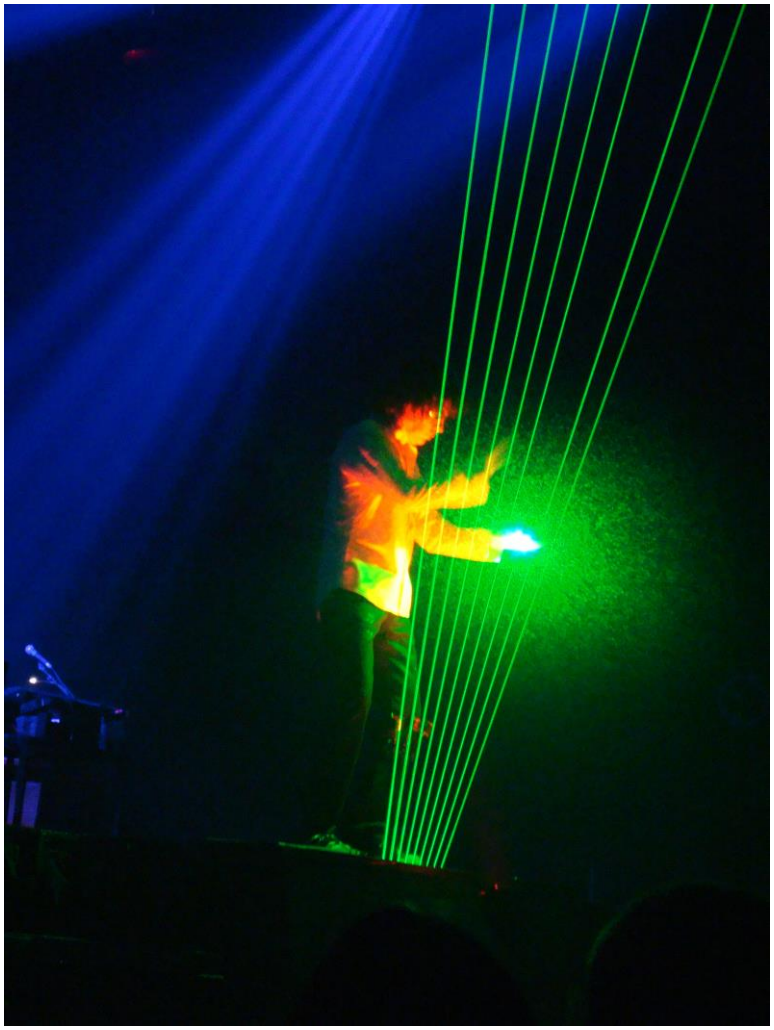
INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

Projet IéTI – Première année – S6

Laser harpe

Pinoche Bérengère – Seurat Rémy – Wei Xinyue



Introduction

The goal of this project is to create a digital musical instrument - Laser Harp, which converts intermittent light into music through lasers and photoelectric receivers. The project is mainly composed of hardware and software. The hardware part uses Nucleo development board, laser and photoelectric receiver, and the software part uses Python and Keil for data processing and music generation. This project demonstrates the possibility of combining traditional musical instruments and modern technology, which has important practical significance for understanding and exploring the digital transformation of music.

1.1 The work group

1.2 Spécifications

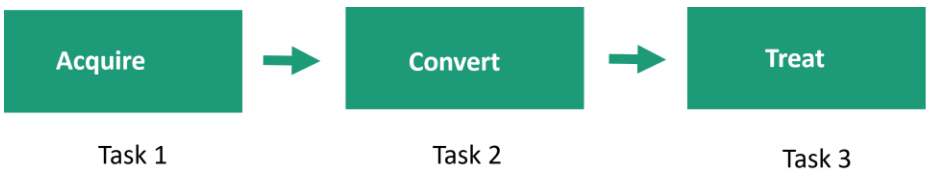
In this experiment, our goal is to build a functional laser harp and understand how it works. We'll build a system that includes a laser transmitter, photoelectric receivers, and a bunch of electronics. This laser harp will be used to trigger musical sounds, simulating "playing" by interrupting the laser beam.

Function	Sector	Standard of appreciation	Standard	Flexibility
FP1 : Create a sound and light show	Security	Audio	In a sound level not harmful to humans (>70dB)	F1
		Visual/laser	Avoid reflections (parasitic and in the direction of the public)	F0
	Alimentation	Laser	P owering the laser at 3V	F0
FC1 : Processing of acquired and transmitted data	Acquisition	Visual	Realize a photo detection system with a SFH 206 photodiode with a resistance of 10 kΩ	F0
		nucleo map	Processing of the analog signal	F0

			whose amplitude is between 0 and 3.3V	
	Treatment	Nucleo map	Emission of a square wave signal with an amplitude between 0 and 3.3V and a frequency corresponding to the note of the desired string (<20HZ and >20 000HZ)	F0
		Filter	Convert the square wave signal into a sinusoidal signal	F0
		Amplifier	Amplify the signal to a human audible signal	F0
FC2 : Have a nice aesthetic	aesthetic	Laser	Seeing laser beams through smoke	F2
		Structure	Compacting the electrical system	F2
		Audio	To have pure musical notes	F1
		Operation	Have a dark space	F2
FC3 : User interface	Communication	Operation	Clear and easy to use	F1
		Audio	Ability to move up and down the frequency range with a rotary knob	F2
		Audio	Be able to hold a note without having your finger in the beam with a push button	F2
		Audio	Use a button to decide whether the harp can play 5 notes or 7	F2

REMARK: rather than trying to create a system ourselves to convert an electrical signal into a sound signal (Function FC1 : treatment), we chose to write a program based on Python's PyGame module, which makes it much easier to generate sounds.

1.3 Functional scheme



Functional scheme

Task 1: Build the laser harp frame: Build the physical structure of the laser harp, arrange the positions of the laser transmitter and photoelectric receiver. This creates a laser beam that can "play".

Circuit Design: Using microcontrollers and other electronic components, design and build a circuit system. When the photoelectric receiver detects that the laser beam is interrupted, this circuit can send a signal to the sound system to generate a corresponding sound.

Task 2: Programming the Microcontroller: Code is written and uploaded to the microcontroller that is used to control the behavior of the laser harp. The code should take the input from the receiver and convert it into a signal that Python can understand.

Task 3: Develop a program based on Python's PyGame module to generate sounds

Function 1: acquire

The electronic assembly of the laser harp is mainly based on photodetection systems whose state (illuminated or not by the lasers) is permanently monitored by a Nucléo card. A switch is also used to select the pitch of the scale to which the notes to be played belong. More precisely, the components used are:

- 1 Nucleo card

- 8 SFH 206 photodiodes

- 2 multi-beam lasers illuminating the 8 photodiodes

- 11 resistors of 10 kΩ

- 1 key switch

- 1 button with the dimensions of the axis of the switch

The button allows to easily select the different positions of the switch once screwed on the axis of the switch (cf. figure 3).

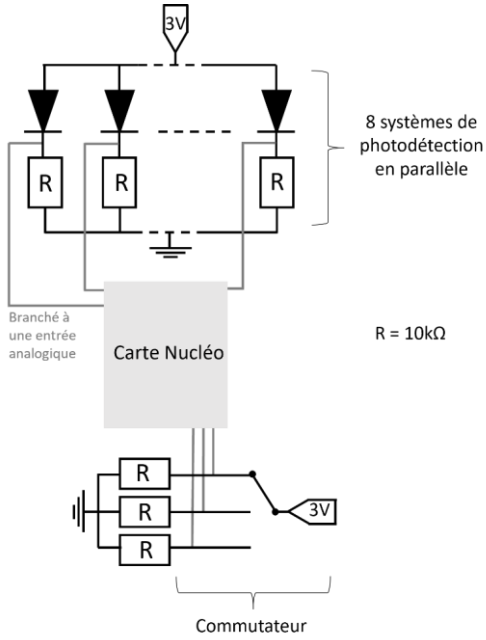


Figure 1: wiring diagrams

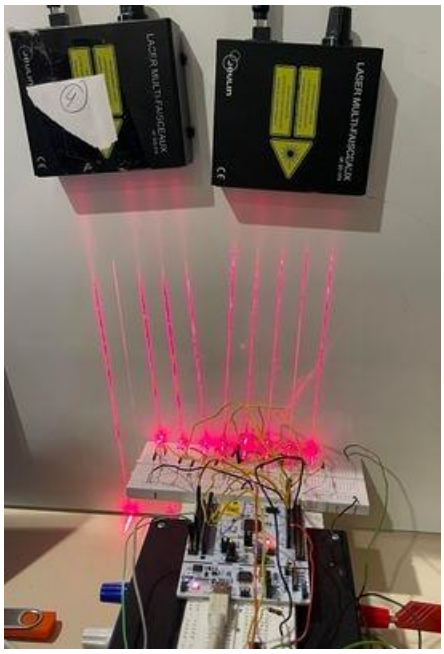


Figure 2: laser harp overview

Figure 3: button screwed on the switch

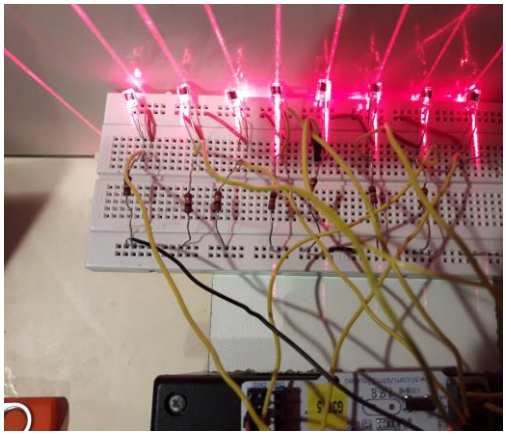


Figure 4: photodetection systems

Function 2: convert

A program written in C language on Keil Studio and then transferred to the Nucleo card allows to store in variables the state of the different photodiodes. In concrete terms, the Nucleo card continuously acquires the value of the voltage across each photodiode. When the voltage at the terminals of one of them is higher than a certain threshold, this means that the photodiode is illuminated by the laser. The user is not playing the note corresponding to this photodiode. The variable describing its state is then zero. On the other hand, if the user places his finger in front of a photodiode to play a note, the voltage at its terminals is zero. It is thus lower than the threshold defined previously and the variable describing its state then takes the value corresponding to the number of the photodiode (if the user places his finger in front of the sixth photodiode, this variable takes the value six).

```
# Ouverture de la classe
controleur = HarpeSound()

# Initialisation de la harpe laser. Cette étape est longue mais n'est réalisé qu'une seule fois
controleur.init()
if __name__ == "__main__":

    # Nom du périphérique série
    serial_port = 'COM5'  # Windows

    # Bauds
    baud_rate = 115200

    ser = serial.Serial(serial_port, baud_rate, timeout=1)

    # Assurez-vous que le port série est ouvert
    if not ser.is_open:
        ser.open()

    try:
        while True:
            # Lire les données d'un microcontrôleur
            data_received = ser.readline().decode("utf-8").strip()
            if data_received:
                print(f"Received: {data_received}")
                signal_array = parse_signal(data_received)
                # Emission du son correspondant aux donnée du microcontroleur
                controleur.sound_creatorcombinaison(signal_array)

            # Latence pour le traitement des données
            time.sleep(0.1)

    # expression de l'interruption du programme dans le shell
    except KeyboardInterrupt:
        print("Le programme a été interrompu")

    finally:
        # Fermez le port série
        ser.close()
```

For the conversion between nucleo card and python, we need to reliate the electronic assembly and the computer. First at all, we need to open the correct serial port to receive the values which are sending by the nucleo card. Then, we check that the device is turned on and we convertize the values in a list.

Function 3: treat

To process the data, we import the pygame.mixer. We tested many others and chose this one which allows to process audio files. The link for the

```
# Bibliothèque utilisée
import pygame
import numpy as np
import serial
import time

def parse_signal(signal):
    return [int(x) for x in signal.split()]

# Définition d'une classe
class HarpeSound:

    # Initialisation des variables
    sampleTimeSec = 1
    samplerate = 44100
    sampleMaxvalue = int(sampleTimeSec*samplerate)

    def __init__(self):
        pygame.mixer.init(frequency=44100, channels=2, size=32, buffer = 4096)

    # Ouverture des fichiers audios
    def init(self):
        self.soundA4 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/A4.wav')
        self.soundB4 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/B4.wav')
        self.soundC4 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/C4.wav')
        self.soundD4 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/D4.wav')
        self.soundE4 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/E4.wav')
        self.soundF4 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/F4.wav')
        self.soundG4 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/G4.wav')
        self.soundA3 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/A3.wav')
        self.soundB3 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/B3.wav')
        self.soundC3 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/C3.wav')
        self.soundD3 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/D3.wav')
        self.soundE3 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/E3.wav')
        self.soundF3 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/F3.wav')
        self.soundG3 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/G3.wav')
        self.soundA5 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/A5.wav')
        self.soundB5 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/B5.wav')
        self.soundC5 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/C5.wav')
        self.soundD5 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/D5.wav')
        self.soundE5 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/E5.wav')
        self.soundF5 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/F5.wav')
        self.soundG5 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/G5.wav')
        self.soundC6 = self.openWaveFile('C:/Users/Bérengère/Documents/IOGS/1A/S6/TP CETI/C6.wav')
```

documentation is : https://www.pygame.org/docs/ref/mixer.html#pygame.mixer.Sound.

In the list received by the program, if the photodiode receives the laser beam, the corresponding position of the list is 0 (this case is removed with the if condition in the function named summation_note()), otherwise the corresponding position of the list is a number between 1 and 8 which represents the position of the laser diode. The last number in the list represents the height of the scale. We used three different scales from -1 to 1. We use a dictionary to associate this pair of numbers with a note and a scale. The use of the dictionary makes it easier to find the right audio file by reducing the number of lines of code.

```
    # Définition d'un dictionnaire qui associe chaque fichier audio à un numero de 1 à 8 pour la note et de -1 à 1 pour la hauteur de la gamme
        self.dictionnaire_sommation =
{(1,0):self.soundC4,(2,0):self.soundD4,(3,0):self.soundE4,(4,0):self.soundF4,(5,0):self.soundG4,(6,0):self.soundA4,(7,0):self.soundB4,(8,0):self.soundC5,(1,-1):self.soundC3,(2,-1):self.soundD3,(3,-1):self.soundE3,(4,-1):self.soundF3,(
5,-1):self.soundG3,(6,-1):self.soundA3,(7,-1):self.soundB3,(8,-1):self.soundC4,(1,1):self.soundC5,(2,1):self.soundD5,(3,1):self.soundE5,(4,1):self.soundF5, (5,1):self.soundG5,(6,1):self.soundA5,(7,1):self.soundB5,(8,1):self.soundC6 }

    # fonction qui permet d'additioner le son de deux fichier afin de superposer les notes
    def sommation_note(self,list):
        l=len(list)
        self.combinaison= np.zeros(HarpeSound.sampleMaxvalue).astype('float32')
        for i in range (l-1):
            if list[i]!=0 :
                self.combinaison=self.combinaison + self.dictionnaire_sommation[(list[i],list[-1])]
        return self.combinaison

    # fonction qui permet d'ouvrir les fichiers audios
    def openWaveFile(self, file):
        sound = pygame.mixer.Sound(file)
        raw = sound.get_raw()
        bytes_array = np.frombuffer(raw, dtype=np.uint8)
        float_array = bytes_array.view(dtype=np.float32)
        return float_array[0:HarpeSound.sampleMaxvalue]

    # fonction qui permet de prendre le son sommet et d'émettre le son
    def sound_creatorcombinaison (self,list):
        self.sommation=self.sommation_note(list)
        pygame.mixer.Sound(self.combinaison[0:HarpeSound.sampleMaxvalue]).play(0)
```

We chose to sum all the audio files, so there will be no delay in the sound due to the order in which the files are opened. In addition, we decided to write the code using classes in order to reduce the number of lines of code and to avoid having to reopen the audio file every time the computer processes a new list. For this reason, the initialization of the program takes time (about 1 minute) but it is not necessary to open the audio file again afterwards which reduces the time of data processing.

Conclusion

In this project, we have successfully implemented a laser piano system, which can receive and process data from Nucleo development board and play corresponding audio according to these data. This is achieved by writing a code that converts the received data into music notes. Then, using an efficient programming construct of classes and dictionaries, these notes are preloaded and played quickly on demand.

Although our system takes a certain amount of time during the initialization phase, in the actual running phase, since all audio files have been preloaded, the response speed of playing audio is very fast, with almost no delay. This design ensures that users can get fast and accurate musical feedback when using our laser piano.

Given the final state of the prototype, three areas for improvement remain. First, the sound quality is not perfect. Indeed, when the user plays a note without interruption, brief and repeated cuts at regular time intervals are heard. However, we think the sound is making by the opening and the closing of the sound channel in pygame. The library pygame shows limit and we need to find a better to solve the problem. Second, we didn't have time to integrate the pedal into our system. Finally, the aesthetics of our prototype can certainly be improved.