Projet INF573
Theremultime: the Ultimate Theremin

Guillaume Pelat & Pascal (Chien-Hsi) Chang

January 5, 2020

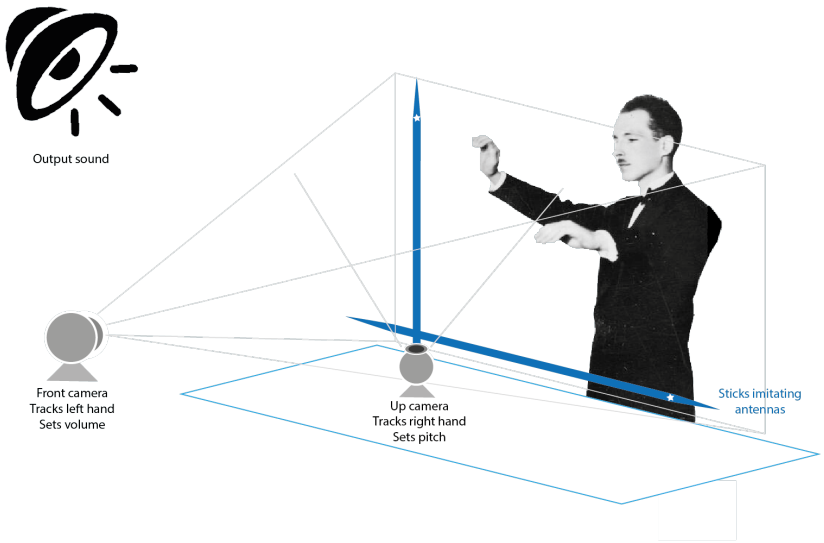
Introduction

L'idée de combiner vision par ordinateur et production de son nous avait plu dès nos premières discussions sur le sujet de ce projet. Aussi, après avoir regardé plusieurs pistes dans cette direction, nous nous sommes intéressés au thérémine.

Le thérémine est un instrument de musique électronique inventé au début du XXème siècle qui a la particularité de se jouer sans être touché par l'instrumentaliste : le joueur fait varier la hauteur et l'intensité de la note en modifiant respectivement la distance de sa main droite à l'antenne verticale et de sa main gauche à l'antenne horizontale.

Ainsi, dans le cadre de ce projet, nous avons décidé de reproduire les fonctionnalités d'un thérémine en utilisant des techniques de vision par ordinateur en C++, à l'aide de la bibliothèque avec OpenCV. Nous présentons dans la suite de ce rapport le dispositif ainsi que les résultats que nous avons obtenus.

1 Dispositif et feuille de route



L'idée principale de ce projet est de proposer une expérience proche de celle du thérémine à l'utilisateur en employant des techniques de vision par ordinateur. Ceci nécessite alors de réaliser trois grandes tâches :

- détecter la position des "antennes" sur l'image ;
- détecter la distance de chaque main à l'antenne correspondante ;
- produire du son.

Le dispositif initial (voir schéma ci-dessus) devait contenir deux caméras : la première, de face, détecterait principalement la hauteur des mains et permettrait de faire varier l'intensité du son produit ; la seconde, du dessous, s'occuperait alors de détecter la distance de la main droite à l'antenne verticale et ferait varier la hauteur de la note. Finalement, l'utilisation d'une seule caméra (celle de face) a pu donner des résultats satisfaisants.

Nous utilisons aussi deux tiges solides disposées en angle droit pour simuler les antennes d'un vrai thérémine. Le logiciel doit donc détecter leur position pour pouvoir produire correctement le son. Les trois parties du projet sont détaillées ci-dessous.

2 Détection de la position des mains

La première étape a été de détecter la distance des mains aux antennes. Pour rappel, les caractéristiques du son produit dépendent de :

- pour **la hauteur** (ou fréquence) : la plus petite distance entre l'antenne verticale et la main droite du joueur (i.e. la distance horizontale entre le point le plus à droite de la main droite et l'antenne)
- pour **l'intensité** (ou volume) : la plus petite distance entre l'antenne horizontale et la main gauche du joueur (i.e. la distance verticale entre le point le plus bas de la main gauche et l'antenne)

Nous enregistrons une vidéo via la webcam de l'ordinateur et procédons à une série de méthodes pour extraire ces distances à partir des *frames* de la vidéo. Parmi les différentes étapes, il y a l'**extraction des mains**, la **détection des mains gauche et droite** et le **calcul des distances**.

2.1 Extraction des mains

Cette première étape consiste à extraire à partir de l'image capturée par la webcam, la zone correspondant aux mains de l'utilisateur. À noter qu'à l'issue de ce premier traitement, nous ne savons pas encore séparer la main gauche de la main droite. Nous faisons dans l'ordre :

- le **débruitage** de l'image grâce à un filtre gaussien de taille 5×5 pixels pour rendre l'image plus lisse ;
- une ***background subtraction*** pour extraire à partir de l'image les parties en mouvement que nous gardons pour le moment dans une image à part ;
- à partir de l'image débruitée, l'**augmentation de la saturation** et la **détection d'une couleur spécifique** (en l'occurrence bleu clair) en passant en coordonnées de couleurs HSV. En effet notre détection des mains diffère des situations habituelles par le fait que la main se trouve souvent à l'horizontale par rapport à la caméra. Les méthodes de détection de mains traditionnelles ne fonctionnent donc pas aussi bien. Nous résolvons le problème en fournissant à l'utilisateur des gants bleus que notre programme a appris à détecter ;
- **multiplication des deux images** précédentes pour obtenir uniquement les objets bleus en mouvement (cela permet d'être robuste à la présence d'autres objets bleus dans la scène) ;

- une opération de ***opening*** (une érosion suivi d'une dilation), cela permet de supprimer les petits îlots de pixels blancs (bruits) restant qui pourraient perturber la détecter de la main.

(a) Image lissée

(b) Background subtraction

(c) Détection de la couleur bleue

(d) Multiplication de (b) et (c)

(e) Erosion suivie d'une dilation (débruitage)

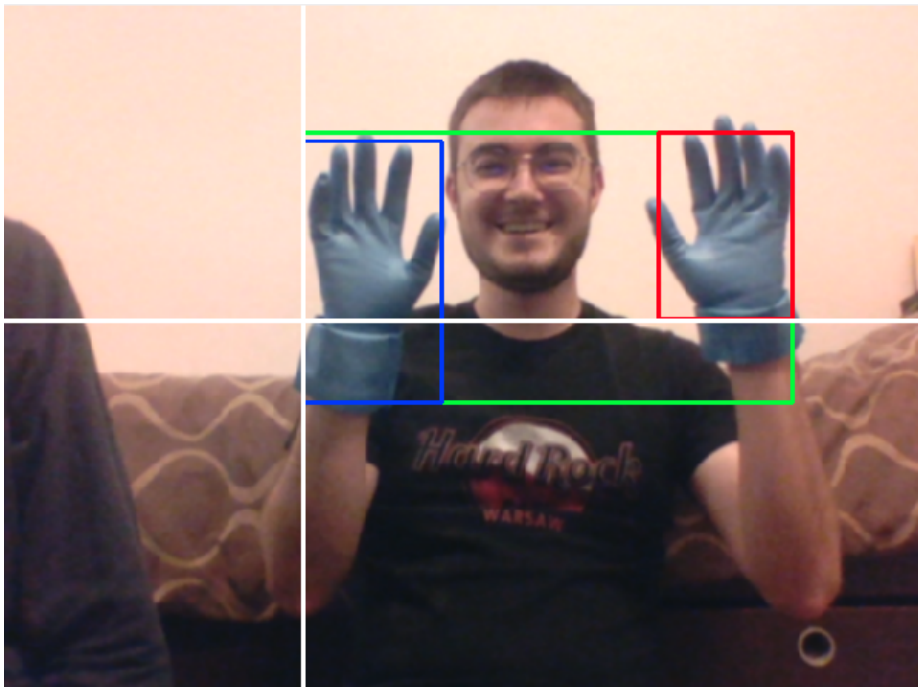


Figure: drawing bounding boxes for each hand

2.2 Détection des mains

Nous avons à présent une image en noir et blanc (*mask*) où le blanc correspond à la zone des mains. Pour séparer la main gauche de la main droite, nous partons de l'hypothèse selon laquelle la main gauche de l'utilisateur sera toujours à gauche de sa main droite (ce qui est le cas lorsque l'on joue avec un vrai thérémine).

Ainsi, nous traçons la *bounding box* correspondant aux deux mains et traçons une ligne verticale au milieu du rectangle. Nous recalculons ensuite une *bounding box* pour chaque partie de l'image séparé par cette ligne. Cela nous donne alors deux rectangles délimitant l'emplacement des deux mains.

2.3 Calcul des distances

Enfin, grâce aux coordonnées des deux rectangles, nous pouvons facilement obtenir la distance en nombre de pixels entre les extrémités des mains et n'importe quel autre objet, comme par exemple les bâtons que nous aurons (théoriquement) détectés. Nous pouvons ensuite convertir cette distance en mètres et appliquer la loi qui relie fréquence émise à la distance pour reproduire les effets d'un thérémine.

2.4 Interface utilisateur

Les conditions d'éclairage et la disposition de la scène peuvent avoir de grandes conséquences sur la performance du logiciel et de la détection des couleurs. Pour résoudre ce problème, nous donnons aussi à l'utilisateur la possibilité de faire varier les paramètres pendant l'exécution, comme par exemple la plage de couleur pour la détection des gants. L'utilisateur pourra ainsi choisir de détecter des gants rouges ou verts, en gardant en tête de choisir une couleur éloignée de la couleur de sa peau et de ses vêtements.

3 Trouver les antennes

Trouver la position des mains est une chose ; pouvoir les repérer par rapport aux antennes (que nous avons remplacé par le matériel à notre disposition, des cannes d'escrime artistique) en est une autre. Nous avons essayé deux techniques pour cela, mais malheureusement aucune n'a donné de résultat satisfaisant.

La première est de procéder de la même façon que pour les mains, et de se baser sur la couleur des bâtons pour déterminer leur position. Le problème de cette méthode est que contrairement au bleu ciel employé pour les mains, leur couleur marron-rouge est beaucoup plus répandue et il est trop facile de les perdre dans l'arrière plan.

Nous avons donc tenté de faire du ***template matching***: imprimer deux motifs que nous placerions aux extrémités de chaque bâton, puis détecter la position de ces motifs dans l'image. Le problème qui s'est posé cette fois est un problème d'échelle: en effet, OpenCV réalise du template matching en calculant une convolution entre le motif et l'image, ce qui nécessite que le motif que l'on cherche à détecter fasse la même taille que l'image utilisée pour sa recherche. Nous avons donc tenté de faire cette détection plusieurs fois, en variant la taille de l'image à chaque itération.

Ce faisant, nous obtenons pour chaque motif une détection pour chaque échelle différente utilisée. Si le motif arrivait à être détecté dans l'immense majorité des cas, il reste à détecter parmis ces échelles celle qui a permis la meilleure détection. Nos résultats dans cette étape n'étaient par contre pas très probants, et pour la suite, nous avons préféré considérer que les bâtons étaient situés à la limite de l'écran.

4 Créer du son

Le but de notre projet étant de recréer un instrument de musique, il était indispensable de savoir émettre des sons.

4.1 The Synthesis Toolkit for C++

La littérature propose habituellement deux façons d'imiter un instrument de musique. La première, et probablement la plus simple, est de disposer d'un enregistrement audio de chaque note produite par l'instrument et de jouer cet enregistrement sur demande. En gérant correctement les transitions entre les différents morceaux, cela permet une certaine flexibilité et un son fidèle à la réalité.

Cependant, le principal problème de cette méthode vient de son fonctionnement. En effet, outre le fait que le thérémine est un instrument suffisamment rare pour qu'il soit compliqué d'obtenir un exemplaire de chacune des 60 notes (estimation basse) qu'il est capable de produire, c'est aussi et surtout un instrument à son "continu". Cela signifie que quelle que soit la position de la main, l'instrument joue quelque chose: nous devrions donc avoir un enregistrement non seulement pour chaque note, mais pour chaque fréquence qu'il est capable de produire.

La seconde méthode est donc de revenir à ce qui caractérise un son: c'est une somme de fonctions sinusoïdales, ayant en général pour fréquence un multiple d'une certaine *fréquence fondamentale*. Celle-ci détermine quelle note est jouée, tandis que la composition des fréquences multiples (aussi appelées *harmoniques*) permet de caractériser un instrument. Il est cependant très complexe de produire un son convaincant à partir de rien, à la fois d'un point de vue pratique (nos tentatives ressemblaient toutes tristement à de la musique 8 bits que n'aurait pas reniée une console de jeu des années 80) et d'un point de vue théorique, chaque système d'exploitation ayant sa façon de gérer la carte son.

Nous avons donc recherché une librairie qui serait au traitement du son ce qu'OpenCV est au traitement d'image, et nous nous sommes tournés vers le **Sythesis Toolkit for C++**. Il s'agit d'un ensemble de classes développées par Perry R. Cook et Gary P. Scavone depuis 1995, et qui permettent d'émuler différents instruments, dont plusieurs à son continu comme le saxophone ou la clarinette. Sous réserve de fournir les bons paramètres au compilateur, le STK est capable de compiler aussi bien sous Linux que sous Mac, les deux systèmes d'exploitation que nous avions au début du projet (par la suite, nous sommes tous deux passés sous Ubuntu).

4.2 Récupérer les informations de la caméra

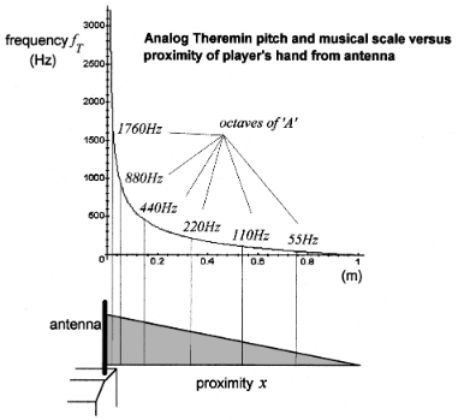
La nécessité de produire un son continu nous a conduits à faire fonctionner notre programme dans deux threads: l'un gère le traitement d'image, et l'autre la création du son. L'avantage de ce système de fonctionnement est double: il permet d'une part un gain de performance, en faisant usage de plusieurs processeurs si l'ordinateur en dispose, et d'autre part une réactivité et une continuité que n'aurait pas un programme linéaire.

En effet, pour créer un son nous générons un buffer audio d'une certaine longueur. Si ce buffer est trop long, nous devons attendre qu'il finisse avant de pouvoir changer la note ;

cela se traduirait par une expérience utilisateur médiocre, car l'utilisateur aurait un retour tardif du programme. Mais à l'inverse, si le buffer est trop court, l'ordinateur arrête de produire des sons en attendant que la fonction de traitement d'image se termine : le son est alors haché, avec des périodes de blanc entre chaque traitement d'image.

Le multithreading a permis la mise en place de la solution suivante: un buffer audio très court, recalculé en permanence et fonction des dernières positions de mains fournies par la fonction gérant la caméra, qui les met à jour chaque fois qu'elle traite une nouvelle image. Les informations échangées sont contenues dans un tableau de 6 entiers, contenant les trois dernières distances (en pixels) connues entre les antennes et les mains correspondantes. Comme la fonction son n'écrit pas dans ce tableau, les risques d'accès concurrents sont minimes et le système fonctionne de façon transparente pour l'utilisateur.

4.3 Un son proche de la réalité ?



La distance calculée par la caméra doit être tranformée en fréquence pour pouvoir être jouée. Pour cela, nous avons tenté plusieurs fonctions. Celles donnant les meilleurs résultats sont des fonctions affines, car l'écart entre deux détections est maîtrisé et la continuité du son est facile à assurer. Cependant, le nombre de notes pouvant être obtenues de cette façon est très réduit, et les notes aiguës difficiles à produire. Nous avons donc tenté une approche basée sur l'instrument originel. Cependant, nous n'avons trouvé aucune étude fournissant une fonction de conversion distance > fréquence. L'information la plus proche trouvée est un graphique représentant cette fonction, dont nous avons expérimentalement trouvé la fonction qui s'avère être :

$$f = \frac{180}{d^{0,6}} - 150$$

Le problème du thérémine est que cette fonction n'est pas linéaire: la fréquence augmente avec l'inverse de la distance, ce qui signifie que plus la main est proche de l'antenne,

plus un petit déplacement aura d'impact sur la fréquence. De ce fait, un changement d'une unité dans la valeur détectée par la caméra peut se traduire par un saut dans la musique jouée, ce qui vient contredire le son continu que nous cherchons à obtenir. Cependant, c'est un problème inhérent à l'informatique et nous sommes conscient que nous ne pouvons faire mieux actuellement. Une piste d'amélioration possible serait d'utiliser une caméra avec une résolution supérieure, mais l'impact sur le temps de traitement engendrerait d'autres problèmes.

Un défaut plus gênant concerne la fiabilité de la détection : en effet, celle-ci peut varier de quelques pixels entre deux images même quand l'utilisateur ne fait (ou n'a l'impression de ne faire) aucun mouvement. Pour palier à ce problème, nous gardons en mémoire les trois dernières détections et jouons la fréquence correspondant à la moyenne de ces détections. La différence de réactivité n'est pas perceptible à moins de vouloir jouer rapidement plusieurs notes éloignées les unes des autres, ce qui nécessite un niveau de pratique que nous n'avons pas ; en revanche, le gain de continuité est notable, particulièrement dans les fréquences graves et moyennes.

Toutefois, nous nous sommes autorisés deux entorses à cette fonction en bloquant la fréquence entre $55Hz$ et $1760Hz$. Cela laisse une amplitude de cinq octaves similaire à celle d'un piano, et permet à la fois d'éviter des sons aigus trop difficiles à reproduire, et des sons trop graves où notre fonction ne reproduit plus l'instrument de façon crédible.

Conclusion

L'objectif de ce projet était de mettre en pratique les compétences acquises en traitement d'image et de reproduire le fonctionnement d'un thérémine de manière entièrement informatisée. En chemin, cela nous a permis d'acquérir quelques notions supplémentaires de gestion du son et de multithreading.

Si le résultat obtenu n'est pas encore pleinement satisfaisant d'un point de vue musical (personne ne le confondrait avec un vrai instrument), nous avons tout de même réussi à générer de façon convaincante un programme fluide et réactif. La détection des mains n'est pas parfaite et le port de gants d'une couleur particulière est toujours nécessaire pour avoir des détections correctes, mais nous sommes satisfaits d'avoir obtenu ces détections sans utiliser de technique de machine learning, une restriction que nous nous étions nous-mêmes fixé.