Année Universitaire
2017-2018

Musique contrôlée

Rapport de projet - EI4 SAGI

Projet réalisé par :
Nathan Dubernard
Alex Charbonnier

Projet encadré par :
Jean-Baptiste Fasquel

IstiA
ÉCOLE D'INGÉNIEURS

université angers

Remerciements

Nous souhaitons remercier l'ensemble des personnes ayant de près ou de loin contribué à l'élaboration de ce projet. Nous remercions tout particulièrement M. Jean-Baptiste Fasquel pour nous avoir proposé ce sujet et pour nous avoir accompagnés et conseillés tout au long de notre travail. Enfin, nous remercions l'ISTIA ainsi que le LARIS pour nous avoir mis à disposition des moyens et du matériel nécessaire au bon déroulement de notre projet.

Table des matières

Introduction

Depuis 1897, date de l'invention du Telharmonium premier instrument de synthèse musicale, la technologie a peu à peu fait sa place dans le monde de la musique jusqu'à devenir indispensable dans les créations d'aujourd'hui. De l'analogique, les instruments de synthèse ont vécu une transition vers le numérique. De nos jours, c'est la structure de l'instrument qui évolue de l'hardware vers le software.

C'est dans cette logique de transition vers le numérique que s'inscrit notre projet de quatrième année d'école d'ingénieur. En effet, notre objectif est de développer une interface utilisant des technologies récentes afin de contrôler des sons et de jouer de la musique d'une manière différente des outils les plus connus tels que les claviers MIDI ou encore les launchpads qui ont la cote ces derniers temps.

Dans ce projet, nous avons été amenés à travailler avec l'environnement de développement Python, à explorer les possibilités d'utilisation d'outils tels que la Leap Motion et la webcam de l'ordinateur ou encore, à faire la liaison entre Sonic Pi (un synthétiseur de musique programmable) et un interpréteur python.

Nous vous présenterons dans un premier temps la gestion du son, le fonctionnement et la logique de Sonic Pi selon nos expériences personnelles. Ensuite, nous aborderons la problématique de la reconnaissance des mouvements de l'utilisateur. Enfin, nous parlerons de l'association des deux points précédents dans une interface graphique exploitable par un utilisateur.

I - Gestion du son

1- Découverte de Mido

Mido est une librairie Python permettant de travailler avec des fichiers MIDI et de gérer les ports. De plus, la librairie dispose de son pré-établi. Mido fonctionne pour à peu près toutes les versions de Python, élément qui nous paraissait important pour commencer notre projet. Mido se veut accessible de par sa facilité d'utilisation. Néanmoins cette facilité aurait pu s'avérer handicapante dans la suite du projet.

C'est dans cette optique de découverte, que ce programme nous a grandement inspiré :

```
#JB Fasquel
import cv2
import numpy as np
import mido

# MIDI part
import mido,time
print(mido.get_output_names())
out_names=mido.get_output_names()
outport = mido.open_output(out_names[0])

# GUI part
capture = cv2.VideoCapture(0)
while(capture.isOpened()): #e.g. cam unplugged
    ret, image = capture.read()
    if ret:
        image_gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY).astype(np.uint8) #BGR
because opencv images are BGR and not RGB
        val=np.mean(image_gray)
        print(val,np.max(image_gray),np.min(image_gray))
        val_note=int(np.round(val*127/255))
        outport.send(mido.Message('note_on', channel=0, note=val_note, velocity=100,
time=0))
        cv2.imshow("Video", image)
        key=cv2.waitKey(8)
        if key == 27: #27: Escape key
            break #On sort de la boucle de lecture/affichage
    else:break

#Release everything if job is finished
capture.release()
cv2.destroyAllWindows()
```

Même si nous avons arrêté Mido par la suite, le programme posait les bases de la capture par vidéo (mais nous le verrons plus précisément dans la partie consacrée à la reconnaissance du déplacement).

2- Présentation de Sonic Pi

Sonic Pi est un synthétiseur de musique programmable avec un langage proche du Python ou du Ruby. Originalement destiné à l'apprentissage du code pour les jeunes enfants, Sonic Pi est un logiciel open source sous licence de la MIT (Massachusetts Institute of Technology à Boston).

Sonic Pi

Aujourd'hui, Sonic Pi est disponible sur de nombreux OS (Systèmes d'exploitation) bien que son utilisation était à la base prévue pour être exécutée sur Raspberry Pi. De nombreuses représentations musicales en live ont été effectuées et en particulier par Sam Aaron, le fondateur du logiciel. Aujourd'hui, une communauté d'utilisateurs publie des tutoriels tels que celui disponible directement sur le logiciel (celui que Nathan a suivi) ou encore, un qui apprend à un utilisateur à composer en prenant comme exemple le titre Aerodynamic du duo français Daft Punk (celui qu'Alex a réalisé).

Exemple d'interface Sonic Pi :

3- Tutoriel de Sonic Pi (Nathan)

Comme dit précédemment, Sonic Pi présente un tutoriel en français inclut directement dans le logiciel. Les notions de bases sont expliquées pas à pas. J'ai essayé de me concentrer sur les parties du tutoriel qui me semblait indispensable pour le projet. Etant donné que le projet nécessitait d'abord la gestion de notes, j'ai commencé par approfondir la fonction **play** et ses paramètres **amp**, **pan**, **release** ...
Puis comme Sonic Pi présente des échantillons pré-enregistrés, il était intéressant d'utiliser la fonction **sample**.

D'autre atouts de Sonic Pi m'ont paru important à connaître dans l'objectif du projet. Même si nous en avons peu utilisé dans le script final, les structures de programmation semblaient essentielles. Vu que le langage de programmation ressemble à celui de Ruby, cela nous a fait découvrir d'autres environnements. J'en ai surtout retenu la manière de faire un **loop**. Voici un exemple simple de batterie fait après la fin de mon tour du tutoriel.

```
live_loop
:drum_heavy_kick do
  sample
:drum_heavy_kick
  sleep 0.375
  sample
:drum_heavy_kick
  sleep 0.125
  sample
:drum_heavy_kick
  sleep 0.75
  sample
:drum_heavy_kick
  sleep 0.25
  sample
:drum_heavy_kick
  sleep 0.25
  sample
:drum_heavy_kick
  sleep 0.25
end
```

```
live_loop
:drum_snare_hard do
  sleep 0.5
  sample
:drum_snare_hard
  sleep 1
  sample
:drum_snare_hard
  sleep 0.5
end
```

```
live_loop
:drum_cymbal_closed do
  sample :drum_cymbal_soft
  sleep 0.5
  sample :drum_cymbal_soft
  sleep 0.5
  sample :drum_cymbal_soft
  sleep 0.5
  sample :drum_cymbal_soft
  sleep 0.5
end
```

Pour voir des exemples plus aboutis, j'ai regardé ceux présents après le tutoriel. On peut ainsi voir le plein potentiel de Sonic Pi. Néanmoins, j'émettrais quelques retenues car je trouve que les exemples se ressemblent un peu. Dans l'optique d'un logiciel de création musicale je pense que ce n'est pas encore assez abouti. En revanche, en tant que synthétiseur pour notre projet, il fait très bien l'affaire.

4- Tutoriel de Sonic Pi (Alex)

En addition à ce que Nathan a pu apprendre dans le tutoriel inclu dans le logiciel, je me suis lancé sur un tutoriel retraçant étape par étape le codage du morceau Aerodynamic des Daft Punk. Au début de ce tutoriel, on y apprend à jouer des samples autres que ceux du logiciel autrement dit, des fichiers musicaux .mp3 ou encore .wav :

```
use_bpm 123

maquillage = "D:/ISTIA/Projet_Musique_Leap_Motion/samples/il-macquillage-lady.wav"

load_sample maquillage
```

Ici, j'ai chargé dans le logiciel un sample musical initialement présent dans les dossiers de mon ordinateur.

En plus de pouvoir jouer ces samples, on peut les adapter au tempo du morceau avec la fonction beat_stretch. On va donc demander au logiciel de découper notre sample en un nombre de sous parties définit en fonction de la durée du sample et de la donnée BPM (battements par minutes) de notre morceau. Ici, notre sample fait environ 7,8 secondes, ce qui correspond à 16 battements à un tempo de 123 BPM :

```
define :sample_chunk do |what, beat, dur, delay|
  beat = beat / 16.0
  dur = dur / 16.0
  sample what, beat_stretch: 16, start: beat, finish: beat + dur
  sleep delay
end
```

Une fois le sample découpé selon le tempo, on peut désormais jouer le sample et le découper tout en restant dans le rythme grâce à la fonction suivante :

```
sample_chunk(maquillage, 0.0, 0.5, 0.5)
sample_chunk(maquillage, 2.5, 1.0, 0.25)
sample_chunk(maquillage, 3.5, 0.5, 0.75)
sample_chunk(maquillage, 0.0, 0.5, 0.5)
sample_chunk(maquillage, 8.5, 0.5, 0.5)
sample_chunk(maquillage, 2.5, 1.0, 1.0)
sample_chunk(maquillage, 7.5, 0.5, 0.5)
```

Dans cette fonction, on a quatre paramètres qui sont : le sample qu'on veut jouer, le temps sur lequel on veut commencer le sample, la durée de lecture du sample et enfin, le temps de pause après lecture de cet extrait.

Enfin, en complément du tutoriel du logiciel, celui-ci nous apprend à créer des phases sur un morceau en entrant nos notes dans une matrice puis en l'associant à une boucle imbriquée :

```
live_loop :solo do
  use_synth :zawa
  use_synth_defaults attack: 0.05, sustain: 0.15, release: 0.125

  phases = [
    [:D4, :Fs3, :B3, :Fs3],
    [:D4, :Gs3, :B3, :Gs3],
    [:G4, :B3, :E4, :B3],
    [:E4, :A3, :Cs4, :A3],

    [:D4, :Fs4, :B3, :Fs4],
    [:D4, :Gs4, :B3, :Gs4],
    [:G4, :B3, :E4, :B3],
    [:E4, :A3, :Cs4, :A3],
  ]

  phases.each do |notes|
    4.times do
      notes.each do |n|
        play n
        sleep 0.25
      end
    end
  end
end
```

II - Reconnaissance du déplacement

1- Adaptation du programme intrusion

Pour reconnaitre le déplacement sur une image, nous avions vu la méthode par “intrusion”. Le programme complet (avec Sonic Pi) se trouve en Annexe 1. Dans cette partie nous nous concentrerons uniquement sur la partie reconnaissance du mouvement.

Voici la partie de l’Annexe 1 qui nous intéresse ici :

```
current_gray = cv2.cvtColor(current_bgr,
cv2.COLOR_BGR2GRAY).astype(np.int16)
result = np.abs(current_gray - previous)
result = np.where(result > seuil, 1, 0).astype(np.uint8)
previous = np.copy(current_gray)
somme_points_blancs = np.sum(result)
taux = float(somme_points_blancs) / float(somme_points)

if taux > tx:

    x, y = np.where(result)
    x_c, y_c = np.mean(x), np.mean(y)
```

Cette méthode utilise la librairie **OpenCV** via l’import cv2. On commence par changer l’image en nuance de gris avec **cvtColor**. Cela va nous permettre de seuiller l’image grâce à **np.where**. Le seuil se situe à 30. Tout ce qui se situe au dessus devient noir sinon ça devient blanc. On pourra alors comparer avec l’image précédente (**previous**). Pour cela, on calcule le pourcentage de points blancs par rapport au total de points. Si le taux est supérieur à 0.01 alors on peut repérer le barycentre des changements grâce à np.where.

Cette méthode de reconnaissance présente un défaut assez important dans le cadre de notre projet. Le codage par “intrusion” n’est pas assez précis pour détecter une position particulière. En effet, vu que nous voulons un repérage précis de nos mouvements, ce type de codage ne peut pas nous servir sur le long terme.

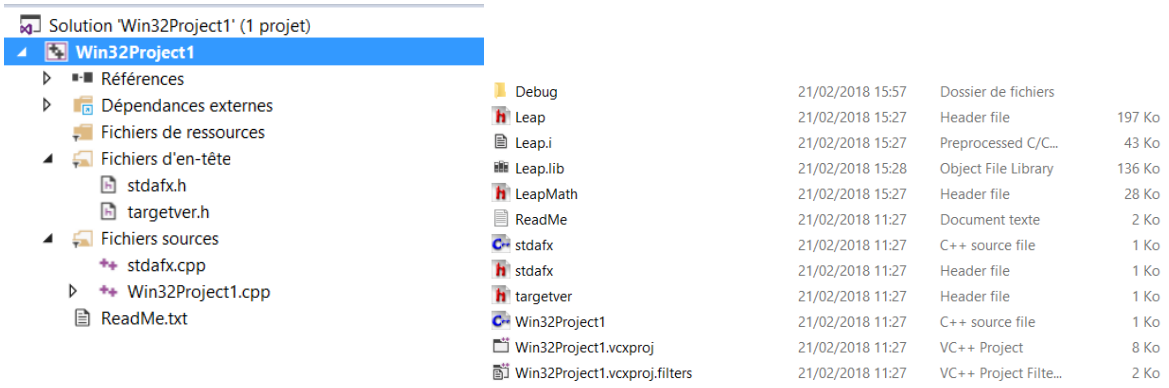
Nous avons donc décidé de changer de méthode de reconnaissance. Après plusieurs idées, nous nous sommes accordé sur la reconnaissance par couleur.

2- Leap Motion & SWIG

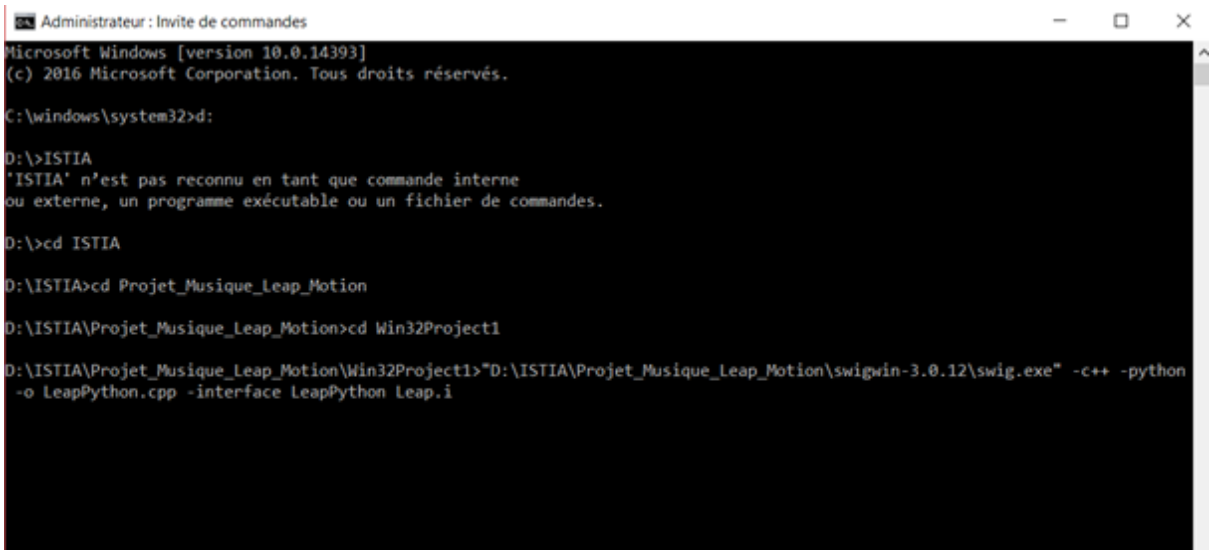
Voulant travailler avec la Leap Motion, notre première problématique a été de rendre son utilisation possible par Python. La Leap Motion est un capteur traqueur de mouvements capable de détecter les variations de l'orientation des deux mains ainsi que de l'inclinaison des phalanges.

Étant un hardware fonctionnant en C++, notre première mission était de rendre possible l'utilisation de la Leap Motion avec un programme en Python. En outre, la dernière librairie de la Leap Motion n'est disponible que pour la version 2.7 de Python, or, nous travaillons sur Python 3. Il nous faut donc procéder autrement, et pour cela, nous nous essayons au connecteur de librairies SWIG.

En effet, SWIG est couramment utilisé pour connecter des logiciels et des librairies en C++ et des logiciels/librairies en Python et autres langages informatiques. Pour cela, on se doit de créer un projet C++ avec Visual Studio dans lequel on copie les librairies de la Leap Motion :



Puis on exécute SWIG depuis la racine de ce projet grâce à l'invite de commande :



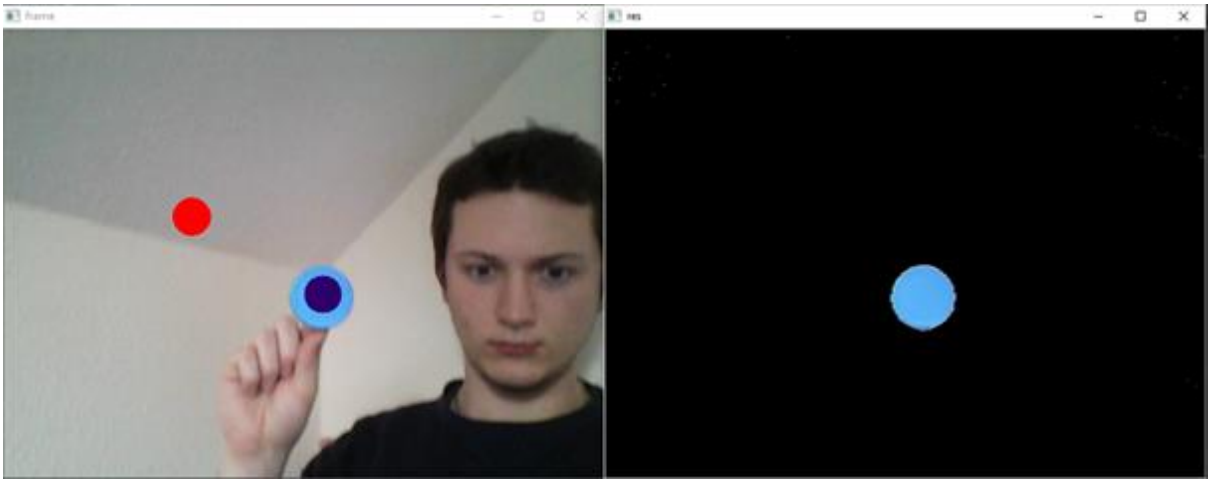
Malheureusement, à ce stade de la manipulation, nous nous retrouvons constamment confrontés à un problème de Windows que nous n'avons pas réussi à clairement identifier :

Ayant passé beaucoup de temps sur SWIG et la Leap Motion, nous décidons de laisser de côté cette solution pour nous concentrer sur l'utilisation de la Webcam.

3- Approche par couleur

On pourra retrouver en Annexe 2 le code permettant de reconnaître le bouchon grâce à sa couleur bleue.

- 1 - - 2 -



1 - Image de base avec le point bleu suivant le bouchon
2- Image avec seulement la composante bleu choisie

Pour récupérer une couleur en particulier, il faut tout d'abord la choisir ! Nous avons choisi le bouchon bleu car il présente deux particularités : c'est facile à bouger sans le cacher et sa couleur n'est pas souvent présente dans le décor. Voici donc l'intervalle de bleu que nous avons choisi :

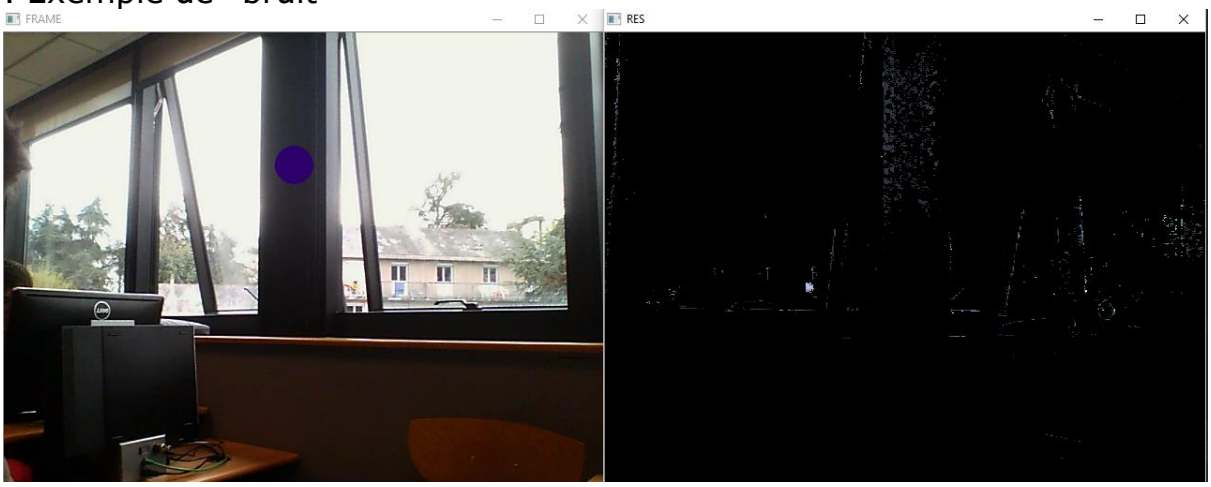
```
ower_blue= np.array([70,50,50]) # teinte basse du bleu
upper_blue= np.array([150,200,255]) # teinte haute du bleu
```

Le principe de ce code est similaire à celui précédent. En fait, au lieu de comparer les images à partir d'images seuillées, on va ajouter un masque grâce à **cv2.inRange()** pour ne garder que les points qui correspondent à l'intervalle de couleurs. Pour un souci de visualisation, on a ajouté une fenêtre avec que la composante bleu (- 2 -), **cv2.bitwise_and()**. On récupère ensuite le barycentre de tous les points bleus. On utilise ces coordonnées pour afficher un point bleu.

4- Essai de connexité

Le principal problème de cette méthode est dans le calcul du barycentre. En effet, il se trouve qu'il y a souvent du "**bruit**" de couleur. La couleur du bouchon peut se retrouver dans le décor, les vêtements … Il y a alors des points qui perturbent le calcul. De plus, la lumière rentre aussi en jeu. Avec certains éclairages tout le bouchon n'est pas repéré ce qui entraîne moins de points et donc plus d'erreurs au niveau du barycentre

. Exemple de "bruit"



Pour corriger ces défauts, l'idée de garder la plus grande composante connexe s'est révélée comme étant la solution idéale. Comme nous nous sommes déjà penchés sur ce sujet en TD, nous avons repris le code.

```
  connectivity=np.ones((3,3))
label,nb_labels=sp.ndimage.measurements.label(mask,connectivity)
hist=np.histogram(label,255)
list_of_labels=[]

for i in range(1,len(hist[0])):
    if hist[0][i]>800 : list_of_labels+=[hist[1][i]]

list_of_connected_components=[]

for value in list_of_labels:
    image_tmp=np.where(label==value,255,0)
    list_of_connected_components.append(image_tmp)
print(list_of_labels)
```

Malheureusement, après une journée de travail dessus, plusieurs problèmes sont apparus. Tout d'abord, le code ne marchait pas exactement comme nous le voulions. De plus, ce bout de code apportait énormément de latence au niveau de la vidéo ce qui est problématique pour une fonctionnalité qui se veut fluide.

Nous avons donc décidé d'abandonner ce correctif car il demandait trop de temps et qu'il fallait passer à d'autres problématiques, notamment ce qui concerne la prochaine partie, l'association entre Sonic Pi et la reconnaissance du bouchon.

III - Association son / déplacement

Maintenant que nous avons a fait un tour d'horizon des différentes solutions envisagées pour la partie son et la partie reconnaissance du déplacement, intéressons-nous à l'association des deux.

1- Intrusion + Sonic Pi

Nous avons vu dans la première partie comment le code par intrusion fonctionnait. Ici, nous allons voir comment émettre un son en fonction de la hauteur du mouvement. Voici la partie du code de l'Annexe 1 qui nous intéresse :

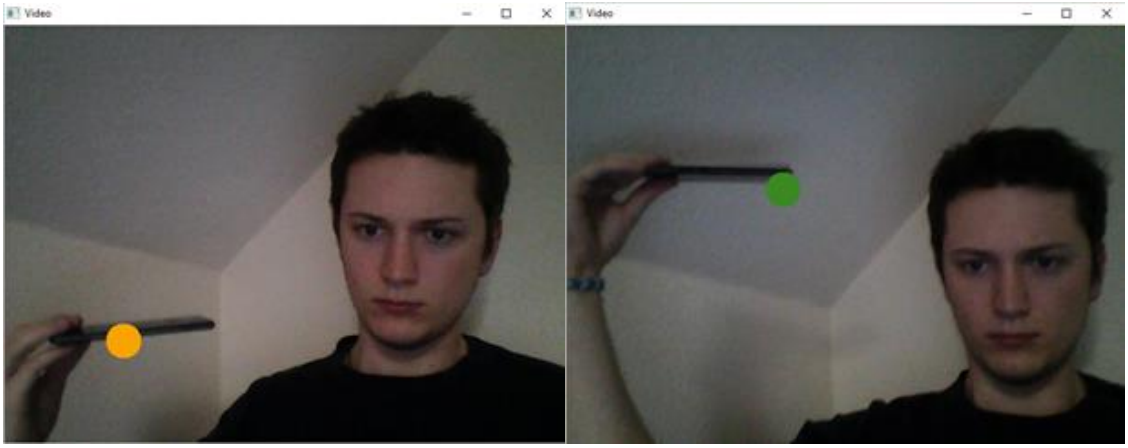
```
if (0 < x_c < 80):
            cv2.circle(current_bgr, (int(np.round(y_c)),
int(np.round(x_c))), 20, (108, 0, 46), thickness=-8,
                       lineType=8)

            if(x_compare!=1):
                play(chord(A3, MINOR), release=0.3)
                x_compare = 1
```

x_c représente la hauteur du barycentre et **y_c** la longueur par rapport au centre (0,0) en haut à gauche de la fenêtre. La fenêtre a une taille de (480,640). On découpe la hauteur en six, chacune ayant une couleur différente. **cv2.circle** permet d'afficher un cercle au niveau du barycentre.

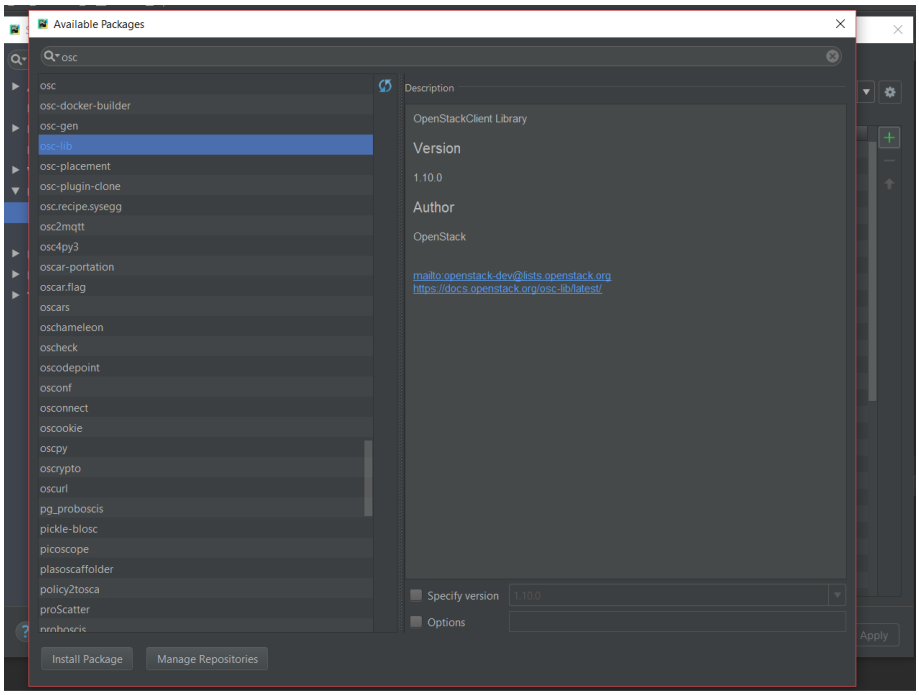
Le **x_compare** permet de savoir si la note a déjà été joué sur la bande horizontale pour la jouer qu'une seule fois. Celui-ci change pour chaque bande.

Pour faire le lien avec Sonic Pi, on va utiliser la librairie **psonic**. Cette librairie permet de pouvoir utiliser directement les fonctions de Sonic Pi. La fonction **Play** va jouer l'accord La 3 mineur lorsque le barycentre se trouve dans la bande supérieure.



2 - Utilisation d'OSC

Dans le monde de la musique, de nombreux appareils sont utilisés pour générer des lignes musicales et le format le plus couramment utilisé est le format MIDI. L'évolution de ce format est le format OSC (Open Sound Control) qui contrairement à MIDI, n'est pas codé en informatique de bas niveau mais en langage interprétable. La transmission d'OSC se fait quant à elle via Ethernet.
Afin d'utiliser le format OSC et de transmettre ces informations à Sonic Pi, nous devons installer la librairie dans l'interpréteur Python :



Afin de transmettre un signal OSC via notre interpréteur python vers Sonic Pi, on insère une live loop qui agit comme un listener pour OSC et qui déclenche l'envoi d'informations OSC vers Sonic Pi :

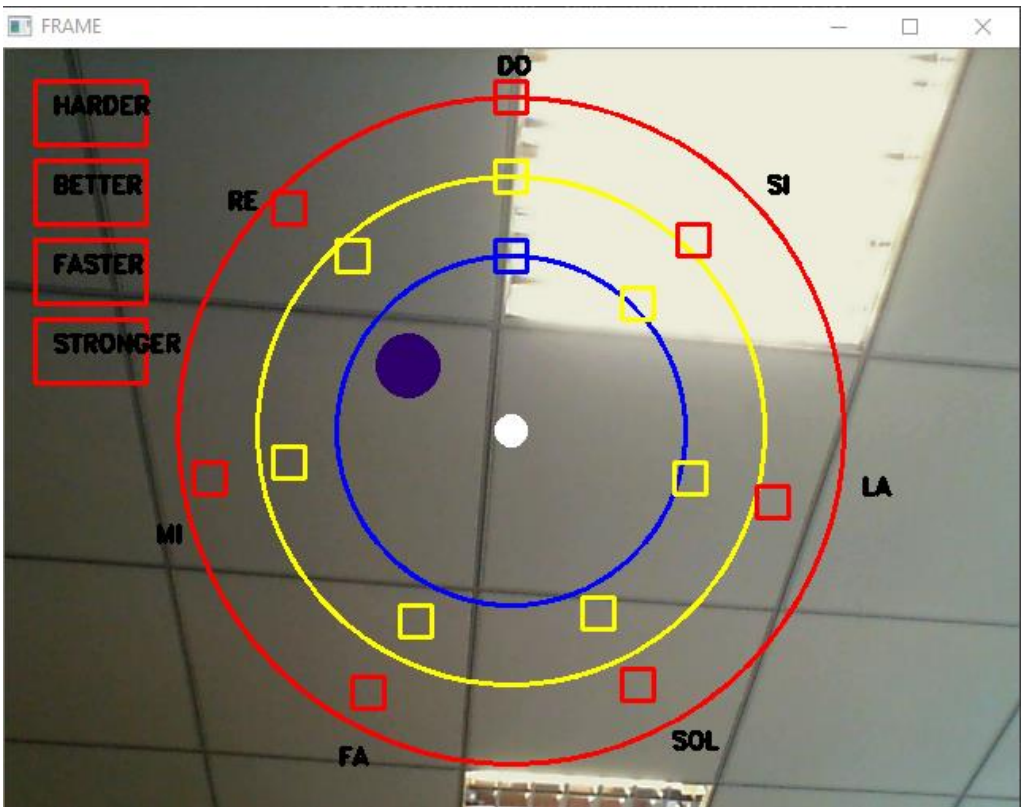
```
live_loop :foo do
  use_real_time
  a, b, c = sync "/osc/trigger/prophet"
  synth :prophet, note: a, cutoff: b, sustain: c
end """)




#### Declenchement du programme
send_message('/trigger/prophet', 60, 100, 10.5)
```

3- Interface graphique

Notre objectif étant de jouer des notes en fonction de nos mouvements devant la caméra, il fallait trouver une interface graphique appropriée aux sept notes de la gamme. Nous nous sommes orientés vers une spirale représentant deux gammes. Plus les points se rapprochent du centre, plus les notes jouées sont aiguës. Chaque gamme a sa couleur, rouge -> Do 3 …, jaune -> Do 4 …, bleu -> Do 5.



Pour créer l'interface, nous avons fait plusieurs fonctions. Etant donné que l'interface est composée uniquement de cercle et de rectangle, nous avons créé une fonction permettant de désigner juste les coordonnées et la couleur.

```
def rectangle(p1,p2,color):
    cv2.rectangle(frame, p1, p2, color, thickness=2, lineType=1)

def text(note,point):
    txt=cv2.putText(frame, note, point, 2, 1 / 2, (0, 0, 0), 2, cv2.LINE_4)
    cv2.flip(txt, 1)

def cercle(centre,rayon,color,thickeness):
    cv2.circle(frame, centre, rayon, color, thickness=thickeness, lineType=2)
```

Dans la fonction text( ), on peut voir qu'il y a une fonction appelée cv2.flip( ). En effet, lorsque la caméra projette sur l'écran, l'image est inversée. Par exemple, lorsque l'on faisait un mouvement vers la droite, nous pouvions voir un mouvement vers la gauche sur l'écran. Il a fallu corriger cela avec un cv2.flip( ). Malheureusement cela modifiait aussi les textes d'où l'appel de la fonction pour inverser de nouveau uniquement les textes.

Une fois ces fonctions fonctionnelles, nous avons créé trois autres fonctions créant l'interface graphique, une pour la spirale, une pour les boutons des Daft Punk et une pour afficher les textes des notes. Voici un exemple pris de l'Annexe 3 :

```
def AffichageSpirale():

    cercle((320, 240), 10, (255, 255, 255),-5)
    cercle((320, 240), 110, (255, 0, 0),2)
    cercle((320, 240), 160, (0, 255, 255),2)
    cercle((320, 240), 210, (0, 0, 255),2)

    rectangle((310, 20), (330, 40), (0, 0, 255))
    rectangle((450, 90), (470, 110), (0, 0, 255))
    rectangle((500, 260), (520, 280), (0, 0, 255))
    rectangle((400, 395), (420, 415), (0, 0, 255))
    rectangle((230, 390), (250, 410), (0, 0, 255))
    rectangle((145, 275), (165, 295), (0, 0, 255))
    rectangle((195, 110), (215, 130), (0, 0, 255))
    rectangle((310, 70), (330, 90), (0, 255, 255))
    rectangle((410, 120), (430, 140), (0, 255, 255))
    rectangle((450, 250), (470, 270), (0, 255, 255))
    rectangle((370, 350), (390, 370), (0, 255, 255))
    rectangle((255, 345), (275, 365), (0, 255, 255))
    rectangle((197, 260), (217, 280), (0, 255, 255))
    rectangle((230, 150), (250, 170), (0, 255, 255))
    rectangle((310, 120), (330, 140), (255, 0, 0))


def AffichageDP( ): ….

def AffichageNotes( ): ….
```

Pour diminuer la gêne dû au bruit, nous avons décidé d'ajouter une interface permettant de savoir s'il y a trop de bruit ou pas. Ainsi lorsque le bruit est assez faible pour pouvoir jouer en toute tranquillité, une autre page s'affiche.

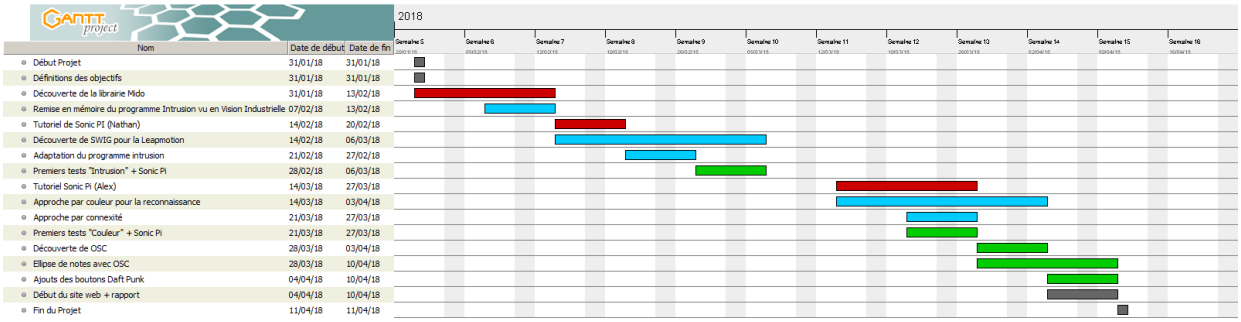
Nous avons expliqué dans la partie “Adaptation du programme Intrusion”, que l’image s’affiche si le pourcentage de points bleu est supérieur à tx. On a donc modifié ce tx pour pouvoir trouver un compromis entre le bruit moyen et le nombre de points bleu lorsque le bouchon n’est pas bien pris en compte. Du coup, voici le code lorsque ce bruit est inférieur à 0.3.

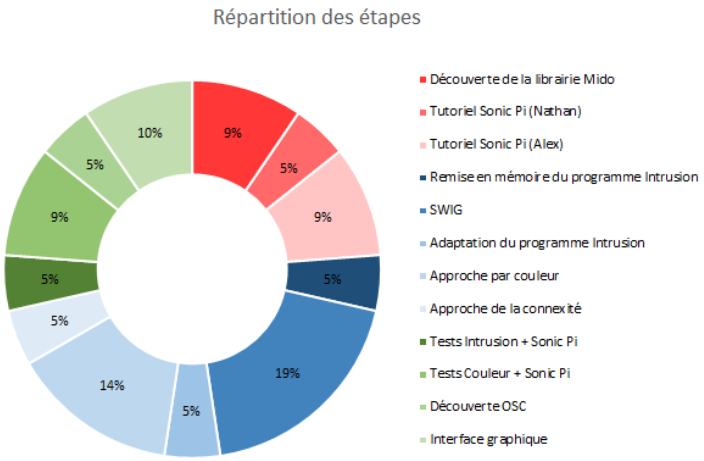
```
        else:
                frame = cv2.flip(frame, 1)
                txt = cv2.putText(frame, "Pour commencer à jouer, dispose le bouchon devant toi !", (100, 200), 2, 1 / 2, (255, 0, 0), 2, cv2.LINE_4)
                cv2.imshow('FRAME', frame)
                cv2.imshow('RES', res)
```

Bilan

Tout au long de ce projet, nous avons été confrontés à différents aspects. En effet, nous avons à la fois traité de vision industrielle ainsi que de programmation Python, mais également de notions globales et basiques de musique. Notre objectif principal, d'interagir à la fois avec la webcam et la Leap Motion n'a pas abouti mais nous avons tout de même réussi à développer une application Python qui joue des sons et des samples, grâce aux mouvements d'un utilisateur face à une webcam.

Le projet possède de nombreux axes d'amélioration en termes de technologie. Il va de soi que l'avenir de ce projet s'il est amené à évoluer, est d'intégrer la Leap Motion dans son utilisation. Pour cela, il faudra tout d'abord passer plus de temps sur SWIG puis, quand cela sera disponible, utiliser la librairie de la Leap Motion pour Python3. En outre, nous sommes convaincus qu'une telle application aurait un meilleur avenir dans un environnement en orienté objet et que Sonic Pi (bien que son utilisation soit ludique) bride le potentiel de ce projet. En changeant de logiciel musical, il sera alors possible de gagner en précision, en facilité de développement et la latence entre le code et l'interaction avec l'utilisateur s'en verra énormément réduite.





Annexes

Annexe 1 :

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from psonic import *

### Variable global ###
tx=0.01
seuil=30
x_compare=1

### Capture de la video avec la webcam ###

capture=cv2.VideoCapture(0)
ret,image=capture.read()

### on récupere la première image en niveau de gris ###

previous=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY).astype(np.int16)
somme_points=previous.shape[0]*previous.shape[1]

## Même code que le TD Intrusion

while(capture.isOpened()):

    ret,current_bgr=capture.read()

    if ret:
        current_gray = cv2.cvtColor(current_bgr, cv2.COLOR_BGR2GRAY).astype(np.int16)

        result = np.abs(current_gray - previous)
        result = np.where(result > seuil, 1, 0).astype(np.uint8)
        previous = np.copy(current_gray)
        somme_points_blancs = np.sum(result)
        taux = float(somme_points_blancs) / float(somme_points)

        if taux > tx:

            x, y = np.where(result)
            x_c, y_c = np.mean(x), np.mean(y)

        #Pour chaque bande, on joue une note via Sonic Pi et on affiche un rond d'une couleur différente

            if (0 < x_c < 80):
                cv2.circle(current_bgr, (int(np.round(y_c)), int(np.round(x_c))), 20, (108, 0, 46), thickness=-8,
                           lineType=8)

                if(x_compare!=1): # le x_compare permet de jouer qu'une seule note si l'intrusion est détectée
sur la même bande horizontale
                    play(chord(A3, MINOR), release=0.3)
                    x_compare = 1


            elif (80 < x_c < 160 ):
                cv2.circle(current_bgr, (int(np.round(y_c)), int(np.round(x_c))), 20, (255, 0, 0), thickness=-8,
                           lineType=8)

                if (x_compare != 2):
                    play(chord(B3, MINOR), release=0.3)
                    x_compare = 2
```

```
        elif (160 < x_c < 240):
            cv2.circle(current_bgr, (int(np.round(y_c)), int(np.round(x_c))), 20, (35, 137, 58), thickness=-8,
                    lineType=8)
            if (x_compare != 3):
                play(chord(C3, MINOR), release=0.3)
                x_compare = 3

        elif (240 < x_c < 320):
            cv2.circle(current_bgr, (int(np.round(y_c)), int(np.round(x_c))), 20, (13, 240, 231), thickness=-8,
                    lineType=8)
            if (x_compare != 4):
                play(chord(D3, MINOR), release=0.3)
                x_compare = 4

        elif (320 < x_c < 400):

            cv2.circle(current_bgr, (int(np.round(y_c)), int(np.round(x_c))), 20, (1, 164, 250), thickness=-8,
                    lineType=8)
            if (x_compare != 5):
                play(chord(E3, MINOR), release=0.3)
                x_compare = 5

        else:

            cv2.circle(current_bgr, (int(np.round(y_c)), int(np.round(x_c))), 20, (0, 0, 255), thickness=-8,
                       lineType=8)
            if (x_compare != 6):
                play(chord(F3, MINOR), release=0.3)
                x_compare = 6


    cv2.imshow("Video",current_bgr)
    key=cv2.waitKey(8)

    if key==27:
        break
  else:break

capture.release()
cv2.destroyAllWindows()
```

Annexe 2 :

```
import cv2
import numpy as np
import scipy as sp
from scipy import ndimage

### Capture de la video avec la webcam ###

cap = cv2.VideoCapture(0)
ret,image=cap.read()

### on récupere la première image en niveau de gris ###
previous=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
### on récupere le nombre de points ###
somme_points=previous.shape[0]*previous.shape[1]


### Variable global ###
tx=0.01
lower_blue= np.array([70,50,50]) # teinte basse du bleu
upper_blue= np.array([150,200,255]) # teinte haute du bleu


## Pour chaque frame ##
while(1):
    ret, frame = cap.read() # on récupère la frame

    rond_rouge=cv2.circle(frame,(200, 200), 20, (0, 0, 255), thickness=-8,lineType=4) # on affiche un cercle rouge ( non utilisé pour l'instant mais qui permettra de tester la détection avec un autre rond pour créer un son )

    ## Si le frame est ok ##
    if ret:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Masque pour récupérer l'image qu'avec les composantes de bleu
        mask = cv2.inRange(hsv, lower_blue, upper_blue)
        previous = np.copy(mask)

        # Permet de joindre deux tableaux, et donc de rajouter la couleur sur le masque
        res = cv2.bitwise_and(frame,frame, mask= mask)

        #On cherche à savoir si il y a assez de points bleu
        somme_points_blancs = np.sum(mask)
        taux = float(somme_points_blancs) / float(somme_points)
        if taux > tx:

            x, y = np.where(mask)  ## donne les indices dans le tableau mask du barycentre des points

            x_c, y_c = np.mean(x), np.mean(y)  # donne les coordonnées


            ##on affiche un point bleu au niveau du barycentre
            cv2.circle(frame, (int(np.round(y_c)), int(np.round(x_c))), 20, (108, 0, 46), thickness=-8,
                       lineType=8)

            cv2.imshow('frame', frame)
            cv2.imshow('res', res)

            k=cv2.waitKey(5) & 0xFF
            if k==5:
                break

cv2.destroyAllWindows()
cap.release()
```

Annexe 3 :

```
## Nouveau code ##
from psonic import *
import cv2
import numpy as np
import scipy as sp
from scipy import ndimage

### Capture de la video avec la webcam ###

cap = cv2.VideoCapture(0)
ret, image = cap.read()

### on récupere la première image en niveau de gris ###
previous = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
### on récupere le nombre de points ###
somme_points = previous.shape[0] * previous.shape[1]

### Variable global ###
tx = 0.3
lower_blue = np.array([70, 50, 50])  # teinte basse du bleu
upper_blue = np.array([150, 200, 255])  # teinte haute du bleu

def rectangle(p1,p2,color):
    cv2.rectangle(frame, p1, p2, color, thickness=2, lineType=1)

def text(note,point):
    txt=cv2.putText(frame, note, point, 2, 1 / 2, (0, 0, 0), 2, cv2.LINE_4)
    cv2.flip(txt, 1)
def cercle(centre,rayon,color,thickeness):
    cv2.circle(frame, centre, rayon, color, thickness=thickeness, lineType=2)

def AffichageSpirale():

    cercle((320, 240), 10, (255, 255, 255),-5)
    cercle((320, 240), 110, (255, 0, 0),2)
    cercle((320, 240), 160, (0, 255, 255),2)
    cercle((320, 240), 210, (0, 0, 255),2)

    rectangle((310, 20), (330, 40), (0, 0, 255))
    rectangle((450, 90), (470, 110), (0, 0, 255))
    rectangle((500, 260), (520, 280), (0, 0, 255))
    rectangle((400, 395), (420, 415), (0, 0, 255))
    rectangle((230, 390), (250, 410), (0, 0, 255))
    rectangle((145, 275), (165, 295), (0, 0, 255))
    rectangle((195, 110), (215, 130), (0, 0, 255))
    rectangle((310, 70), (330, 90), (0, 255, 255))
    rectangle((410, 120), (430, 140), (0, 255, 255))
    rectangle((450, 250), (470, 270), (0, 255, 255))
    rectangle((370, 350), (390, 370), (0, 255, 255))
    rectangle((255, 345), (275, 365), (0, 255, 255))
    rectangle((197, 260), (217, 280), (0, 255, 255))
    rectangle((230, 150), (250, 170), (0, 255, 255))
    rectangle((310, 120), (330, 140), (255, 0, 0))


def AffichageDP():
```

```
    rectangle((550, 20), (620, 60), (0, 0, 255))
    rectangle((550, 70), (620, 110), (0, 0, 255))
    rectangle((550, 120), (620, 160), (0, 0, 255))
    rectangle((550, 170), (620, 210), (0, 0, 255))



def AffichageNotes():

    text("DO", (310, 15))  # C
    text("SI", (480, 90))  # D
    text("LA", (540, 280))  # E
    text("SOL", (420, 440))  # F
    text("FA", (210, 450))  # G
    text("MI", (95, 310))  # A
    text("RE", (140, 100))  # B


## Pour chaque frame ##

while (1):

    ret, frame = cap.read()  # on récupère la frame


    ## Si le frame est ok ##
    if ret:

        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Mask pour récupérer l'image qu'avec les composantes de bleu
        mask = cv2.inRange(hsv, lower_blue, upper_blue)
        previous = np.copy(mask)

        # Permet de joindre deux tableaux, et donc de rajouter la couleur sur le mask
        res = cv2.bitwise_and(frame, frame, mask=mask)

        # On cherche à savoir si il y a assez de points bleu
        somme_points_blancs = np.sum(mask)
        taux = float(somme_points_blancs) / float(somme_points)

        if taux > tx:
            x, y = np.where(mask)  ## donne les indices dans le tableau mask du barycentre des points
            x_c, y_c = np.mean(x), np.mean(y)  # donne les coordonnées
            cercle((int(np.round(y_c)), int(np.round(x_c))), 20, (108, 0, 46),-8)

            AffichageSpirale()
            AffichageDP()

            #### SPIRALE NOTES ####

            if (x_c>20 and x_c<40 and y_c>310 and y_c<330):
                run("""
                    play :C3
                    live_loop :foo do
                        use_real_time
                        a, b, c = sync "/osc/trigger/prophet"
                        synth :prophet, note: a, cutoff: b, sustain: c
                end """)
```

```
    run("""live_loop :foo do
            use_real_time
            a, b, c, d = sync "/osc/trigger/mod_fm"
            synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
          end """)
if (x_c > 90 and x_c < 110 and y_c > 450 and y_c < 470 ):
    run("""
                    play :D3
                    live_loop :foo do
                        use_real_time
                        a, b, c = sync "/osc/trigger/prophet"
                        synth :prophet, note: a, cutoff: b, sustain: c
                end """)

    run("""live_loop :foo do
                        use_real_time
                        a, b, c, d = sync "/osc/trigger/mod_fm"
                        synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                      end """)
if (x_c > 260 and x_c < 280 and y_c > 500 and y_c < 520 ):
    run("""
                    play :E3
                    live_loop :foo do
                        use_real_time
                        a, b, c = sync "/osc/trigger/prophet"
                        synth :prophet, note: a, cutoff: b, sustain: c
                end """)

    run("""live_loop :foo do
                        use_real_time
                        a, b, c, d = sync "/osc/trigger/mod_fm"
                        synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                      end """)
if (x_c > 395 and x_c < 415 and y_c > 400 and y_c < 420):
    run("""
                    play :F3
                    live_loop :foo do
                        use_real_time
                        a, b, c = sync "/osc/trigger/prophet"
                        synth :prophet, note: a, cutoff: b, sustain: c
                end """)

    run("""live_loop :foo do
                        use_real_time
                        a, b, c, d = sync "/osc/trigger/mod_fm"
                        synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                      end """)
if (x_c > 390 and x_c < 410 and y_c > 230 and y_c < 250):
    run("""
                    play :G3
                    live_loop :foo do
                        use_real_time
                        a, b, c = sync "/osc/trigger/prophet"
                        synth :prophet, note: a, cutoff: b, sustain: c
                end """)

    run("""live_loop :foo do
                        use_real_time
                        a, b, c, d = sync "/osc/trigger/mod_fm"
                        synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                      end """)
if (x_c > 275 and x_c < 295 and y_c > 145 and y_c < 165):
    run("""
                    play :A3
                    live_loop :foo do
```

```
                    use_real_time
                    a, b, c = sync "/osc/trigger/prophet"
                    synth :prophet, note: a, cutoff: b, sustain: c
                end """)

    run("""live_loop :foo do
                    use_real_time
                    a, b, c, d = sync "/osc/trigger/mod_fm"
                    synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                end """)

if (x_c > 110 and x_c < 130 and y_c > 195 and y_c < 215):
    run("""
                play :B3
                live_loop :foo do
                    use_real_time
                    a, b, c = sync "/osc/trigger/prophet"
                    synth :prophet, note: a, cutoff: b, sustain: c
                end """)

    run("""live_loop :foo do
                    use_real_time
                    a, b, c, d = sync "/osc/trigger/mod_fm"
                    synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                end """)

if (x_c > 70 and x_c < 90 and y_c > 310 and y_c < 330):
    run("""
                play :C4
                live_loop :foo do
                    use_real_time
                    a, b, c = sync "/osc/trigger/prophet"
                    synth :prophet, note: a, cutoff: b, sustain: c
                end """)

    run("""live_loop :foo do
                    use_real_time
                    a, b, c, d = sync "/osc/trigger/mod_fm"
                    synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                end """)

if (x_c > 120 and x_c < 140 and y_c > 410 and y_c < 430 ):
    run("""
                play :D4
                live_loop :foo do
                    use_real_time
                    a, b, c = sync "/osc/trigger/prophet"
                    synth :prophet, note: a, cutoff: b, sustain: c
                end """)

    run("""live_loop :foo do
                    use_real_time
                    a, b, c, d = sync "/osc/trigger/mod_fm"
                    synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                end """)

if (x_c > 250 and x_c < 270 and y_c > 450 and y_c < 470):
    run("""
                play :E4
                live_loop :foo do
                    use_real_time
                    a, b, c = sync "/osc/trigger/prophet"
                    synth :prophet, note: a, cutoff: b, sustain: c
                end """)
```

```
    run("""live_loop :foo do
                                    use_real_time
                                    a, b, c, d = sync "/osc/trigger/mod_fm"
                                    synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                                  end """)

if (x_c > 350 and x_c < 370 and y_c > 370 and y_c < 390):
    run("""
                                  play :F4
                                  live_loop :foo do
                                      use_real_time
                                      a, b, c = sync "/osc/trigger/prophet"
                                      synth :prophet, note: a, cutoff: b, sustain: c
                                end """)

    run("""live_loop :foo do
                                      use_real_time
                                      a, b, c, d = sync "/osc/trigger/mod_fm"
                                      synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                                    end """)

if (x_c > 345 and x_c < 365 and y_c > 255 and y_c < 275):
    run("""
                                    play :G4
                                    live_loop :foo do
                                        use_real_time
                                        a, b, c = sync "/osc/trigger/prophet"
                                        synth :prophet, note: a, cutoff: b, sustain: c
                                  end """)

    run("""live_loop :foo do
                                        use_real_time
                                        a, b, c, d = sync "/osc/trigger/mod_fm"
                                        synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                                      end """)

if (x_c > 260 and x_c < 280 and y_c > 197 and y_c < 217 ):
    run("""
                                      play :A4
                                      live_loop :foo do
                                          use_real_time
                                          a, b, c = sync "/osc/trigger/prophet"
                                          synth :prophet, note: a, cutoff: b, sustain: c
                                    end """)

    run("""live_loop :foo do
                                          use_real_time
                                          a, b, c, d = sync "/osc/trigger/mod_fm"
                                          synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                                        end """)

if (x_c > 150 and x_c < 170 and y_c > 230 and y_c < 250 ):
    run("""
                                        play :B4
                                        live_loop :foo do
                                            use_real_time
                                            a, b, c = sync "/osc/trigger/prophet"
                                            synth :prophet, note: a, cutoff: b, sustain: c
                                      end """)

    run("""live_loop :foo do
                                            use_real_time
                                            a, b, c, d = sync "/osc/trigger/mod_fm"
                                            synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
```

```
                                        end """)

        if (x_c > 120 and x_c < 140 and y_c > 310 and y_c < 330):
            run("""
                                            play :C5
                                            live_loop :foo do
                                                use_real_time
                                                a, b, c = sync "/osc/trigger/prophet"
                                                synth :prophet, note: a, cutoff: b, sustain: c
                                        end """)

            run("""live_loop :foo do
                                                use_real_time
                                                a, b, c, d = sync "/osc/trigger/mod_fm"
                                                synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                                            end """)


        #### DAFT PUNK ####

        if (x_c > 20 and x_c < 60 and y_c > 550 and y_c < 620 and one1==True):
            run("""
                    use_bpm 123
                    sample "C:/Users/Nathan/Documents/GitHub/Projet-Musique/Projet-
Musique/PrEi4/samples/Harder.wav"
                    """)

            run("""live_loop :foo do
                                            use_real_time
                                            a, b, c, d = sync "/osc/trigger/mod_fm"
                                            synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                                        end """)
            one1 = False

        if (x_c > 70 and x_c < 110 and y_c > 550 and y_c < 620 and one2==True):
            run("""
                        use_bpm 123
                        sample "C:/Users/Nathan/Documents/GitHub/Projet-Musique/Projet-
Musique/PrEi4/samples/Better.wav"
                    """)
            run("""live_loop :foo do
                                                    use_real_time
                                                    a, b, c, d = sync "/osc/trigger/mod_fm"
                                                    synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                                                end """)
            one2 = False

        if (x_c > 120 and x_c < 160 and y_c > 550 and y_c < 620 and one3==True):
            run("""
                    use_bpm 123


                    sample "C:/Users/Nathan/Documents/GitHub/Projet-Musique/Projet-
Musique/PrEi4/samples/Faster.wav"
                    """)

            run("""live_loop :foo do
                                            use_real_time
                                            a, b, c, d = sync "/osc/trigger/mod_fm"
                                            synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                                        end """)
            one3 = False
```

```
        if (x_c > 170 and x_c < 210 and y_c > 550 and y_c < 620 and one4==True):
            run("""
                use_bpm 123
                sample "C:/Users/Nathan/Documents/GitHub/Projet-Musique/Projet-
Musique/PrEi4/samples/Stronger.wav"
                """)

            run("""live_loop :foo do
                                        use_real_time
                                        a, b, c, d = sync "/osc/trigger/mod_fm"
                                        synth :mod_fm, note: a, cutoff: b, sustain: c,release: d
                                      end """)
            one4 = False

        if (y_c < 550 or x_c > 210):
            one1 = True
            one2 = True
            one3 = True
            one4 = True

        frame = cv2.flip(frame, 1)

        AffichageNotes()

        text("HARDER", (30, 40))
        text("BETTER", (30, 90))
        text("FASTER", (30, 140))
        text("STRONGER", (30, 190))

        cv2.imshow('FRAME', frame)
        cv2.imshow('RES', res)

    else:
        frame = cv2.flip(frame, 1)
        txt = cv2.putText(frame, "Pour commencer a jouer, dispose le bouchon devant toi !", (100, 200), 2, 1 / 2,
(255, 0, 0), 2, cv2.LINE_4)
        cv2.imshow('FRAME', frame)
        cv2.imshow('RES', res)

    k = cv2.waitKey(5) & 0xFF
    if k == 5:
        break

cv2.destroyAllWindows()
cap.release()
```

Bibliographie/Sitographie

OpenCV :

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html

Leap Motion :

https://support.leapmotion.com/hc/en-us/articles/223784048

https://developer.leapmotion.com/documentation/python/devguide/Project_Setup.html

OSC :

http://opensoundcontrol.org/what-difference-between-osc-and-midi

Sonic Pi :

https://github.com/samaaron/sonic-pi/blob/master/etc/doc/tutorial/12.1-Receiving-OSC.md

https://aimxhaisse.com/aerodynamic-everything-en.html

https://www.geekzone.fr/2016/09/12/sonic-pi-et-le-code-devint-musique/

SWIG :

https://fr.wikipedia.org/wiki/SWIG

http://www.swig.org/

Histoire de la musique :

http://synthmuseum.com/magazine/0102jw.html

Musique contrôlée

Projet réalisé par : Nathan Dubernard et Alex Charbonnier
Projet encadré par : Jean-Baptiste Fasquel

Contrôler la musique grâce à nos mouvements dans l'espace. Telle était la mission de deux étudiants ingénieurs en 4ème année à l'ISTIA dans le cadre de leur projet. L'objectif initial de pouvoir faire de la musique grâce aux mouvements a été réussi. Deux fonctionnalités ont été étudiées : une permettant de jouer simplement des notes grâce à une interface originale et l'autre permettant de lancer directement des samples.
La reconnaissance des mouvements s'est faite grâce à la couleur de l'objet tenu face à la caméra. Concernant la musique, le logiciel Sonic Pi a été utilisé en guise de synthétiseur. Relié avec Python grâce au format de musique OSC, le logiciel a parfaitement fonctionné.

#Musique #MusiqueContrôlée #Webcam #SonicPi #Python #OSC

Controlling music with moves in space. This, was two students' mission in the context of their 4th year of engineering school project at the ISTIA. The initial goal was to make music thanks to moves and it has been succeeded. Two functionalities have been studied: a first one allowing to simply play music notes thanks to an original interface and a second one allowing to directly play samples.
The camera can acknowledge moves thanks to the color of the object standing in front of it. About music, the software Sonic Pi has been used as a synthesizer. Linked with Python thanks to OSC music format, the software perfectly worked.

#Music #ControlledMusic #Webcam #SonicPi #Python #OSC