# Asservissement en position d'un banc laser

Dany Zakharia
João Luiz de Oliveira Pacheco
Maxime Nurwubusa
Nicolas Guenaux
Tancrède Esnouf

---

## I - Descriptif du projet

### 1) Contexte

Sur le nouveau site d'Etretat, le laboratoire de l'IOGS est sujet à des perturbations liées aux vagues qui rendent inexploitables les mesures et résultats d'un certain nombre d'expériences. En effet, les perturbations désaxent les laser sur les bancs optiques. Pour pallier ce problème, nous nous proposons d'asservir en position une photodiode par rapport à un faisceau laser qui se déplacent sous l'action d'une perturbation extérieure. Expérimentalement, l'étude sera menée grâce à une photodiode de type *TSL201R-LF,* dont les caractéristiques sont données en annexe. Ainsi qu'un laser de 6V émettant dans le bleu et dont la position est ici simulée comme s' il s'agissait d'une perturbation sismique pour le laboratoire d'Etretat.

Figure 1: Lieu choisi par les architectes pour le nouveau site de Supoptique Etretat.

**Cahier des Charges :**

Pour répondre à ces besoins, notre produit :

- Est capable de réagir à des secousses de l'ordre de quelques centimètres par secondes (faibles vagues)
- Est capable de trouver sa position d'équilibre en moins de 2 secondes.
- - Propose un interfaçage simple et agréable, pour les étudiant.e.s et professeur.e.s du LenSe

## 2) Explications approfondies du montage

### 2.1) Servomoteurs

Les positions du capteur et du laser sont contrôlées via deux servomoteurs. Ces moteurs permettent grâce à une commande PWM, de contrôler en position angulaire les palmes du moteur et donc de contrôler en translation le laser. Généralement, la période de la commande PWM est fixée, c'est la valeur du temps haut qui influence la position angulaire du servomoteur. Cela revient à changer le rapport cyclique. Pour câbler les servomoteurs, nous les alimentons avec une tension comprise entre 4,8V et 6V, la tension choisie comme indiquée sur la figure, influencera la vitesse des palmes. La tension d'alimentation correspond au câble rouge, le noir à la masse et l'orange reçoit la commande PWM.

Pour l'expérience, nous les alimentons avec une tension de l'ordre de 5V par mesure de sécurité car il serait fâcheux d'endommager un moteur.

Afin de simuler les perturbations mécaniques qui gênent l'alignement du laser, nous avons décidé de coder un déplacement oscillant du chariot portant le laser (cf. figure 2). Le code est détaillé en ANNEXE (cf. Code Perturbations).
Par la suite, ce code vous permettra également de vérifier le bon branchement des servomoteurs.

Modelcraft RS-2 Servo

Specifications

| | |
|---|---|
| Modulation: | Analog |
| Torque: | **4.8V:** 45.30 oz-in (3.26 kg-cm)<br>**6.0V:** 49.60 oz-in (3.57 kg-cm) |
| Speed: | **4.8V:** 0.19 sec/60°<br>**6.0V:** 0.17 sec/60° |
| Weight: | 1.38 oz (39.2 g) |
| Dimensions: | **Length:** 1.61 in (41.0 mm)<br>**Width:** 0.79 in (20.0 mm)<br>**Height:** 1.65 in (42.0 mm) |
| Motor Type: | (add) |
| Gear Type: | Plastic |
| Rotation/Support: | Bushing |
| Rotational Range: | 203° |
| Pulse Cycle: | 20 ms |
| Pulse Width: | 540-2470 µs |
| Connector Type: | JR |

| | |
|---|---|
| Brand: | MODELCRAFT |
| Product Number: | (add) |
| Typical Price: | (add) |
| Compare: | view |

Figure 2: Caractéristiques des servomoteurs utilisés.

## 2.2) Capteur photodiode

Le capteur utilisé dans cette étude est composé de 64 pixels disposés linéairement. Les pixels sont numérotés dans le code de 1 à 64. Le but étant de positionner le laser autour du 32 ème pixel, car il s'agit du centre du capteur photodiode.

Nous pouvons avoir un aperçu du fonctionnement de la barette CCD sur le schéma présenté en ANNEXE. Celle-ci explique que deux signaux PWM doivent être fournis en entrée de la photodiode, correspondant aux signaux SI (pour Serial Input) et Clock (Horloge). Le SI permet de démarrer l'acquisition des 64 valeurs de pixels, cela signifie qu'à chaque lecture du Heaviside, le capteur commence l'acquisition des intensités perçues sur chacun des 64 pixels. Le signal d'horloge "s'exécute" avant l'acquisition de chaque pixel : donc au moins 64 fois plus souvent. Ainsi, en pratique dans le code, on pourra implémenter ces deux signaux par des PWM avec un rapport cyclique de 50% et des période $T_{int}$ et $\frac{T_{int}}{64}$, la valeure de $T_{int}$ restant à déterminer : c'est le temps d'intégration des pixels.

Selon le temps d'intégration, les signaux recueillis par les pixels peuvent être saturés, insensibles au signal laser, ou bien adaptés. Dans ce dernier cas, nous pourrions distinguer le signal du laser de la lumière ambiante. Ainsi, il sera préférable de s'isoler de toute lumière parasite. Cela ne représente pas d'inconvénient majeur pour notre application, étant donné la portabilité de notre montage électronique.

L'enchaînement des signaux en entrées de la photodiode (SI et Clock) et en sortie (AO) s'effectue selon le schéma de principe suivant :

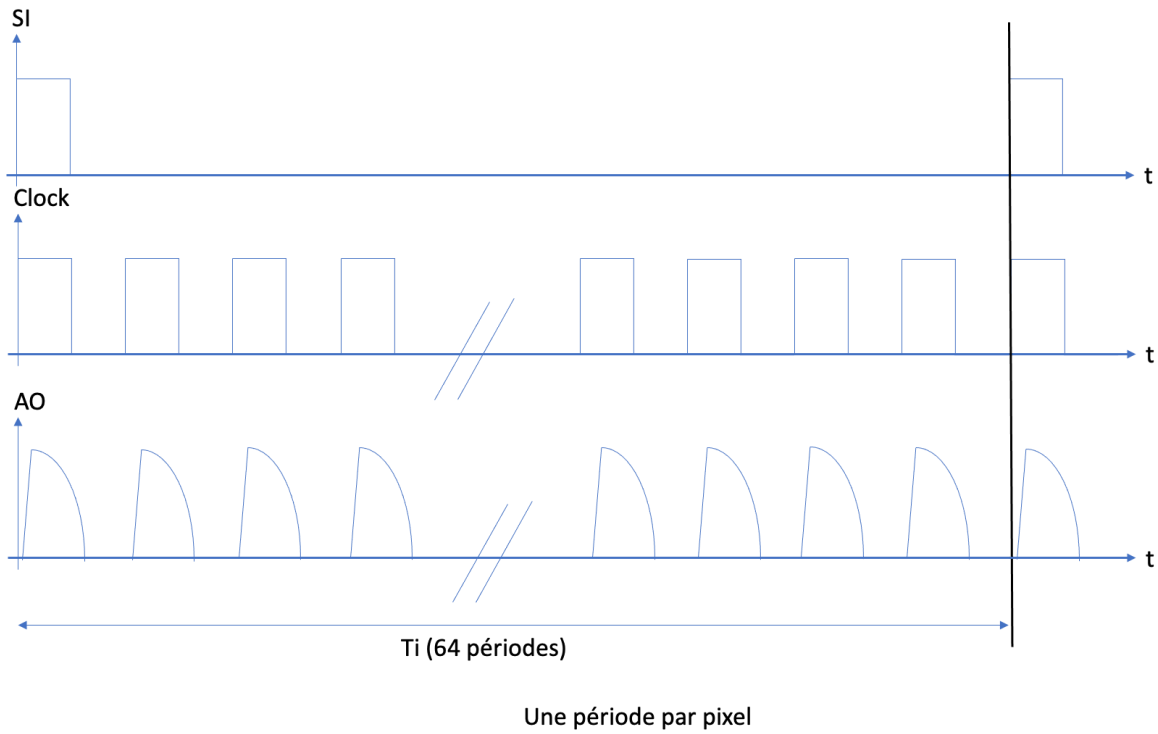



Figure 3: Explications des signaux SI, Clock et Ao.

Nous observons un léger temps de montée qui sépare chaque "porte" du signal d'horloge du signal de sortie AO : c'est le temps de réponse d'un pixel du capteur. En pratique, ce temps ne représente pas d'obstacle significatif dans l'exécution de l'algorithme.

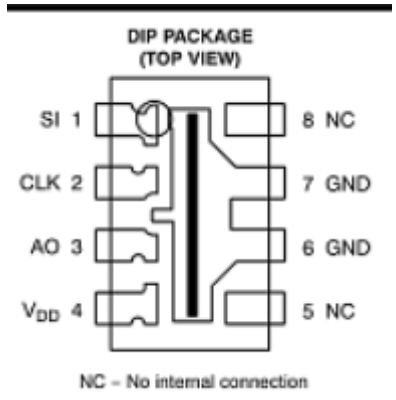



Figure 4: Schéma des sorties et entrées du capteur photodiode. Une vue plus détaillée est donnée en annexe.

Les branchement de la photodiode s'effectuent en tenant compte des positions de sorties et d'entrées sur le capteur (cf. Figure 4). L'alimentation $V_{DD}$ du capteur est de 5V continu et les bornes SI et CLK sont connectées à des sorties PWM et la borne A0 à une entrée analogique sur la carte Nucleo.

### 2.3) Montage complet.

Le montage final est représenté sur la figure suivante. Des alimentations externes alimentent les composants car la puissance de la nucleo n'est pas suffisante pour faire tourner le montage. De plus, veiller à connecter la masse de la carte à celle des alimentations.

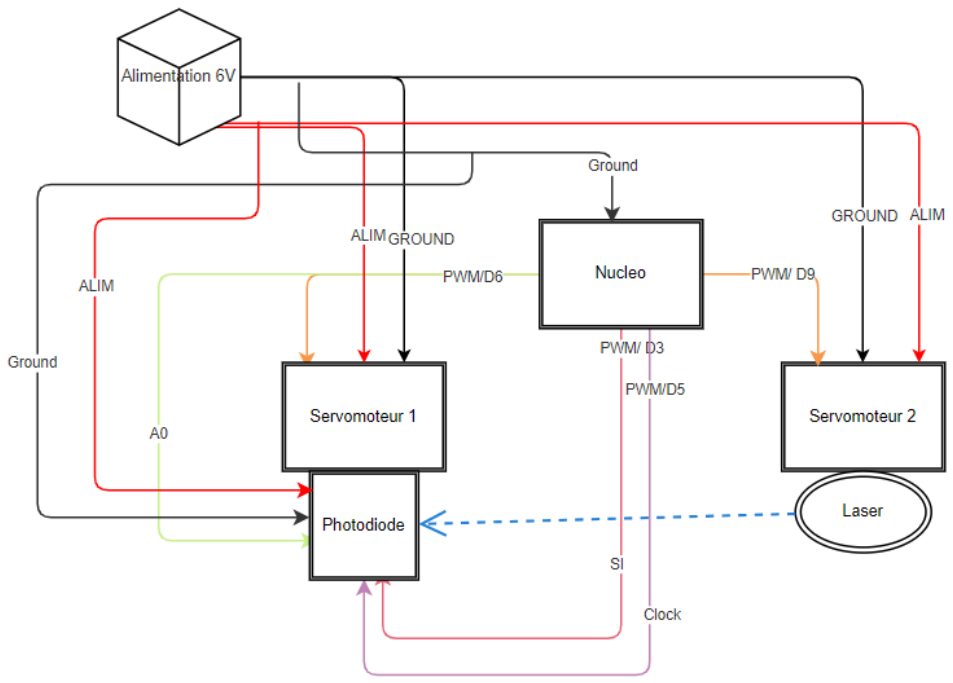



Figure 5: Schéma du câble électrique du montage.

## II- Méthode d'asservissement

### 1) Schéma d'asservissement

Le principe de l'asservissement en position est décrit sur la figure suivante (cf. figure 6). L'asservissement est mené selon une consigne définie comme la position du centre de la photodiode. Le programme compare la position du pixel ayant reçu le maximum d'intensité avec la position de consigne.
Nous avons fait apparaître du bruit qui peut être occasionné par de la lumière parasite par exemple; Nous n'avons pas de moyen de l'estimer mais nous en tenons

compte simplement pour informer que le système est soumis à des perturbations extérieures.
L'interface affiche en temps réel, la valeur des intensités reçues sur chacun des pixels de la barrette, le code et le maniement de cette interface est précisé plus loin.

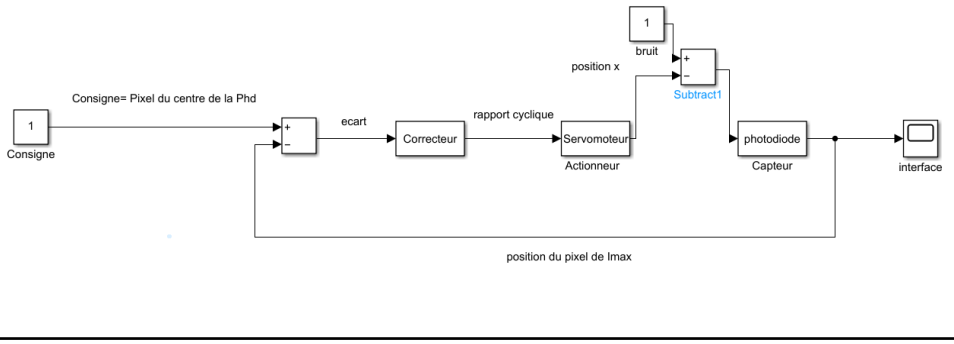



Figure 6: Schéma d'asservissement en position du laser.

## 2) Asservissement

L'asservissement se déroule en 5 étapes qui idéalement s'enchaînent de manière simultanée :

- Le servomoteur laser simule un déplacement, sinusoïdal par exemple.
- Le capteur linéaire CCD acquiert localement l"information sur le flux de photon reçu, donnant l'information sur la position du laser. Le pixel le plus éclairé sur la barrette photosensible correspond au désaxement du laser le long de l'axe optique représenté par le pixel du milieu de la barrette.
- La carte nucleo sur laquelle est implémentée le code, calcule alors la réponse et l'asservissement nécessaire pour aligner le capteur. Pour cela, il calcule la différence de pixel entre celui détecté et la consigne puis cette différence est exprimée en écart de longueur d'onde de position angulaire pour le servomoteur.
- Le servomoteur CCD exécute la commande d'asservissement. Il corrige la position angulaire du laser.
- Les données sont représentées sur l'interface système, dans un but de lisibilité.

Pour une perturbation donnée, l'asservissement sera réussi si la photodiode ne perd pas de vue le faisceau laser. Dès lors, on peut augmenter l'amplitude des perturbations et réitérer l'analyse. Selon les cas, les coefficients d'asservissement (Proportionnel, Intégral, Dérivé) optimaux peuvent différer. Pour cela, se référer à la partie Asservissement du code, en annexe.

L'asservissement utilisé est un correcteur PID. Nous additionnons une correction, proportionnelle à l'erreur, une correction intégrale et une correction dérivée.

## III - Interfaçage

Pour contrôler l'asservissement, une interface graphique a été construite en python (les programmes sont accessibles ici ou en annexe). Cette interface permet de lancer les programmes de remise au centre, d'asservissement et permet également une vision en temps réel de l'intensité reçue sur chacun des pixels. La figure ci-dessous représente l'interface finale.

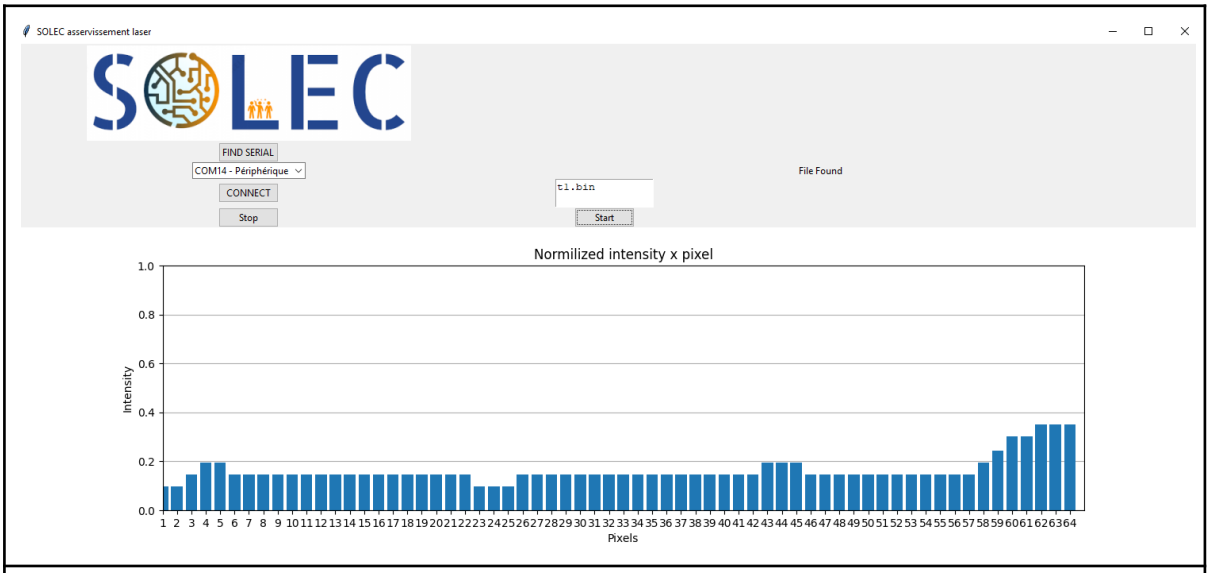



Figure 7: Vue de l'interface qui montre l'intensité des pixels au cours du temps.

Pour utiliser cette interface sur Windows, il faut installer le logiciel python. Dans l'ordinateur de l'IOGS, trois commandes supplémentaires doivent être mises sur la commande (terminal):

-pip install matplotlib (pour afficher les résultats sur un graphique)

-pip install pyserial (pour manipuler la liaison de la carte Nucléo)

-pip install pywin32 (pour trouver le chemin du fichier contenant le code MBED)

L'interface est composée de trois fichiers: un fichier "acquisition.py" , un fichier "__exec.py" et un fichier "SerialUSB_Nucleo.py" .Le fichier "acquisition.py" est le programme principal où on a défini les classes python tkinter permettant d'affecter des fonctions à des boutons sur l'interface. "SerialUSB_Nucleo.py" est un programme contenant tous les paramètres de la connexion avec la carte Nucleo.Le fichier "__exec.py" programme que l'on appelle sur la console windows pour lancer l'interface.

Pour faire fonctionner l'interface, on choisit la bonne connexion dans la fenêtre find Serial.On appuie sur le bouton "Connect" et on rentre dans la fenêtre à côté du nom du programme MBED en format "bin"(le programme doit être dans le même dossier).Puis on appuie sur le bouton "start", on envoie le programme à la carte Nucleo le système démarre et on affiche l'intensité normalisé de chaque pixel sous le format d'un histogramme.

## IV- Conclusion

En combinant le programme mbed (fichier .bin compilé) avec l'interface python, on a pu contrôler le moteur, faire bouger le laser et capter les valeurs d'intensité obtenues par la photodiode. Cependant, les réglages des temps caractéristiques du correcteur PID doivent être améliorés afin qu'il se déplace comme souhaité. Il faut aussi faire attention au temps de mise à jour de l'interface pour la visualisation des données, pour que cela n'interfère pas avec les performances du système.

## V- ANNEXE

### Code de simulation d'une perturbation latérale du laser - Gauche-droite-gauche-droite à l'infini :

```
// Blinking rate in milliseconds
#define BLINKING_RATE_MS   500
PwmOut moteur1(D9); //Sortie pilotant le servomoteur
Serial pc(USBTX,USBRX);

int main(){
    double rc1=0.06; // On définit le rapport cyclique initial
    int N=60; // Nombre de positions prises par le chariot lors d'un allé
    double compteur=0;
    double v=1; // coefficient permet de changer le sens de translation
    double alpha=(0.1235-0.02)/N; // Cela correspond au pas du chariot
    // Le numérateur correspond à la dimension spatiale parcourable par le chariot en rc
    moteur1.period_ms(20); // On fixe la période du signal PWM
    moteur1.write(rc1); // Son rc est initialement rc1 qui correpsond à une position au centre
    // De la course du chariot sur l'axe.
    while (1){
        rc1= rc1+v*alpha; // on incrémente le rc par pas d'alpha.
        compteur+=v/abs(v);
        moteur1.write(rc1);
        /*wait(1);*/ // Peut etre rajouté mais n'est pas nécessaire
        pc.printf("%lf",v);
         pc.printf("%lf",compteur);
        if (abs(compteur)==10)
        {v=-v;}

        }}
```

---

### Fiche technique de la photodiode TSL 201 R-LF:

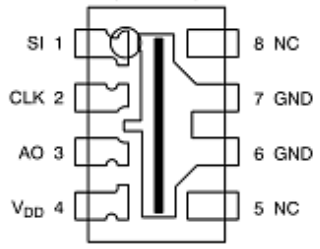
TAOS TEXAS ADVANCED OPTOELECTRONIC SOLUTIONS®

# TSL201R–LF
# 64 × 1 LINEAR SENSOR ARRAY

TAOS077B – MAY 2007

- 64 × 1 Sensor-Element Organization
- 200 Dots-Per-Inch (DPI) Sensor Pitch
- High Linearity and Uniformity
- Wide Dynamic Range . . . 2000:1 (66 dB)
- Output Referenced to Ground
- Low Image Lag . . . 0.5% Typ
- Operation to 5 MHz
- Single 5-V Supply
- Replacement for TSL201 and TSL201R
- RoHS Compliant

DIP PACKAGE
(TOP VIEW)



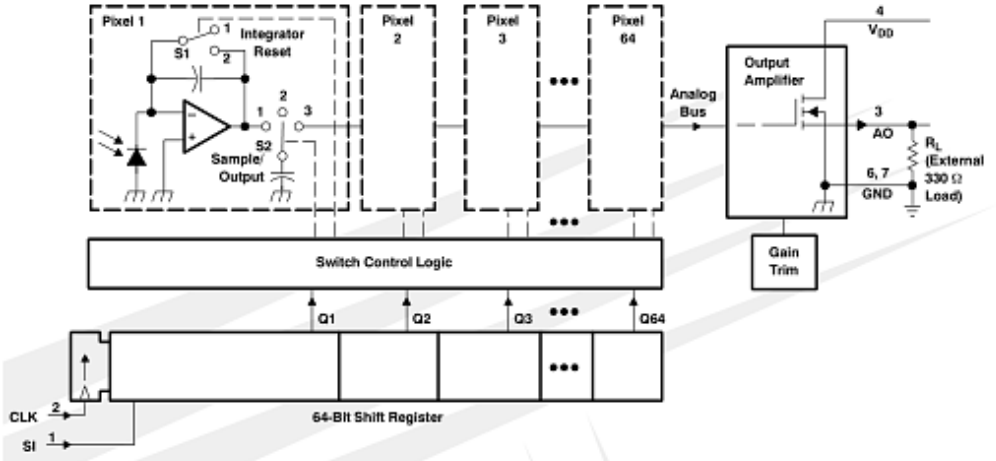

NC – No internal connection

## Description

The TSL201R–LF linear sensor array consists of a 64 × 1 array of photodiodes and associated charge amplifier circuitry. The pixels measure 120 μm (H) by 70 μm (W) with 125-μm center-to-center spacing and 55-μm spacing between pixels. Operation is simplified by internal control logic that requires only a serial-input (SI) signal and a clock.

The TSL201R–LF is intended for use in a wide variety of applications including mark detection and code reading, optical character recognition (OCR) and contact imaging, edge detection and positioning as well as optical linear and rotary encoding.

## Functional Block Diagram








Texas Advanced Optoelectronic Solutions Inc.
1001 Klein Road • Suite 300 • Plano, TX 75074 • (972) 673-0759
www.taosinc.com



---

# Codes Interfaçage :

## Acquisition.py

```
#26/02/2022
#João Luiz de Oliveira Pacheco IOGS 2A
#Dany Zakharia  IOGS 2A
#acquisition.py defines a class of function related to the tkinter interface for controlling NUCLEO-L476RG
#adaptation of http://lense.institutoptique.fr/ressources-protis/, create by Professor Julien VILLEMEJANE  IOGS


# All necessary 'tools '
from tkinter import * #interface
from tkinter.ttk    import * #interface
import time #time controll
import os #system  path-folder
from os import listdir #system  path-folder
from os.path import isfile, join #system  path-folder
import pathlib #system  path-folder
import sys
import glob       #linux path
import matplotlib.pyplot as plt #plot
import SerialUSB_Nucleo as sUSB #USB connection
from PIL import Image, ImageTk #SOLEC logo
from matplotlib.backends.backend_tkagg import (
   FigureCanvasTkAgg, NavigationToolbar2Tk) #animate plot
from matplotlib.backend_bases import key_press_handler #animate plot
from matplotlib.figure import Figure #plot
import numpy as np #math
import matplotlib.animation as animation #animate plot



class Acquisition():
#class acquisiton: function __init__; animate; actionFind; actionConnect;action_readfilename; interfacage; actionStop; serialList; run
#parameter self allows to acess the attributes and methods of the class  (all variables )
   def __init__(self):
      #Creates all the variables used for create buttons and bind functions

      self.running=False #Did the user pressed start?
      self.found=False #Did I find a connection
      self.envoye=False #Did I send the file yet?

      #connection variables
```

```
        self.serialUSB = sUSB.SerialUSB_Nucleo() #class from  SerialUSB_Nucleo.py
                self.ports = self.serialUSB.serial_ports()    #calling function from SerialUSB_Nucleo.py
        self.ports_name = [] #connection name
        self.ports_number = []
        self.ports_value = 0

        #create main window
        self.mainWindow = Tk() # Main window
        self.mainWindow.title("SOLEC asservissement laser") #tilte


        #Logo solec
        image = Image.open('solec.png')#image
        photo = ImageTk.PhotoImage(image)
        label = Label( image = photo)
        label.image = photo
        label.grid(row=0)#posisiton

        # Buttons - and actions

        #crating a teextbox to write the .bin name
        self.textBox=Text(self.mainWindow, height=2, width=15)
        self.textBox.grid(row=3, column = 1)#textbox position

        #text to verify if there is a connection
        self.varPortConnected = StringVar()
                self.labelPortConnected = Label(self.mainWindow, textvariable = self.varPortConnected)
        self.labelPortConnected.grid(row=2, column = 2)


          self.bFindSerial = Button(self.mainWindow, text ='FIND SERIAL', command =self.actionFind)#Find serial button see function actionFind for its acction
              self.bConnect = Button(self.mainWindow, text ='CONNECT', command =self.actionConnect) #Connect  button see function actionConnect for its action

        self.bFindSerial.grid(row=1, column = 0)#Find serial button position
        self.bConnect.grid(row=3, column = 0)#Connect  button position

                self.bfilename = Button(self.mainWindow, text ='Start', command =self.action_readfilename)#Start button see function action_readfilename for its acction
        self.bfilename.grid(row=4, column=1)#Start button position
```

```
            self.bDataStop = Button(self.mainWindow, text ='Stop', command =self.actionStop)#Start button see function actionStop for its acction
        self.bDataStop.grid(row=4, column=0)#Stop button position


       #creating plot (animated histogram)
        self.figure = Figure(figsize=(15, 4), dpi=100)#figure size
        self.ax = self.figure.add_subplot(111)#definig plot's position


        self.x_data = [i+1 for i in range(64)]#initital data
        #print(self.x_data)
        self.y_data = [0.25 for i in range(64)]#initital data
          self.plot = self.ax.bar(self.x_data, self.y_data)[0]#initial graph based on intial data

        #axes limits
        self.ax.set_ylim(0, 1)
        self.ax.set_xlim(1, 65)

        self.ax.set_axisbelow(True) #set grid position below data
        self.ax.grid(axis = 'y')#grid only in y

        self.ax.set_xticks(np.arange(len(self.x_data))+1)#xlabel fit
        self.ax.set_xticklabels(self.x_data)#x label fit
        self.ax.title.set_text('Normilized intensity x pixel')#title
        self.ax.set_xlabel('Pixels')
        self.ax.set_ylabel('Intensity')

           #creating tkinter variable ion main Window
        self.canvas1 = FigureCanvasTkAgg(self.figure, master = self.mainWindow)
        self.canvas1.draw()
                self.canvas1.get_tk_widget().grid(row=5, column = 0, columnspan = 5)#positioning

    def animate(self,intensite):
        #create an histogram with a vector (intensite)

        self.y_data=intensite #y values



        self.ax.clear() #clear old data
        #same procedure as creating intial plot
        self.ax.bar(self.x_data, self.y_data)[0]
```

```
        self.ax.set_axisbelow(True)
        self.ax.grid(axis = 'y')
        print(self.y_data)#just to see data in terminal
        self.ax.set_xlim(1, 65)
        self.ax.set_ylim(0,1)
        self.ax.set_xticks(np.arange(len(self.x_data))+1)
        self.ax.set_xticklabels(self.x_data)
        self.ax.title.set_text('Normilized intensity x pixel')
        self.ax.set_xlabel('Pixels')
        self.ax.set_ylabel('Intensity')

        self.canvas1.draw_idle()  # redraw plot


    #actions for the buttons  (core of the program)

    def actionFind(self):
            #associated with the button Find
        self.serialList() #see funtion serialList



    def actionConnect(self):
            #associated with the button connect. make the conetion with the selected USB, to be used after actionFind
        if(self.ports !=0): #if there is USB connection
            self.ports_value = self.listPorts.get()#changing port name to current USB
            self.varPortConnected.set("Connected") #status = connected
            self.labelPortConnected.grid(row=1, column = 3)
              self.serialUSB.change_port(self.ports_value) #changing the port in the USB function program



    def action_readfilename(self):
            #find the .bin program (program that will be sent), activate with button start
          inputValue=self.textBox.get("1.0","end-1c")#program name (a string written in the textbox)
            mypath=pathlib.Path(__file__).parent.resolve()#find current path (there are some diffenrences between Linux and windows)
          onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]#getting all files in path
        i=1
        while(i<len(onlyfiles)):
            if inputValue==onlyfiles[i]:#if we find the .bin program
                self.varPortConnected.set("File Found")#printing in the interface
```

```
            self.labelPortConnected.grid(row=2, column = 2)
            self.found=True#changing variables controls
            self.running=True
            return
          i+=1
        self.varPortConnected.set("FIle not Found")#if we do not find, print this
        self.labelPortConnected.grid(row=2, column = 2)


    def interfacage(self):
            #main function in the program. It runs it self every second and once every variables are set, send the program
            #to the NUCLEO
        if (self.found==True):
            if (self.running==True):
                if (self.envoye==False):#if we did not send the file to start l'asservissement
                    fl=self.textBox.get("1.0","end-1c")
                        sr=str(pathlib.Path(__file__).parent.resolve())+'/'+fl #get the .bin file path
                    dst=r"E:"   #NUCLEO directory
                    self.serialUSB.send_file(sr,dst)#send file (same as drag)
                    self.envoye=True
                   intensite=self.serialUSB.test_nucleo()#on appele la function test_nucleo() de la classe SerialUSB_NUcleo, where we read the NUCLEO output (in this application 65 numbers )
                self.animate(intensite)   #we plot intensite in an histogram
            else:#if the variables are not set, idle state
                self.varPortConnected.set("waiting for file")
                self.labelPortConnected.grid(row=3, column = 2)
        self.mainWindow.after(1000,self.interfacage) #recall the function

    def actionStop(self):
            #send another .bin file to stop all action in NUCLEO. Only if we click in stop and the program is running
        if(self.found==True):
            if(self.running==True):
                sr=str(pathlib.Path(__file__).parent.resolve())+'/'+'stop.bin'
                dst=r'E:'
                self.serialUSB.send_file(sr,dst)
                self.running=False#set every thing to initial state
                self.found=False
                self.envoye=False
```

```
    def serialList(self):
        #Read the USB connections and return them
        self.ports_value = StringVar(self.mainWindow) #string of lists to be displayed in the interface
                self.ports = self.serialUSB.serial_ports()    #calling serial_ports() from SerialUSB_Nucleo.py to read serial connections (USB)
        self.listPorts = Combobox(self.mainWindow)#creating a list in mainWIndow
        self.ports_name = []
        self.ports_number = []

        # Serial communication zone
        if(self.ports ==0): #if there is no USB connection we indicate it
            self.ports_name.append('No Serial / Click on Find Serial')
            self.ports_number.append(-1)
            self.ports_value.set(self.ports_number[0])
            self.listPorts.insert(self.ports_number[0], self.ports_name[0])

        else: #if we find a conncetion the function goes here
            for i in range(0, len(self.ports)):# in the range of available connections we add them to a list do be displayed
                #print(self.ports[i]) if we wanna print the name of the connection in the terminal
                self.ports_name.append(self.ports[i])
                self.ports_number.append(i)
                self.listPorts.insert(self.ports_number[i], self.ports_name[i])
                # self.listPorts.current(-1)
        self.listPorts.grid(row=2, column = 0)# list postiton

    def run(self):
        #call interfacage function when  phyton __exec.py
        self.mainWindow.after(1000,self.interfacage)
        self.mainWindow.mainloop()
```

**SerialUSB_Nucleo.py**

```
#26/02/2022
#João Luiz de Oliveira Pacheco IOGS 2A
#Dany Zakharia  IOGS 2A
#SerialUSB_Nucleo.py defines a class of function related to the NUCLEO-L476RG connetion, reading data and sending files
#adaptation of http://lense.institutoptique.fr/ressources-protis/, create by Professor Julien VILLEMEJANE  IOGS
```

```
# All necessary 'tools '
import time
import os
import sys
import glob #linux
import matplotlib.pyplot as plt
import serial
import shutil #copy file
import win32com.client #for windows
from serial import tools #library for serial connections
from serial.tools import list_ports

class SerialUSB_Nucleo:
#class SerialUSB_Nucleo: function __init__; change_port; serial_ports; send_file;test_nucleo
#parameter self allows to acess the attributes and methods of the class  (all variables )
   def __init__(self):
        #Creates all the variables used for create buttons and bind functions in its initial state
      self.nucleo_connected = 0
      self.serial_connected = 0
      self.nb_port = 0
      self.list_port = []
      self.serialPortSelected = 0
      self.param_fe = 0
      self.param_nbPoints = 0
      self.param_sync = 0
      self.param_updated = 0
      self.portname='aaa'


   def change_port(self, portSelected):
         self.serialPortSelected = portSelected #connect with the desire serial port "portSelected", called in acquisition.py

   def serial_ports(self):
        #Function that returns all ports available (serial connection), called in acquisition.py
      self.nb_port = 0
      ports = list(serial.tools.list_ports.comports())
      self.list_port = []

      for port in ports:
```

```
            self.nb_port += 1
            self.list_port.append(str(port))
            self.portname=str(port).partition(' ')[0]
        if(self.nb_port != 0):
            return self.list_port
        else:
            return 0


    def send_file(self,src,dst):
            #send a file located in src to dst
           if(self.nucleo_connected != 0): #checking to see if ther is a connection. Maybe we should replace the position of this part
            return 2


           serialSelected = serial.Serial(self.portname,115200,timeout=100)#port name is the USB connection, baudrate=115200 same as in .bin

        if(serialSelected.isOpen() == False):  #if connnection is not made yet
            serialSelected.open() #open connection
            self.serial_connected = 1
        else:
            self.serial_connected = 1

           shutil.copy2(src, dst)#copying file from src to dst, located in NUCLEO for this application


     def test_nucleo(self):#maybe change the name of this function to something more intuitive
        #read data from NUCLEO, if sucessful, return 64 numbers (pixel intensity)
           if(self.nucleo_connected != 0): #checking to see if ther is a connection. Maybe we should replace the position of this part
            return 2

           serialSelected = serial.Serial(self.portname,115200,timeout=100)#port name is the USB connection, baudrate=115200 same as in .bin

        if(serialSelected.isOpen() == False):    #if connnection is not made yet
            serialSelected.open()#open connection
            self.serial_connected = 1
        else:
            self.serial_connected = 1
```

```
        dataReceived = serialSelected.read(1)#read 1 byte
                    dataReceived=int.from_bytes(dataReceived,   byteorder=sys.byteorder)
#transform it to integer

         while(dataReceived!=255): #search for a specific byte indicating the begining of
the meas
           dataReceived = serialSelected.read(1) #read 1 byte
           dataReceived=int.from_bytes(dataReceived, byteorder=sys.byteorder)
        valeur=[]

        while(len(valeur)<64):#we are reading the data
                                        valeur.append(int.from_bytes(serialSelected.read(1),
byteorder=sys.byteorder)/123)    #rescaling the intensity with 1/123 because of the
way NUCLEO send it
        print(valeur)
        print("here")
        return valeur
```

**___exec.py**

```
# coding=utf-8
import acquisition as acq


# programme principal
appli_data = acq.Acquisition() #getting the class of function from acquisition.py

appli_data.run()#all that we need to do is call the function run
```

---

## Code Asservissement :

```
/* mbed Microcontroller Library
 * Copyright (c) 2019 ARM Limited
 * SPDX-License-Identifier: Apache-2.0
 */
#include "mbed.h"
#include "QEI.h"
#include "platform/mbed_thread.h"
#define n 65
#include "math.h"
```

```
Serial pc(USBTX, USBRX);           //déclaration de la liaison série --> TeraTerm
PwmOut moteur(D6);
AnalogIn capteur(A0);   //Tension délivrée par le capteur A0
PwmOut SI(D5); //SI
PwmOut Clock(D3); //Clock
//AnalogOut Clock(A1);
//AnalogOut Clock(A2);

AnalogOut  myDAC(A2);


Ticker ticker1; // Déclaration du ticker1  --> fonction Clock - Horloge

Ticker ticker2; // Déclaration du ticker2  --> fonction SI

Ticker deplalaser;

PwmOut moteur1(D9);

double rc1=0.06;
double compteur=0;
double v=1;


double fclock_min = 5000;//Hz (5kHz) --> fréquence d'échantillonage minimale indiquée dans la datasheet du capteur
double fclock_max = 5000000;//Hz (5MHz) --> fréquence d'échantillonage maximale indiquée dans la datasheet du capteur

double Tint_min = (1/fclock_max)*n;//s =200ns*64 --> Temps d'intégration minimal
double Tint_max = 0.005;//s --> Temps d'intégration maximal

double Tint =0.05;//valeur déterminée de manière empirique : ne fait pas saturer les pixels et permet de distinguer clairement le signal laser

double ts = 185;//ns (temps de montée)

int i=0; // variable globale utile dans les fonctions si_clock et pix --> permet de savoir à quel pixel on est rendu, i = 64 représentant la fin de la barette
int initialisation=0; // est initialisé le système après un parcours complet de la barette CCD : utile pour l'asservissement
```

```
double rc=0.06;
int N=30;
double k;
double alpha=(0.1235-0.02)/N;
int t;
int milieu=32;
double somme_erreurs=0;
double variation_erreurs=0;
double erreur=0;
double erreur_prec=0;
double Kd=0;
double Kp=0.1;
double Ki=0.01;
double erreur_pond=0;
int res;
double d1;
double d2;
int d;
int d3;

// VDD Alimentation 5V --> n'apparaît pas dans le code

//Acquisition des données du capteur linéaire CCD

double pixels[n]; //65=n  --> tableau accueillant les valeurs de chaque pixel en direct
double phd[n-1]; //65=n --> tableau accueillant les valeurs de chaque pixel après un tour : sauvegarde du tableau pixels précédent tous les Tint
double phd_lisse[n-1]; //65=n --> tableau accueillant les valeurs de chaque pixel en direct

void si_clock(void);   // déclaration de la fonction gérant les signaux d'horloge et SI
void pix(void); // déclaration de la fonction gérant l'acquisition des valeurs de chaque pixel
int indice_max(double tab[n-1]); // déclaration de la fonction déterminant la position du laser par la méthode du maximal d'intensité
//void affiche_phd(void);
void lisser_phd(int k); // Fonction lissant le tableau phd sur k valeurs
void affiche_phd_lisse(void); // Fonction affichant le tableau phd lissé sur Teraterm : utile pour débuguer
void deplacement(void); // Pertubation du laser
void asservissement(void);// Asservissement de la photodiode sur la position du faisceau laser

int main()
```

```
{
        moteur.period_ms(20);
        moteur.write(rc);

        moteur1.period_ms(20);
        moteur1.write(rc1);

        pc.baud(115200); // Teraterm

        ticker1.attach(&si_clock, Tint); // attribution de la fréquence d'exécution
        ticker2.attach(&pix, Tint/65); // attribution de la fréquence d'exécution --> 65 fois plus rapide
        deplalaser.attach(&deplacement, 0.1);


        while (1){      // Les fonctions s'exécutent
        }



        }


int indice_max(double tab[n-1]){ // Fonction calculant l'indice correspondant à la valeur maximale dans un tableau
        int j=0;
        int i_max=0;
        double max=tab[0];

        for(j=0;j<n-1;j++){
        if(tab[j]>max){
        max = tab[j];
        i_max=j;
        }
        }
        return i_max;
        }




void deplacement(void){//Déplace le laser-gauche-droite-gauche sur un temps “infini”
```

```
        rc1= rc1+v*alpha; // le rapport cyclique fait des pas de + ou - alpha (v = -1 ou 1 selon qu'on se déplace à gauche ou à droite )
        compteur+=v/abs(v);
        moteur1.write(rc1);
        /*wait(1);*/
        pc.printf("%lf",v);
        pc.printf("%lf",compteur);
        if (abs(compteur)==10) // si on a fait 10 pas on est au bout du chemin et il faut faire demi-tour
        {v=-v;}} // changement de sens


void asservissement(void){
        lisser_phd(6); // moyennage du tableau pour s'affranchir du bruit
        double seuil=0.35 ; // seuil de detection pour s'affranchir du bruit
        int ind = indice_max(phd);//+lisse
        erreur=(ind-32); // 32 correspond au pixel central de la photodiode
        somme_erreurs+=erreur; // Pour la correction intégrale
        variation_erreurs=erreur-erreur_prec; // Pour la correction derivée
        if(phd[ind]>seuil){
        //rc= Kp*erreur+Ki*somme_erreurs+Kd*variation_erreurs +0.06;
        rc=0.06+Kp*alpha*erreur;
        }
        erreur_prec=erreur;

        moteur.write(rc); // on applique la correction

        }




void si_clock(void){
        //pc.printf("in the si_clock\r\n");
        SI.write(1/65.0);    // Cf schéma du principe d'utilisation de la barette CCD du rapport technique (PWM "en haut" un 65 ième du temps)
        Clock.write(1/2.0); // Horloge
        i = 0;
        if(initialisation){ // Cela signifie que la barette CCD a déjà été parcouru unr fois en entier
        asservissement();   //On exécute l'asservissement
        // affiche_phd();      // Observation sur teraterm
```

```
        //affiche_phd_lisse();
        }
        }

void pix(void){
        if(i==0){
        wait_ns(ts); // Au début, on attend un temps ts correspondant au temps de montée ou temps de réponse du pixel (délai entre le signal d'horloge et la sortie)
        }
        else{
        pixels[i-1]=capteur.read(); // Lecture et acquisition du pixel i-1 dans le tableau pixels
        }
        i=i+1;
        if(i==65){  // Si on est en bout de barrette
        initialisation=1;   // On est initialisée (jusqu'à la fin)
        int j;
        for(j=0; j<65; j++)
        phd[j]=pixels[j]; //phd est la copie du tableau pixels : il est actualisé une fois toutes les 64 valeurs de pixels acquises
        //phd est donc actualisé tous les Tint

        }
        }

void affiche_phd(void){
  pc.printf("nouvelle acquisition \r\n");
  int k;
  pc.printf("[ ");
  for(k=0;k<n-1;k++) { pc.printf("%lf ", phd[k]);}
  pc.printf("]");
        }

void  affiche_phd_lisse(void){


        pc.putc(255);
        for(d3=0;d3<n-1;d3++) {
        d=int(123*phd_lisse[d3]);
        pc.putc(d);}

        }
```

```
void lisser_phd(int k){       //on lisse sur k pixel : k pair

        int j=0;
        double s_i=0;
        for(j=0;j<n-1;j++){
        s_i=s_i+phd[j];
        }
        double phd_i=s_i/k;          //Valeur initiale des tableaux (sur le bord gauche)

        j=n-2-k;
        double s_f=0;
        for(j=n-2-k;j<n-1;j++){
        s_f=s_f+phd[j];
        }
        double phd_f=s_f/k;          //Valeur initiale des tableaux (sur le bord droit)

        int l;
        for(l=0;l<n-1;l++){

        if(l<k/2){  //si on est sur le bord gauche
        phd_lisse[l] = phd_i;
        }

        if(l>=n-1-k/2){ //si on est sur le bord droit
        phd_lisse[l] = phd_f;
        }

        else{
        j=l-k/2;
        double s=0;
        for(j=l-k/2;j<l+k/2;j++){
        s=s+phd[j];
        }
        double phd_l=s/k;   //Valeur lissée
        phd_lisse[l]=phd_l;
        }}}
```

---

## VI. Sources


- Image de la figure 1: [etretat - Bing images]

- Explication des servomoteurs: Nucléo – Contrôler un mouvement angulaire – LEnsE (institutoptique.fr)