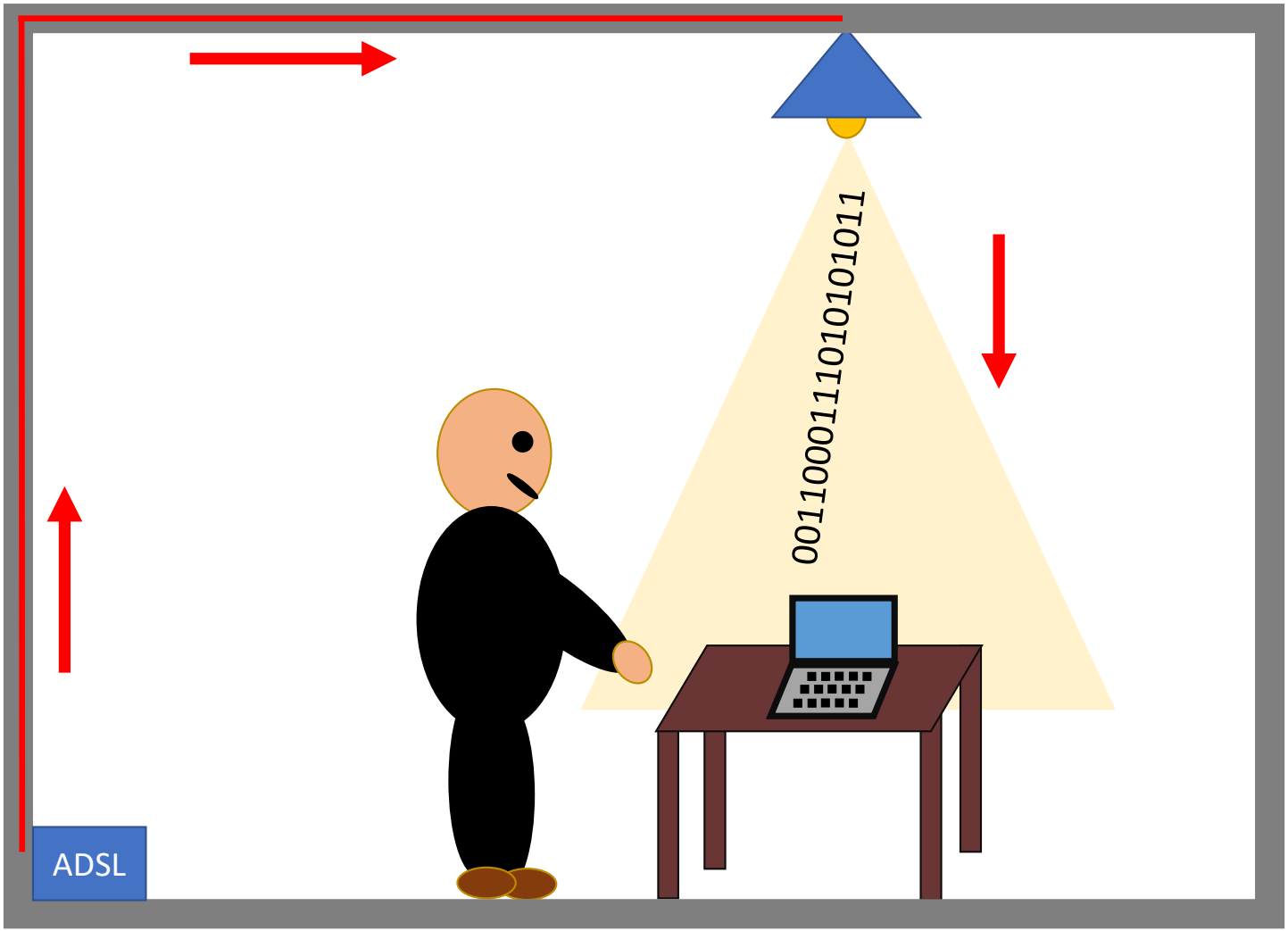
Transmission et perturbation d'un signal en lumière visible

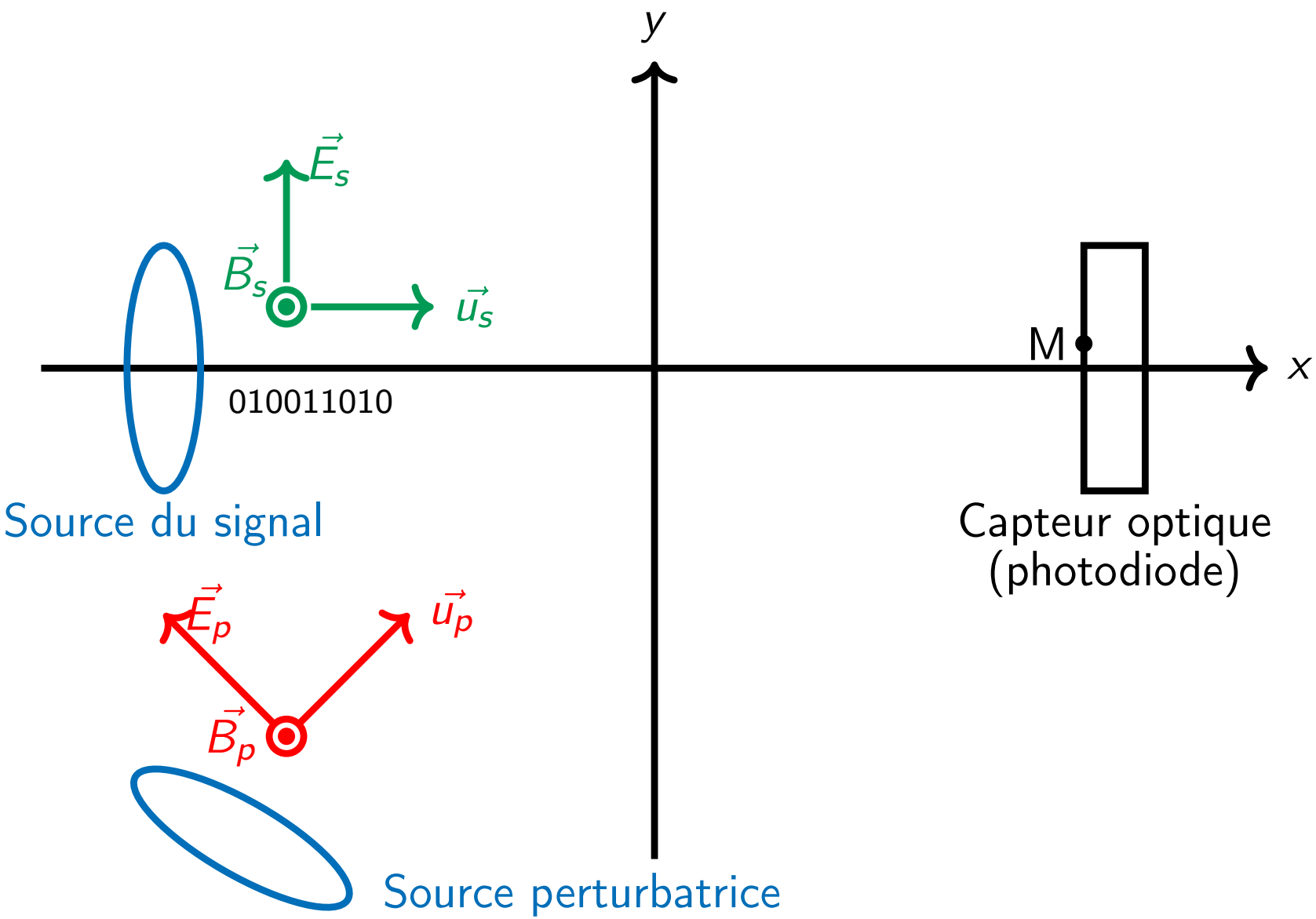
Alexandre HUMBERT n° 45794

Épreuve de TIPE

Session 2019

Principe de fonctionnement



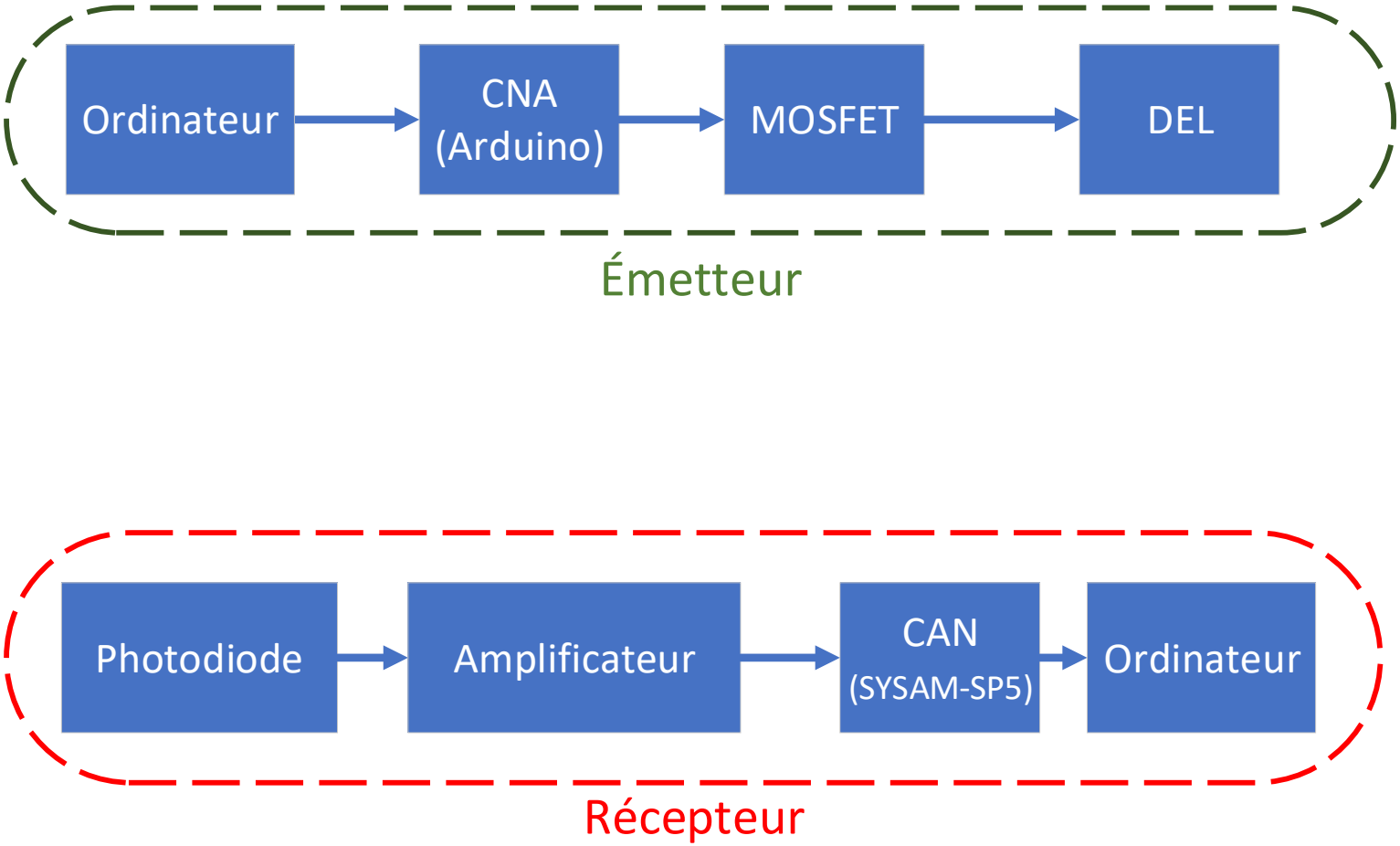

y
$\vec{E}_s$
$\vec{B}_s$
$\vec{u}_s$
010011010
M
x
Source du signal
Capteur optique
(photodiode)
$\vec{E}_p$
$\vec{u}_p$
$\vec{B}_p$
Source perturbatrice

- Lumière blanche $\lambda_0 \approx 550nm$
- $T_{capteur} = 10^{-7}s$
- $F_s \sim 1kHz$ et $F_p \sim 1kHz$

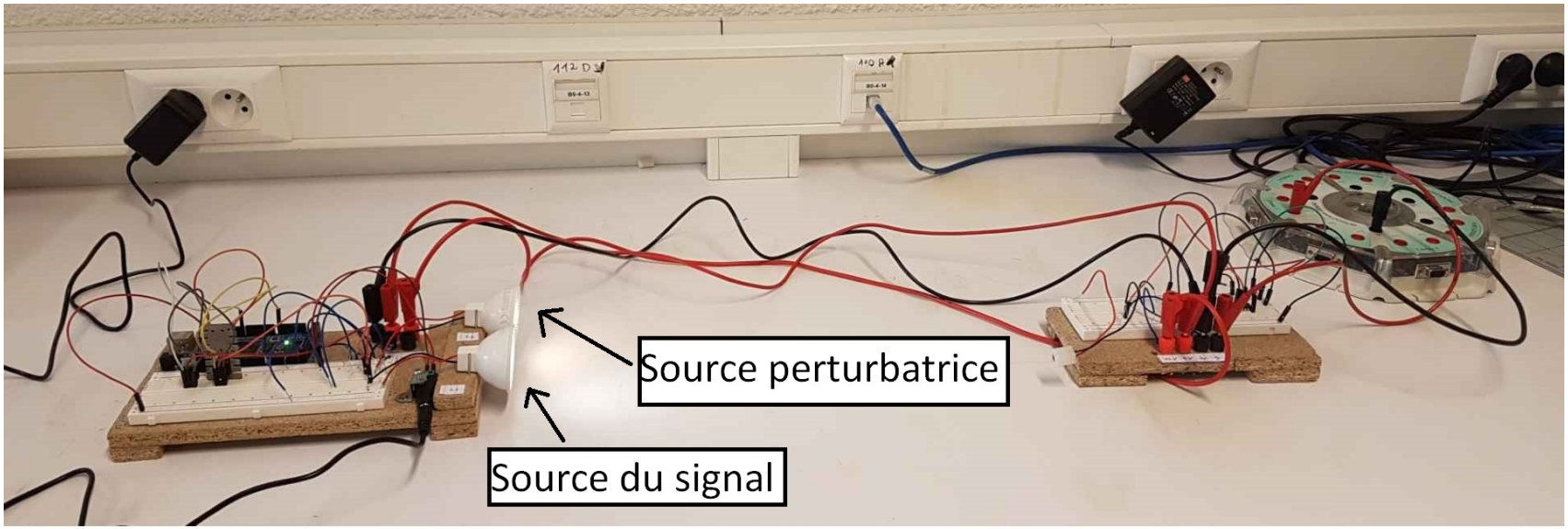
- $s_s(M,t) = \underbrace{s_{s0}(M,t)}_{001010} \cos(\omega_s t - \phi_s(M,t))$
- $s_p(M,t) = s_{p0}(M,t) \cos(\omega_p t - \phi_p(M,t))$
- $I_s(M,t) = \frac{1}{T_{capteur}} \int_t^{t+T_{capteur}} s_s^2(M,t)dt$
- $I_p(M,t) = \frac{1}{T_{capteur}} \int_t^{t+T_{capteur}} s_p^2(M,t)dt$

- Capteur assimilé à un unique point M
- $s(t) = s_s(t) + s_p(t)$
- $I(t) = \frac{1}{T_{capteur}} \int_t^{t+T_{capteur}} s^2(t)dt$
- $S_{s,mb}(t) = \frac{1}{T_{capteur}} \int_t^{t+T_{capteur}} s_{s0}^2(t)dt$
- $S_{p,mb}(t) = \frac{1}{T_{capteur}} \int_t^{t+T_{capteur}} s_{p0}^2(t)dt$
- Sources non cohérentes $\omega_s \neq \omega_p$ (pas d'interférences)
- $I(t) = I_s(t) + I_p(t) = \underbrace{\frac{S_{s,mb}(t)}{2}}_{001010} + \frac{S_{p,mb}(t)}{2}$

Montage


Ordinateur
CNA (Arduino)
MOSFET
DEL
Émetteur
Photodiode
Amplificateur
CAN (SYSAM-SP5)
Ordinateur
Récepteur

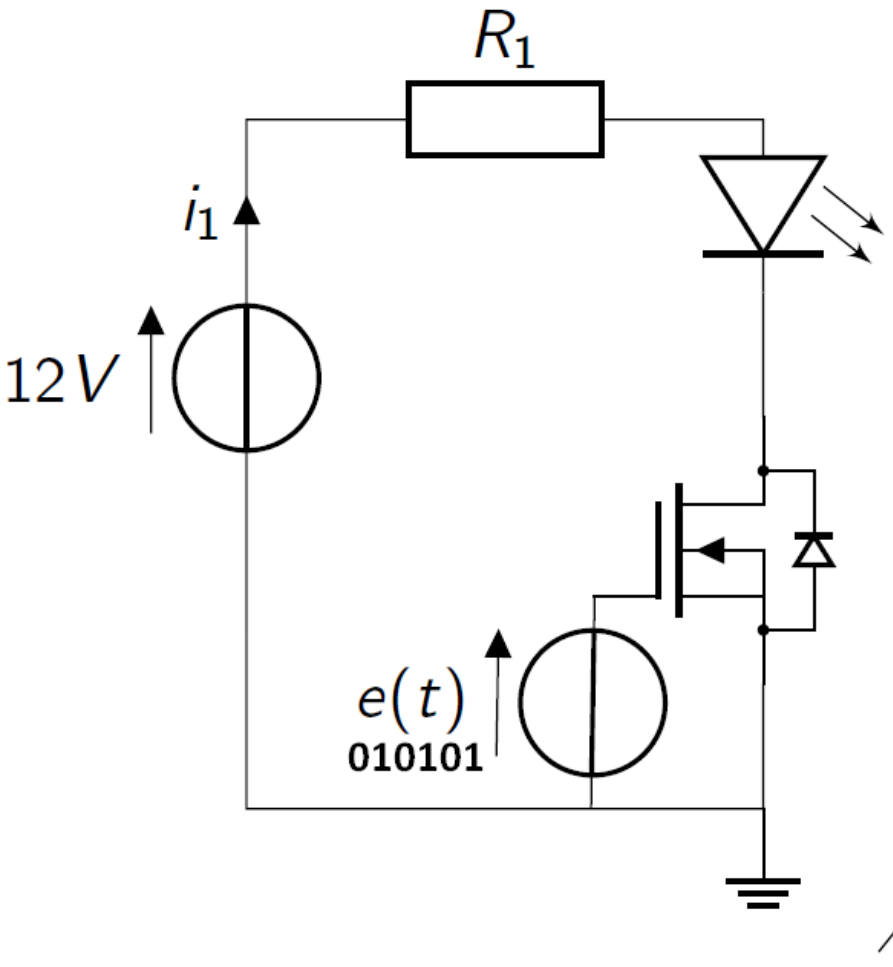
Montage expérimental



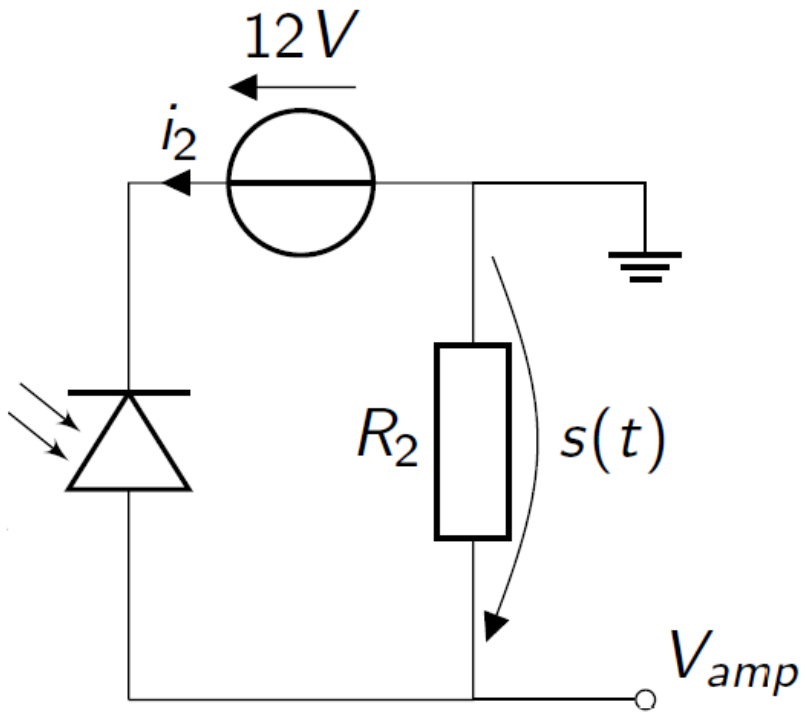
Arduino MOSFET Lampe

Photodiode Amplificateur CAN

Émetteur du signal

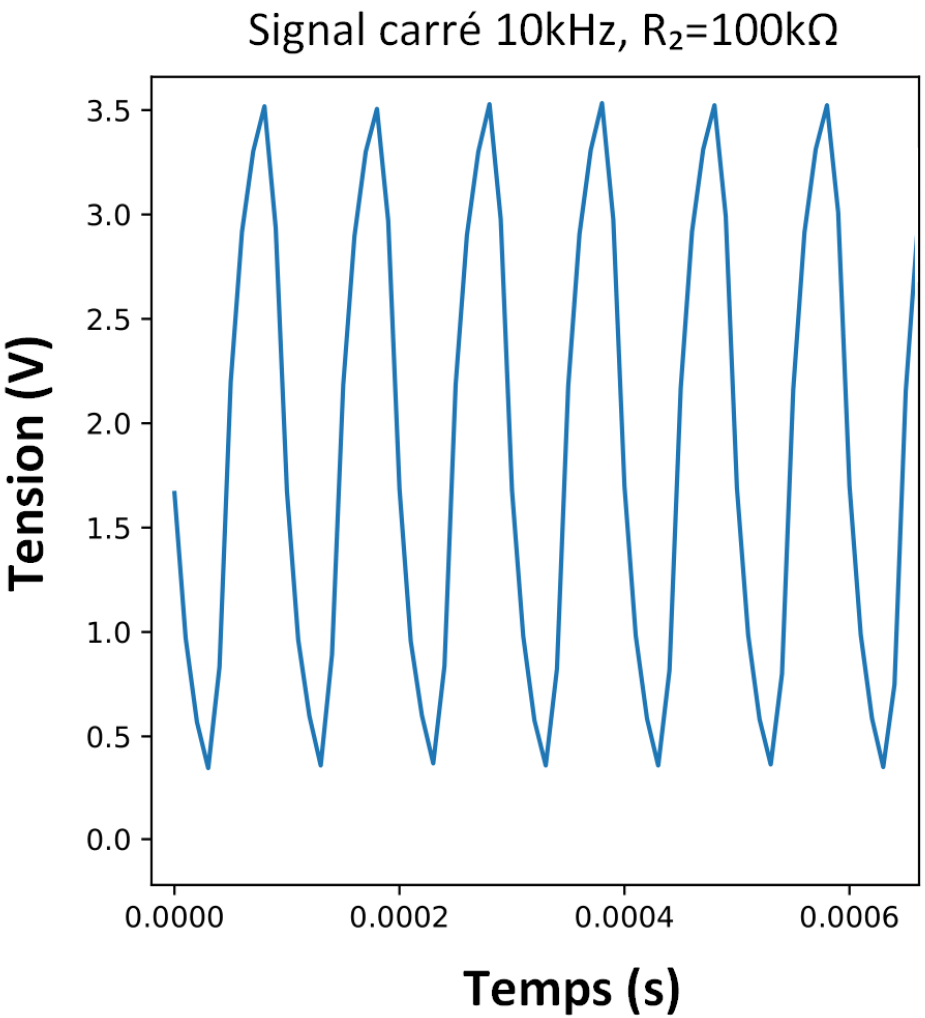


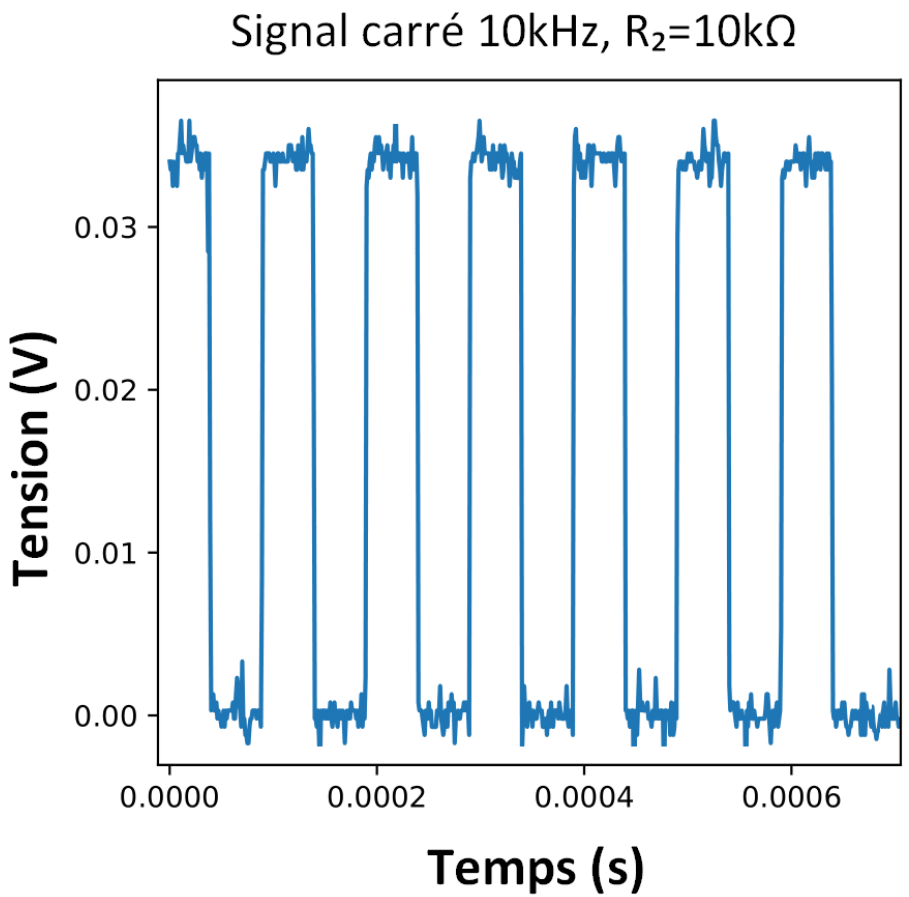
Récepteur



+ source perturbatrice

Choix des composants





Sources de perturbations analysées

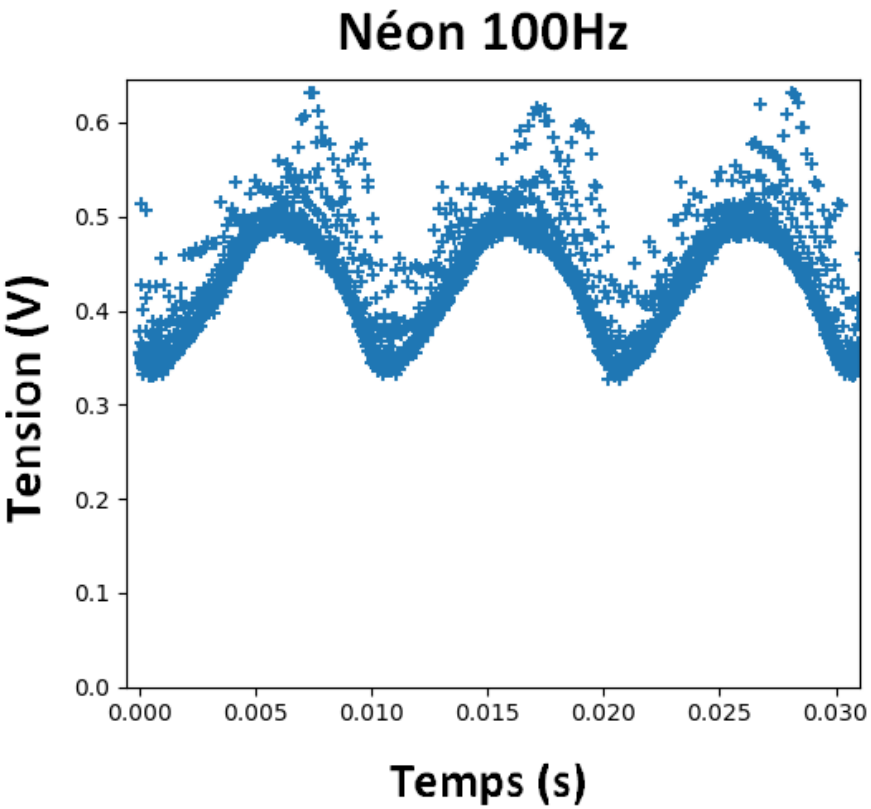
Sources continues

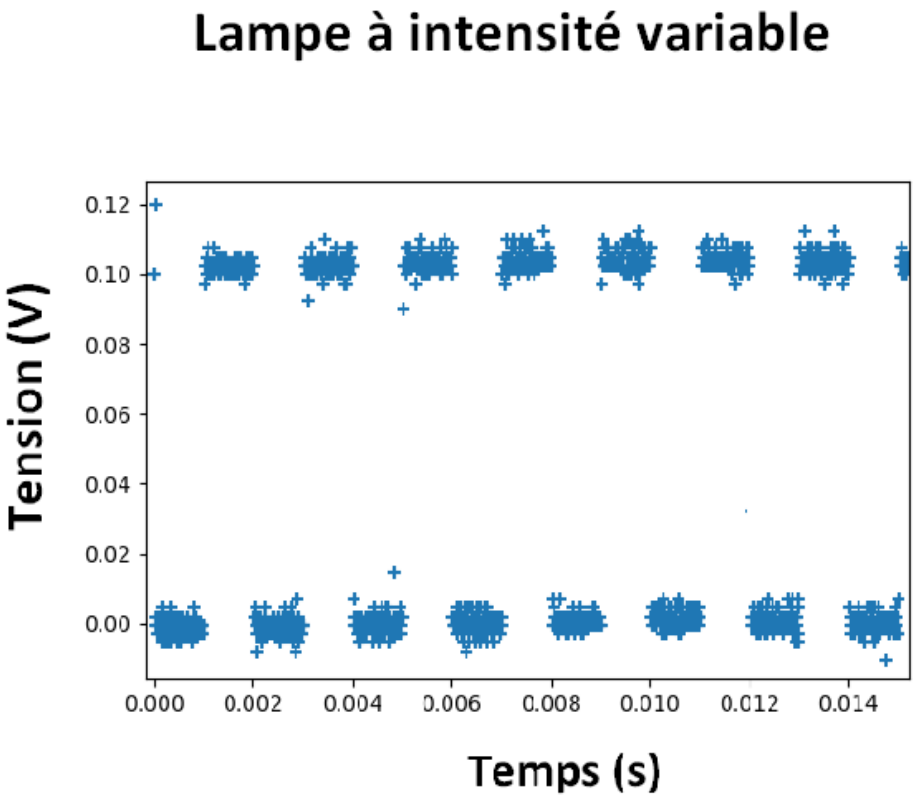
- Soleil
- Lampe LED bonne qualité

Sources alternatives

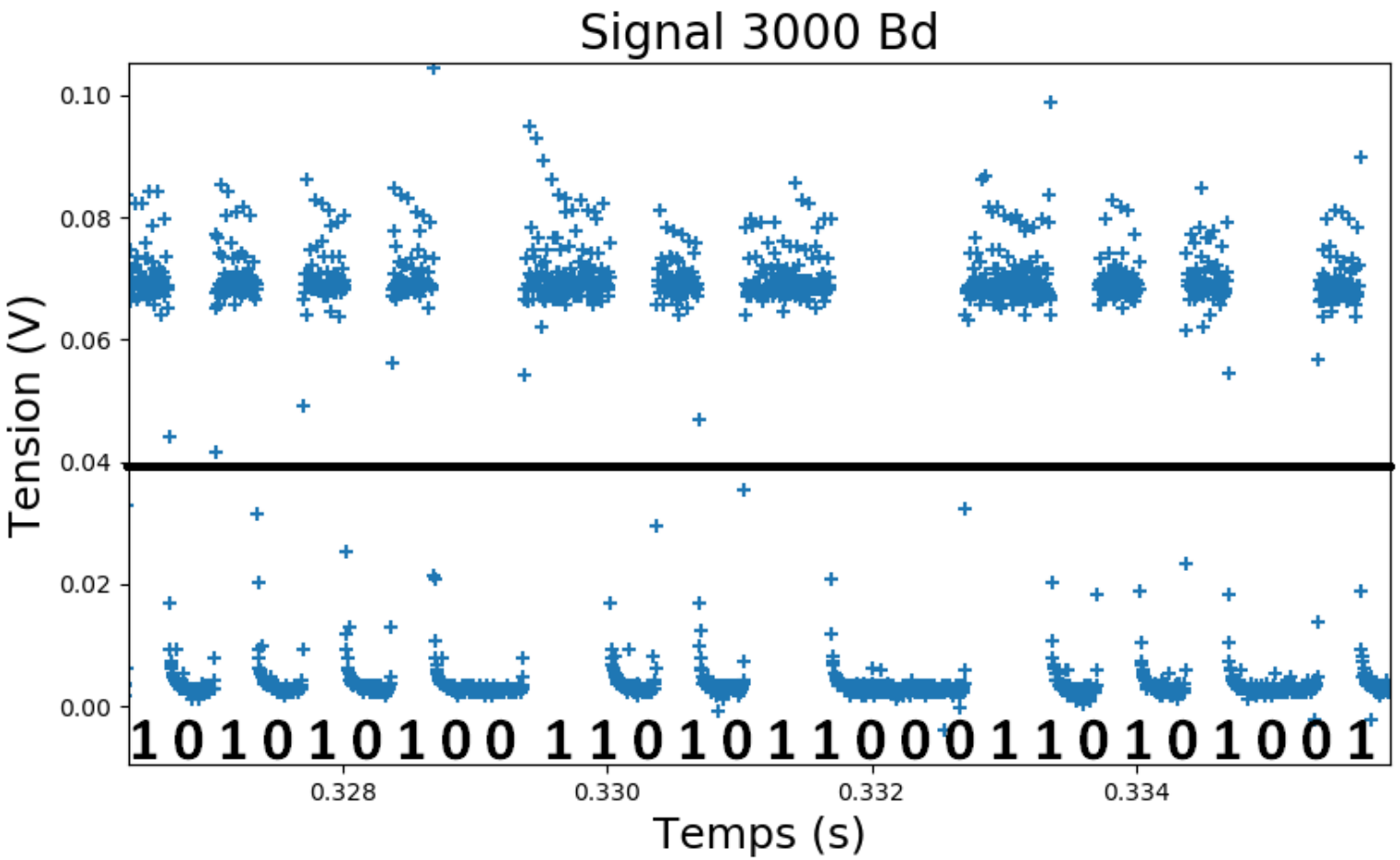
- Tubes néons (100 Hz)
- Lampe LED mauvaise qualité (50-100 Hz)
- Lampe LED avec variateur de luminosité (100Hz-10kHz)

Deux exemples de perturbations

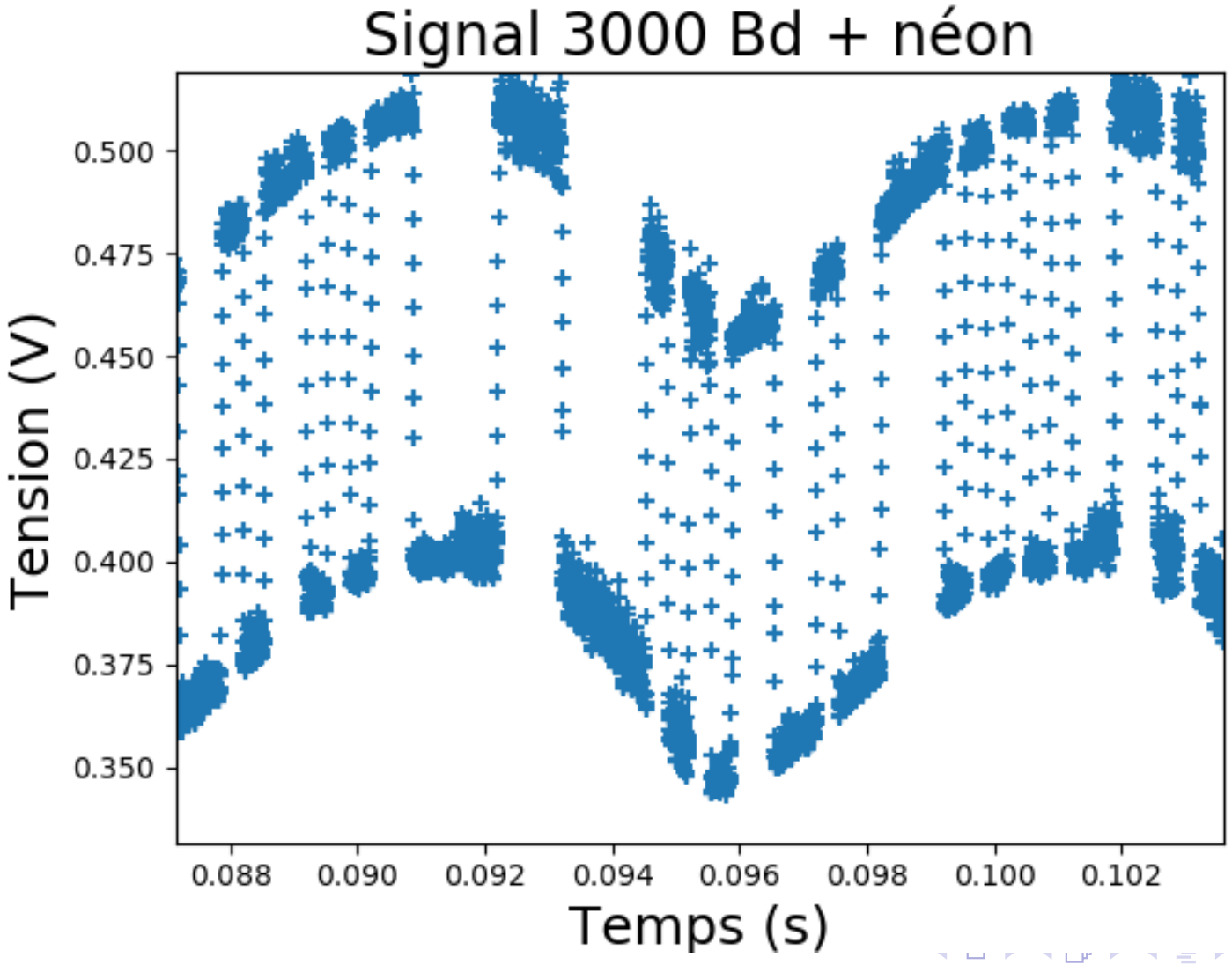




Signal transmis en lumière visible



Transmission en présence de néons



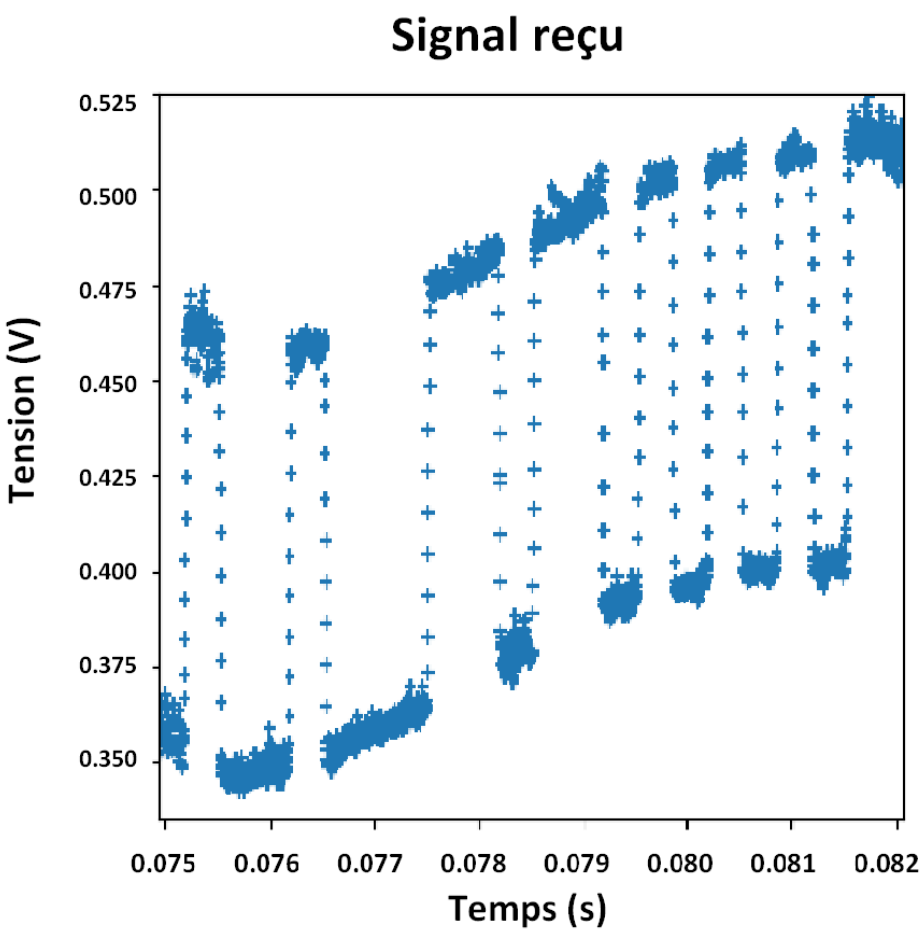
Traitement de la perturbation

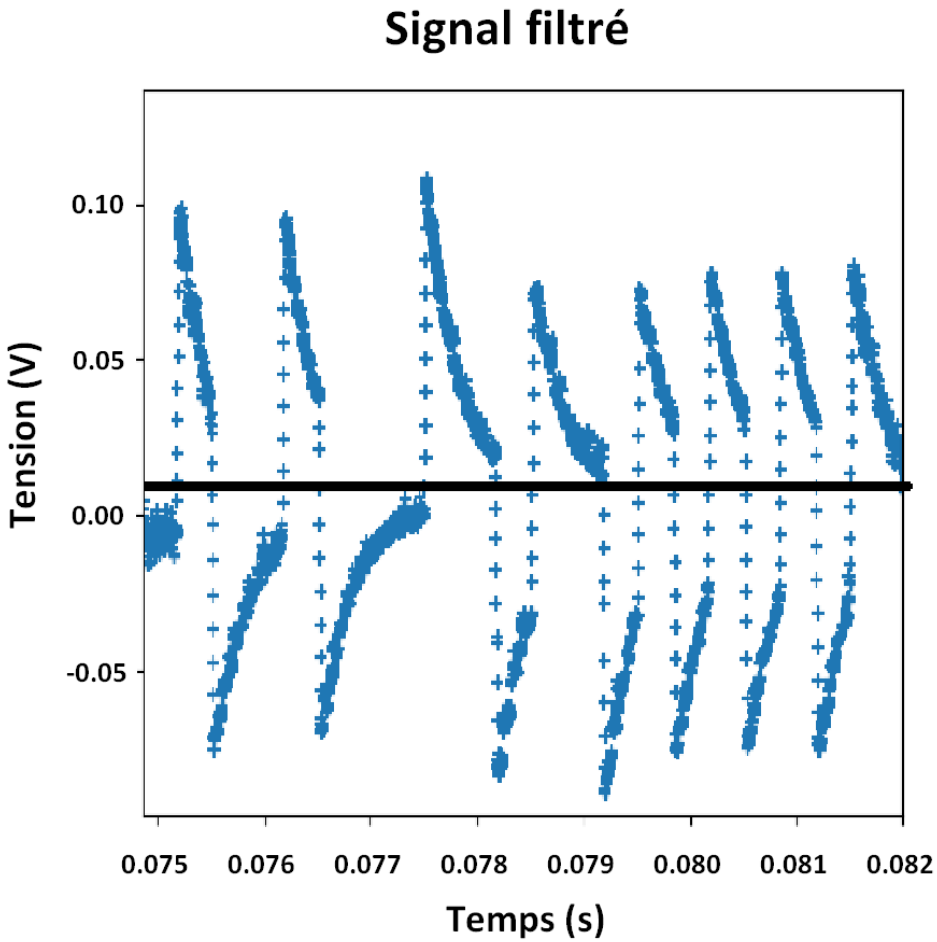
Choix du filtre

- Signal 3000 Bd et néon 100Hz
- Filtre passe-haut d'odre 1

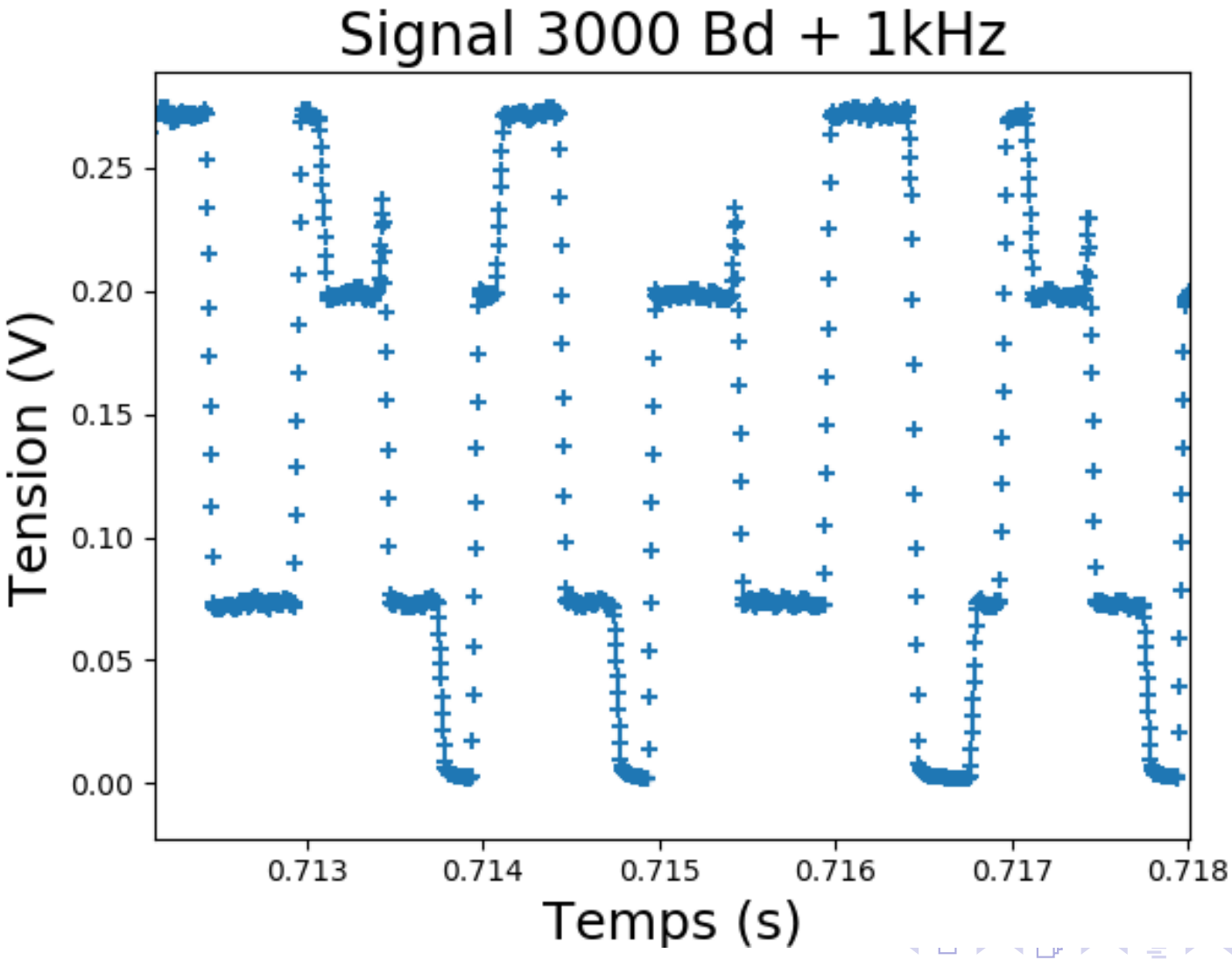
Expression du filtre

- $\underline{H}(j\omega) = \dfrac{j\frac{\omega}{\omega_C}}{1+j\frac{\omega}{\omega_C}}$ avec $f_c = 500Hz$
- $s_{ech}[i] = \dfrac{e_{ech}[i]-e_{ech}[i-1]+s_{ech}[i-1]}{w_c*h+1}$ (méthode d'Euler explicite)


Signal reçu
Tension (V)
0.525
0.500
0.475
0.450
0.425
0.400
0.375
0.350
0.075
0.076
0.077
0.078
0.079
0.080
0.081
0.082
Temps (s)


Signal filtré
Tension (V)
0.10
0.05
0.00
-0.05
0.075
0.076
0.077
0.078
0.079
0.080
0.081
0.082
Temps (s)

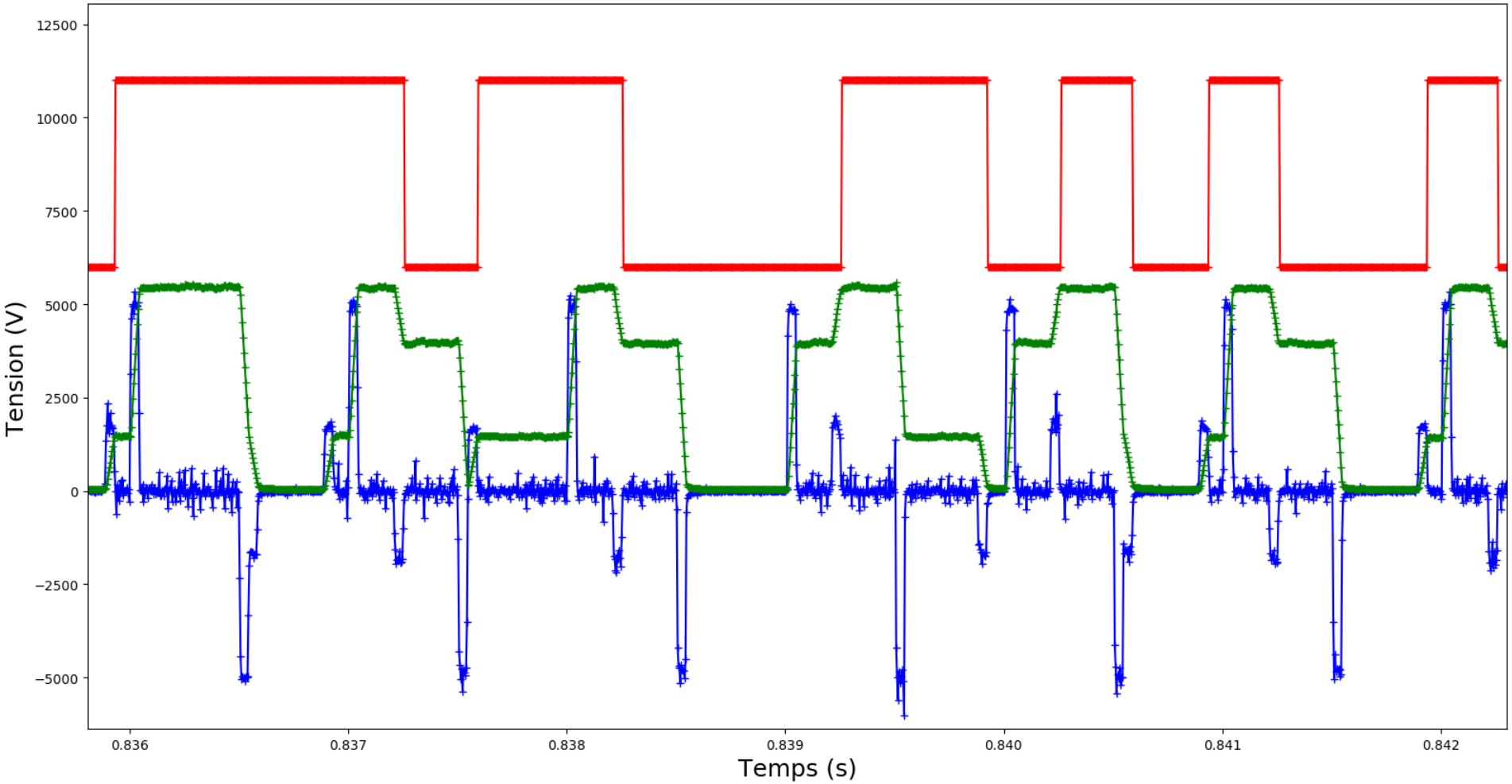
Perturbation par une lampe à intensité variable



Traitement de la perturbation

- Fréquences proches $\Rightarrow$ filtrage impossible
- Amplitude du signal $\neq$ amplitude de la perturbation
- Méthode de détection des fronts montants
- Utilisation de la dérivée du signal

Traitement du signal

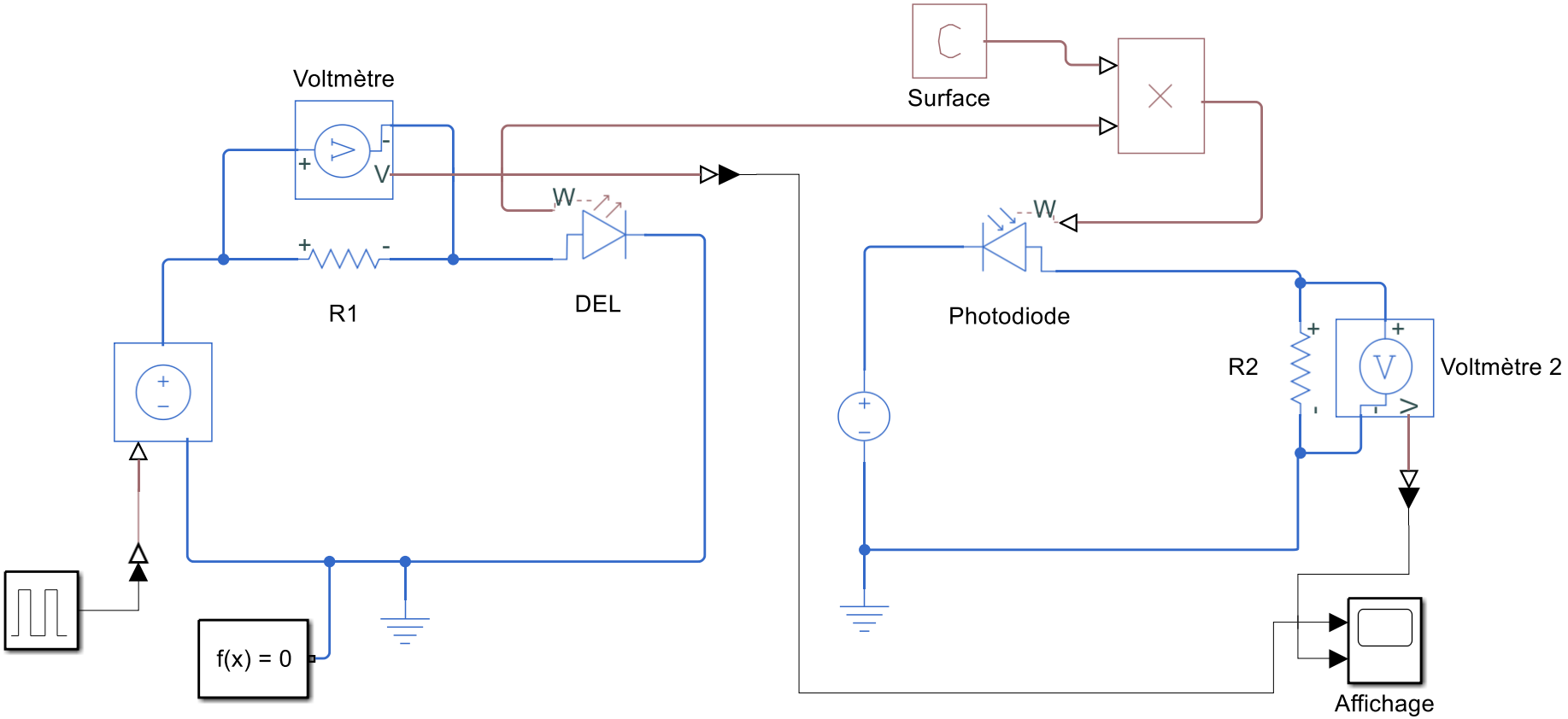


Conclusion

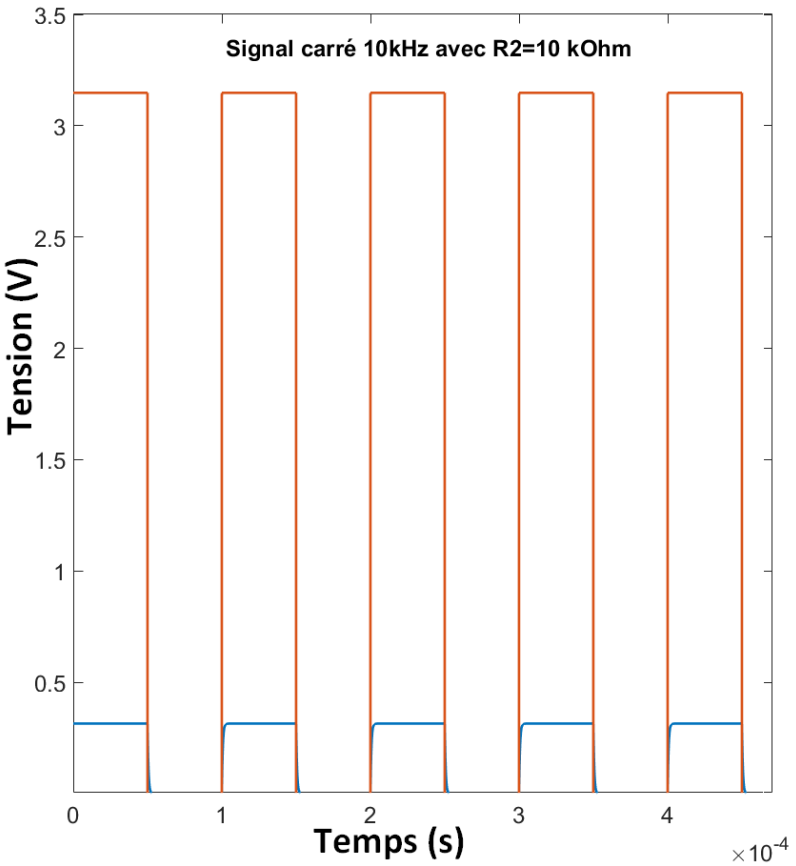
- Nombreuses sources de perturbations identifiées
- Traitement du signal possible et efficace
- Enjeux : communication entre véhicules

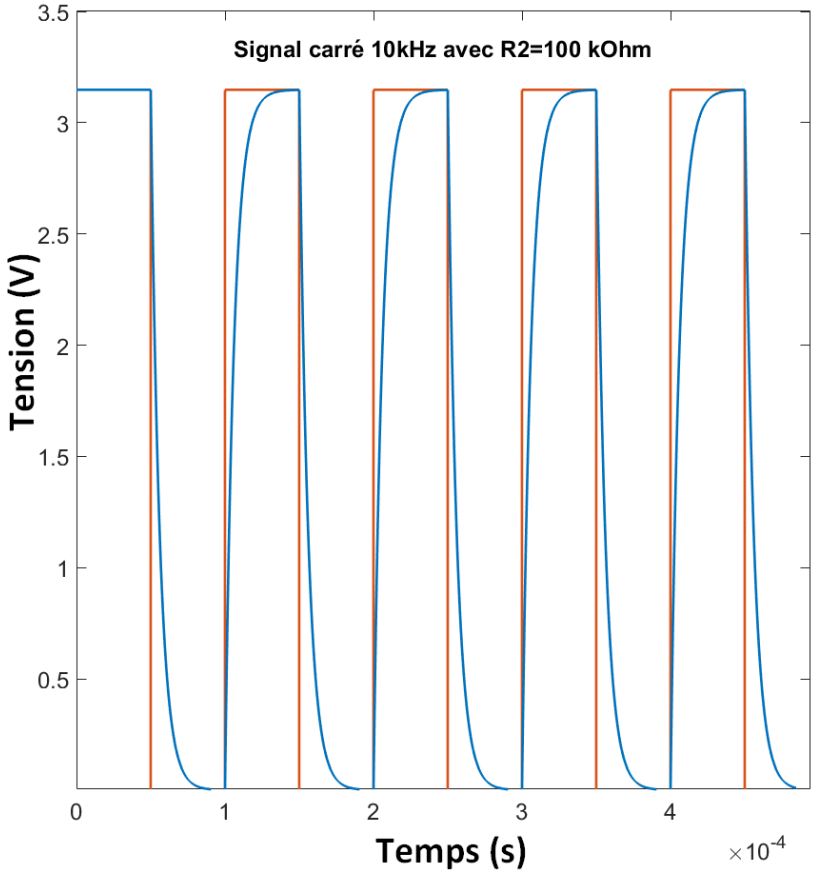
Annexes

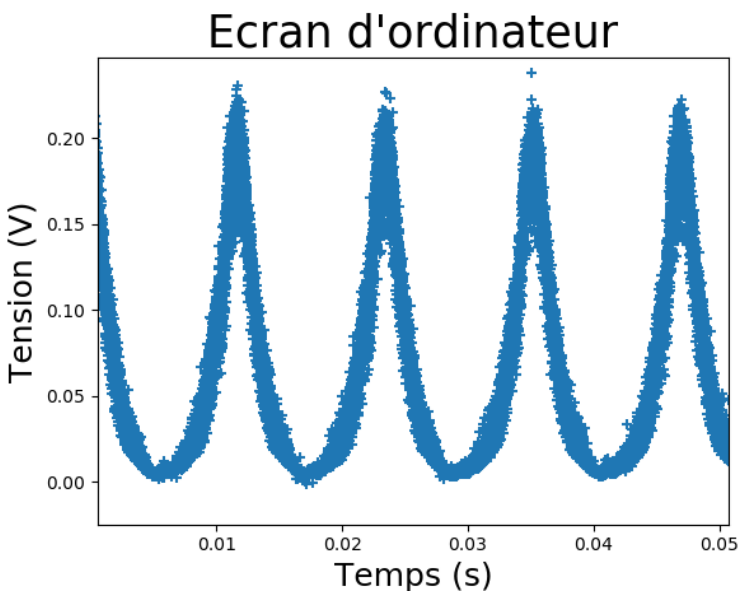
Simulation multi-physiques (Matlab-Simulink)



Simulation multi-physiques (Matlab-Simulink)






Ecran d'ordinateur
Tension (V)
0.00
0.05
0.10
0.15
0.20
0.01
0.02
0.03
0.04
0.05
Temps (s)

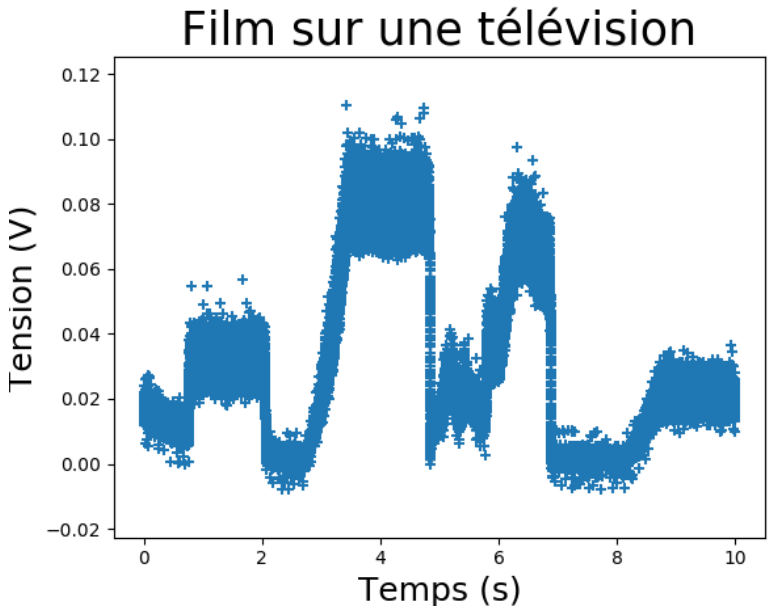
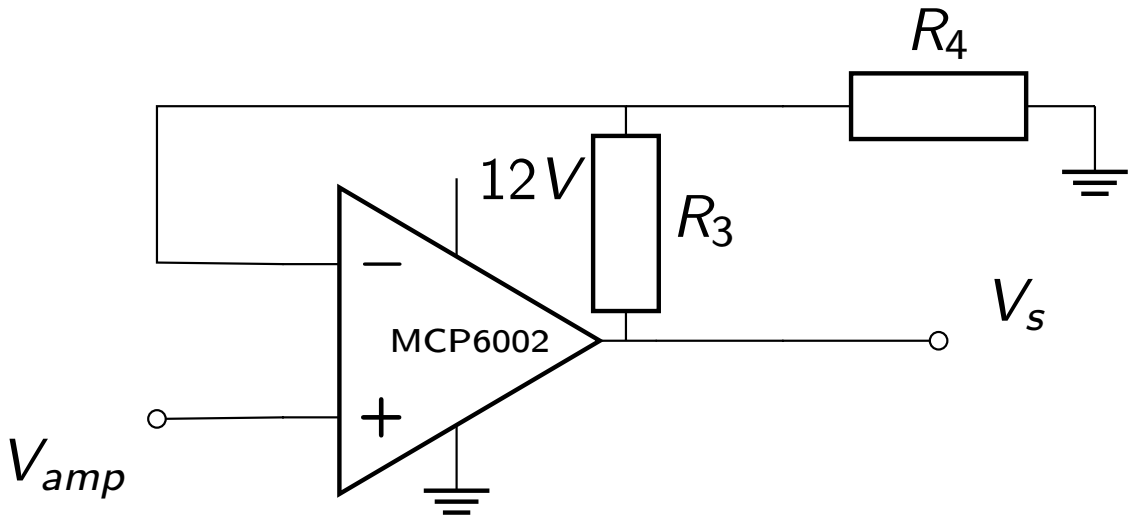

Film sur une télévision
Tension (V)
−0.02
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
2
4
6
8
10
Temps (s)

Schéma électrique



$$G = 1 + \frac{R_3}{R_4}$$

Code téléversé sur la carte Arduino

```
void setup()
{
  Serial.begin(9600);
  Serial2.begin(5000);
  Serial3.begin(3000);
}

void loop()
{
  Serial2.println("Ceci est un message transmis en lumière visible");
  Serial3.println("Ceci est un message transmis en lumière visible");
  tone(8,1000);
  tone(9,5000);
  tone(10,1000);
  delay(10);
}
```

Code Python

```
import matplotlib.pyplot as plt
import numpy as np

#importation du fichier de points
X = []
Y = []
f = open("C:...../courbe_1.txt", "r")
for x in f:
    a = x.split(';')
    X.append(float((a[0]).replace(',','.')))
    Y.append(float((a[1]).replace(',','.')))


def fourier(A,B):
""" Renvoie la transformée de Fourier d'un signal """
    fourier = np.fft.fft(B)
    spectre = np.absolute(fourier)
    freq = np.fft.fftfreq(len(A),X[1])
    return(freq,spectre)
```

```python
#filtres
def passe_haut_ordre_1(X,Y,f_c):
    t = X[1]
    n_points = len(X)
    e_ech = Y
    w_c = 2*np.pi*f_c
    s_ech = np.zeros(n_points)
    for i in range(1, n_points):
        s_ech[i] = ((e_ech[i] - e_ech[i-1]) + s_ech[i-1])/(w_c*t + 1)
    return s_ech

def passe_bas_ordre_1(X,Y,f_c):
    t = X[1]
    n_points = len(X)
    e_ech = Y
    w_c = 2*np.pi*f_c
    s_ech = np.zeros(n_points)
    for i in range(1, n_points):
        s_ech[i] = s_ech[i-1] + w_c * t * (e_ech[i-1] - s_ech[i-1])
    return s_ech
```

```
def moyenne(X,n):
""" Renvoie la moyenne mobile d'une courbe sur n valeurs """
    Y=X[:n]
    for i in range(n,len(X)):
        Y.append(mean(X[i-n:i]))
    return Y

#Affichage des courbes
plt.figure(2)
plt.scatter(X,moyenne(Y,10), marker='+')
plt.ylabel('Tension (V)',fontsize=18)
plt.xlabel("Temps (s)",fontsize=18)
plt.title("Signal 3000 Bd + néon",fontsize=20)
plt.show()
```

```
def sig_to_sigb(X):
""" Transforme un signal en chronogramme """
    moy = mean(X)
    B = []
    for x in X:
        if x>moy:
            B.append(1)
        else:
            B.append(0)
    return B

def sigb_to_bin(X,Y,n):
""" Transforme le chronogramme en nombres binaires """
    S = []
    A = []
    t = X[1]-X[0]
    fs = 1/t
    l_bit =fs/n
    i = 0
    while i < len(Y) - 20*l_bit:
        while Y[int(i)] == 1 and i < len(Y)-20*l_bit:
            i+=1
        for k in range(0,10):
            i+= l_bit/2
            S.append(Y[int(i)])
            i += l_bit/2
    return S
```

```python
def bin_to_char(A):
""" Renvoie le caractère ASCII d'un nombre binaire """
    b=0
    for i in range(0,len(A)):
        b += A[i]*2**i
    return(chr(b))

def bin_to_str(A):
""" Renvoie le chaine de caractère codée en binaire """
    i = 0
    message = ""
    while i < len(A) - 10:
        message += bin_to_char(A[i+1:i+9])
        i += 10
    return message
    #return message.encode("latin1").decode()
```

```
def decoder(X,Y,n):
""" Décode un signal avec les fonctions précédentes """
    B = sig_to_sigb(Y)
    N = sigb_to_bin(X,B,n)
    donnee = bin_to_str(N)
    return donnee

def derive(X,Y):
""" Dérive une fonction """
    F1 = []
    t = X[1] - X[0]
    for i in range(0,len(Y)-1):
        F1.append((Y[i+1]-Y[i])/t)
    F1.append((Y[-2]-Y[-1])/t)
    return F1
```

```
def trouver_amplitudes(X,Y,n):
""" Trouve les amplitudes des deux signaux """
    t = X[1] - X[0]
    fs = 1/t
    debut = int(len(Y)/2 -10*fs/n)
    fin = int(len(Y)/2 + 10*fs/n)
    E = Y[debut:fin]
    nb_valeurs = []
    amplitudes = []
    maxi = max(E)
    for i in range(10,100,5):
        compteur = 0
        for x in E:
            if x > i/100*maxi:
                compteur +=1
        amplitudes.append(i/100*maxi)
        nb_valeurs.append(compteur)
    l = len(nb_valeurs)
    return [amplitudes[int(l/4)],amplitudes[3*int(l/4)]]
```

```
def detecter_front(X,amplitudes):
""" Détecte les fronts montants d'un signal """
    a1 = amplitudes[0]
    a2 = amplitudes[1]
    S = len(X)*[0]
    for i in range(0,len(X)):
        if X[i] >= a1 and X[i] <= a2:
            S[i] = 1
        if X[i] > a2:
            S[i] = 2
        if X[i] <= -a1 and X[i] >= -a2:
            S[i] = -1
        if X[i] < -a2:
            S[i] = -2
    return S
```

```
def gen_signal(A):
""" Génère un chronogramme à partir des fronts montants """
    X = A
    i = 0
    while i < len(A):
        if X[i] != 0:
            c = X[i]
            i += 1
            while i < len(A) and X[i] == 0:
                X[i] = c
                i += 1
        i += 1
    return A
```