Communication par laser à grande distance, listings de code

Table des matières

1 Émission

```
1  from machine import Pin, PWM, ADC # type: ignore
2  from time import sleep
3
4  def format_bis(entier, nb_bits):
5      """
6      Convertit un entier en binaire sur un nombre fixe de bits.
7
8      Args:
9          entier (int): L'entier à convertir.
10         nb_bits (int): Le nombre de bits pour la représentation binaire.
11
12     Returns:
13         str: La représentation binaire de l'entier sur le nombre de bits
               spécifié.
14     """
15     # Convertit l'entier en binaire
16     binaire = ""
17     while entier > 0:
18         binaire = str(entier % 2) + binaire
19         entier //= 2
20
21     # Ajoute des zéros de remplissage si nécessaire
22     while len(binaire) < nb_bits:
23         binaire = "0" + binaire
24
25     # Tronque la chaîne si elle est trop longue
26     if len(binaire) > nb_bits:
```

```
27          binaire = binaire[-nb_bits:]
28
29      return binaire
30
31  led = Pin('LED', Pin.OUT)
32  laser = Pin(16, Pin.OUT)
33
34  # Alphabet personnalisé codé sur 32 caractères (chacun pouvant ainsi être
        codé sur 5 bits)
35  alphabet = {'a':1,'b':2,'c':3,'d':4,'e':5,'f':6,'g':7,'h':8,'i':9,'j':10,
        'k':11,'l':12,
36              'm':13,'n':14,'o':15,'p':16,'q':17,'r':18,'s':19,'t':20,'u'
                    :21,'v':22,'w':23,
37              'x':24,'y':25,'z':26,' ':27,',':28,'.':29,'!':30,'?':31,"'"
                    :32}
38
39  # Inversion du dictionnaire pour obtenir une correspondance valeur ->
        lettre
40  tebahpla = {i: j for j, i in alphabet.items()}
41
42  def text_to_binary(message): #Entrée : str
43      # Stocker le résultat
44      binary_message = ""
45      # Parcourir chaque caractère
46      for c in message:
47          if c in alphabet:
48              # On attribue une valeur au cractère
49              value = alphabet[c]
50              # Convertir la valeur en binaire sur 5 bits
51              binary_c = format_bis(value, 5)
52              binary_message += binary_c
53          else:
54              print("Caractère '{c}' non supporté dans l'alphabet
                    personnalisé.")
55      return binary_message
56
57  def hamming_encode(str):
58      reverse_str=str[::-1]
59      n=len(str)
60      k = 0
61      while (2 ** k) < (n + k + 1): #Calcul du nombre k de bits de contrôle
62          k += 1
63      result = [0 for u in range (n+k)]
64      i = 0
65      for x in range(1,n+k+1): #Ajoute les bits d'informations aux bons
            emplacements
66          if x & (x-1) != 0: #Vérifie si x n'est pas une puissance de 2, &
                est l'opérateur ET logique qui compare en binaire x et x-1.
67              #L'idée avec x et x-1 est que si x est une puissance de 2,
                    son binaire ne possède qu'un 1, x-1 inverse alors tous les
                     bits.
68              #Donc aucun des bits ne sera commun à x et x-1 donc "l'
```

```
                indicatrice" & ne s'allume jamais et on a 0.
            result[x-1]=reverse_str[i]
            i+=1
    bits_1=[]
    bin_bits_1=[]
    for x in range(n+k):
        if result[x] == '1':
            bits_1.append(x+1) #stock la position de tous les bits valant
                1
    for x in bits_1:
        bin_bits_1.append(format_bis(x,k))
    parity=[0 for i in range(k)]
    for x in bin_bits_1:
        for i in range(len(x)):
            parity[i]+=int(x[::-1][i]) #Calcule la parité de chaque bit
                de contrôle
    for u in range(len(parity)):
        if parity[u]%2==1:
            result[(2**u)-1]='1' #Ajuste la valeur du bit de contrôle
                selon sa parité
    final=''
    for y in result[::-1]:
        final+=y
    return(final)

def emission(binary,f):
    binary2="1"+str(binary)+"1"
    for x in binary2:
        if x == "1":
            laser.high()
            sleep(1/f)
            laser.low()
        else:
            sleep(1/f)

def text_to_light(message,f):
    return(emission(text_to_binary(message),f))

def text_to_light_hamming(message,f):
    return(emission(hamming_encode(text_to_binary(message)),f))
```

---

2 Réception (code principal)

```
from machine import Pin, ADC # type: ignore
import kmean
import error_detection
import convertors
import hamming
```

```
6  from time import sleep
7
8  photo = ADC(Pin(26))
9  base = ADC(Pin(27))
10 switch = Pin(0, Pin.IN)
11
12 f_emit = 7
13 bit_duration = 3
14 f_recep = f_emit*bit_duration
15
16 data = []
17
18 while switch.value() == 0:
19     pass
20
21 while switch.value() == 1:
22     data.append([photo.read_u16(), base.read_u16()])
23     sleep(1/f_recep)
24
25 centers = [[min(point[0] for point in data), min(point[1] for point in
       data)],[max(point[0] for point in data), max(point[1] for point in
       data)]]
26
27 centers, labels = kmean.k_means(data, centers, n = 100)
28 bits = error_detection.clean_signal(labels, bit_duration)
29 bits = convertors.list_to_str(bits)
30
31 ### sans hamming
32 message = convertors.binary_to_text(bits)
33
34 ### avec hamming
35 hamminged_message = hamming.hamming_decode(bits)
36
37 message = convertors.binary_to_text(hamminged_message)
38 print(message)
```

---

3 Réception (K-mean)

```
1  from math import sqrt
2
3  def distance(p1, p2):
4      return sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)
5
6  def k_means(data, centers, n=100):
7      n_points = len(data)
8      n_clusters = len(centers)
9
10     labels = [0]*n_points
```

```
11
12      for _ in range(n):
13          # update labels
14          new_labels = []
15          for i in range(n_points):
16              # compute all distances, attribute new centers
17              distances = [distance(data[i], center) for center in centers]
18              new_labels.append(distances.index(min(distances)))
19
20          # update clusters
21          new_centers = []
22          for k in range(n_clusters):
23              cluster_points = [data[i] for i in range(n_points) if
                    new_labels[i] == k]
24              if cluster_points != []: # for not empty clusters
25                  mean_x = sum(point[0] for point in cluster_points) / len(
                        cluster_points)
26                  mean_y = sum(point[1] for point in cluster_points) / len(
                        cluster_points)
27                  new_centers.append([mean_x, mean_y])
28
29              else: # for empty clusters
30                  new_centers.append(centers[k])
31
32          # check for convergence
33          if new_centers == centers:
34              break
35          centers = new_centers
36          labels = new_labels
37
38      print("labels")
39      print(labels)
40
41      return centers, labels
```

---

4 Réception (Traitement)

```
1  def clean_signal(bits, bit_duration):
2      output = rmv_first_zeros(bits, bit_duration)
3      output = filtering(output, bit_duration)
4      output = rmv_last_zeros(output)
5
6      output = output[1:-1]
7
8      print("clean_signal")
9      print(output)
10
11     return output
```

```
12
13 def rmv_first_zeros(bits, bit_duration):
14     init = True
15
16     while init :
17         while bits[0] == 0:
18             bits.pop(0)
19
20         mean = round(sum(bits[:bit_duration])/bit_duration)
21
22         if mean == 1:
23             init = False
24
25         else:
26             bits.pop(0)
27
28     return bits
29
30 def filtering(bits, bit_duration):
31     n = len(bits)
32     groups = [bits[i:i+bit_duration] for i in range(0, n, bit_duration)]
33     output = []
34
35     for group in groups:
36         mean = round(sum(group)/bit_duration)
37         output.append(mean)
38
39     return output
40
41 def fintering2(bits, bit_duration):
42     n = len(bits)
43     groups = [bits[i:i+bit_duration] for i in range(0, n, bit_duration)]
44     output = []
45
46     for group in groups:
47         mean = round(sum(group[1:-1])/(bit_duration-2))
48         output.append(mean)
49
50     return output
51
52 def rmv_last_zeros(bits):
53     while bits[-1] == 0:
54       bits.pop(-1)
55
56     return bits
```

---

5 Réception (Hamming)

```
1  def format_bis(entier, nb_bits):
2      """
3      Convertit un entier en binaire sur un nombre fixe de bits.
4
5      Args:
6          entier (int): L'entier à convertir.
7          nb_bits (int): Le nombre de bits pour la représentation binaire.
8
9      Returns:
10         str: La représentation binaire de l'entier sur le nombre de bits
               spécifié.
11     """
12     # Convertit l'entier en binaire
13     binaire = ""
14     while entier > 0:
15         binaire = str(entier % 2) + binaire
16         entier //= 2
17
18     # Ajoute des zéros de remplissage si nécessaire
19     while len(binaire) < nb_bits:
20         binaire = "0" + binaire
21
22     # Tronque la chaîne si elle est trop longue
23     if len(binaire) > nb_bits:
24         binaire = binaire[-nb_bits:]
25
26     return binaire
27
28 def hamming_decode(stri:str)->str:
29     reverse_str=list(stri[::-1])
30     n=len(stri)
31     k = 0
32     while (2 ** k) < (n + 1): #Cherche le nombre de bits de contrôle
33         k += 1
34     result=[]
35     bits_1=[]
36     bin_bits_1=[]
37     for x in range(n):
38         if reverse_str[x] == '1':
39             bits_1.append(x+1) #stock la position de tous les bits valant
                   1
40     for x in bits_1:
41         bin_bits_1.append(format_bis(x,k))
42     parity=[0 for i in range(k)]
43     for x in bin_bits_1:
44         for i in range(len(x)):
45             parity[i]+=int(x[::-1][i]) #Calcule la parité de chaque bit
                   de contrôle
46     ok=0 #Tant que ok=0, aucune erreur détectée
47     bit_error = 0
48     for u in range(len(parity)):
49         if parity[u]%2==1:
```

```
50              ok+=1
51              bit_error+=2**u
52              index_bit_error=bit_error-1
53      if ok == 0:
54          print("Aucune erreur détectée")
55      else:
56          print("Erreur détectée au bit",bit_error)
57          reverse_str[index_bit_error] = '0' if reverse_str[index_bit_error
                ] == '1' else '1'
58      for u in range(n):
59          if u+1 & u != 0:
60              result.append(reverse_str[u])
61      final=''
62      for y in result[::-1]:
63          final+=y
64      return final
```

---

6 Réception (Conversion)

```
1  # personalized alphabet coded on 32 characters (each character can be
       coded on 5 bits)
2  alphabet = {'a':1,'b':2,'c':3,'d':4,'e':5,'f':6,'g':7,'h':8,'i':9,'j':10,
       'k':11,'l':12,
3              'm':13,'n':14,'o':15,'p':16,'q':17,'r':18,'s':19,'t':20,'u'
                   :21,'v':22,'w':23,
4              'x':24,'y':25,'z':26,' ':27,',':28,'.':29,'!':30,'?':31,"'"
                   :32}
5
6  # inverting the dictionary to get a value -> letter
7  tebahpla = {i: j for j, i in alphabet.items()}
8
9  def list_to_str(list:list) -> str:
10     output = ""
11     for i in list :
12         output += str(i)
13     return output
14
15
16 def binary_to_text(binary_message, tebahpla:dict=tebahpla) -> str:
17     """Convert a "01" string to a string of ASCII characters."""
18
19     if type(binary_message) != str:
20         binary_message = list_to_str(binary_message)
21
22     message = ""
23
24     # parse the binary string into groups of 5 bits
25     for i in range(0, len(binary_message), 5):
```

```
26          bit = binary_message[i:i+5]
27
28          # convert the 5 bits to an integer, convert the integer to a
                character
29          try :
30              message += tebahpla[int(bit, 2)]
31
32          # if wrong char
33          except KeyError:
34              message += "*"
35
36      return message
```

---

7 Topographie de l'irradiance

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  z_max = 3
5  r_max = 5
6  z_r = 1
7  w_0 = 1
8  I_0 = 1
9
10 z_list = np.linspace(0, z_max, 300)
11 r_list = np.linspace(-r_max, r_max, 300)
12 Z, R = np.meshgrid(z_list, r_list)
13
14 def div(z, A, B):
15     return A * np.sqrt(1 + (z / B) ** 2)
16
17 def gaussian(r, A, B):
18     return A * np.exp(-2 * r**2 / B**2)
19
20 def radiance(z, r):
21     w = div(z, w_0, z_r)
22     return gaussian(r, I_0 * (w_0 / w)**2, w)
23
24 I = radiance(Z, R)
25
26 plt.figure('Irradiance faisceau laser gaussien')
27 extent = [z_list.min(), z_list.max(), r_list.min(), r_list.max()]
28 plt.imshow(I, extent=extent, aspect='auto', cmap='inferno')
29 plt.xlabel('Distance laser/réception (m)')
30 plt.ylabel('Rayon (mm)')
31 plt.title('Irradiance du faisceau laser gaussien (valeurs arbitraires)')
32 plt.colorbar(label=r'Irradiance ($W \cdot m^{-2}$)')
33 plt.tight_layout()
```

```
34 plt.show()
```

---

8 Caractérisation du faisceau Gaussien :

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import sqlite3
4  from scipy.optimize import curve_fit
5  import seaborn as sns
6
7  def gauss(x, A, B):
8          y = A*np.exp(-2*x**2/(B**2))
9          return y
10
11 def div(x,A,B):
12         y = A*np.sqrt(1+(x/B)**2)
13         return y
14
15 def get_r_v(z):
16     global cur
17     cur.execute(f'SELECT data.Radius, data.Voltage FROM data WHERE data.
           distance = {z} AND data.radius >= {0} ORDER BY data.Radius')
18     rows = cur.fetchall()
19     r = np.array([row[0] for row in rows])
20     v = np.array([row[1] for row in rows])
21     return np.array(r),np.array(v)
22
23 conn = sqlite3.connect('V2/data2.db')
24 cur = conn.cursor()
25
26 z_list = cur.execute('SELECT DISTINCT data.distance FROM data ORDER BY
       data.distance').fetchall()
27 z_list = [z[0] for z in z_list]
28 W_list = []
29
30 for z in z_list:
31     R,V = get_r_v(z)
32     parameters,covariance = curve_fit(gauss, R, V)
33     fit_A = parameters[0]
34     fit_B = parameters[1]
35
36     W_list.append(fit_B)
37
38     fit_r = np.linspace(min(R), max(R), 100)
39     fit_v = gauss(fit_r, fit_A, fit_B)
40
41     plt.figure(f'Matrice de covariance gaussienne z={z}m')
42     plt.title(f'Matrice de covariance gaussienne z={z}m')
```

```
43      sns.heatmap(covariance, annot=True, xticklabels=['Amplitude (V)', 'w(
            z) (mm)'], yticklabels=['Amplitude (V)', 'w(z) (mm)'], cmap='
            viridis')
44      plt.figure(f'Tension au fil du rayon (z={z})')
45      plt.plot(R,V, marker='+', linestyle='dotted', markersize=10, label='
            Mesures')
46      plt.plot(fit_r, fit_v, color='orange', label='Régression')
47      plt.grid()
48      plt.xlabel('Rayon (mm)')
49      plt.ylabel('Tension (V)')
50      plt.title(f'Évolution de la tension au fil du rayon, z={z}m')
51      plt.tight_layout()
52      plt.legend()
53
54  z_list, W_list = np.array(z_list),np.array(W_list)
55
56  parameters,covariance = curve_fit(div, z_list, W_list)
57  fit_A = parameters[0]
58  fit_B = parameters[1]
59
60  plt.figure('Matrice de covariance rayon caractéristique')
61  plt.title('Matrice de covariance rayon caractéristique')
62  sns.heatmap(covariance, annot=True, xticklabels=[r'${W_0}$ (mm)', r'${z_r
        }$ (m)'], yticklabels=[r'${W_0}$ (mm)', r'${z_r}$ (m)'], cmap='viridis
        ')
63
64  print(f'W0 = {fit_A}, zr = {fit_B}')
65
66  fit_z = np.linspace(min(z_list), max(z_list), 100)
67  fit_w = div(fit_z, fit_A, fit_B)
68
69  plt.figure('Rayon caractéristique distance')
70  plt.plot(z_list, W_list, marker='+', linestyle='dotted', markersize=10,
        label='Mesures')
71  plt.plot(fit_z, fit_w, color='orange', label='Régression')
72  plt.xlabel('Distance laser/réception (m)')
73  plt.ylabel('Rayon caractéristique (mm)')
74  plt.title('Rayon caractéristique au fil de la distance')
75  plt.legend()
76  plt.tight_layout()
77  plt.grid()
78  plt.show()
```

---

9 Caractérisation des photorésistances :

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
```

```
4  # res de mesure
5  R_mes = 215
6
7  I_global = []
8  U_photo_global = []
9  i_mes_global = []
10 E = np.array([0,0.49,0.98,1.47,1.96,2.45,2.94])
11
12 def store(I,U_R):
13     U_photo = E - U_R
14     i_mes = U_R / R_mes
15
16     I_global.append(I)
17     U_photo_global.append(U_photo)
18     i_mes_global.append(i_mes)
19
20 I_global = np.array(I_global)
21 U_photo_global = np.array(U_photo_global)
22 i_mes_global = np.array(i_mes_global)
23
24 ### FIRST PLOT ###
25 fig = plt.figure('Données brutes')
26 ax = fig.add_subplot(projection='3d')
27
28 for i in range(len(I_global)) :
29     ax.plot(i_mes_global[i],U_photo_global[i], I_global[i], marker = '+',
           markersize=10)
30
31 plt.title('Lien entre intensité, tension et éclairement lumineux')
32 ax.set_xlabel('Intensité (A)')
33 ax.set_ylabel('Tension (V)')
34 ax.set_zlabel('Éclairement lumineux (lx)')
35 plt.tight_layout()
36 ax.grid()
37
38 ### LINEAR REGRESSION ###
39 R = []
40 R_square = []
41
42 for i in range(len(I_global)) :
43     vec = np.polyfit(i_mes_global[i], U_photo_global[i],1)
44     R.append(vec[0])
45     U_reg = i_mes_global[i]*vec[0]
46     err_quadratique = np.sum((U_photo_global[i] - U_reg)**2)
47     ecart_type = np.sum((U_photo_global[i] - np.mean(U_photo_global[i]))
         **2)
48     R_square.append(1-err_quadratique/ecart_type)
49
50 R = np.array(R)
51
52 min_R_square = min(R_square)
53 min_R_square = round(min_R_square,3)
```

```
54
55 fig = plt.figure('Régression linéaire 1')
56
57 plt.plot(I_global,R, marker='+', linestyle='dotted', markersize=10)
58 plt.title(r'Lien entre éclairement lumineux et resistance, $R^2 = $' +
       str(min_R_square))
59 plt.xlabel('Éclairement lumineux (lx)')
60 plt.ylabel(r'Resistance ($\Omega$)')
61 plt.tight_layout()
62 plt.grid()
63
64 ### LOI PHOTORESISTANCE ###
65 new_R = 1/R
66
67 vec = np.polyfit(I_global,new_R,1)
68 I_lin = np.linspace(min(I_global),max(I_global),100)
69 reg = vec[0]*I_lin + vec[1]
70
71 err_quadratique = np.sum((U_photo_global[i] - U_reg)**2)
72 ecart_type = np.sum((U_photo_global[i] - np.mean(U_photo_global[i]))**2)
73
74 R_square = (1-err_quadratique/ecart_type)
75 R_square = round(R_square,4)
76
77 fig = plt.figure('Régression linéaire 2')
78 plt.plot(I_global,new_R, marker='+', linestyle='dotted', markersize=10,
       label='Mesures')
79 plt.plot(I_lin,reg, color='orange', label='Régression')
80 plt.title(r'Régression linéaire, $R^2 = $' + str(R_square))
81 plt.xlabel('Éclairement lumineux (lx)')
82 plt.ylabel(r'1/R ($\Omega^{-1}$)')
83 plt.legend()
84 plt.tight_layout()
85 plt.grid()
86
87 R_reg = 1/reg
88
89 fig = plt.figure('Loi fonctionnement photorésistance')
90 plt.plot(I_global,R, marker='+', linestyle='dotted', markersize=10, label
       ='Mesures')
91 plt.plot(I_lin,R_reg, color='orange', label='Régression')
92 plt.title('Loi de fonctionnement de la photorésistance')
93 plt.xlabel('Éclairement lumineux (lx)')
94 plt.ylabel(r'Résistance ($\Omega$)')
95 plt.legend()
96 plt.tight_layout()
97 plt.grid()
98 plt.show()
```