
            ML.append((derl_p[2]+P[0][2])//2)
            n_J.append(ML)
        if l1%2==1 and L1%2==0:
            MC=[]
            MC.append((derc_p[0]+P[0][0])//2)
            MC.append((derc_p[1]+P[0][1])//2)
            MC.append((derc_p[2]+P[0][2])//2)
            n_K.append(MC)
    return([P,n_J,n_K])

def dim1():#Retourne les dimensions paires de l'image
    l,L=i.size
    l_=(l//2)*2
    L_=(L//2)*2
    return(L_,l_)


def list_moy():
    E=dim1()
    L,l=E[0],E[1]
    a_m=-2
    a_M=-1
    R=[]
    M=[]
    for n in range(0,int((l/2)*l)):
        C1=[]
        C2=[]
        T=[]
        a_m+=2
        a_M+=2
        for k in range(0,L):
            pix1=i.getpixel((a_m,k))
            C1.append(pix1)
            pix2=i.getpixel((a_M,k))
            C2.append(pix2)
        T.append(C1)
        T.append(C2)
        a,b=0,1
        for z in range(int(L/2)):
            I=[]
            for p in range(0,3):#3 répétitions pour les 3 coords des pixels
                m=(T[0][a][p]+T[0][b][p]+T[1][a][p]+T[1][b][p])/4
                if m%1>=0.5 :
                    m+=1
                m=int(m)
                I.append(m)
            a+=2
            b+=2
            M.append(I)
    return(M)

def binary(A):
    J=[]
    for d in range((len(A))):
        a=A[d][0]
        b=A[d][1]
        c=A[d][2]
        J.append((bin(a)[2:]).zfill(8))
        J.append((bin(b)[2:]).zfill(8))
        J.append((bin(c)[2:]).zfill(8))
    return(J)


def compil_():
    A=list_moy()#Append list_moy
    B=col_lign_plus_moy ()
    if B[1]!=[]:
        for k in range (len(B[1])):
            A.append(B[1][k])#Append dernière ligne
    if B[2]!=[]:
        for m in range (len(B[2])):
            A.append(B[2][m])#Append dernière colonne
    lc,Li=i.size
    if Li%2==1 and lc%2==1:#Append dernier pixel si image impaire
        G=[]
        G.append(int(B[0][0][0]))
        G.append(int(B[0][0][1]))
        G.append(int(B[0][0][2]))
        A.append(G)
    return A

def data_to_laser():
    C=binary(compil_())
    F=[]
    for k in range(len(C)):
        for i in range(8):
            F.append(C[k][i])
    return(len(F))

def script():
    text2=open('data.txt')
    for k in data_to_laser():
        if k[0]==0:
            a=str(1)
            text2.write(a)
        elif k[0]==1:
            b=str(0)
            text2.write(b)
    text2.close()
    return(True)
```

Transmission de données par modulation lumineuse

Annexes :

```
def lifi_laser():
    time.sleep(1.68)
    A=data_to_laser()
    ser=serial.Serial('COM3', 9600)
    if ser.isOpen()==True:
        for k in range(0,len(A)):
            c=str(A[k])
            ser.write(str.encode(c))
            time.sleep(0.025)

#ser=serial.Serial('/dev/cu.usbmodem1401', 9600)

def Lifi_Detector():
    txt2=open('data3.txt','x')
    c=0
    while True and c<((len(compil_())*3*8)):
        c+=1
        ligne=ser.read()
        a=int.from_bytes(ligne,byteorder='little')
        if a==0:
            a=1
        else:
            a=0
        b=str(a)
        txt2.write(b)
    txt2.close()
    Conv_Final()

def decod():
    F=[]
    txt3=open('data.txt','r')
    L=txt3.read
    print(L)
    a=0
    b=8
    for l in range(int(len(L)/24)):
        A=[]
        t1=L[a:b]
        A.append(int(t1,2))
        a+=8
        b+=8
        t2=L[a:b]
        A.append(int(t2,2))
        a+=8
        b+=8
        t3=L[a:b]
        A.append(int(t3,2))
        a+=8
        b+=8
        F.append(A)
    return F

def reorga():
    A=decod()
    l,L=1,size
    LF=[]
    if (L%2==0 and l%2==0):#Image paire
        a=0
        b=(L//2)
        for k in range(int(l/2)):
            C=[]
            for m in range(a,b):
                x=A[m][0]
                y=A[m][1]
                z=A[m][2]
                C.append((x,y,z))
            a+=(L//2)
            b+=(L//2)
            LF.append(C)
    if (l%2==0 and l%2==1) or (l%2==1 and l%2==0):#Colonnes impaires uniquement OU Lignes impaire uniquement
        a=0
        b=(L//2)
        while((len(LF))<(l//2)):#Tant que l'image paire n'est pas consitituée dans LF
            for k in range((l//2)):
                C=[]
                for m in range(a,b):
                    x=A[m][0]
                    y=A[m][1]
                    z=A[m][2]
                    C.append((x,y,z))
                a+=(L//2)
                b+=(L//2)
                LF.append(C)
        C_=[]
        for n in range(((L//2)*(l//2)), len(A)):#Colonne impaire uniquement OU Ligne
            x_=A[n][0]
            y_=A[n][1]
            z_=A[n][2]
            C_.append((x_,y_,z_))
        LF.append(C_)
    if(L%2==1 and l%2==1):#Image impaire
        while((len(LF))<(l//2)):#Tant que l'image paire n'est pas consitituée dans LF
            a=0
            b=(L//2)
            for k in range((l//2)):
                C=[]
                for m in range(a,b):
                    x=A[m][0]
                    y=A[m][1]
                    z=A[m][2]
                    C.append((x,y,z))
                a+=(L//2)
                b+=(L//2)
                LF.append(C)
        C_=[]
        for n in range(((L//2)*(l//2)),(((L//2)*(l//2))+((l-1)//2))):#Ligne impaire uniquement
            x_=A[n][0]
            y_=A[n][1]
            z_=A[n][2]
            C_.append((x_,y_,z_))
        LF.append(C_)
        C__=[]
        for p in range(((L//2)*(l//2))+((l-1)//2)),(((L//2)*(l//2))+((l-1)//2)+((L-1)//2))):#Colonne impaire uniquement
            x__=A[p][0]
            y__=A[p][1]
            z__=A[p][2]
            C__.append((x__,y__,z__))
        LF.append(C__)
        C___=[]
        x___=A[int((len(A))-1)][0]
        y___=A[int((len(A))-1)][1]
        z___=A[int((len(A))-1)][2]
        C___.append((x___,y___,z___))
        LF.append(C___)
    return(LF)
```

Transmission de données par modulation lumineuse

Annexes :

```
def Conv_Final():
    a_=time.clock()
    R=reorga()
    l,L=i.size
    img=i.size
    nl=len(R)
    img=Image.new("RGB",(l,L))
    pixel=img.load()
    if (l%2==0 and L%2==0):#Nombre de colonnes et lignes paire
        for k in range(int(l//2)):
            for m in range(int(L//2)):
                A=R[k][m]
                pixel[2*k,2*m]=A
                pixel[2*k,(2*m)+1]=A
                pixel[(2*k)+1,2*m]=A
                pixel[(2*k)+1,(2*m)+1]=A
    img.save('img_finale.png','PNG')
    if (L%2==1 and l%2==0):  #Nombre de lignes impaire et nombre colonnes paire
        for k in range(int(l//2)):
            for m in range(int(L//2)):
                A=R[k][m]
                pixel[2*k,2*m]=A
                pixel[2*k,(2*m)+1]=A
                pixel[(2*k)+1,2*m]=A
                pixel[(2*k)+1,(2*m)+1]=A
        for p in range(int(len(R[l//2]))):
            B=R[l//2][p]
            pixel[2*p,L-1]=B
            pixel[(2*p)+1,L-1]=B
    img.save('img_finale.png','PNG')
    if (L%2==0 and l%2==1):  #Nombre de colonnes impaires et nombre lignes paires
        for k in range(int(l//2)):
            for m in range(int(L//2)):
                A=R[k][m]
                pixel[2*k,2*m]=A
                pixel[2*k,(2*m)+1]=A
                pixel[(2*k)+1,2*m]=A
                pixel[(2*k)+1,(2*m)+1]=A
        for p in range(int(len(R[l//2]))):
            B=R[l//2][p]
            pixel[l-1,2*p]=B
            pixel[l-1,(2*p)+1]=B
    img.save('img_finale.png','PNG')
    if (l%2==1 and L%2==1):  #Nombre de colonnes et lignes impaires
        for k in range(int(l//2)):
            for m in range(int(L//2)):
                A=R[k][m]
                pixel[2*k,2*m]=A
                pixel[2*k,(2*m)+1]=A
                pixel[(2*k)+1,2*m]=A
                pixel[(2*k)+1,(2*m)+1]=A
        for p in range(int(len(R[l//2]))):
            B=R[l//2][p]
            pixel[2*p,L-1]=B
            pixel[(2*p)+1,L-1]=B
        for p in range(int(len(R[(l//2)+1]))):
            B=R[(l//2)+1][p]
            pixel[l-1,2*p]=B
            pixel[l-1,(2*p)+1]=B
        pixel[l-1,L-1]=R[(l//2)+2][0]
    img.save('img_finale.png','PNG')
    b_=time.clock()
    return (b_-a_)
```

```
## Composantes connexes

def img_decomp():
    R=[]
    G=[]
    B=[]
    l,L=i.size
    for k in range (0,L):
        lignR=[]
        lignG=[]
        lignB=[]
        for j in range (0,l):
            lignR.append(i.getpixel((j,k))[0])
            lignG.append(i.getpixel((j,k))[1])
            lignB.append(i.getpixel((j,k))[2])
        R.append(lignR)
        G.append(lignG)
        B.append(lignB)
    return(R,G,B)

def img_decomp2():
    R,G,B=img_decomp()
    a=open("RGB.txt","a")
    for k in R:
        a.write(str(k))
    a.write("\n")
    for l in G:
        a.write(str(l))
    a.write("\n")
    for m in B:
        a.write(str(m))
    a.write("\n")
    a.close()


def verif_dim(i,j,n,p):
    if i<=(n-1) and i>=0 and j>=0 and j<=p-1:
        return(True)
    else:
        return(False)

def verif_dim2(A,lign,col):
    if A[0]<=(lign-1) and A[0]>=0 and A[1]>=0 and A[1]<=(col-1):
        return(True)
    else:
        return(False)

def voisins1(A,n,p):#A=[x,y] du point a tester, et n,p les dimensions de l'image
    L=[]
    i=int(A[0])
    j=int(A[1])
    if verif_dim(i,j-1,n,p):
        L.append((i,j-1))
    if verif_dim(i,j+1,n,p):
        L.append((i,j+1))
    if verif_dim(i-1,j,n,p):
        L.append((i-1,j))
    if verif_dim(i+1,j,n,p):
        L.append((i+1,j))
    return(L)
```

Annexes :

```
def voisins(A,lign,col):#A est de la forme (y,x) avec y num de la lign et x le num de la colonne
    L=[]
    x=A[1]
    y=A[0]
    B=[(y,x+1),(y,x-1),(y+1,x),(y-1,x)]
    for k in B:
        if verif_dim2(k,lign,col)==False:
            B.remove(k)
    return(B)


def carto_pos(n,p):
    C=[]
    for l in range(0,n):
        for c in range(0,p):
            C.append((l,c))
    return(C)

def is_not_in(A,B):#si A n'est pas dans B
    c=0
    for k in (B):
        if A==k:
            c+=1
    if c==0:
        return(True)

def list_color():
    p,n=i.size
    R_color=[]
    G_color=[]
    B_color=[]
    R,G,B=img_decomp()
    for l in range(0,n):
        for c in range(0,p):
            if is_not_in(R[l][c],R_color):
                R_color.append(R[l][c])
            if is_not_in(G[l][c],G_color):
                G_color.append(G[l][c])
            if is_not_in(B[l][c],B_color):
                B_color.append(B[l][c])
    return(sorted(R_color),sorted(G_color),sorted(B_color))

def list_comp_conv():
    LR,LG,LB=[],[],[]
    a=0
    R,G,B=list_color()
    for k in range(0,len(R)):
        LR.append(a)
    for l in range(0,len(G)):
        LG.append(a)
    for m in range(0,len(B)):
        LB.append(a)
    return(LR,LG,LB)

def list_coord():
    l,L=i.size
    C=[]
    for k in range(0,L):
        for j in range(0,l):
            C.append((k,j))
    return(C)
```

```
def return_pos(a,M): # a->Couleur, M->Liste des couleurs
    pos=0
    for m in range(0,len(M)):
        if M[m]==a:
            return(m)

def compo_connexeR():
    a=time.clock
    I=[]
    p,n=i.size
    R_color=img_decomp()[0]
    L=list_coord()
    while L!=[]:
        for k in L:
            B=voisins(k,n,p)
            compo=[]
            compo.append(k)
            L.remove(k)
            while B!=[]:
                for v in B:
                    if not is_not_in(v,L):
                        if R_color[k[0]][k[1]]==R_color[v[0]][v[1]]:
                            L.remove(v)
                            compo.append(v)
                            for j in voisins(v,n,p):
                                B.append(j)
                    B.remove(v)
            I.append(compo)
    b=time.clock
    return(I)

def compo_connexeG():
    I=[]
    p,n=i.size
    G_color=img_decomp()[1]
    L=list_coord()
    while L!=[]:
        for k in L:
            B=voisins1(k,n,p)
            compo=[]
            compo.append(k)
            L.remove(k)
            while B!=[]:
                for v in B:
                    if not is_not_in(v,L):
                        if G_color[k[0]][k[1]]==G_color[v[0]][v[1]]:
                            L.remove(v)
                            compo.append(v)
                            for j in voisins1(v,n,p):
                                B.append(j)
                    B.remove(v)
            I.append(compo)
    return(I)
```

Annexes :

```
def compo_connexeB():
    I=[]
    p,n=i.size
    B_color=img_decomp()[2]
    L=list_coord()
    while L!=[]:
        for k in L:
            B=voisins(k,n,p)
            compo=[]
            compo.append(k)
            L.remove(k)
            while B!=[]:
                for v in B:
                    if not is_not_in(v,L):
                        if B_color[k[0]][k[1]]==B_color[v[0]][v[1]]:
                            L.remove(v)
                            compo.append(v)
                            for j in voisins(v,n,p):
                                B.append(j)
                    B.remove(v)
            I.append(compo)
    return(I)

def color_reorga(L_color,compo_conn,decomp_img):#avec L_color:list_color()[0,1,2]
    LR=[]
    for k in (L_color):
        A=[]
        for g in range(len(compo_conn)):
            if decomp_img[compo_conn[g][0][0]][compo_conn[g][0][1]]==k:
                A.append(compo_conn[g])
        LR.append(A)
    return(LR)

#color_reorga(list_color()[0],compo_connexeR(),img_decomp()[0])
#contour(color_reorga(list_color()[0],compo_connexeR(),img_decomp()[0]))


def contour(col_reorga): #Argument : color_reorga(...)
    p,n=i_.size
    F=[]
    for k in range(0,len(col_reorga)):
        E=[]
        for t in range(0, len(col_reorga[k])):
            G=[]
            for j in col_reorga[k][t]:
                V=voisins(j,n,p)
                for l in V:
                    if is_not_in(l,col_reorga[k][t]):
                        V.remove(l)
                if len(V)!=4:
                    G.append(j)
            E.append(G)
        F.append(E)
    return(F)

def val_entiere(n):
    n=int(n)
    return(n)

def is_btw(x,a,b):
    if x>=a and x<b:
        return True
```

```
def copie(A):
    B=[]
    for k in A:
        B.append(k)
    return(B)

## Compression d'images

def compr_color(p):#p poucentage de compression par couleur (RGB)
    if p==100:
        return([0,255])
    else:
        a=1-(p/100)
        n=a*255
        nbr_col=val_entiere(n)
        id_col=val_entiere(255/nbr_col)
        intervalles=[0]
        for k in range (0,nbr_col):
            nbr=intervalles[k]+id_col
            if nbr<255:
                intervalles.append(nbr)
        intervalles.append(255)
    return(intervalles)

def compr_color1(p):
    if p==100:
        return([0,255])
    else:
        a=1-(p/100)
        pro=a*(255*255*255)
        nbr_col=int((pro)**(1/3))+1
        id_col=int(255/nbr_col)
        intervalles=[0]
        print(id_col)
        for k in range (0,nbr_col):
            nbr=intervalles[k]+id_col
            if nbr<255:
                intervalles.append(nbr)
        intervalles.append(255)
    return(intervalles)


def compr_list_color(p):
    L_fin=[]
    I=compr_color(p)
    L=list_color()
    for k in L:
        A=[]
        for i in k:
            comt=0
            for l in range(0,len(I)-1):
                if i>I[l] and i<I[l+1]:
                    A.append(I[l])
            for n in range(0,len(I)):
                if i==I[n]:
                    A.append(I[n])
            A=list(set(A))
            A=sorted(A)
        L_fin.append(A)
    return(L_fin)
```

Annexes :

```
def compr_img_decomp(p):
    A=img_decomp()
    L=compr_list_color(p)
    L_final=[]
    for k in range(0,len(A)):
        A_0=A[k]
        LN=[]
        for j in range(0,len(A_0)):
            B=[]
            for l in A_0[j]:
                for z in range(0,len(L[k])-1):
                    if l>L[k][z] and l<L[k][z+1]:
                        B.append(L[k][z])
                for y in range(0,len(L[k])):
                    if l==L[k][y]:
                        B.append(L[k][y])
                if l>L[k][len(L[k])-1]:
                    B.append(L[k][len(L[k])-1])
            LN.append(B)
        L_final.append(LN)
    return(L_final)

def compr_screen(p):
    a=time.clock()
    t,L=i.size
    A=compr_img_decomp(p)
    img=Image.new("RGB",(t,L))
    pixel=img.load()
    for l in range(0,L):
        for j in range(0,t):
            pixel[j,l]=(A[0][l][j],A[1][l][j],A[2][l][j])
    b=time.clock()
    img.save('img_finale.png','PNG')
    return(b-a)

def compr_compo_connexeR(R_color):#R_color=compr_img_decomp(p)[0]
    I=[]
    p=int(len(R_color[0]))
    n=int(len(R_color))
    L=compr_list_coord()
    while L!=[]:
        for k in L:
            B=voisins(k,n,p)
            compo=[]
            compo.append(k)
            L.remove(k)
            while B!=[]:
                for v in B:
                    if not is_not_in(v,L):
                        if R_color[k[0]][k[1]]==R_color[v[0]][v[1]]:
                            L.remove(v)
                            compo.append(v)
                            for j in voisins(v,n,p):
                                B.append(j)
                    B.remove(v)
            I.append(compo)
    return(I)
```

```
def compr_compo_connexeG(G_color):#G_color=compr_img_decomp(p)[1]
    I=[]
    p=int(len(G_color[0]))
    n=int(len(G_color))
    L=list_coord()
    while L!=[]:
        for k in L:
            B=voisins(k,n,p)
            compo=[]
            compo.append(k)
            L.remove(k)
            while B!=[]:
                for v in B:
                    if not is_not_in(v,L):
                        if G_color[k[0]][k[1]]==G_color[v[0]][v[1]]:
                            L.remove(v)
                            compo.append(v)
                            for j in voisins(v,n,p):
                                B.append(j)
                    B.remove(v)
            I.append(compo)
    return(I)

def compr_compo_connexeB(B_color):#B_color=compr_img_decomp(p)[2]
    I=[]
    p=int(len(B_color[0]))
    n=int(len(B_color))
    L=list_coord()
    while L!=[]:
        for k in L:
            B=voisins(k,n,p)
            compo=[]
            compo.append(k)
            L.remove(k)
            while B!=[]:
                for v in B:
                    if not is_not_in(v,L):
                        if B_color[k[0]][k[1]]==B_color[v[0]][v[1]]:
                            L.remove(v)
                            compo.append(v)
                            for j in voisins(v,n,p):
                                B.append(j)
                    B.remove(v)
            I.append(compo)
    return(I)

def compr_list_coord():
    l,L=i_.size
    C=[]
    for k in range(0,L):
        for j in range(0,l):
            C.append((k,j))
    return(C)
```

Annexes :

```python
def compr_comp_connexe_final(p):
    Final=[]
    A=compr_img_decomp(p)
    list_color_R=compr_list_color(p)[0]
    list_color_G=compr_list_color(p)[1]
    list_color_B=compr_list_color(p)[2]
    R_color=compr_img_decomp(p)[0]
    G_color=compr_img_decomp(p)[1]
    B_color=compr_img_decomp(p)[2]
    R=compr_compo_connexeR(p,R_color)
    G=compr_compo_connexeG(p,G_color)
    B=compr_compo_connexeB(p,B_color)
    R_c=contour(color_reorga(list_color_R,R,R_color))
    G_c=contour(color_reorga(list_color_G,G,G_color))
    B_c=contour(color_reorga(list_color_B,B,B_color))
    Final.append(R_c)
    Final.append(G_c)
    Final.append(B_c)
    return(np.array(Final))

def compr_message(p): #TROP LOURD
    mess=[]
    list_color=compr_list_color(p)
    A=compr_comp_connexe_final(p)
    c,l=i.size #Dimensions de l'image : 2x12 bits
    C=str((bin(c)[2:]).zfill(12))
    L=str((bin(l)[2:]).zfill(12))
    for k in range(0,12):
        mess.append(C[k])
    for r in range(0,12):
        mess.append(L[r])
    for idRGB in range(0,3):
        nbr_r=len(list_color[idRGB]) #Nbr clr R
        N_R=str((bin(nbr_r)[2:]).zfill(8))
        for u in range(0,8):
            mess.append(N_R[u])
        for idclr in range(0,len(list_color[idRGB])):
            col=str((bin(list_color[idRGB][idclr])[2:]).zfill(8)) #Couleur
            for ar in range(0,8):
                mess.append(col[ar])
            nbr_ccr=len(A[idRGB][idclr]) #Nbr CC pour une couleur associée
            NBR_CCR=str((bin(nbr_ccr)[2:]).zfill(12))
            for nbccr in range(0,12):
                mess.append(NBR_CCR[nbccr])
            for cc_r in range(0,nbr_ccr):#CC de la couleur
                for compo_cc in A[idRGB][idclr][cc_r]:
                    a=((bin(compo_cc[0])[2:]).zfill(12))
                    b=((bin(compo_cc[1])[2:]).zfill(12))
                    for e1 in range(0,12):
                        mess.append(a[e1])
                    for e2 in range(0,12):
                        mess.append(b[e2])
    return(len(mess))
```

```python
## Division d'images

def img_div():
    c,l=i.size
    if (c<=256 and l<=256):
        decomp=[]
        img_=[]
        img_.append(img_decomp()[0])
        img_.append(img_decomp()[1])
        img_.append(img_decomp()[2])
        decomp.append(img_)
        return(decomp)
    elif c>256 and l<=256:
        k1=c//256
        decomp1=[]
        a1=-256
        b1=0
        for j in range(0,k1):
            a1+=256
            b1+=256
            img1=[]
            R1=[]
            G1=[]
            V1=[]
            for i1 in range(0,l):
                B1R=[]
                B1G=[]
                B1B=[]
                for j1 in range(a1,b1):
                    B1R.append(i.getpixel((j1,i1))[0])
                    B1G.append(i.getpixel((j1,i1))[1])
                    B1B.append(i.getpixel((j1,i1))[2])
                R1.append(B1R)
                G1.append(B1G)
                B1.append(B1B)
            img1.append(R1)
            img1.append(G1)
            img1.append(B1)
            decomp1.append(img1)
        decomp1_f=[] #Dernière partie de l'image
        R1f=[]
        G1f=[]
        B1f=[]
        for k in range (0,l):
            lignR1f=[]
            lignG1f=[]
            lignB1f=[]
            for f in range (b-1,c):
                lignR1f.append(i.getpixel((f,k))[0])
                lignG1f.append(i.getpixel((f,k))[1])
                lignB1f.append(i.getpixel((f,k))[2])
            R1f.append(lignR1f)
            G1f.append(lignG1f)
            B1f.append(lignB1f)
        decomp1_f.append(R1f)
        decomp1_f.append(G1f)
        decomp1_f.append(B1f)
        decomp1.append(decomp1_f)
        return(decomp1)
    elif c<=256 and l>256:
        k2=l//256
        decomp2=[]
        a2=-256
        b2=0
```

Transmission de données par modulation lumineuse

Annexes :

```
        for g in range(0,k2):
            a2+=256
            b2+=256
            img2=[]
            R2=[]
            G2=[]
            V2=[]
            for i2 in range(a2,b2):
                B2R=[]
                B2G=[]
                B2B=[]
                for j2 in range(0,c):
                    B2R.append(i.getpixel((j2,i2))[0])
                    B2G.append(i.getpixel((j2,i2))[1])
                    B2B.append(i.getpixel((j2,i2))[2])
                R2.append(B2R)
                G2.append(B2G)
                B2.append(B2B)
            img2.append(R2)
            img2.append(G2)
            img2.append(B2)
            decomp2.append(img2)
        decomp2_f=[] #Dernière partie de l'image
        R2f=[]
        G2f=[]
        B2f=[]
        for k in range (b-1,l):
            lignR2f=[]
            lignG2f=[]
            lignB2f=[]
            for f in range (0,c):
                lignR2f.append(i.getpixel((f,k))[0])
                lignG2f.append(i.getpixel((f,k))[1])
                lignB2f.append(i.getpixel((f,k))[2])
            R2f.append(lignR2f)
            G2f.append(lignG2f)
            B2f.append(lignB2f)
        decomp2_f.append(R2f)
        decomp2_f.append(G2f)
        decomp2_f.append(B2f)
        decomp2.append(decomp2_f)
        return(decomp2)
    else:
        decomp3=[]
        k3=l//256
        k4=c//256
        a3=-256
        b3=0
        for li in range(0,k3):
            a3+=256
            b3+=256
            a4=-256
            b4=0
            for co in range(0,k4):
                img=[]
                a4+=256
                b4+=256
                R=[]
                G=[]
                B=[]
                for lign in range(a3,b3):
                    Rl=[]
                    Gl=[]
                    Bl=[]
                    for coll in range(a4,b4):
                        Rl.append(i.getpixel((coll,lign))[0])
                        Gl.append(i.getpixel((coll,lign))[1])
                        Bl.append(i.getpixel((coll,lign))[2])
                    R.append(Rl)
                    G.append(Gl)
                    B.append(Bl)
                img.append(R)
                img.append(G)
                img.append(B)
                decomp3.append(img)
            imgT=[] #Dernière image restante par 256 lignes
            Rf=[]
            Gf=[]
            Bf=[]
            for lignf in range(a3,b3):
                Rlf=[]
                Glf=[]
                Blf=[]
                for collf in range(b4,c):
                    Rlf.append(i.getpixel((collf,lignf))[0])
                    Glf.append(i.getpixel((collf,lignf))[1])
                    Blf.append(i.getpixel((collf,lignf))[2])
                Rf.append(Rlf)
                Gf.append(Glf)
                Bf.append(Blf)
            imgf.append(Rf)
            imgf.append(Gf)
            imgf.append(Bf)
            decomp3.append(imgf)
        a5=-256 #Dernière ligne d'images restantes (sauf dernière image en bas à droite)
        b5=0
        for co_f in range(0,k4):
            imgff=[]
            a5+=256
            b5+=256
            Rff=[]
            Gff=[]
            Bff=[]
            for lignff in range(b3,l):
                Rlff=[]
                Glff=[]
                Blff=[]
                for collff in range(a5,b5):
                    Rlff.append(i.getpixel((collff,lignff))[0])
                    Glff.append(i.getpixel((collff,lignff))[1])
                    Blff.append(i.getpixel((collff,lignff))[2])
                Rff.append(Rlff)
                Gff.append(Glff)
                Bff.append(Blff)
            imgff.append(Rff)
            imgff.append(Gff)
            imgff.append(Bff)
            decomp3.append(imgff)
        imgfff=[] #Dernière image en bas à droite
        Rfff=[]
        Gfff=[]
        Bfff=[]
        for lignfff in range(b3,l):
            Rlfff=[]
            Glfff=[]
            Blfff=[]
            for collfff in range(b5,c):
                Rlfff.append(i.getpixel((collfff,lignfff))[0])
                Glfff.append(i.getpixel((collfff,lignfff))[1])
                Blfff.append(i.getpixel((collfff,lignfff))[2])
            Rfff.append(Rlfff)
            Gfff.append(Glfff)
            Bfff.append(Blfff)
        imgfff.append(Rfff)
        imgfff.append(Gfff)
        imgfff.append(Bfff)
        decomp3.append(imgfff)
        return(decomp3)

def dim1():
    A=img_div()
    for k in range(0,len(A)):
        print(len(A[k][0][0]),len(A[k][0]),len(A))
    return('fin')

def dim(nom):
    i1=Image.open(nom)
    p,n=i1.size
    return(p,n)


def div_screen():
    A=img_div()
    for k in range(0,len(A)):
        img=Image.new("RGB",(len(A[k][0][0]),len(A[k][0])))
        pixel=img.load()
        for l in range(0,len(A[k][0])):
            for j in range(0,len(A[k][0][0])):
                pixel[j,l]=(A[k][0][l][j],A[k][1][l][j],A[k][2][l][j])
        img.save('{0}.png'.format(k),'PNG')

def div_list_coord(nom):
    pos=nom.find(".png")
    id_img=int(nom[:pos])
    decomp=img_div()[id_img]
    l=int(len(decomp[0][0]))
    L=int(len(decomp[0]))
    C=[]
    for k in range(0,L):
        for j in range(0,l):
            C.append((k,j))
    return(C)

# Approche n°1 : Division puis compression de chaque image 256*256

def div_list_color(nom):
    pos=nom.find(".png")
    id_img=int(nom[:pos])
    decomp=img_div()[id_img]
    p=int(len(decomp[0][0]))
    n=int(len(decomp[0]))
    R_color=[]
    G_color=[]
    B_color=[]
    R,G,B=img_decomp()
    for l in range(0,n):
        for c in range(0,p):
            if is_not_in(R[l][c],R_color):
                R_color.append(R[l][c])
            if is_not_in(G[l][c],G_color):
                G_color.append(G[l][c])
            if is_not_in(B[l][c],B_color):
                B_color.append(B[l][c])
    return(sorted(R_color),sorted(G_color),sorted(B_color))
```

Annexes :

```
def div_compr_list_color(p,nom):
    L_fin=[]
    I=compr_color(p)
    L=div_list_color(nom)
    for k in L:
        A=[]
        for i in k:
            comt=0
            for l in range(0,len(I)-1):
                if i>I[l] and i<I[l+1]:
                    A.append(I[l])
            for n in range(0,len(I)):
                if i==I[n]:
                    A.append(I[n])
            A=list(set(A))
            A=sorted(A)
        L_fin.append(A)
    return(L_fin)

def div_img_decomp(nom):
    R=[]
    G=[]
    B=[]
    (l,L)=dim(nom)
    for k in range (0,L):
        lignR=[]
        lignG=[]
        lignB=[]
        for j in range (0,l):
            lignR.append(i.getpixel((j,k))[0])
            lignG.append(i.getpixel((j,k))[1])
            lignB.append(i.getpixel((j,k))[2])
        R.append(lignR)
        G.append(lignG)
        B.append(lignB)
    return(R,G,B)

def div_compr_img_decomp(p,nom):
    A=div_img_decomp(nom)
    L=div_compr_list_color(p,nom)
    L_final=[]
    for k in range(0,len(A)):
        A_0=A[k]
        LN=[]
        for j in range(0,len(A_0)):
            B=[]
            for l in A_0[j]:
                for z in range(0,len(L[k])-1):
                    if l>L[k][z] and l<L[k][z+1]:
                        B.append(L[k][z])
                for y in range(0,len(L[k])):
                    if l==L[k][y]:
                        B.append(L[k][y])
                if l>L[k][len(L[k])-1]:
                    B.append(L[k][len(L[k])-1])
            LN.append(B)
        L_final.append(LN)
    return(L_final)
```

```
def div_compr_compo_connexeR(pr,nom):
    a=time.clock
    I=[]
    R_color=div_compr_img_decomp(pr,nom)[0]
    (p,n)=dim(nom)
    L=div_list_coord(nom)
    while L!=[]:
        for k in L:
            B=voisins(k,n,p)
            compo=[]
            compo.append(k)
            L.remove(k)
            while B!=[]:
                for v in B:
                    if not is_not_in(v,L):
                        if R_color[k[0]][k[1]]==R_color[v[0]][v[1]]:
                            L.remove(v)
                            compo.append(v)
                            for j in voisins(v,n,p):
                                B.append(j)
                    B.remove(v)
            I.append(compo)
    b=time.clock
    return(I)

def div_compr_compo_connexeG(pr,nom):
    I=[]
    G_color=div_compr_img_decomp(pr,nom)[1]
    p=int(len(G_color[0]))
    n=int(len(G_color))
    L=div_list_coord(nom)
    while L!=[]:
        for k in L:
            B=voisins(k,n,p)
            compo=[]
            compo.append(k)
            L.remove(k)
            while B!=[]:
                for v in B:
                    if not is_not_in(v,L):
                        if G_color[k[0]][k[1]]==G_color[v[0]][v[1]]:
                            L.remove(v)
                            compo.append(v)
                            for j in voisins(v,n,p):
                                B.append(j)
                    B.remove(v)
            I.append(compo)
    return(I)
```

Annexes :

```
def div_compr_compo_connexeB(pr,nom):
    I=[]
    B_color=div_compr_img_decomp(pr,nom)[2]
    p=int(len(B_color[0]))
    n=int(len(B_color))
    L=div_list_coord(nom)
    while L!=[]:
        for k in L:
            B=voisins(k,n,p)
            compo=[]
            compo.append(k)
            L.remove(k)
            while B!=[]:
                for v in B:
                    if not is_not_in(v,L):
                        if B_color[k[0]][k[1]]==B_color[v[0]][v[1]]:
                            L.remove(v)
                            compo.append(v)
                            for j in voisins(v,n,p):
                                B.append(j)
                    B.remove(v)
            I.append(compo)
    return(I)

def div_color_reorga(L_color,compo_conn,decomp_img,n):#avec L_color:div_list_color()[0,1,2]
    a=time.clock
    LR=[]
    for k in (L_color):
        A=[]
        B=decomp_img[n]
        for g in range(len(compo_conn)):
            if B[compo_conn[g][0][0]][compo_conn[g][0][1]]==k:
                A.append(sorted(compo_conn[g]))
        LR.append(A)
    b=time.clock
    return(LR)

#div_color_reorga(compr_list_color(95)[0], div_compr_compo_connexeR(95,"0.png"), div_compr_img_decomp(95,"0.png"))

def div_contour(col_reorga,nom): #Argument : div_color_reorga(...)
    a=time.clock()
    pos=nom.find(".png")
    id_img=int(nom[:pos])
    decomp=img_div()[id_img]
    p=int(len(decomp[0][0]))
    n=int(len(decomp[0]))
    F=[]
    comp=0
    for k in range(0,len(col_reorga)):
        E=[]
        for compo in col_reorga[k]:
            A=[]
            for j in compo:
                V=voisins(j,n,p)
                for l in V:
                    if is_not_in(l,compo):
                        V.remove(l)
                if len(V)!=4:
                    comp+=1
                    A.append(j)
            A=sorted(A)
            E.append(A)
        F.append(E)
    b=time.clock()
    return(F)#,([comp/(p*n))*100),b-a]
```

Annexes :

```
def color_reorga_final(pr,nom): #col_reorga = color_reorga_final(pr,nom)[A] avec A=0->R A=1->V A=2->B
    return(div_color_reorga(compr_list_color(pr)[0], div_compr_compo_connexeR(pr,nom),
div_compr_img_decomp(pr,nom),0),div_color_reorga(compr_list_color(pr)[1], div_compr_compo_connexeG(pr,nom),
div_compr_img_decomp(pr,nom),1),div_color_reorga(compr_list_color(pr)[2], div_compr_compo_connexeB(pr,nom),
div_compr_img_decomp(pr,nom),2))

#div_color_reorga(compr_list_color(95)[0], div_compr_compo_connexeR(95,"0.png"), div_compr_img_decomp(95,"0.png"))


def signal(pr):#col_reorga = color_reorga_final(pr,nom) et color_reorga[A] avec    A=0->R     A=1->V     A=2->B
    TIME1=time.clock()
    SIGNAL=[]
    p,n=i.size
    coord1=(bin(p)[2:]).zfill(8)
    coord2=(bin(n)[2:]).zfill(8)
    for z in range(8):
        SIGNAL.append(coord1[z])
    for z1 in range(8):
        SIGNAL.append(coord2[z1])
    r=0        #NOMBRE D'IMAGES
    r1=0
    q=p//256
    w=n//256
    if p%256!=0:
        r=1
    if p%256!=0:
        r1=1
    nbr_img=int((w*q)+(r*w)+(q*r1)+(r*r1))
    BIN_nbr_img=(bin(nbr_img)[2:]).zfill(8)
    for t in range (8):
        SIGNAL.append(BIN_nbr_img[t])
    for img in range(nbr_img):   #DIMENSIONS DE L'IMAGE
        if img==(nbr_img%2) :
            print('50%')
        nom=str(img)+".png"
        if dim(nom)!=(256,256):
            SIGNAL.append('1')
            x=(bin((dim(nom))[0])[2:]).zfill(8)
            y=(bin((dim(nom))[1])[2:]).zfill(8)
            for dimx in range (8):
                SIGNAL.append(x[dimx])
            for dimy in range (8):
                SIGNAL.append(y[dimy])
        else:
            SIGNAL.append('0')
        for RGB in range (3):       #RGB
            list_col=div_compr_list_color(pr,nom)[RGB]
            nbr_col=len(list_col)      #NOMBRE COULEURS
            BIN_nbr_col=(bin(nbr_col)[2:]).zfill(8)
            for u in range (8):
                SIGNAL.append(BIN_nbr_col[u])
            CONTOUR=div_contour(color_reorga_final(pr,nom)[RGB],nom)
            for id_col in range(len(list_col)):            #COULEUR
                col=list_col[id_col]
                BIN_col=(bin(col)[2:]).zfill(8)
                for o in range (8):
                    SIGNAL.append(BIN_col[o])
                nbr_comp=len(CONTOUR[id_col])          #NOMBRE DE COMPO CONNEXES POUR LA COULEUR
                BIN_nbr_comp=(bin(nbr_comp)[2:]).zfill(8)
                for m in range (8):
                    SIGNAL.append(BIN_nbr_comp[m])
                for id_comp in range(nbr_comp):  #COMPO CONNEXE
                    size_comp=len(CONTOUR[id_col][id_comp]) #TAILLE DE LA COMPO CONNEXE
                    BIN_size_comp=(bin(size_comp)[2:]).zfill(8)
```

Transmission de données par modulation lumineuse

Annexes :

```
                for l in range (8):
                    SIGNAL.append(BIN_size_comp[l])
                for id_int_compo in range (size_comp):
                    x1=CONTOUR[id_col][id_comp][id_int_compo][0]
                    x2=CONTOUR[id_col][id_comp][id_int_compo][1]
                    element1=(bin(x1)[2:]).zfill(8)
                    element2=(bin(x2)[2:]).zfill(8)
                    for b1 in range (8):
                        SIGNAL.append(element1[b1])
                    for b2 in range (8):
                        SIGNAL.append(element2[b2])
    TIME2=time.clock()
    return(SIGNAL)

def signification(A,x,y) :
    b=''
    for k in range(x,y):
        b+=A[k]
    return(int(b,2))


def reconstruction_signal(SIGNAL):
    RESTIT=[]
    a=0
    coord1=''              #DIMENSIONS IMAGE TOTALE
    coord2=''
    for z in range(a,a+8):
        coord1+=SIGNAL[z]
    a+=8
    p_o=int(coord1,2)
    for z1 in range(a,a+8):
        coord2+=SIGNAL[z1]
    a+=8
    n_o=int(coord2,2)
    RESTIT.append((p_o,n_o))
    b=''                   #NOMBRE IMAGES
    for k in range(a,a+8):
        b+=SIGNAL[k]
    nbr_img=int(b,2)
    a+=8
    RESTIT.append(nbr_img)
    for img in range(nbr_img):
        IMG=[]
        if SIGNAL[a]=='1':    #DIMENSION IMAGE
            a+=1
            b1=''
            b2=''
            for t1 in range(a,a+8):
                b1+=SIGNAL[t1]
            p=int(b1,2)
            a+=8
            for t2 in range(a,a+8):
                b2+=SIGNAL[t2]
            n=int(b2,2)
            a+=8
        else :
            p=256
            n=256
            a+=1
        IMG.append([p,n])
        for RGB in range(3):       #RGB
            RGB_list=[]
            c=''                   #NOMBRE COULEURS
            for r in range (a,a+8):
                c+=SIGNAL[r]
            a+=8
            nbr_col=int(c,2)
            for id_col in range(nbr_col):
                COLOR=[]
                c1=''                  #COULEUR
                for r1 in range(a,a+8):
                    c1+=SIGNAL[r1]
                a+=8
                col=int(c1,2)
                COLOR.append(col)
                c2=''                  #NOMBRE COMPO CONNEXE POUR LA COULEUR
                for r2 in range(a,a+8):
                    c2+=SIGNAL[r2]
                a+=8
                nbr_comp=int(c2,2)
                for id_comp in range(nbr_comp):
                    COMP=[]
                    c3=''
                    for r3 in range(a,a+8):
                        c3+=SIGNAL[r3]
                    a+=8
                    size_comp=int(c3,2)
                    for comp_int in range(size_comp):
                        x=signification(SIGNAL,a,a+8)
                        a+=8
                        y=signification(SIGNAL,a,a+8)
                        a+=8
                        COMP.append((x,y))
                    COLOR.append(COMP)
                RGB_list.append(COLOR)
            IMG.append(RGB_list)
        RESTIT.append(IMG)
    return(RESTIT)

def img_vide_256(x,y): #x nombre de colonnes, y nombre de lignes
    IMG=[]
    for lign in range (y):
        L=[]
        for col in range(x):
            L.append(256)
        IMG.append(L)
    return(IMG)

def img_vide(x,y): #x nombre de colonnes, y nombre de lignes
    IMG=[]
    for lign in range (y):
        L=[]
        for col in range(x):
            L.append([])
        IMG.append(L)
    return(IMG)

def reconstruction_images(SIGNAL): #SIGNAL=raconstruction_signal(signal(pr))
    p_dim=SIGNAL[0][0]
    n_dim=SIGNAL[0][1]
    nbr_img=SIGNAL[1]
    for id_img in range(0,nbr_img):
        id_img+=2 #Car a partir de la position 3 dans la liste (suivi de dim_tot et de nbr_img)
        p=SIGNAL[id_img][0][0]
        n=SIGNAL[id_img][0][1]
        img_=Image.new("RGB",(p,n))
        pixel=img_.load()
        IMG=img_vide(p,n)
        for id_rgb in range(3):
            id_rgb+=1 #Car à partir de la position 1 dans la liste (suivi de dim_img)
```

Annexes :

```
    RGB=img_vide_256(p,n)
    nbr_col=len(SIGNAL[id_img][id_rgb])
    for id_col in range (nbr_col):
        col=SIGNAL[id_img][id_rgb][id_col][0]
        nbr_comp=(len(SIGNAL[id_img][id_rgb][id_col]))-1
        for id_comp in range (nbr_comp):
            id_comp+=1 #Car à partir de la position 1 dans la liste (suivi de col)
            for comp in (SIGNAL[id_img][id_rgb][id_col][id_comp]):
                x_n=comp[0]
                y_p=comp[1]
                RGB[x_n][y_p]=col
    for x1_n in range(n):
        for y1_p in range(p):
            if RGB[x1_n][y1_p]==256:
                RGB[x1_n][y1_p]=RGB[x1_n][(y1_p)-1]
            IMG[x1_n][y1_p].append(RGB[x1_n][y1_p])
for a in range (n):
    for c in range (p):
        r=IMG[a][c][0]
        g=IMG[a][c][1]
        b=IMG[a][c][2]
        pixel[c,a]=(r,g,b)
nom=str(id_img-2)+'_1.png'
img_.save(nom,'PNG')
```

Photo utilisées :

SpaceX: Twitter - SpaceX

Laser : https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.dhresource.com%2F0x0s%2Ff2-albu-g5-M01-76-05-rBVaI1lloR6APRNgAAMKTz_6ISs890.jpg%2Fmodule-de-capteur-de-t-te-laser-arduino-adapt.jpg&imgrefurl=https%3A%2F%2Ffr.dhgate.com%2Fproduct%2Flaser-head-sensor-module-arduino-suitable%2F402096000.html&docid=Du-5I7l7lxGYpM&tbnid=tvD57X5dSpV61M%3A&vet=10ahUKEwjNiYvUieHiAhXiD2MBHd_HD3sQMwjGASgEMAQ..i&w=960&h=960&bih=1281&biw=2560&q=laser%20arduino&ved=0ahUKEwjNiYvUieHiAhXiD2MBHd_HD3sQMwjGASgEMAQ&iact=mrc&uact=8

capteur : https://www.google.com/imgres?imgurl=https%3A%2F%2Fimages-na.ssl-images-amazon.com%2Fimages%2FI%2F41gorPxujTL._SX355_.jpg&imgrefurl=https%3A%2F%2Fwww.amazon.fr%2FWaveshare-Laser-Receiver-Sensor-Transmitter%2Fdp%2FB00NJNYQ9G&docid=qDE5HmMsIG3TbM&tbnid=ClmUf98-6VWpTM%3A&vet=10ahUKEwjNiYvUieHiAhXiD2MBHd_HD3sQMwjEASgCMAI..i&w=355&h=266&bih=1281&biw=2560&q=laser%20arduino&ved=0ahUKEwjNiYvUieHiAhXiD2MBHd_HD3sQMwjEASgCMAI&iact=mrc&uact=8

sensor : https://www.google.com/imgres?imgurl=https%3A%2F%2F4.imimg.com%2Fdata4%2FPF%2FXL%2FMY-3287828%2Farduino-photoresistor-detection-photosensitive-light-sensor-500x500.jpg&imgrefurl=https%3A%2F%2Fwww.indiamart.com%2Fproddetail%2Farduino-photo-resistor-detection-photosensitive-light-sensor-13120746130.html&docid=-c5RevlnW5oLSM&tbnid=UpqNibtAp4bC1M%3A&vet=10ahUKEwiC086IiuHiAhUh3uAKHTyCD2UQMwh9KAEwAQ..i&w=500&h=500&bih=1281&biw=2560&q=light%20sensor%20%20arduino&ved=0ahUKEwiC086IiuHiAhUh3uAKHTyCD2UQMwh9KAEwAQ&iact=mrc&uact=8

arduino : https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.gotronic.fr%2Fori-carte-arduino-uno-12420.jpg&imgrefurl=https%3A%2F%2Fwww.gotronic.fr%2Fart-carte-arduino-uno-12420.htm&docid=culT2N5av9oBuM&tbnid=6vvsjj_XKpcSNM%3A&vet=10ahUKEwi7tKu1iuHiAhUJ0uAKHXIaDocQMwioASgBMAE..i&w=600&h=600&bih=1281&biw=2560&q=arduino%20uno&ved=0ahUKEwi7tKu1iuHiAhUJ0uAKHXIaDocQMwioASgBMAE&iact=mrc&uact=8

elegoo: https://www.google.com/imgres?imgurl=https%3A%2F%2Fimages-na.ssl-images-amazon.com%2Fimages%2FI%2F71DQ3MmmswL._SY355_.jpg&imgrefurl=https%3A%2F%2Fwww.amazon.fr%2FElegoo-ATmega328P-ATMEGA16U2-Controller-Microcontr%25C3%25B4leur%2Fdp%2FB01N91PVIS&docid=Yf-FvJmB-oSGRM&tbnid=UTNhjfN4MLvyfM%3A&vet=10ahUKEwivkf-7iuHiAhVOAGMBHYnuBnEQMwhMKAEwAQ..i&w=355&h=355&bih=1281&biw=2560&q=elegoo%20uno&ved=0ahUKEwivkf-7iuHiAhVOAGMBHYnuBnEQMwhMKAEwAQ&iact=mrc&uact=8

Logo PC: https://www.google.com/imgres?imgurl=https%3A%2F%2Fthumbs.dreamstime.com%2Ft%2Ftischrechner-pc-linie-ikone-entwurfsvektorzeichen-lineares-artpiktogramm-lokalisiert-auf-wei%25C3%259F-88391920.jpg&imgrefurl=https%3A%2F%2Fde.dreamstime.com%2Fstock-abbildung-tischrechner-pc-linie-ikone-entwurfsvektorzeichen-lineares-artpiktogramm-lokalisiert-auf-wei%25C3%259F-image88391920&docid=wQoG6EJ04BpkjM&tbnid=-E1h5ka_sG5ueM%3A&vet=1&w=160&h=160&bih=424&biw=1620&ved=2ahUKEwjV_Ou2juHiAhWByIUKHZZgDFcQxiAoBnoECAEQGw&iact=c&ictx=1

Logo USB: https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwjM96TojuHiAhUK-YUKHbIQAlEQjRx6BAgBEAU&url=https%3A%2F%2Fjch.blog4ever.com%2Fusb-quels-type-de-connecteurs-et-quelles-versions&psig=AOvVaw0sHHbKYFjFukPr95K5pXvj&ust=1560331735893750

Sensor scheme: https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwiEgcrYm-HiAhUNQhoKHfTcBeAQjRx6BAgBEAU&url=https%3A%2F%2Fcreate.arduino.cc%2Fprojecthub%2Falbertoz%2Fvibration-sensor-module-c88067&psig=AOvVaw2-h08jXor97B9nuKnWC2tf&ust=1560335190698155