
Projet Musique & Tech 3 : GuitAR – Rapport



ISEN

ALL IS DIGITAL!

LILLE



Remerciements :

La réalisation de ce projet a été possible grâce au soutien de plusieurs personnes que nous souhaitons ici remercier en ces quelques lignes.

Nous voudrions tout d'abord adresser notre reconnaissance à notre tuteur Monsieur Arthur Paté, pour sa patience, sa disponibilité et surtout ses précieux conseils quant à la réalisation du projet, dans toutes ses étapes, du début jusqu'à l'accomplissement de ce dernier.

Nous désirons également remercier Monsieur Paul Cambourian qui a travers sa thèse nous a inspiré certaines idées et qui a pu nous aiguiller sur des solutions à nos problématiques.

Introduction :

Dans le cadre de notre formation d'ingénieur, nous avons pu réaliser un projet d'une durée d'un mois pour terminer notre 1ère année de cycle ingénieur.

La thématique s'est articulée autour d'une option commune aux quatre membres du groupe, la musique et la technologie.

C'est de ces disciplines qu'est né le projet GuitAR, donc « Guitare en réalité augmentée ».

Les objectifs étaient tout d'abord de découvrir, acquérir et consolider des notions telles que le traitement d'image, la réalité augmentée, le développement logiciel, tout ça dans le cadre d'une application artistique et musicale.

Notre projet a eu pour but de modifier l'aspect visuel d'un instrument, la guitare dans notre cas, à l'aide de techniques de réalité augmentée (AR).

Nous avons pu poser l'hypothèse suivante : on suppose que l'identification d'une marque ou la reconnaissance de différents bois utilisés dans la conception d'une guitare ou de n'importe quel instrument, pourrait permettre à un musicien d'avoir une attente sur la sonorité, ainsi que sur le rendu vibratoire de l'instrument ; ce qui pourrait modifier sa perception de l'instrument, et même sa façon d'en jouer.

Pour grossir le trait, si un instrument présente le visuel d'une guitare habituellement utilisée pour jouer du *Metal*, un musicien pourrait ne pas s'attendre à ce que le son qu'elle va produire puisse "sonner jazz" par exemple, et il pourrait ne même pas s'y essayer.

Les programmes présentés ci-dessous doivent nous permettre de tester cette hypothèse.

L'objectif a été d'utiliser de la captation optique en temps réel du guitariste et surtout de l'instrument grâce à une webcam, de traiter l'image avec des algorithmes en langage Python et la librairie open-source OpenCV pour identifier les zones d'intérêt (le manche, les frettes, etc...). Nous permettant enfin modifier le retour visuel de l'instrument (c'est-à-dire une image modifiée visible sur un écran), soit à travers une nouvelle texture/couleur du manche, soit afficher une matrice de points sur chaque corde et frette, pour d'autres applications.

Notre rapport s'organisera en quatre grandes parties et une annexe, abordant d'abord notre rapport à la gestion de projet, notre organisation et notre rendu final.

Ensuite nous verrons comment s'est déroulée la première étape de reconnaissance du manche, nos techniques pour y parvenir et le premier rendu de ces techniques.

En découleront les différentes approches d'appropriation de cette reconnaissance, et plus particulièrement dans quelles voies nous nous sommes dirigés pour mener à bien ce projet et exploiter au maximum notre programme. Principalement la modification du retour visuel du manche, qui pourra nous aider pour notre hypothèse principale, mais également à la construction de la matrice de points sur le manche, qui sera ici plus dans un but éducatif (apprentissage de différents accords, ou encore l'objectif d'apprendre des chansons entières).

Enfin, nous conclurons notre rapport, à savoir si nous pouvons avoir une idée plus précise sur le sujet et répondre à notre hypothèse de départ. Ainsi que sur les améliorations et directions que notre projet pourrait prendre dans le futur.

Nous présenterons également une annexe en fin de rapport qui expliquera plus en détail les différentes idées et essais que nous avons réalisés et qui nous tiennent à cœur pour la suite du projet, à savoir les différentes améliorations et projections que nous envisageons pour la poursuite de notre travail.

I. Gestion de projet :

Nous avons commencé le projet le 25 mai. En premier lieu, nous avons fait de l'analyse fonctionnelle afin de définir précisément notre objectif. Nous avons fait un diagramme bête à cornes :

Notre objectif consiste à offrir une dimension visuelle à un joueur de guitare pour que son expérience ne soit pas qu'auditive.

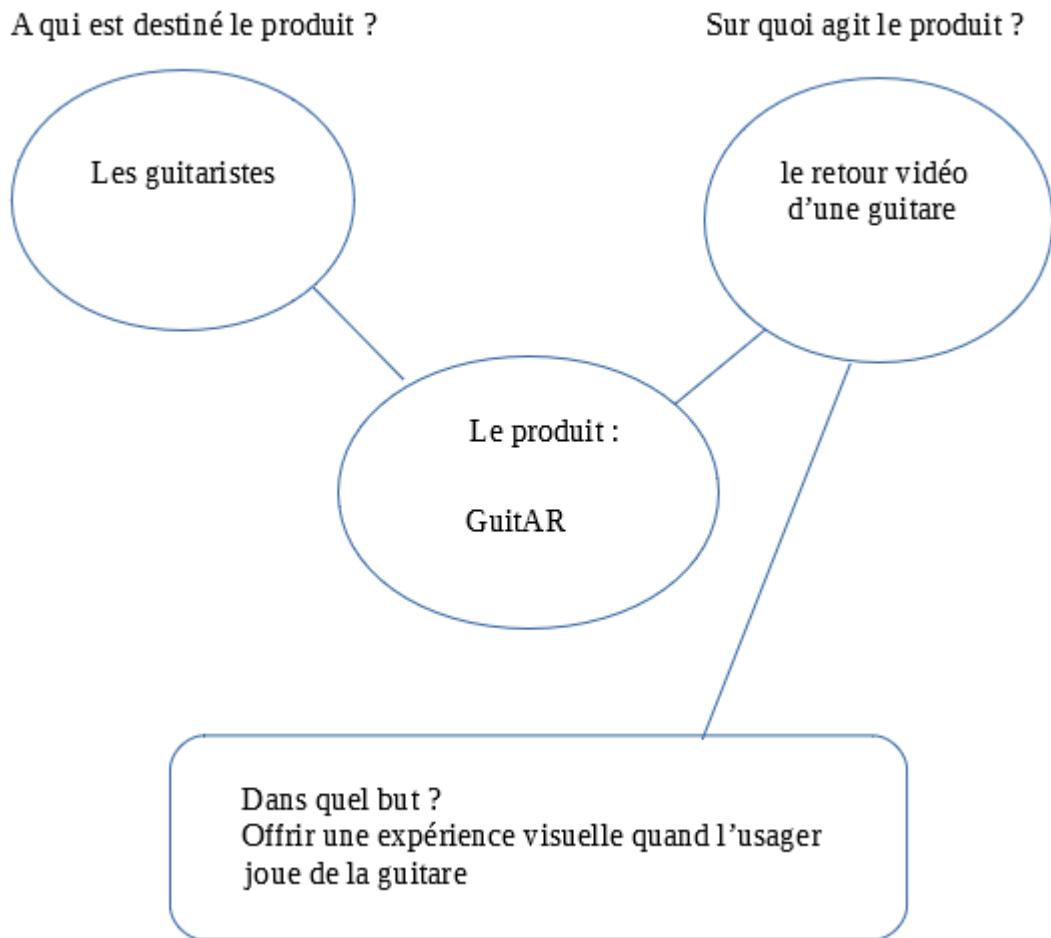


Figure 1A1 : Diagramme à tête à cornes changement de texture

Après avoir réfléchi sur notre projet, un objectif secondaire nous est apparu. Le second objectif est d'aider les débutants à apprendre la guitare en affichant des informations sur le manche.

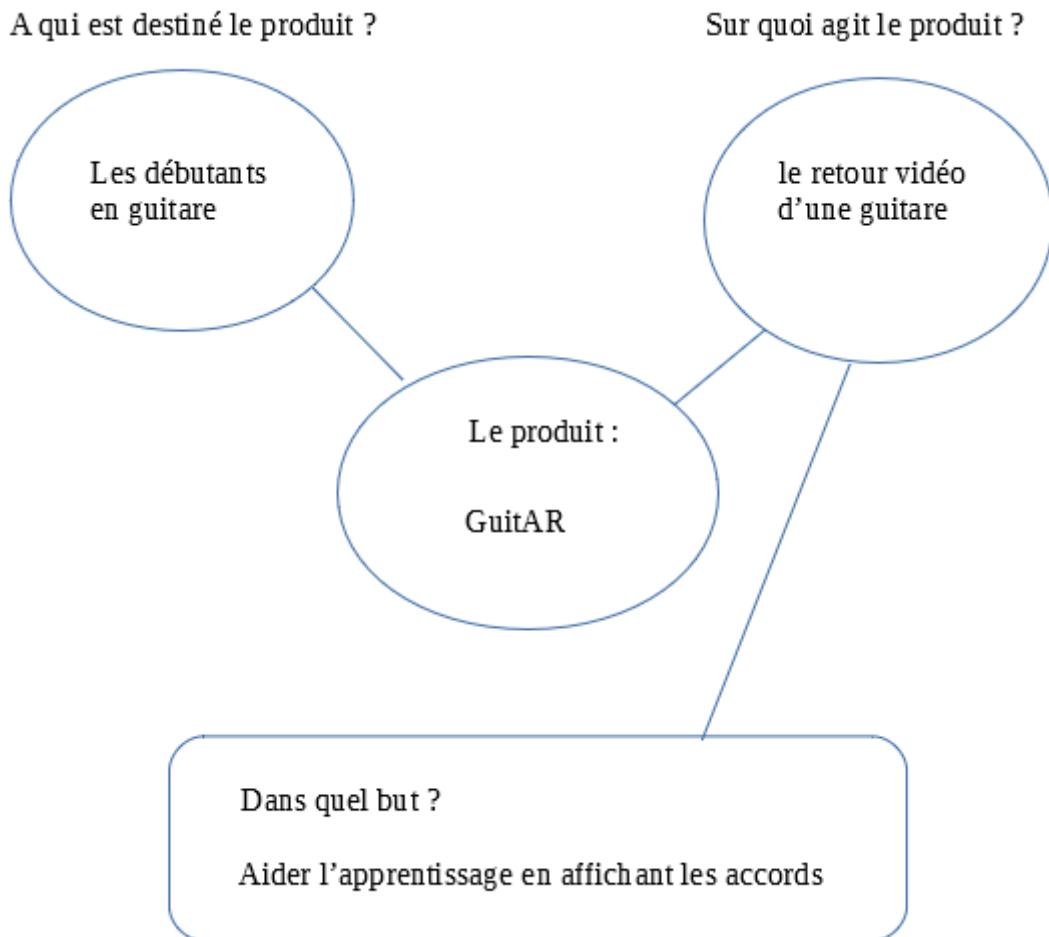


Figure 1A2 : Diagramme à tête à cornes matrice apprentissage

Nous avons décidé que ces deux objectifs étaient importants et bien que le premier fût prioritaire, nous avons aussi travaillé sur le développement du second.

Nous avons ensuite réfléchi aux différentes solutions possibles pour que le projet fonctionne. Nous nous sommes documentés et avons vu que *OpenCV* (lien en annexe) pouvait nous permettre de faire du traitement d'image en temps réel. Nous avons eu plusieurs idées pour accomplir les différentes tâches du projet que nous avons décidé de réaliser ou non en fonction des priorités et du temps qu'il nous restait. Nous avons alors établi un premier diagramme de Gantt, consultable figure 1A3, pour ne pas passer trop de temps sur des fonctions moins indispensables que d'autres.

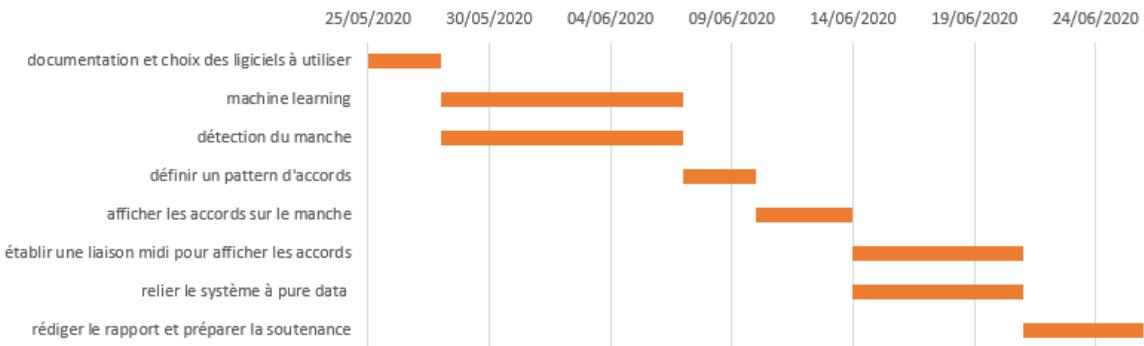


Figure 1A3 : Diagramme de Gantt initial

Selon nous, la partie la plus importante du projet est l'affichage sur le manche. C'est donc la partie où nous avons passés le plus de temps. Au bout de deux semaines, nous avions les premiers résultats de la détection du manche, nous avons eu une meilleure opinion de ce qu'il était possible de faire avec le temps imparti et ce qui était indispensable. Nous avons donc revu le planning à travers un nouveau diagramme de Gantt, que l'on peut voir figure 1A4.

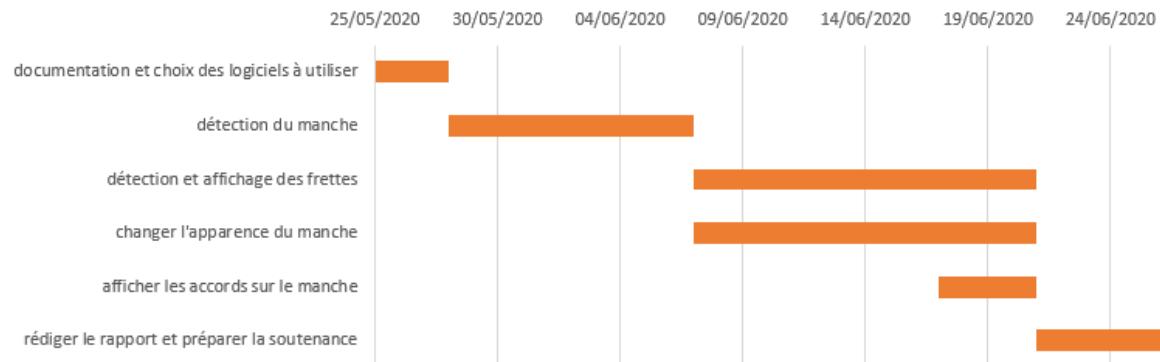


Figure 1A4 : Diagramme de Gantt final

Le projet devait être mené pendant les conditions particulières de la crise du Covid-19. En vue de respecter la distanciation sociale, les membres du groupe étaient donc amenés à travailler à distance.

Le projet a été développé sur *Python* (lien en annexe) qui nous permettait d'utiliser différentes fonctions de la librairies *OpenCV*.

Nous avons partagé nos codes via *GitHub* qui était pour la plupart une première expérience avec celui-ci. Les différentes images ou vidéo montrant l'avancée du projet ont été partagées via des lien *GoogleDrive* et via *Facebook Messenger*.

De cette manière nous pouvions suivre et comprendre l'avancée du projet correctement. Les réunions étaient au nombre de deux par semaine avec notre tuteur au début. Puis nous sommes passé à des réunions « bilan » une fois par semaine.

L'équipe s'est organisée suivant les axes de développement après la documentation, et après chaque réunion « bilan » nous rediscussions de la distribution des tâches.

À présent, commençons à nous intéresser à la partie technique principale.

II. Reconnaissance du manche :

La première étape du projet consiste à reconnaître le manche. Plusieurs choix s'offrent alors à nous : une méthode générique utilisant la reconnaissance de forme, une méthode exploitant le *machine learning* (détails de ces méthodes en annexe p.29), ou la reconnaissance de couleurs. Pour plusieurs raisons (contraintes de temps, efficacité des différentes méthodes), nous avons opté pour la troisième méthode.

L'idée est d'utiliser des marqueurs de couleurs placés stratégiquement sur la guitare. Utiliser des couleurs "atypiques", comme par exemple du vert fluo permet notamment d'être plus précis :



Figure 2A1 : Marqueurs placés sur la guitare à des endroits stratégiques

Ces marqueurs sont positionnés à des points stratégiques : au niveau du sillet et au niveau de l'intersection entre le manche et la caisse.

Il faut maintenant exploiter ces marqueurs. Pour cela, nous allons d'abord convertir notre image RGB en HSV. RGB, de l'anglais « *red, green, blue* », est un système de codage informatique des couleurs. Les écrans d'ordinateurs reconstituent une couleur par synthèse additive à partir de ces trois couleurs primaires. L'image captée par la caméra et que l'on reçoit est codée en RGB, et nous devons convertir cette image en HSV : c'est une méthode de description des couleurs qui utilise trois paramètres : teinte, saturation, et luminosité (ou valeur) (en anglais HSV pour hue, saturation, value).

Chaque paramètre permet de décrire un aspect de la couleur : la teinte est exprimée selon l'angle qui lui correspond sur le cercle des couleurs. La saturation, qui décrit l'intensité de la couleur, varie entre 0 et 100%. La valeur (ou luminosité) exprime la “clarté” de la couleur, elle varie aussi entre 0 et 100%. Un article explique très bien le codage des couleurs sur le site « <http://primatice.phpnet.org> » (voir le lien en annexe p29)

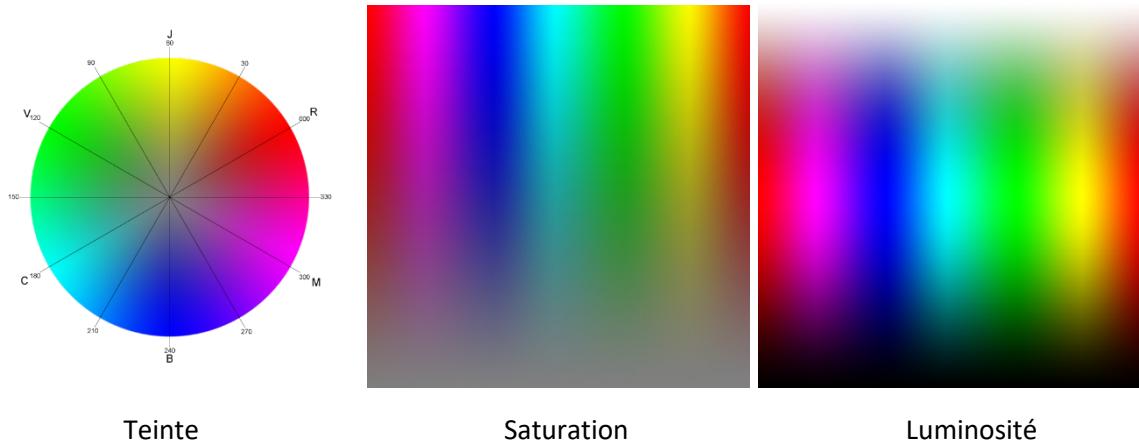


Figure 2A2 : visuel sur la teinte,
saturation et luminosité

Afin de définir quelle couleur nous intéresse, nous devons utiliser deux seuils (“threshold”) pour chaque paramètre. Nous créons donc deux curseurs (valeur minimum et valeur maximum) pour chaque paramètre, cela nous permettant d’être plus efficaces pour trouver les bonnes valeurs, en donnant une sorte de « plage » de valeurs (chaque curseur a des valeurs comprises entre 0 et 255).

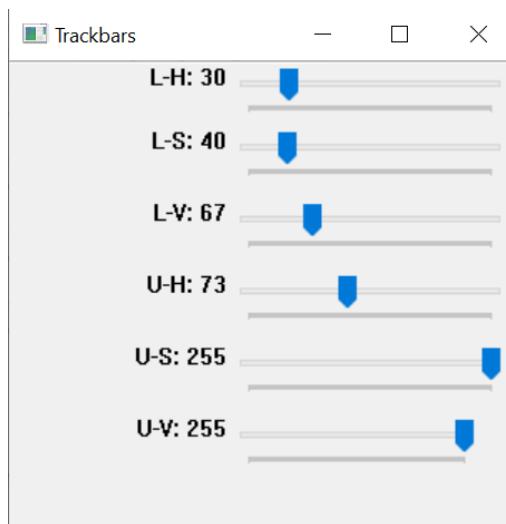


Figure 2A3 : Curseurs utilisés pour définir les seuils haut et bas de chaque paramètre

A partir de ces valeurs, on peut alors créer deux tableaux (valeurs minimum et maximum) et utiliser la fonction `inRange` d'*OpenCV*, qui permet de vérifier si les éléments de l'image HSV se trouvent entre les éléments des deux autres tableaux. On peut à présent créer un masque, et bouger les curseurs (figure 2A3) afin de n'afficher que ce qui nous intéresse :

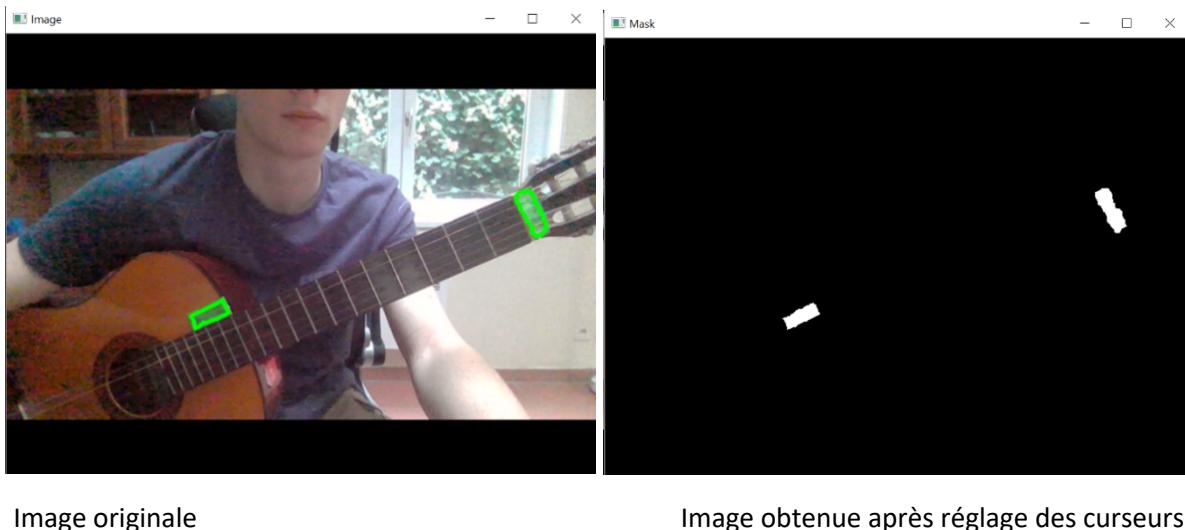


Figure 2A4 : Image originale et masque créé

Exploitons ce masque pour trouver les contours : on utilise la fonction *d'OpenCV* `findContours`. Il suffit maintenant de parcourir les contours trouvés à l'aide d'une boucle `for`, et d'y ajouter quelques conditions pour définir quel contour correspond à quel marqueur. Le but est d'obtenir trois points de référence. Par soucis de compréhension et pour faciliter la lecture, nous allons nommer les coordonnées les points de référence (cf figure 3A3 pour une illustration plus visuelle) :

(x_{11}, y_{11}) : point situé à l'intersection manche-caisse

(x_{21}, y_{21}) : point situé au sillet, corde 1 (mi grave)

(x_{22}, y_{22}) : point situé au sillet, corde 6 (mi aigu)

Autour de chaque contour est créé un rectangle (ci-dessous représenté en noir), grâce à la commande `x, y, w, h = cv2.boundingRect`

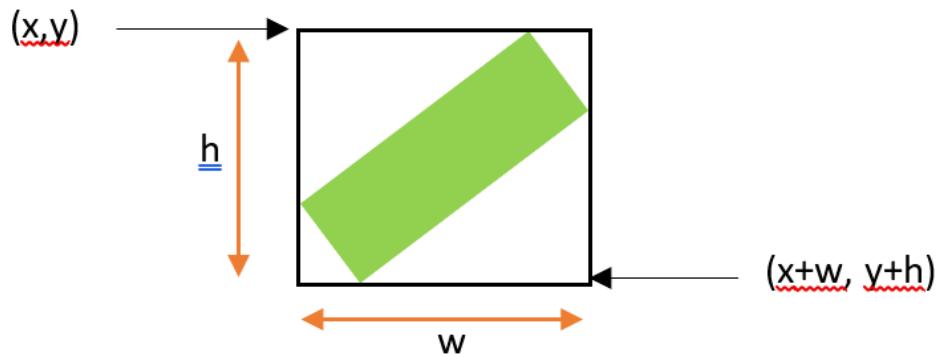


Figure 2A5 : Rectangle créé autour des marqueurs

Afin d'être plus précis, on fait une moyenne spatiale des coordonnées des contours trouvés : si x est inférieur à la moitié de la largeur de l'écran, alors cette coordonnée est ajoutée à `sommeX1`, et `moyenneX1` est ainsi recalculée (on en profite aussi pour faire la moyenne des y). Même principe pour la seconde moitié de l'écran, avec `moyenneX2`. Ainsi, si la coordonnée x est proche, à 10 pixels près, de `moyenneX1`, alors les coordonnées du point auquel on s'intéresse seront définies comme étant le point situé à l'intersection manchecaisse ($x_{11}, y_{11} = x+w, y+h$). A l'inverse, si la coordonnée x est proche, à 10 pixels près, de `moyenneX2`, alors les coordonnées du point auquel on s'intéresse seront définies comme étant le point situé au sillet, corde 1 (mi grave) (x_{21}, y_{21}). Les coordonnées du troisième point (x_{22}, y_{22}), situé au sillet corde 6 (mi aigu), valent ($x_{21}+w, y_{21}+h$).

L'étape de reconnaissance du manche étant à présent terminée, il faut maintenant exploiter les coordonnées trouvées pour afficher sur le manche ce que l'on souhaite.

III. Exploitation de la reconnaissance :

A. Projeter des accords sur le manche

Afin de pouvoir afficher des accords ou des notes sur le manche (sous la forme de points), nous avons besoin d'utiliser une matrice de coordonnées. Cette matrice devra suivre l'inclinaison du manche, mais aussi respecter l'espacement inter-frettes de la guitare, qui n'est pas linéaire. En effet, la longueur L de corde entre le chevalet et la frette détermine la note jouée. Plus cette longueur L diminue, plus la fréquence augmente. Cependant cette loi n'est pas linéaire en raison du caractère logarithmique des fréquences. La figure 3A1 illustre bien cette non-linéarité.

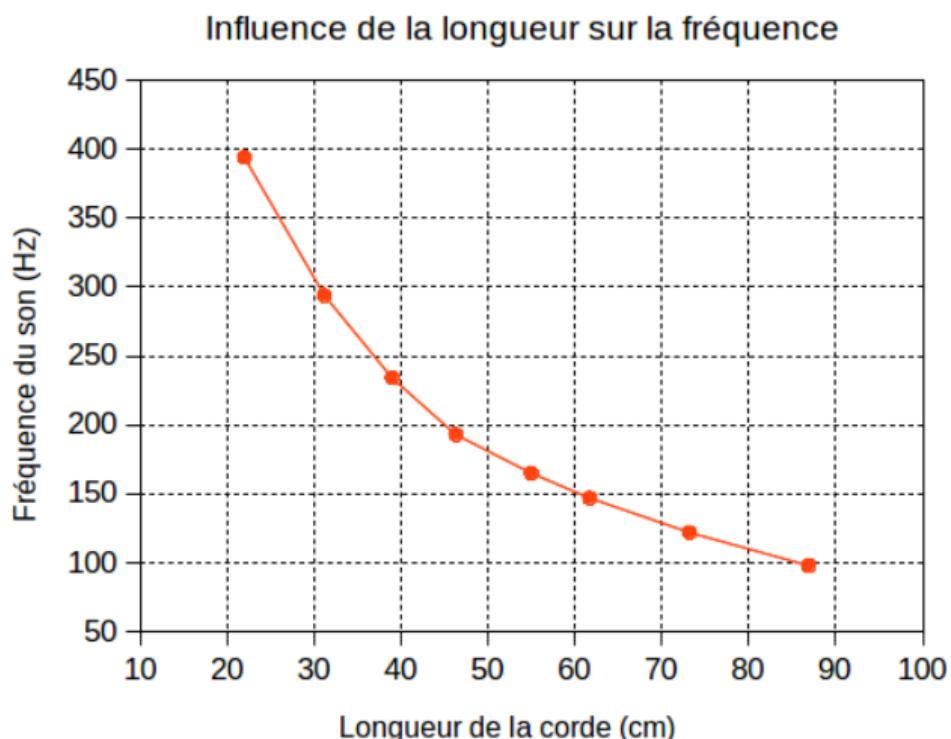


Figure 3A1 : Graphique de la fréquence du son en fonction de la longueur de la corde

Source : "Physique de la corde de guitare", sur « <https://zestedesavoir.com> »

Commençons par créer notre matrice. L'idée est de créer deux vecteurs de coordonnées : un vecteur pour les x allant de x_{21} à x_{11} , et un pour les y allant de y_{21} à y_{11} . On utilise alors la fonction `np.meshgrid(x, y)`, qui va retourner deux matrices de coordonnées `x_1` et `y_1`. On utilise ensuite `np.array(x_1, y_1)`, qui va créer la matrice de coordonnées finale. Le résultat obtenu en affichant tous les points de la matrice est visible en figure 3A2 :

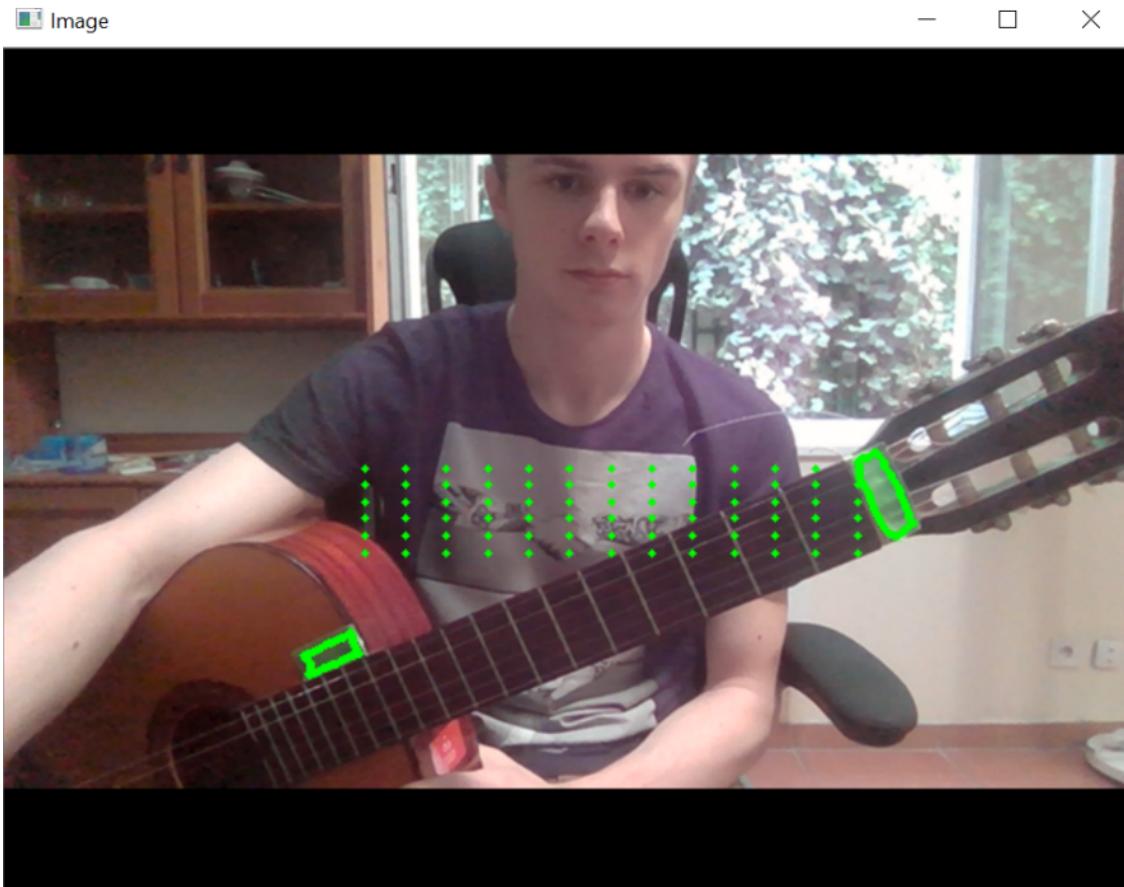


Figure 3A2 : Matrice de coordonnées obtenue avant rotation et modification inter-frettes

A présent, il faut que cette matrice suive l'inclinaison de la guitare. Pour cela, nous copions la matrice obtenue, et nous parcourons tous les points à l'aide d'une boucle `for`. L'instruction est la suivante : chaque point est tourné autour du point (x_{21}, y_{21}) (sillet corde 1), suivant l'angle de la ligne, qui est calculé grâce à la formule suivante :

$$\text{angleManche} = \arctan\left(\frac{x_{21}-x_{11}}{y_{21}-y_{11}}\right) * \left(\frac{180}{\pi}\right)$$

En affichant tous les points de la matrice tournée, on obtient le résultat suivant visible en figure 3A3 :

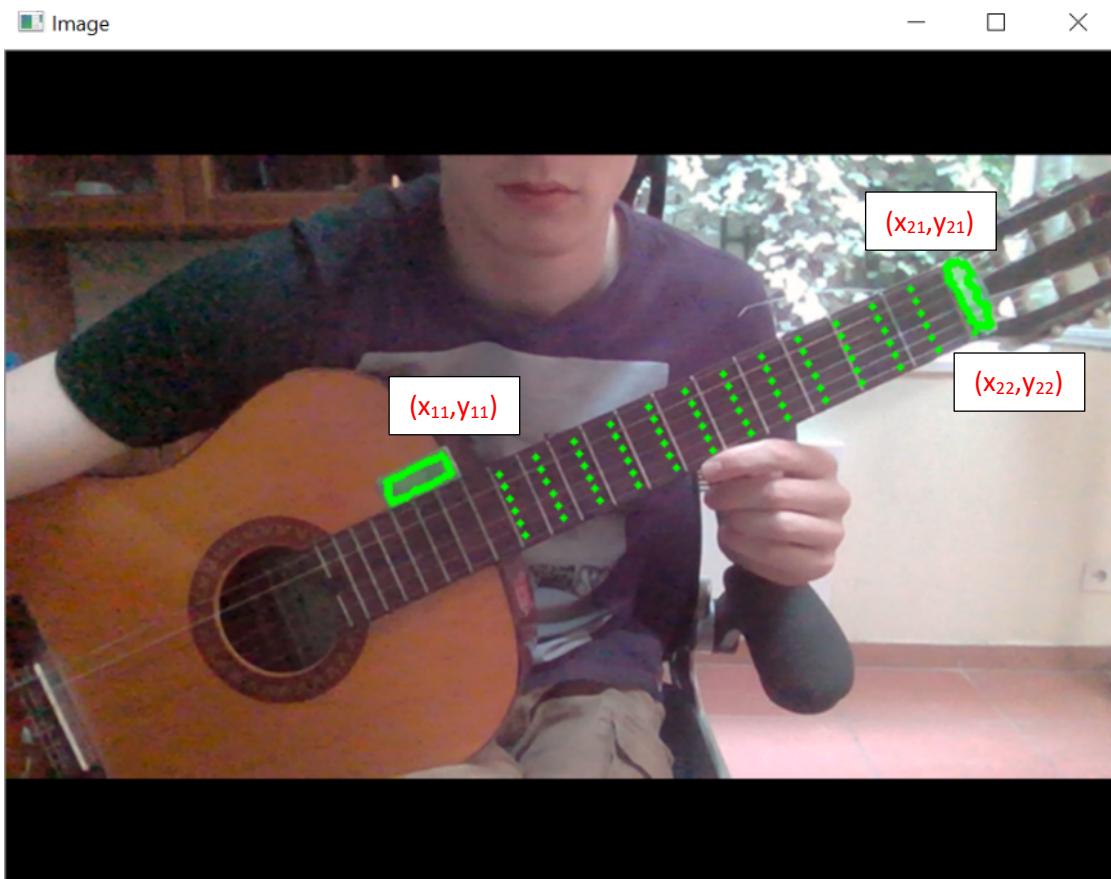


Figure 3A3 : Matrice obtenue après rotation

La dernière étape est de respecter les distances inter-frettes. Le seul axe impacté par cette non-linéarité est celui des x. L'idée est donc de modifier chaque coordonnée x dès la création de la matrice. Pour cela, il faut d'abord créer un vecteur (noté M) de pourcentages (dont les valeurs sont comprises entre 0 et 1).

Ce vecteur exprime la distance entre le sillet et la i-ème frette en pourcentage, la valeur 1 correspondant à la longueur du diapason. Nous nous sommes aidés de l'article présent en annexe p20 ("Calcul des distances inter-frettes") pour déterminer une formule. La première valeur (la distance entre le sillet et la première frette) est égale à $\left(\frac{\text{longueurDiapason}}{17,817} \right)$ (Donc $\frac{1}{17,817}$).

Pour la deuxième valeur, on prend la longueur du diapason moins la longueur de la case 1 et on divise à nouveau par 17,817, et ainsi de suite pour les autres

distances. Ainsi, on établit la formule de récurrence suivante : (M est le vecteur des distances sillet-frette)

$$M_{[i]} = M_{[i-1]} + \frac{1 - M_{[i-1]}}{17,817}$$

Le vecteur de distances sillet-frettes en pourcentages a donc finalement les valeurs suivantes :

```
0.0
0.05612617163383286
0.10910219612539525
0.15910487917386607
0.20630110305140628
0.25084838356712685
0.2928953957708016
0.332582470150843
0.3700420609825856
0.4053991883899726
0.4387718555960133
0.470271442754569
0.5000030786778688
```

Figure 3A4 : Valeurs en pourcentage des distances sillet-frettes

Une fois ce vecteur obtenu, l'idée est de modifier chaque coordonnée x dès la création du vecteur de coordonnées x. Pour cela, une fois que ce dernier est créé, nous le parcourons et modifions la valeur de chaque x : la nouvelle valeur de chaque x est alors égale à :

$$x_{[i]} = x_{21} - (M_{[i]} * \text{longueurDiapason})$$

Pour connaître la longueur du diapason, il faut savoir au préalable quelle est la frette au niveau de la caisse. En effet, pour certaines guitares c'est la 12^{ème} frette qui est au niveau de la caisse, et pour d'autres c'est la 14^{ème}, il y a plusieurs cas (pour l'instant, nous avons exploré uniquement ces deux cas car ce sont les plus courants).

Le but est de déterminer la longueur du diapason à partir de la seule longueur que nous connaissons, celle du manche.

Dans tous les cas, la longueur du diapason est égale à 2 fois la distance sillet-12ème frette. Ainsi, si c'est la 12^{ème} frette qui est au niveau de la caisse, alors la longueur du diapason est égale à 2*(la longueur du manche).

Si c'est la 14^{ème} frette, alors on établit une distance entre la frette 12 et 14 (en pourcentage), qui est environ égale à 0.1028571429 (la distance entre la frette 12 et 14 est environ égale à 10% de la longueur du manche).

La longueur du diapason sera donc égale à (*longueurManche* – (*distance1214* * *longueurManche*)) * 2

Tous les éléments sont à présent réunis pour pouvoir modifier les coordonnées x. La figure 3A5 nous illustre le résultat de la matrice finale :

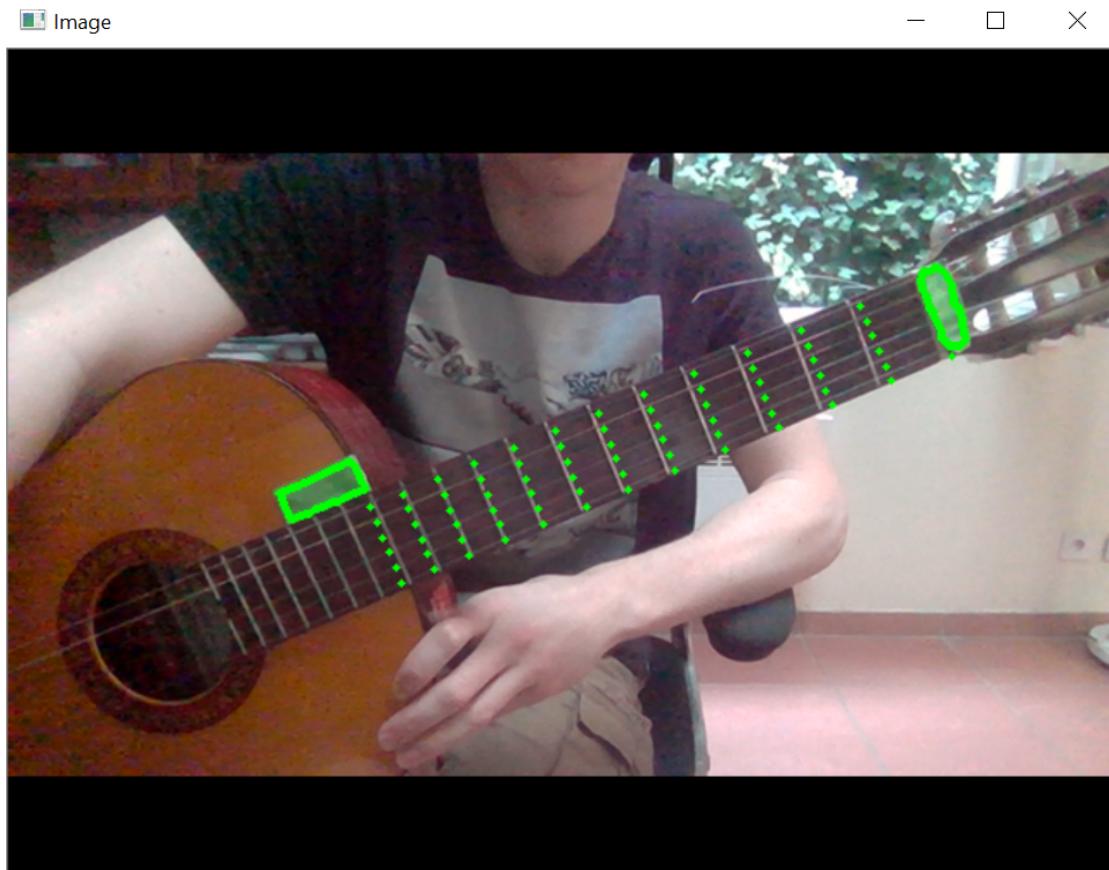


Figure 3A5 : Matrice finale

Pour dessiner une note, il suffit maintenant de renseigner le numéro de frette et le numéro de corde (voir en figure 3A6) :

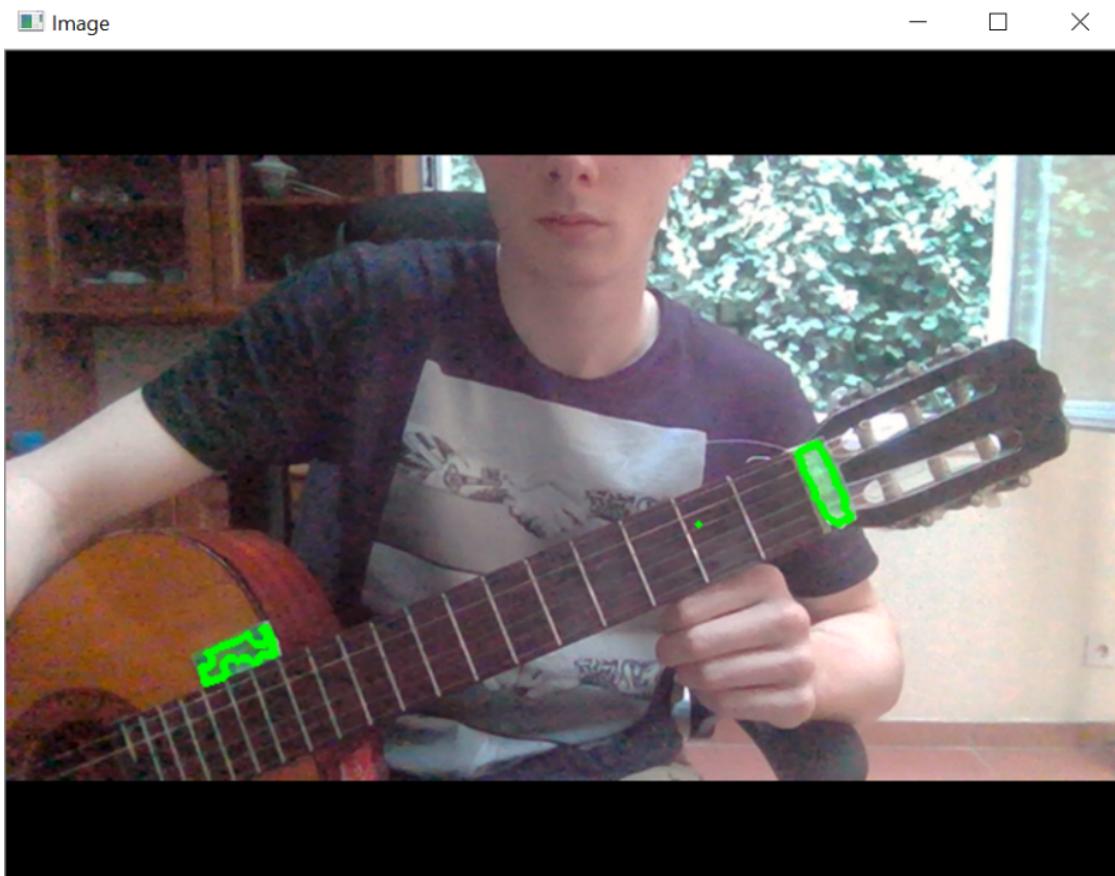


Figure 3A6 : On dessine la note à la corde 3, case 2

Si l'on peut dessiner une note, on peut aussi dessiner un accord. Précisons que cette idée nous a été inspirée par un article, “*Augmented Reality Scenarios for Guitar Learning*” par F. Liarokapis. Pour dessiner un accord, nous créons une fonction accord(ac), qui prend en argument le nom d'un accord, par exemple « Fa ». Cette fonction va retourner un tableau d'indices. Pour l'accord « Fa », cette fonction va retourner

$$[(1,1), (2,3), (3,3), (4,2), (5,1), (6,1)]$$

Chaque couple correspond à des indices pour les coordonnées x et y : par exemple, lorsque l'on veut dessiner une note à la 3ème corde, 3ème case, on va envoyer le couple (3,3)

On parcourt alors ce tableau pour afficher tous ces points. On peut voir le résultat de l'affichage de l'accord “Fa” en figure 3A7 :

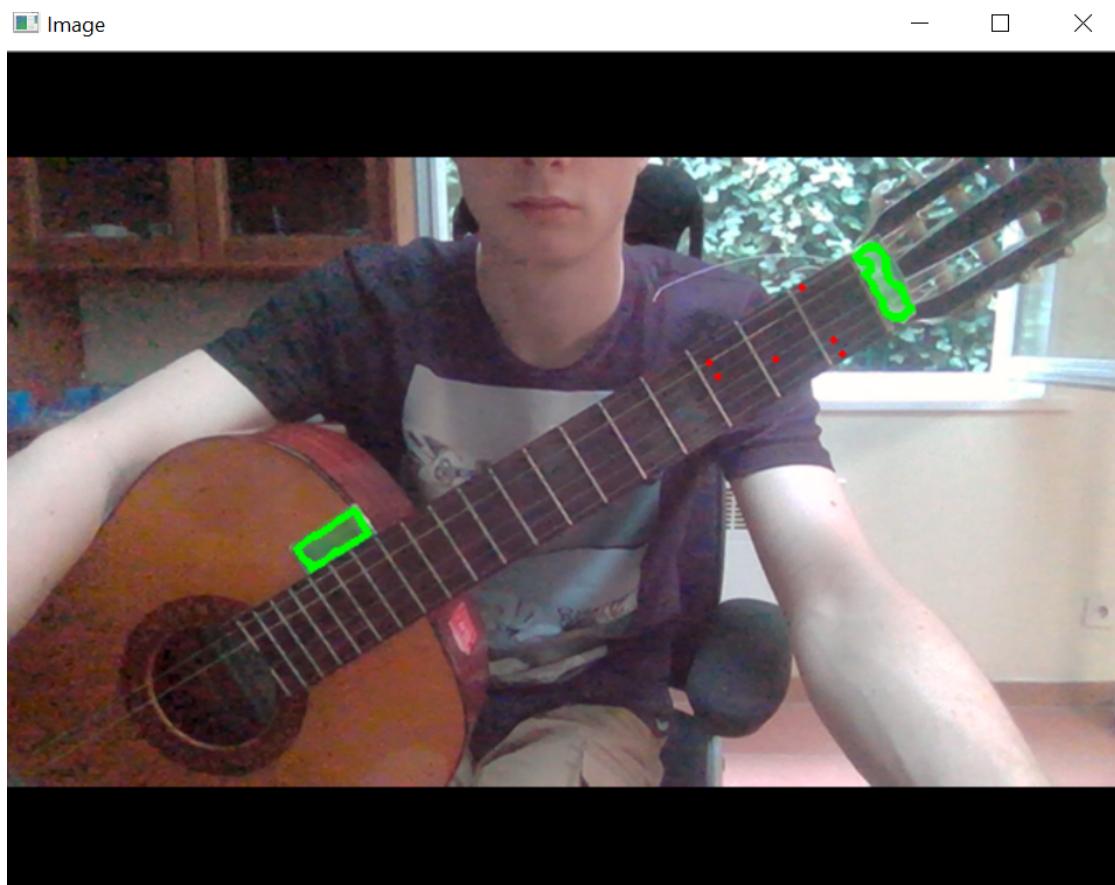


Figure 3A7 : On dessine l'accord « Fa » (majeur)

Nous avons donc réussi à afficher des notes et des accords sur le manche, grâce à une matrice de coordonnées qui suit le manche.

A présent, intéressons-nous au changement d'apparence du manche

B. Changer l'apparence du manche

Nous sommes partis sur l'idée de projeter une image sur le flux vidéo de la webcam. A partir des points de référence que nous avons, nous pouvons définir une « boîte » (une zone image par image du flux vidéo) où coller une texture de manche.



Figure 3B1 : Exemple de fichier .png de manche utilisé

La méthode consiste alors à d'abord effectuer une rotation de l'image de la texture suivant l'inclinaison du manche réel de la guitare sur lequel on veut superposer notre image.

Puis redimensionner l'image de texture aux dimensions du manche réel sur l'image car pour utiliser la méthode de superposition d'image avec *OpenCV*, il faut absolument que les dimensions de l'image correspondent à celles de la zone où on veut l'insérer.

Ensuite, il était impossible d'appliquer une matrice de rotation sur la « boîte » où l'on voulait insérer l'image. (cf figure 3B2)

```
#insermentation d'une image sur le flux vidéo webcam
img[Y:Y+dim[0] , X:X+dim[1] ] = weighted_img
```

Figure 3B2 : Insermentation d'image sur Python

En effet une autre contrainte est que la « boîte » se définit sur l'image à partir d'un seul point de référence (sur l'image il s'agit du point vert de coordonnée (X,Y) utiliser dans la ligne de code (cf figure 3B2)).

La « boîte » est ainsi définie par les coordonnées d'un point et les dimensions de l'image (dim[0]= « hauteur de l'image » et dim[1]= « largeur de l'image »)

(cf figure 3B3).



Figure 3B3 : Définition de la « boîte »

À partir de ces contraintes, différentes stratégies ont été développées.

En réutilisant l'angle du manche calculé précédemment nous pouvons faire une rotation sur l'image et la redimensionner aux dimensions approximatives du manche :

```
#rotation de l'image contenant suivant l'angle du manche et redimensionnement de l'image au dimensionnement
#resized_img = cv2.resize(img_manche, (img_width,img_height), interpolation = cv2.INTER_AREA)
rotated_img=rotate_image(img_manche,angle=90)
resized_img = cv2.resize(rotated_img, (img_width,img_height), interpolation = cv2.INTER_AREA)
```

Figure 3B4 : Redimensionnement aux dimensions approximatives du manche ($img_width * img_height$)

Nous avons défini les variables img_width et img_height en fonction des points de référence pour que la texture ait les bonnes dimensions :

$$img_{width} = (x_{vert} - x_{bleu})$$

$$img_{height} = (y_{bleu} - y_{jaune})$$

Un problème apparaît alors lorsque le manche n'est plus horizontal.

Une première solution pour éviter d'avoir un rectangle noir sur la figure 3B3 était d'utiliser une image .png préalablement éditer avec *Gimp* de manière à avoir des zones transparentes autour de la texture.

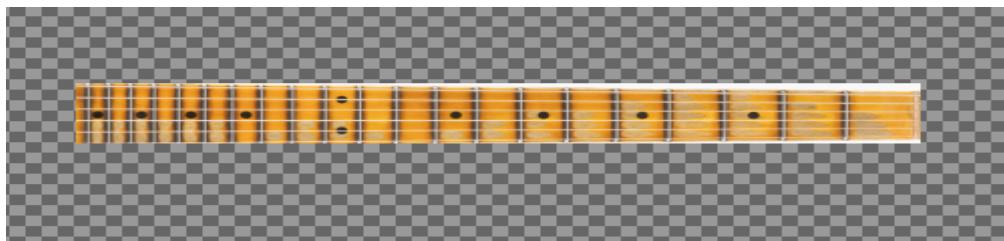


Figure 3B5 : Texture_bord_transparent.png

Cependant ce genre d'image possède une quatrième couche, la **couche alpha**, en plus des trois couches : rouge, verte et bleue. Cette couche ne modifie pas les couleurs de l'image et sert dans la plupart des cas à gérer la transparence de l'image. (Référence en annexe)

La méthode de superposition d'image via *OpenCV* requiert que les images aient le même nombre de canaux (le flux webcam en possède 3 car en RGB, mais la texture.png en possède 4 car RGB + canal Alpha).

Une deuxième solution est donc de remplacer ces zones transparentes par des couleurs unies qui ne seraient « pas trop dérangeantes ».



Figure 3B6 : Texture_bord_unis.png

De plus avec la fonction `cv2.addWeighted(src1, alpha, src2, 1-alpha)` on peut préparer une image calque qui va être superposée avec une transparence variable.

Le principe de la fonction `cv2.addWeighted` repose sur l'équation :

$$g(x) = (1 - \alpha)f_0(x) + \alpha f_1(x)$$

Qui définit une relation de densité (*linear blend operator*) entre les pixels de l'image g et ceux de f_0 et f_1 , où f_0 représente l'image de texture et f_1 l'image de fond (le flux vidéo de la webcam).

Ainsi on a un mélange des deux images avec α comme pourcentage de présence de l'image f_0 . (Référence en annexe)



Figure 3B7 : Incrustation de la texture.png (figure 3B1) sur le flux vidéo avec $\alpha = 0,4$

Il faut maintenant essayer de contourner la contrainte d'avoir un seul point de référence pour placer la boîte.

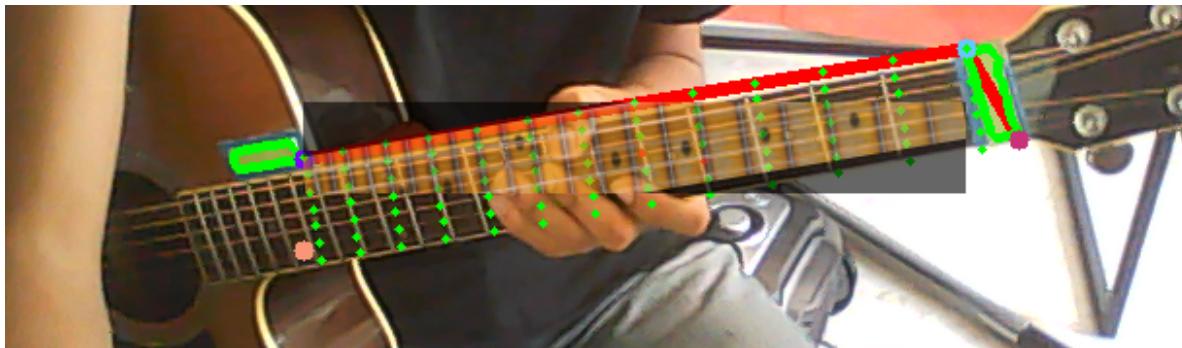


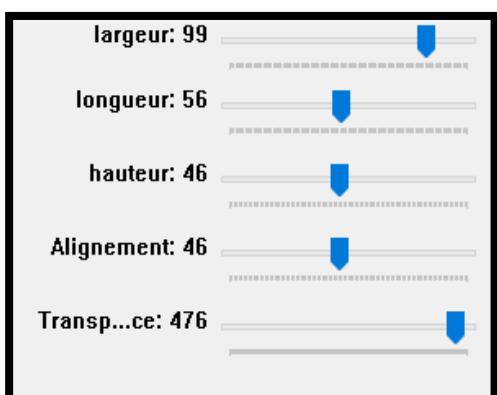
Figure 3B8 : Problème de recouvrement du manche

En effet en positionnant la boîte à partir d'un unique point lorsque que le manche est incliné on risque alors une texture superposée qui ne couvre pas l'intégralité du manche.

Pour pallier ce problème il faut que l'image de texture ait des « bords suffisamment épais » pour laisser plus d'espace au manche virtuel pour pivoter.



Figure 3B9 : Utilisation de Texture_bords_unis.png (figure 3B6)



Nous avons rajouté des curseurs pour permettre à l'utilisateur de faire quelques ajustements (transparence, alignement frettes réel/virtuelle, hauteur sur l'image, largeur, longueur) afin de pouvoir utiliser d'autres images de texture et ce sur différents manches.

On redéfinit ainsi les dimensions dans lesquelles l'image de texture est redimensionnée avec les curseurs correctifs de largeur et longueur :

$$img_{width} = (x_{bleu} - x_{beige}) + L_{correctif}$$

$$img_{height} = (y_{bleu} - y_{violet}) + l_{correctif}$$

Nous essayons de surmonter la contrainte d'avoir un seul point de référence pour la « boîte » en définissant la zone d'incrustation avec le code présenté en figure 3B10 :

```

204     Y=abs(y11+y21)//2-offset #Y= 1/2 *(Ybleu-Ymauve)+correctif
205     X=x11-fin
206
207     # Mix de la texture et du background image/image
208     weighted_img = cv2.addWeighted(img[Y:Y+dim[0],X:X+dim[1],:],alpha,resized_img,1-alpha,0)
209
210     #print("H ",img_height>=20,"L ",img_width>=300 )
211     #print("largeur: ",img_width,"H: ",img_height)
212
213     #incrustation d'une image sur le flux vidéo webcam
214     img[Y:Y+dim[0] , X:X+dim[1] ] = weighted_img
  
```

Figure 3B10 : Code définissant la zone d'incrustation

La « boîte » est donc positionnée à une hauteur $Y = \frac{(Y_{bleu}-Y_{mauve})}{2} + H_{correctif}$
 qui s'étend jusqu'à $Y = \left[\frac{(Y_{bleu}-Y_{mauve})}{2} + H_{correctif} \right] + img_height$. Et
 s'étend en longueur de $X = x_{violet} - x_{correction alignement frette}$ jusqu'à
 $X = [x_{beige} - x_{correction alignement frette}] + img_width$



Figure 3B11 : Flux vidéo avec la boîte incrustée sur la guitare correctement

Finalement après que l'utilisateur fasse ses petits correctifs, l'incrustation de la texture est correcte. (cf vidéo texture.mkv)

Conclusion / ouverture :

Ce projet nous a donc montré dans sa finalité, différentes possibilités pour modifier la perception que nous pouvons avoir d'une guitare, et comment utiliser ces outils pour voir autrement le manche de l'instrument.

En effet, grâce aux différentes techniques mises en œuvre pour changer l'aspect de l'instrument, nous pouvons actuellement dire que le challenge de changer l'apparence de la guitare est concluant, cependant même si l'aspect technique est un succès, pour répondre entièrement à notre problématique il aurait fallu faire tester notre programme avec différents guitaristes pour avoir une réponse plus précise à notre hypothèse.

La matrice de point, à visée éducative, est quant à elle une bonne appropriation de notre algorithme de reconnaissance du manche, et l'objectif serait par la suite de pouvoir afficher des suites de notes MIDI pour pouvoir faire jouer à un musicien une chanson entière, ce qui pourrait largement aider l'apprentissage.

Nous tirerons de ce projet de nouvelles connaissances en informatique, et notamment en traitement d'images. La réalisation de ce projet nous a tous motivé à en découvrir davantage et nous a passionné au fur et à mesure de notre travail.

Annexes

Voici les différentes méthodes qui ont été explorées mais pas abouti, pour des raisons de temps ou d'efficacité :

Pour reconnaître le manche de la guitare, une option était le machine learning : à partir de photos “positives” et “négatives” (photos contenantes ou non un manche de guitare), l'idée était de labelliser ces dernières et d'établir une sorte de base de données, sous forme d'un fichier qui serait par la suite lu dans notre programme *Python*, le but étant qu'il reconnaisse instantanément le manche de guitare. Cela fonctionne comme la reconnaissance de visage par exemple.

Cette méthode aurait pu s'avérer extrêmement efficace, mais après quelques recherches nous nous sommes rendu compte que c'est une méthode relativement compliquée à mettre en place, et surtout très chronophage. Nous ne sommes donc pas allés plus loin, faute de temps.

Une deuxième option pour reconnaître le manche était la reconnaissance de formes : au lieu d'utiliser des marqueurs de couleurs, l'idée était de reconnaître les formes, notamment les lignes (le manche de guitare étant composé de plusieurs lignes). Pour cela, nous avons utilisé un « threshold adaptatif Gaussien ». Cette méthode utilise la valeur d'un seuil qui est dynamiquement modifié (selon l'image capturée par la caméra) afin de rendre compte de l'intensité locale des différentes zones de l'image (en niveau de gris). Une fois cette méthode appliquée à notre image, nous avions les masques des figures Ann1 et Ann2 :



Figure Ann1 : Masque à contrejour

Figure Ann2 : Masque au soleil

L'avantage de cette méthode était qu'il n'y avait pas besoin de curseurs pour régler l'image : le « seuil adaptatif Gaussien » permet justement de s'adapter à tout type d'environnement. Que la guitare soit en plein soleil, à contrejour ou dans un endroit un peu sombre, le masque obtenu reste relativement le même.

Ainsi, à partir de ce masque, nous avons détecté les lignes grâce à la fonction `HoughLinesP` d'*OpenCV*. Chaque ligne possède alors quatre coordonnées : x_1, y_1, x_2, y_2 , qui sont les coordonnées des points définissant la ligne. On peut alors définir la plus grande ligne (qui vaut 0 de base) : si la longueur de la ligne (qui vaut $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$) est plus grande que la longueur de la plus grande ligne, alors celle-ci devient la plus grande ligne. Ainsi, la plus grande ligne détectée est souvent une ligne appartenant au manche. On calcule ensuite l'angle de cette ligne, et pour chaque ligne parcourue, si l'angle de la ligne est égal, à 1 degré près, à celui de la plus grande ligne, alors on prend en compte cette ligne (sur les images qui suivent, la plus grande ligne s'affiche en bleu)

Voici les résultats obtenus avec cet algorithme :

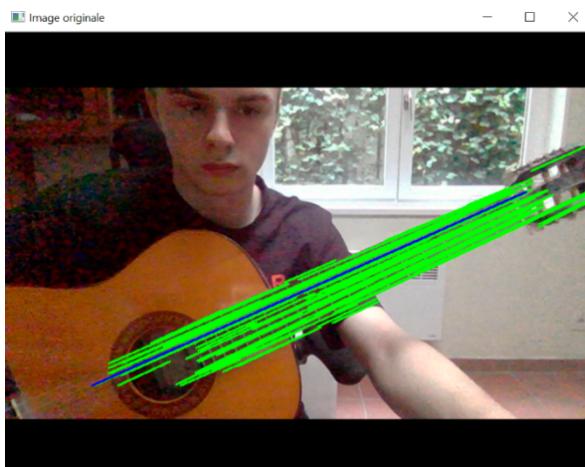


Figure Ann3 : A contrejour



Figure Ann 4 : En plein soleil

On peut donc voir que les lignes du manche sont bien reconnues. C'est une méthode qui pourrait être prometteuse, mais malheureusement elle peut s'avérer ne pas être très stable de temps en temps : en effet, l'image analysée étant un flux vidéo, le masque a sans arrêt beaucoup de « bruit » et donc les lignes sont très volatiles. Difficile alors de déterminer précisément les coordonnées des points de référence.

OpenCV (pour Open Computer Vision) est une bibliothèque graphique libre spécialisée dans le traitement d'images en temps réel :

<https://opencv.org/>

Python est un langage de programmation très populaire, favorisant la programmation impérative structurée, fonctionnelle et orientée objet :

<https://www.python.org/>

Le codage des couleurs :

<http://primatec.phpnet.org/logiciels/chromoweb/aide/codage.htm#:~:text=Pour%20repr%C3%A9senter%20les%20couleurs%2C%20les,allant%20de%200%20%C3%A0%20255.>

Calcul des distances inter-frettes :

<https://guitare-et-couleurs.com/comment-est-calcule-lespace-ment-des-frettes-sur-un-manche-de-guitare/>

Fonction `numpy.meshgrid` :

<https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html>

Fonction `numpy.array` :

<https://numpy.org/doc/stable/reference/generated/numpy.array.html>

Fonction `cv2.boundingRect` :

https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#!?highlight=boundingrect

Fonction `cv2.findContours` :

https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#!

Canal Alpha :

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

Add image to a live camera feed using OpenCV-Python:

<https://theailearner.com/2019/03/18/add-image-to-a-live-camera-feed-using-opencv-python/>

Fonction `cv2.addWeighted`:

https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html