Projet de sonification d'interface pour PSA

Romain Le Bail – Nathalie Lurin

Encadrants Entreprise : Vincent Roussarie

Sébastien Denjean

Encadrant ECN : Jean-François Petiot

Option PDS 2014-2015

Table des matières

Introduction

Ce projet de sonification d'interface est effectué en collaboration avec le département de recherche et développement de PSA. Il s'agirait d'améliorer le système de commande de certaines fonctions dans l'habitacle d'une voiture telles que l'autoradio, le GPS etc… La tendance voudrait que de plus en plus de fonctions soient commandées à l'aide d'un écran tactile à la droite du conducteur. Pour une question de sécurité, il semble avantageux que le conducteur puisse les actionner sans avoir à détourner son regard de la route, PSA a donc pensé à intégrer un système de reconnaissance de gestes. Le conducteur n'aurait donc qu'à bouger sa main dans la zone prévue à cet effet pour commander certains utilitaires. Le problème est alors de fournir à l'utilisateur un retour sonore sur ses gestes, lui permettant de savoir si l'action qu'il voulait a bien été effectuée.

I- Présentation du projet

1- Demande

Il nous a été demandé, dans le cadre de ce projet d'étudier les problèmes suivants :

- Connexion entre le capteur vidéo et le logiciel de production du son
- Mapping sonore d'un nombre réduit de gestes
- Proposition de sons adaptés au problème

Il s'agit donc de proposer un prototype fonctionnel permettant de reconnaitre un petit nombre de gestes proposés par PSA et de faire le design de sons modulables permettant un retour sur ces gestes. Le système doit être simple et intuitif afin de nécessiter un apprentissage minimal de la part de l'utilisateur, sans quoi il ne serait pas fonctionnel. Les gestes à sonoriser sont les suivants :

- Swipe : mouvement rectiligne du doigt dans les directions droite/gauche et haut/bas
- Cercle : pointe du doigt décrivant un cercle
- Keytap : mouvement d'appui sur une touche
- Screentap : mouvement d'aller-retour du doigt pointant vers l'écran

Il faut par ailleurs guider l'utilisateur vers la zone de détection et indiquer sa sortie de la zone.

Les problèmes d'adéquation des sons à la marque ainsi que ceux du masquage possible des sons par les bruits de l'environnement (bruits de moteur, frottements de l'air…) seront mis de côté pour ce projet. Ce sont des problèmes qui seront à considérer à un stade plus avancé du projet : le retour sonore doit être efficace et le rendu des sons est toujours différent dans un contexte bruyant.

2- Outils

Le capteur

Le capteur à notre disposition est un capteur « Leap Motion ». Ce capteur comprend 2 capteurs vidéo et 3 leds infrarouge. Il a été développé pour permettre le pilotage d'un écran à l'aide des mains. Il se place à plat, leds dirigées vers le plafond. La connexion avec l'ordinateur se fait par l'intermédiaire d'un port USB. D'une précision de l'ordre du millimètre, la Leap Motion est capable de repérer en temps réel la position d'une (ou plusieurs) main(s) : os, articulations et avant-bras. La

détection peut se faire à une distance de 2 à 60cm au-dessus du capteur et dans un cône d'angle 150° au-dessus du Leap Motion.

Le Leap Motion est livré avec une interface de programmation (API) disponible en 6 langages (C++, C#, Unity, Objective-C, Java, Python, JavaScript et Unreal Engine). Elle permet de récupérer des mesures physiques telles que des distances en millimètres, des vitesses en millimètres pas seconde et des angles en radians. Le Leap Motion envoie des objets « frame » correspondant à l'état des mains à un instant donné : positions et vitesses des différents composants de la main et gestes reconnus par exemple. Les gestes reconnus sont les suivants : swipe, cercle, keytap et screentap.

Pure Data

Pour ce projet, les sons seront commandés à l'aide du langage Pure Data, langage de programmation visuelle adapté au traitement du son. C'est un équivalent libre de Max/MSP. Il s'agit donc de relier des objets et fonctions entre eux afin d'arriver au résultat désiré. Pure Data exécute en continu le programme et les données peuvent être passées à deux fréquences différentes : 44100Hz pour les données de type « signal » et à une fréquence plus faible pour les autres types de données. Il existe deux modes d'utilisation qui permettent d'interagir avec les objets de manière différente : le mode édition et le mode exécution. Pure Data est adapté à la fois à la production de sons et à leur modulation en temps réel.

Bases de données sonores libres

Pour nous procurer des sons de base nous avons trouvé un certain nombre de bibliothèques de sons libres de droit :

https://www.freesound.org

http://www.universal-soundbank.com/

http://soundbible.com/free-sound-effects-1.html

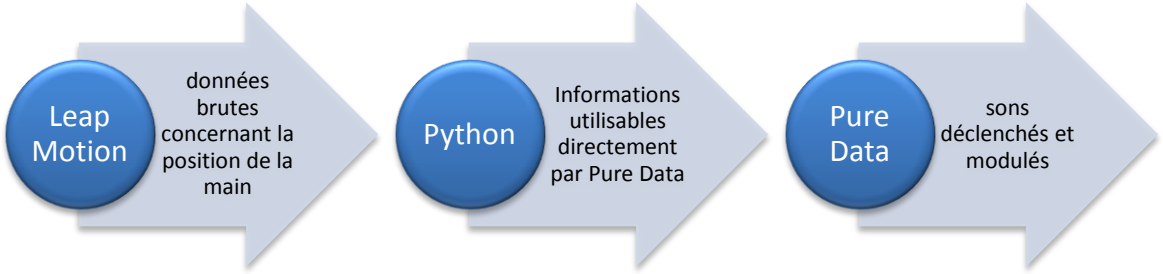
http://www.grsites.com/archive/sounds/

http://www.soundsnap.com/

Ces bases de données permettent des recherches par mots-clefs de sons enregistrés par d'autres utilisateurs. Les sons de sont pas toujours très propres mais ils donnent une idée de ce que pourrait être le rendu final.

3- Solution générale proposée

Notre solution se décompose de la façon suivante :



Notre solution comporte de plus deux thèmes sonores différents qui consistent en 4 sons chacun. Chaque thème se veut cohérent et permettre une ambiance sonore différente.

II- Mapping et choix des sons

Nous avons choisi de présenter 3 types de sons :

- Des sons ponctuels déclenchés par un événement
- Des sons ponctuels déclenchés par un événement et modulés par certains paramètres
- Un son de zone destiné à informer l'utilisateur de sa position dans la zone de détection

Nous allons présenter dans cette partie les gestes pour lesquels ils ont été choisis, les caractéristiques qu'ils nécessitent et la manière dont ils ont été implémentés en Pure Data.

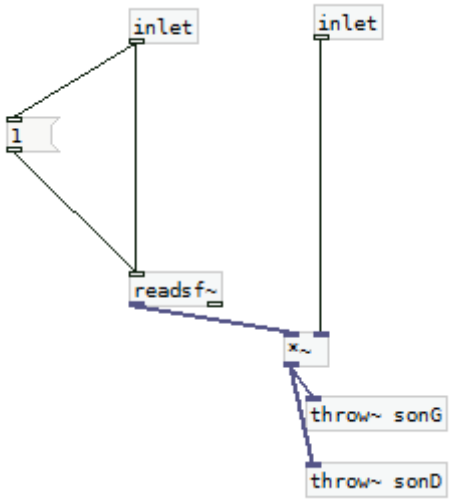
1- Sons ponctuels

Son de validation

Nous avons pensé accompagner les gestes keytap et screentap d'un son de validation. Ce sont en effet des gestes courts qui permettent de valider un choix ou de choisir un élément. Il pourrait être nécessaire par la suite de changer le son en fonction de la fonction concernée, mais un son de validation commun paraît un bon départ. Ce son est aussi déclenché lorsqu'un cercle est terminé.

Un son de validation ponctuel doit être court et avoir une consonance positive. On se dirige donc vers un son relativement aigu et cristallin, il doit être le plus intuitif possible.

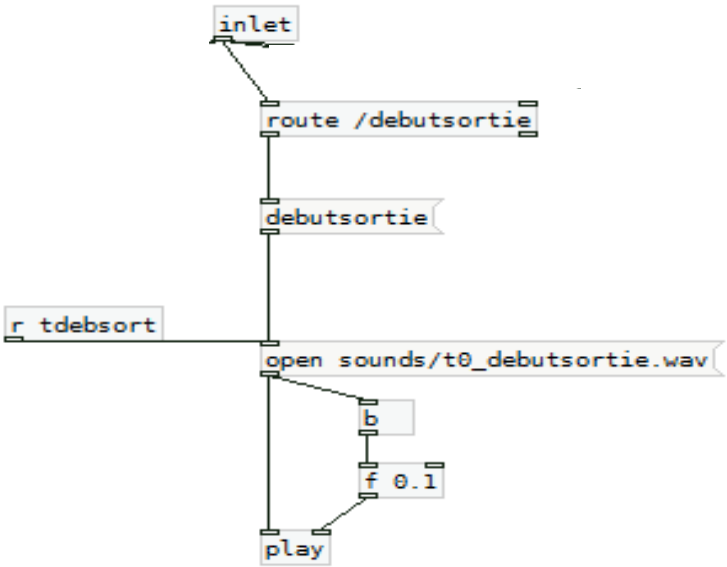
Ce son doit être déclenché lorsque Pure Data reçoit l'information qu'un geste de keytap ou de screentap a été effectué. Le fichier correspondant doit alors être lu :



Cette fonction prend en entrée un message indiquant le message à lire, l'envoie à la fonction readsf~ et envoie le son résultant à la sortie Dac~ par l'intermédiaire de la fonction throw.

Son de sortie de la zone

Nous avons pensé qu'il était important de donner un retour à l'utilisateur sur sa sortie de la zone : si, lorsqu'un utilisateur effectue un geste, il sort de la zone, il risque une mauvais détection du geste, il est donc intéressant de l'en informer. Pour ne pas surcharger le dispositif, le son doit être court et intuitif. Il en sort que le son doit être sec et mat et transmettre une impression négative.

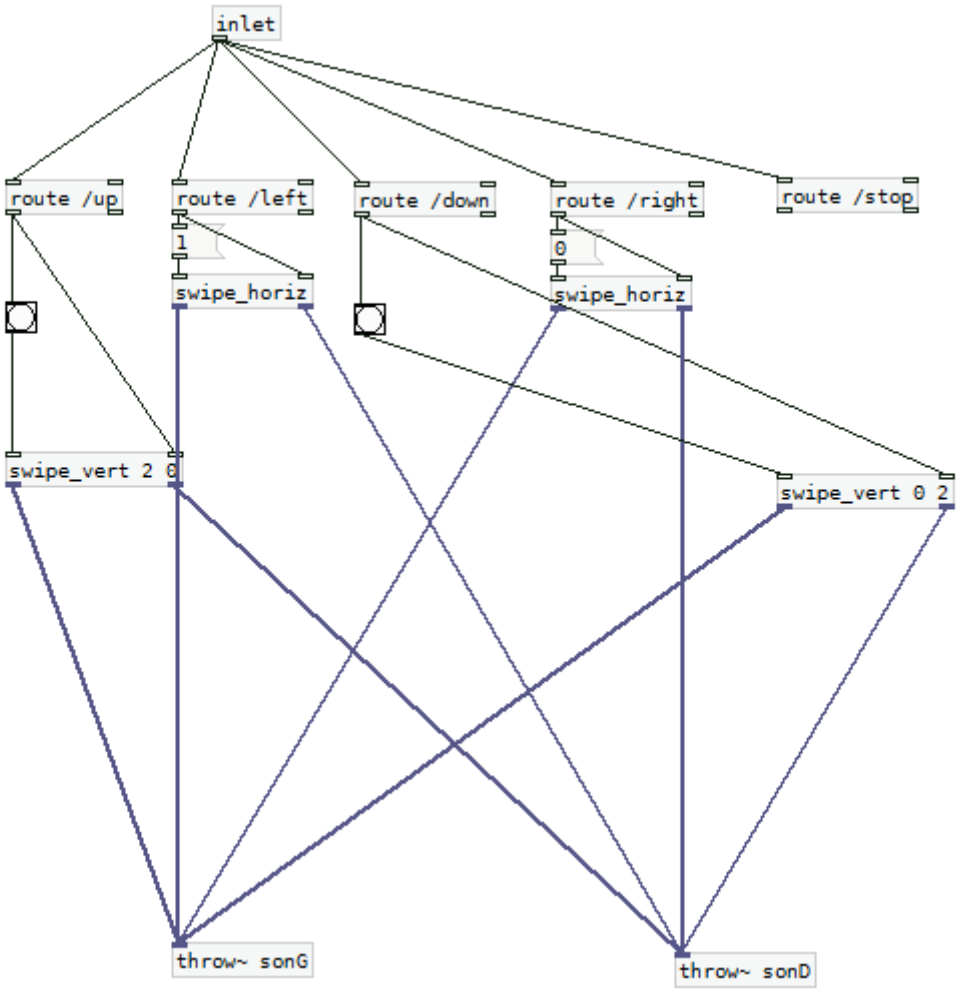


Le son de sortie de zone est inclus dans le patch chargé de s'occuper des sons en rapport avec la zone de détection. Une fois l'événement reçu, le patch reçoit le nom du fichier à ouvrir et le lit.

2- Sons modulés

Swipe

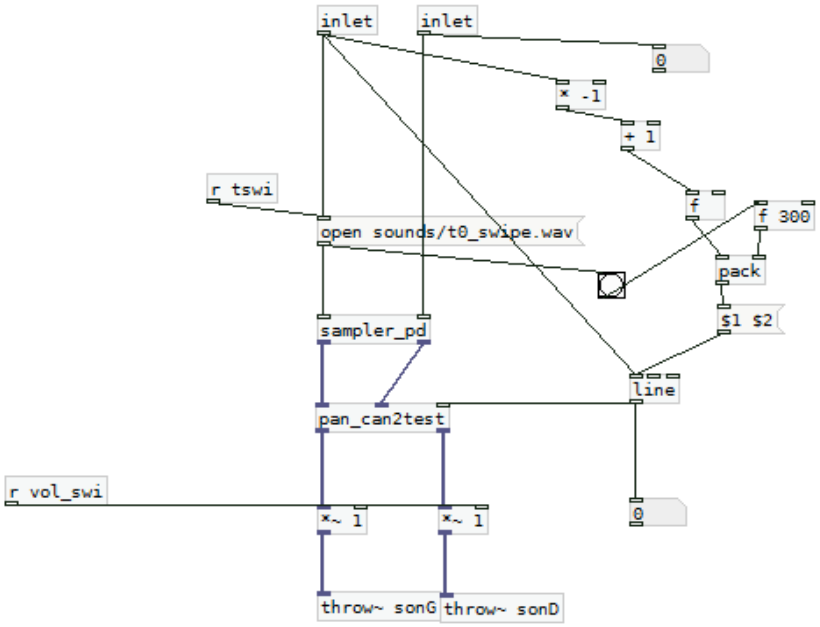
Le swipe est un mouvement rectiligne dans la direction haut/bas ou droite/gauche. Le geste n'étant pas trop court, nous avons pensé intéressant de pouvoir le moduler. L'information à sonoriser est la direction du swipe. Pour donner davantage de vie au dispositif, une modulation en fonction de la vitesse d'exécution du geste semble possible : on a ainsi un retour en temps réel sur le geste. La structure de patch Pure Data dédié au swipe est la suivante :



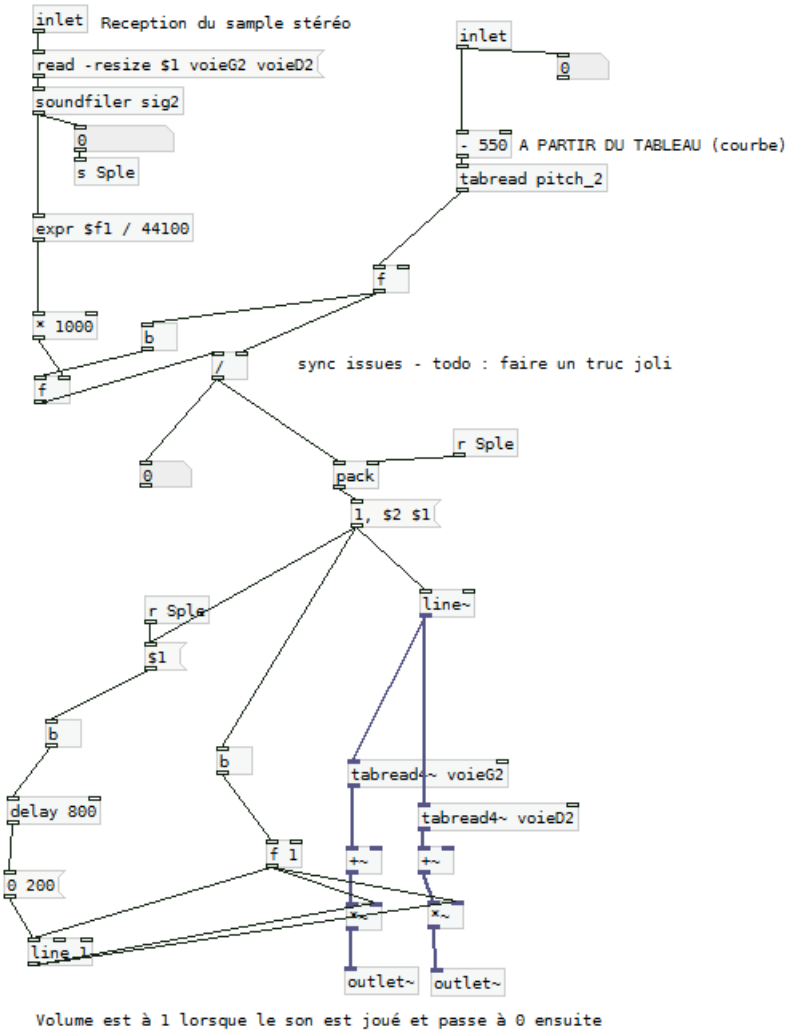
Il appelle deux abstractions, une pour les swipe horizontaux et une pour les swipe verticaux.

Nous avons choisi une spatialisation de droite à gauche pour indiquer la direction d'un swipe horizontal : un swipe de droite à gauche entraîne un mouvement ressenti du son de la droite à la gauche. La spatialisation a été réalisée à l'aide de deux patchs Pure Data fournis par PSA : un patch de panning (variation de volume entre les deux oreilles) (*pan_can2test.pd*) et un patch de delay (variation du temps d'arrivée du son entre les deux oreilles) . La vitesse de lecture du son est ensuite modifiée par la vitesse initiale du geste. L'abstraction *sampler.pd* est inspirée de deux patchs fournis par PSA : le patch de delay et un sampler permettant de changer la vitesse de lecture du son.

swipe_horiz.pd :

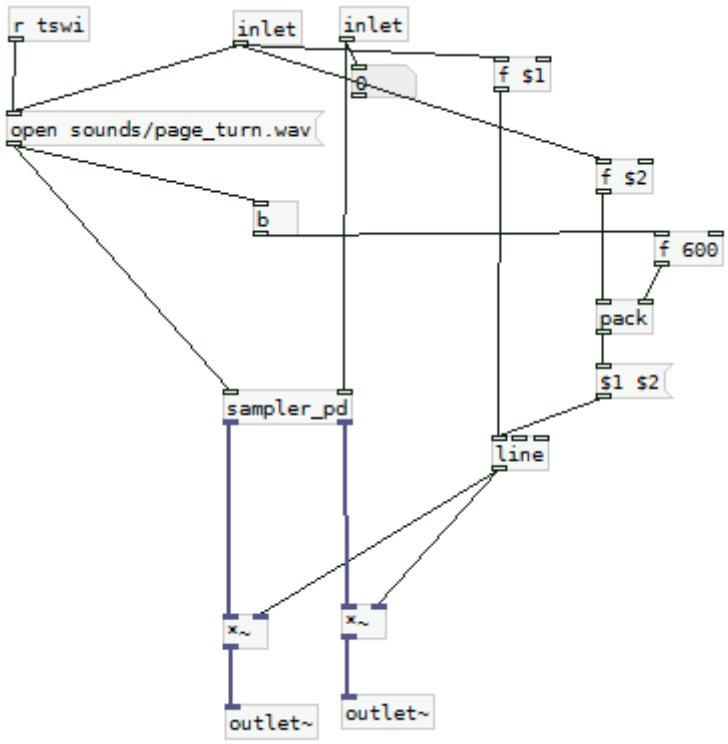


sampler.pd :



Nous avons choisi une modulation de volume pour indiquer la direction haut/bas : le volume du son augmente lorsque le geste est dirigé vers le haut et descend lorsque le geste est dirigé vers le bas. C'est une modulation assez évidente, la deuxième à laquelle nous avions pensé était la modulation en fréquence : la fréquence pourrait augmenter lorsque le geste monte et descendre dans le cas contraire. Cette modulation à l'avantage d'être plus facile à détecter pas l'oreille humaine qui n'est pas très sensible au volume. Cependant elle est beaucoup plus compliquée à mettre en œuvre sous Pure Data.

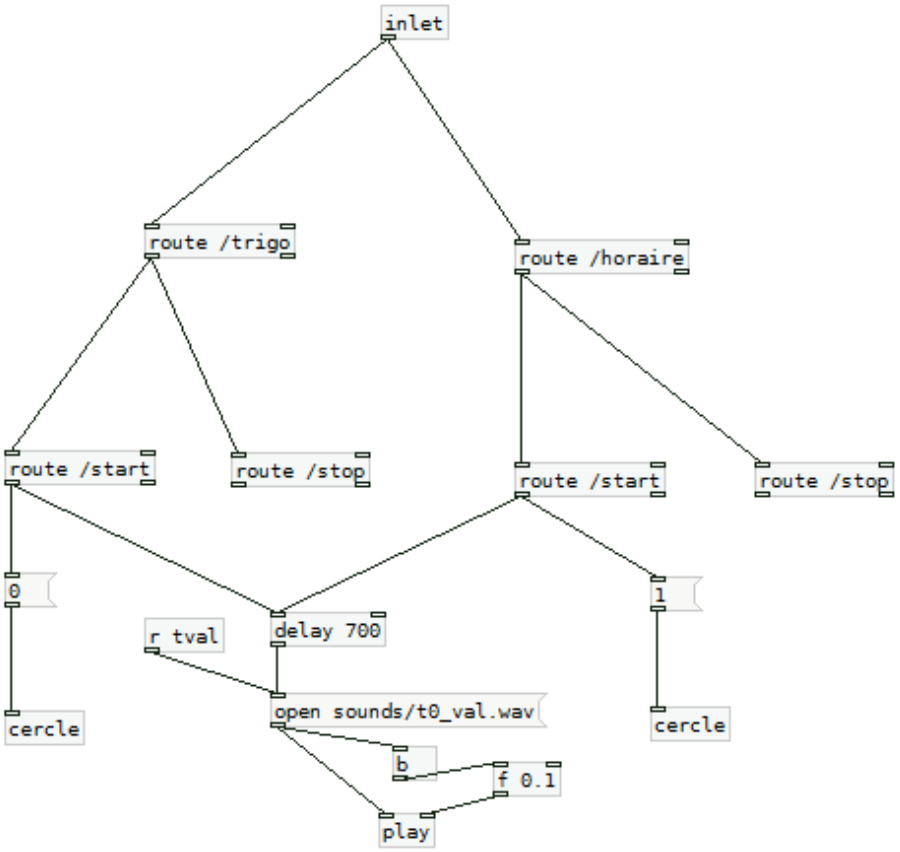
La mise en œuvre de la modulation choisie est faite dans l'abstraction *swipe_vert* de la façon suivante :



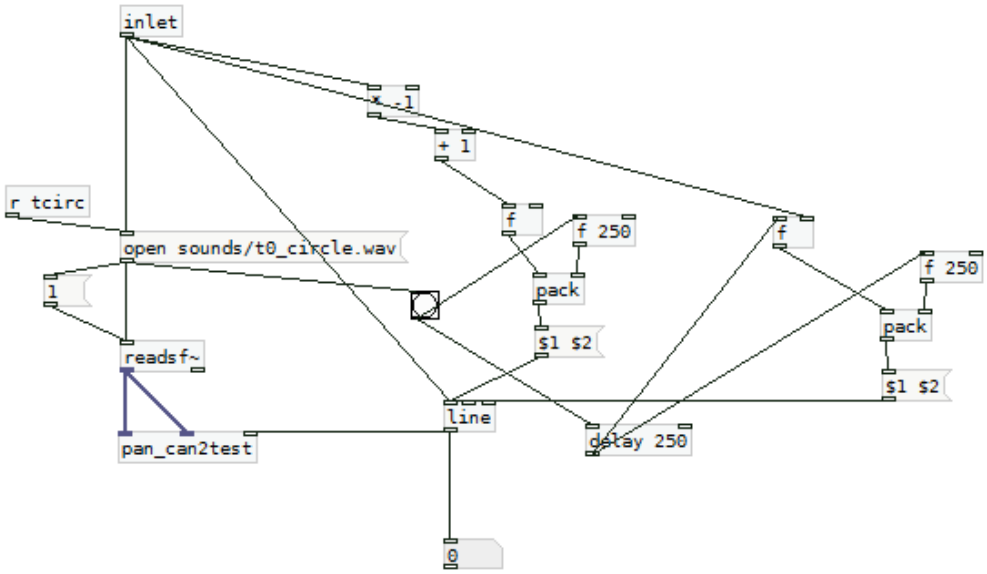
L'abstraction reçoit le nom de fichier à lire et la vitesse initiale du geste. Elle prend en argument les paramètres de la rampe en volume à effectuer (rampe descendante si le swipe est vers le bas par exemple. La vitesse de lecture est modifiée par *sampler_pd* et l'amplitude est ensuite multipliée par la rampe formée.

Cercle

Nous avons pensé que le cercle aussi pourrait bénéficier d'une spatialisation : il faut transmettre l'idée d'un retour au point de départ. Nous avons donc choisi que le cercle trigonométrique serait associer d'une spatialisation droite/gauche/droite tandis que le cercle horaire serait spatialisé gauche/droite/gauche. Par ailleurs nous avons une fois de plus fait varier la vitesse de lecture du fichier en fonction de la vitesse initiale du geste. Le schéma général est le suivant :



Pure Data reçoit l'information de l'exécution d'un cercle par l'utilisateur. Le son du cercle dépend du message reçu par l'abstraction *cercle* (0 ou 1). On envoie alors une double rampe à *pan_can2test* Pour avoir un aller-retour dans la spatialisation. Les paramètres de ces rampes ont été choisis de manière à ce que le rendu soit le plus adapté aux sons que nous avons choisis.



3- Son de zone

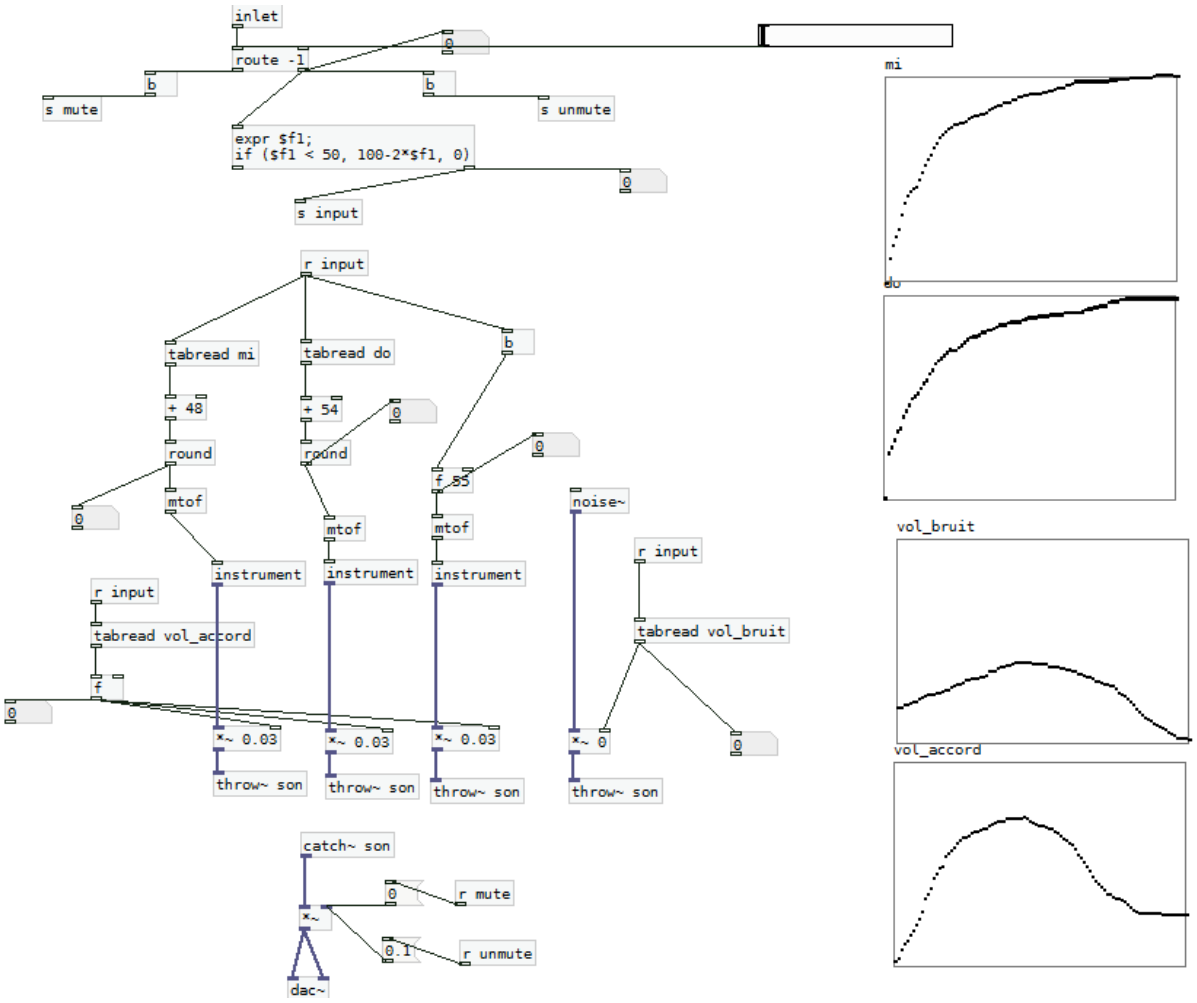
L'une des problématiques les plus importantes du projet est celle du son de zone : il faut indiquer de manière sonore à l'utilisateur où il se situe par rapport à la zone de détection afin qu'il n'ait pas à quitter la route des yeux. Il y a plusieurs stratégies possibles.

Une première possibilité est de modulé un son de manière à ce que l'utilisateur sache la direction dans laquelle bouger sa main pour entrer dans la zone. Le son reste présent mais de volume plus faible lorsque l'utilisateur est dans la zone de détection et l'informe de son éloignement. Le passage hors zone/in zone est très rapide, la modulation sera à adapter précisément. Une autre possibilité est alors de guider l'utilisateur vers une zone plus réduite de la zone de détection : on informe l'utilisateur de sa position optimale dans la zone et la modulation se fait sur une distance plus importante. Cependant la zone de détection risque de se trouver au-dessus du levier de vitesse, zone dans laquelle l'utilisateur put avoir à mettre la main régulièrement, il vaudrait donc peut-être mieux arrêter le son une fois l'utilisateur dans la zone et ne le re-déclencher que s'il commence à en sortir.

La solution que nous avons implémentée est la première. Notre son de zone est par ailleurs commun aux deux thèmes sonores proposés. Le principe est un accord de 3 sons dissonants tendant vers un accord consonant lorsque l'utilisateur se dirige vers la zone. Nous avons de plus rajouté un bruit blanc dont le volume diminue avec le volume de l'accord et s'éteint lorsque la main est entièrement dans la zone. Le mouvement des notes de l'accord est contrôlé par la lecture d'une courbe de la fréquence en fonction de la distance. Les accords sont joués par un instrument défini par la synthèse additive d'un certain nombre d'harmoniques.

Pour pouvoir adapter le son de zone aux deux thèmes avec la troisième possibilité, nous avons pensé à un son de train à vapeur ou de de cigale avec une double modulation en vitesse et fréquence : la fréquence diminuerait tandis que le volume atteindrait un maximum aux limites de la zone avant de s'arrêter progressivement, à la manière de l'arrivée en gare d'un train.

Zone.pd :



Pour pouvoir adapter le son de zone aux deux thèmes avec la troisième possibilité, nous avons pensé à un son de train à vapeur ou de de cigale avec une double modulation en vitesse et fréquence : la fréquence diminuerait tandis que le volume atteindrait un maximum aux limites de la zone avant de s'arrêter progressivement, à la manière de l'arrivée en gare d'un train.

4- Thèmes sonores et sons choisis

La problématique est de proposer des thèmes sonores pouvant se décliner en plusieurs sons simples et intuitifs pour l'utilisateur et adaptés au contexte : les sons ne doivent pas déranger l'utilisateur mais plutôt l'accompagner dans son utilisation du véhicule. Les sons appartenant à chaque thème doivent de plus être faciles à différencier.

Nous proposons donc les deux thèmes sonores suivants :

Thème « mécanique »

Nos actions de tous les jours sont associées au bruit des machines/outils/appareils avec lesquels nous les effectuons. Il nous semble donc naturel et intuitif d'associer à chaque geste un bruit correspondant à l'équivalent mécanique du geste effectué. Par ailleurs ce thème permet d'évoquer l'histoire de l'automobile pour passer d'un engin purement mécanique aux véhicules informatisés d'aujourd'hui.

Type de son	Son proposé	Justification
Swipe	Mécanisme d'horloge	Le son est celui d'un engrenage donc une série de cliquetis métalliques. Il transmet assez bien l'idée d'une progression, du défilement d'un menu par exemple. Par ailleurs, il s'adapte très bien à un changement de vitesse de lecture.
Cercle	Manivelle	Le son est celui d'une manivelle mal huilée en train d'être tournée. La manivelle semble l'élément mécanique le plus intuitivement associé à une rotation.
Keytap/screentap	Cloche de machine à écrire	Il nous fallait un son cristallin transmettant une idée de validation/succès. Ce son n'est peut-être pas le plus adapté au thème « mécanique », mais il semble très intuitif.
Sortie	Bois brisé	Le son de sortie doit être assez sec et négatif d'où l'idée du bruit d'un objet cassant ou d'un coup. On pourrait aussi choisir le bruit d'un choc dans un mécanisme.

Thème « papier »

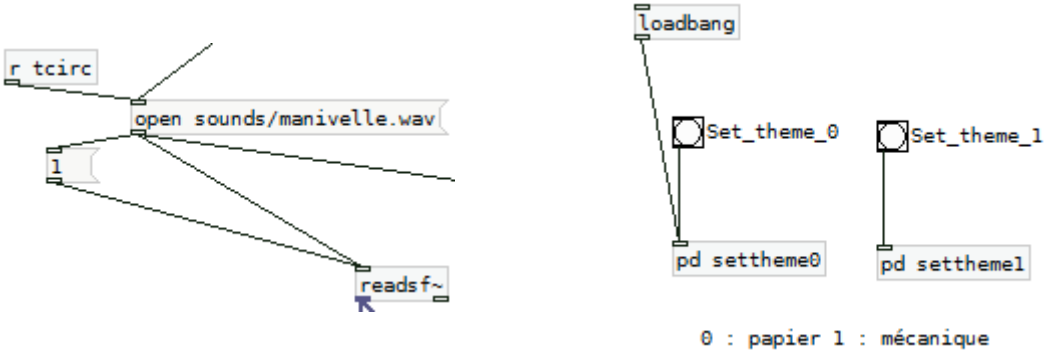
Le papier et l'écriture sont des éléments de nos vies de tous les jours. Des bruits tels que celui d'un page se tournant ont déjà été utilisés dans de nombreux contextes car ils semblent aussi très intuitifs. Pour nous l'idée d'un thème centré sur le papier apporterait une idée de confort car ce sont des sons chaleureux et organiques. Il transmettrait aussi l'idée de luxe : de nos jours les images d'un bon papier ou d'un stylo plume sont de ce registre, n'étant plus autant utilisés qu'avant.

Type de son	Son proposé	Justification
Swipe	Page tournée	Le son est celui d'un page de livre qui se tourne. Ce son est facilement identifiable et il transmet bien l'idée de passer d'une page à une autre donc d'un élément d'un menu au suivant.
Cercle	feutre	Le son est celui d'une d'un feutre traçant un cercle sur une feuille de papier. On reconnaît aisément le bruit d'un dessin. Le cercle semble le geste le plus adapté à ce son, étant lui aussi un dessin dans l'espace.

Keytap/screentap	agrafeuse	Une fois de plus, c'est un son facilement identifiable. C'est le bruit de l'appui sur un appareil (ici une agrafeuse), il est donc adapté à sonoriser un geste de keytap et ses sonorités positives en font un bon candidat pour tous les sons de validation.
Sortie	Papier déchiré	Les deux sons les plus négatifs en rapport avec le papier auxquels nous avons pensé sont les sons de papier déchiré ou froissé. Le son de papier froissé s'apparentait trop au bruit blanc à l'écoute et nous semblait trop dur à identifier pour être intuitif.

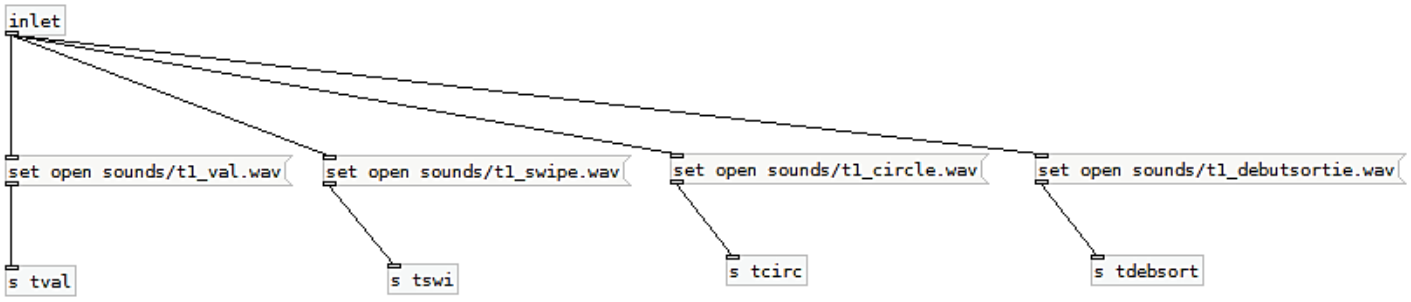
Passage d'un thème à un autre

Pour que l'utilisateur puisse choisir l'ambiance sonore qu'il préfère, ainsi que pour faciliter la comparaison de différents thèmes sonores nous avons implémenté dans Pure Data un interrupteur



permettant de passer de l'un à l'autre. Le rajout d'un thème peut de plus se faire aisément :dans Pure Data un interrupteur permettant de passer de l'un à l'autre. Le rajout d'un thème peut de plus se faire aisément :

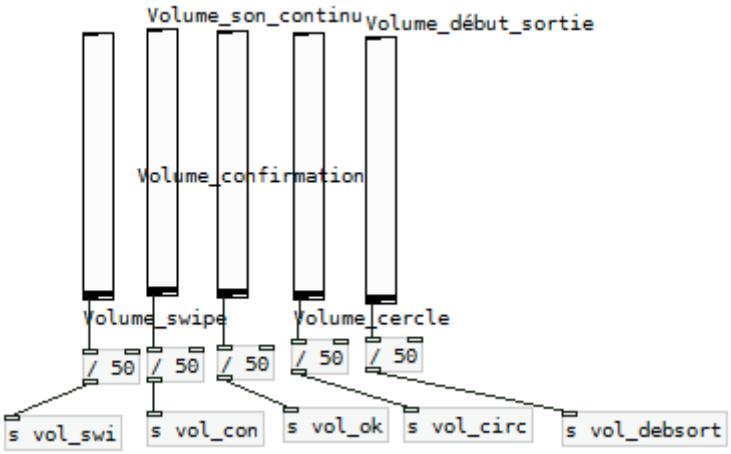
Il suffit pour changer de thème de choisir l'interrupteur correspondant, celui-ci envoyant un « bang » à un sous-patch « setthemei ». Ce sous-patch envoie des messages contenant les noms des différents fichiers correspondant aux sons du thème aux fonctions concernées par l'intermédiaire d'un bloc « send » dans des variables communes à l'ensemble du programme. Les fonctions lisant les sons contiennent donc des blocs « receive ». Ci-dessous le sous-patch « settheme1 » et un exemple de réception du message :



La variable tcirc change le contenu du message qui permet d'ouvrir le fichier et de lire le son.

5- Contrôle du volume

Afin de pouvoir faire des ajustements rapidement et d'avoir un rendu agréable, nous avons rapidement réalisé sous Puredata un mixer pour les différents sons :



Ce mixer permet d'ajuster indépendamment les niveaux du son continu indiquant la distance à la zone, du son de confirmation, des deux sons d'évènements complexes (cercle et swipe) et enfin de régler le volume du son de sortie.

Encore une fois, l'utilisation des objets send et receive sous puredata rend très facile la réalisation du mixer.

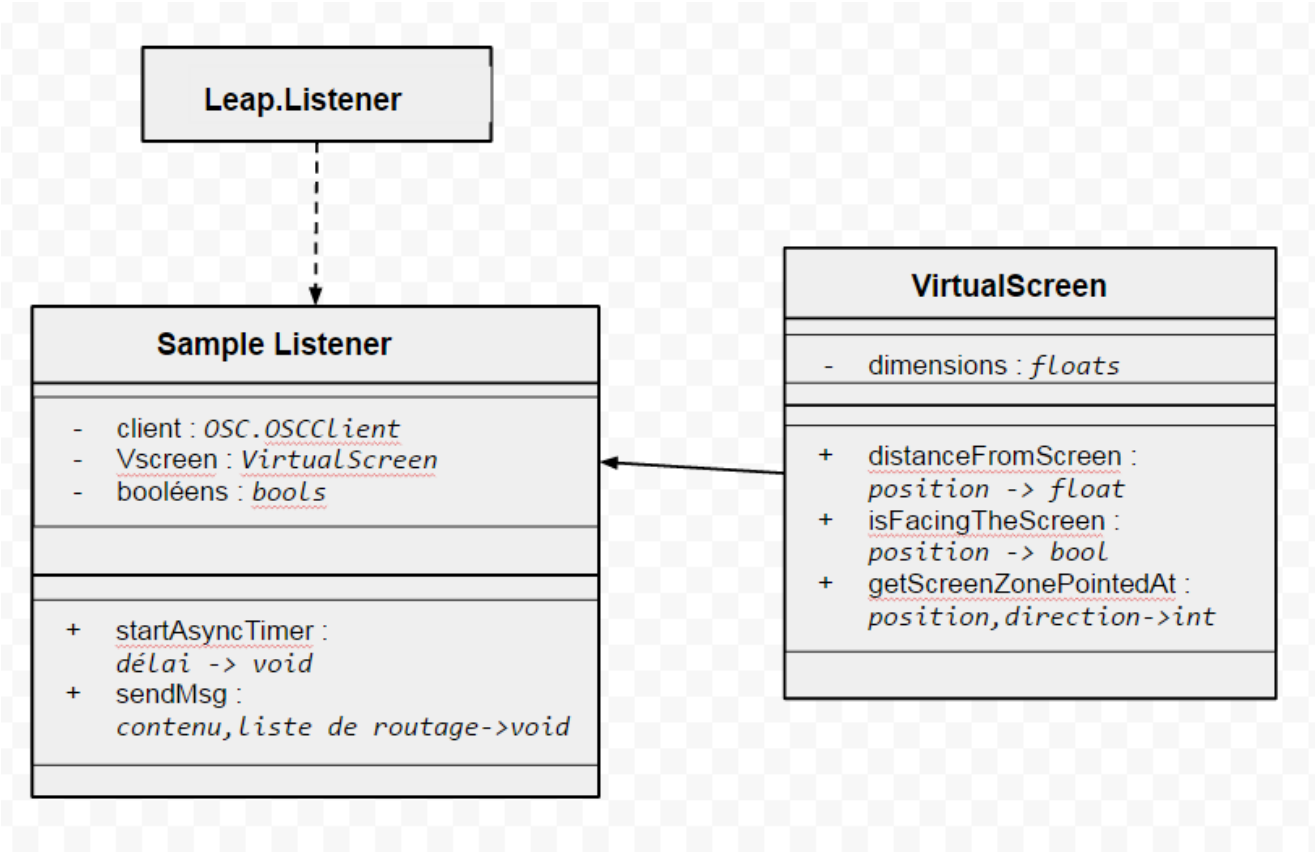
III- Récupération des informations de la Leap Motion

1- Présentation générale

Au cours de cette partie, nous nous efforcerons de présenter la partie du projet écrite en Python, sans rentrer autant dans les détails techniques que précédemment; en effet, bien qu'importante, cette partie était plus une nécessité pour pouvoir travailler le son qu'un véritable objectif du projet et est moins liée à l'option Perception et Design sonore que la partie précédente.

Si un lecteur intéressé voudrait se référer au code brut, il est disponible à cette adresse avec l'intégralité du projet : https://github.com/IIazertyuiopII/PDS_sonification , téléchargeable et réutilisable sans aucune restriction.

Ci-dessous un diagramme de classe dans lequel quelques libertés ont été prises avec la syntaxe habituelle, par souci de clarté et de simplicité de lecture; au cours de cette partie, il sera fait de nombreuses référence à ce diagramme afin d'en expliquer chacun des éléments.



Notre classe principale hérite donc de Leap.Listener, ce qui nous permet d'avoir accès à toutes les fonctions prédéfinies de l'API Python du Leap, qui est par ailleurs très bien documentée sur le site du constructeur (https://developer.leapmotion.com/documentation/python/index.html). Nous nous intéresserons ici uniquement aux classes que nous avons écrites.

Afin de répondre à nos besoins, nous avons fait deux ajouts majeurs à la classe Listener par défaut du Leap, qui permet grosso modo de recevoir les informations des capteurs et d'exécuter du code dans un nouveau thread chaque fois qu'une image arrive du LeapMotion (environ 100 fois par seconde).

Premier ajout : communication avec Puredata

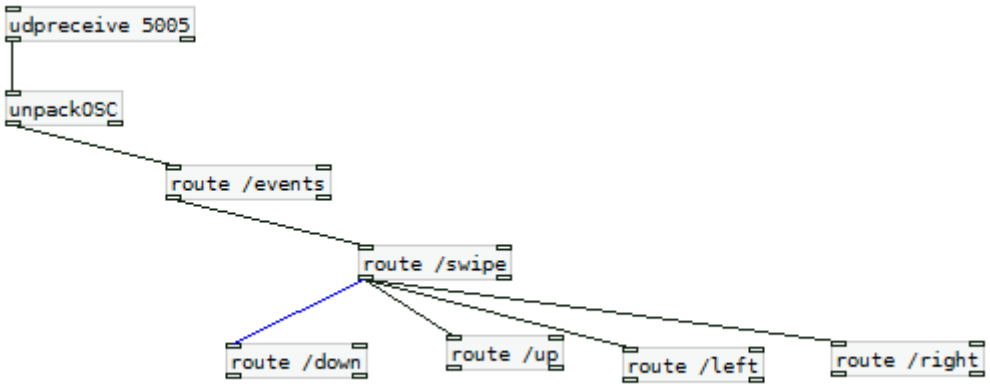
Bien évidemment, nous avons besoin de transmettre des informations à l'étage de synthèse sonore. Pour ceci, nous nous sommes appuyés sur le protocole OSC (open sound control), implémenté en de nombreux langages et qui permet de définir un standard pour l'encodage des données binaires et leur transmission par UDP (protocole de transfert de données permettant à deux entités possédant une adresse IP de communiquer entre elles).

Ce premier ajout correspond à l'attribut client dans le diagramme de classe; client appartient à la classe de la librairie OSC utilisée pour le projet : nous n'avons pas eu besoin de retoucher cette librairie pour l'adapter à nos besoins.

En outre, nous avons codé une méthode sendMSG afin de disposer d'une syntaxe agréable pour envoyer des messages : avec cette fonction, l'envoi d'un message devient :

```
self.sendMSG(velocity,["events","swipe",swipeDirection])
```

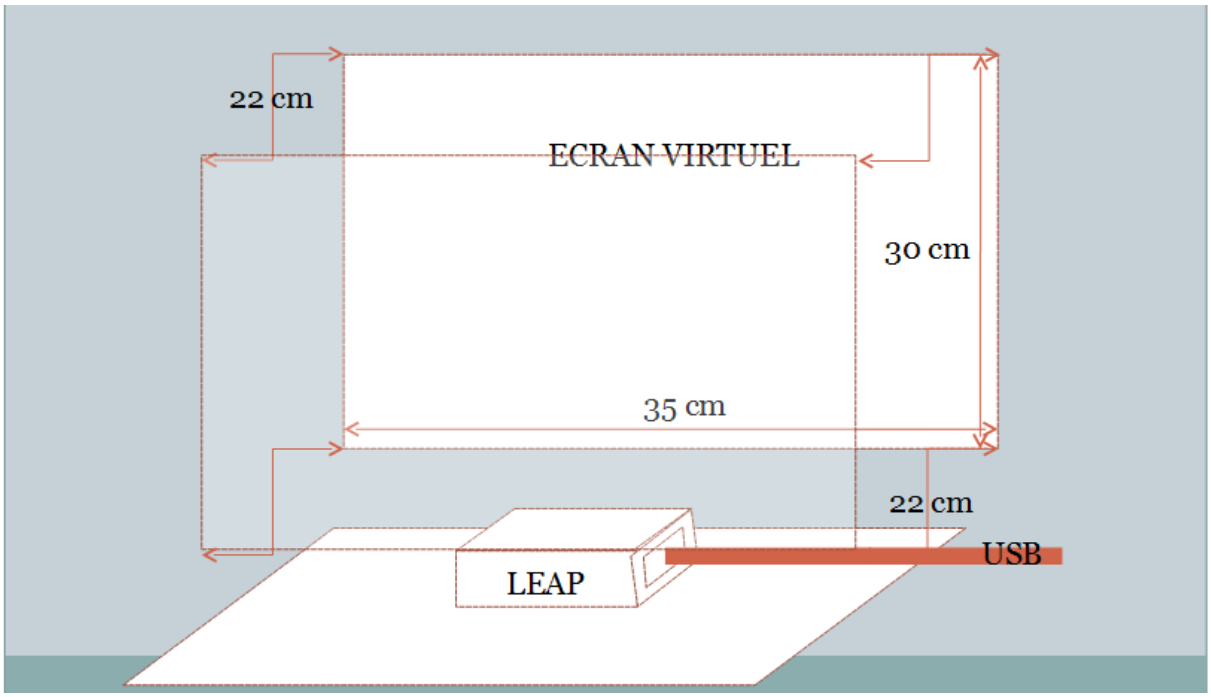
et sa réception (si swipeDirection peut valoir up, down, left, right) :



On voit clairement ici l'intérêt de la syntaxe qui permet de rajouter autant de paramètres de routage que l'on souhaite, afin de regrouper les messages en catégories et la simplicité de la réception sous puredata grâce à l'usage de l'objet route.

Deuxième ajout : prise en compte de la zone

La deuxième fonction importante que nous avons ajoutée au LeapMotion est la prise en compte d'un écran virtuel placé devant celui-ci :



Cet écran correspond à la classe VirtualScreen sur le diagramme de classes, et donc à l'attribut VScreen de notre Listener amélioré.

Il permet les fonctions suivantes :

- Savoir si un point est dans la zone en face de l'écran (fonction isFacingTheScreen)
- Savoir la distance d'un point à l'écran (fonction distanceFromScreen)
- Savoir quelle zone l'utilisateur pointe avec son doigt (fonction getScreenZonePointedAt)

Le principal intérêt de cette classe est d'envoyer directement des informations de contrôle des sons à Puredata (booléens, entiers) et non des données brutes du Leapmotion, ce qui évite de devoir faire les calculs sous Puredata, qui est bien moins adapté pour ceci que Python.

Avec ces deux ajout majeurs, l'API du LeapMotion est presque convenable pour commander les sons qui nous intéressent et donc nous consacrer à la partie design sonore abordée précédemment, à un détail près que nous allons présenter dans la partie qui vient.

2- Gestion des évènements

L'API de base du LeapMotion contient les quatre gestes qui nous intéressent (c'est d'ailleurs pour celà que nous les avons choisis) : swipe, cercle, keytap et screentap. Cependant, la fiabilité de la détection est mauvaise de base, et ce même en ajustant les paramètres de détection : soit les paramètres sont trop sensibles et le Leapmotion envoie beaucoup de faux positifs, soit ils ne le sont pas assez et il devient très difficile d'être reconnu de manière consistante.

La solution simple implémentée

Notre solution devait être rapide à implémenter, car nous n'avions pas le temps de nous consacrer à une véritable amélioration des évènements du LeapMotion, ce qui aurait en effet constitué un projet à part entière, voire plus.

Elle repose sur deux principes :

- Régler le Leapmotion sur des paramètres très sensibles et empêcher les faux positifs par l'ajout d'un timer de 300ms à chaque fois qu'un évènement est déclenché.

Ainsi, les gestes de l'utilisateur sont bien reconnus une et une seule fois; la gestion du timer s'appuie sur les fonctions startAsyncTimer et les différents booléens de la classe SampleListener : un thread est lancé à chaque appel du timer, qui met un booléen (attribut de SampleListener) à true, attend 300 ms puis le remet à false.

Par conséquent, quand la fonction on_frame de SampleListener (cf API de base) veut savoir si elle doit déclencher un évènement, il lui suffit de regarder la valeur de ce booléen et de ne rien faire s'il est vrai, puisque cela signifie que le timer est en cours.

- S'appuyer uniquement sur l'index de l'utilisateur pour déclencher les évènements

Le fait de ne considérer que l'index permet de limiter les faux positifs et de simplifier le processus d'apprentissage pour l'utilsateur; en outre, il est tout à fait naturel de faire des gestes avec l'index tendu, et ce n'est donc pas une contrainte lourde d'imposer à l'utilisateur d'avoir l'index visible lorsqu'il souhaite faire des gestes.

De plus, suivre la position du bout de l'index permet d'avoir un point supplémentaire utile pour améliorer la fiabilité de la détection de l'entrée/sortie de la zone ou encore d'avoir des informations de vitesse plus fiables lors des swipes.

Une autre solution étudiée : réseau de neurones

Il pourrait être intéressant de désactiver la reconnaissance rudimentaire de gestes proposée par l'API du Leap (au lieu de tenter de l'améliorer tant bien que mal comme nous l'avons fait) pour la remplacer par une solution plus puissante. Dans cette optique, nous avons écrit un script de capture de données pour l'apprentissage d'un réseau de neurons, ainsi qu'un utilitaire court permettant de simplifier la manipulation de fichiers .pkl (sérialisation de classes Python) dans ce cadre précis.

Les données capturées correspondent à 30 valeurs de position du bout de l'index et 30 valeurs de direction (en 3D) de ce même index, ce qui serait à priori suffisant pour reconnaître un grand nombre de gestes plus ou moins complexes.

Le fonctionnement hypothétique serait le suivant : une fois le réseau entraîné sur ces 2*30 valeurs (couche d'entrée de 60 neurones), à chaque fois que le LeapMotion envoie une nouvelle image, les 30 dernières images seraient envoyées au réseau de neurones pour qu'il indique s'il détecte un mouvement connu sur les 30 dernières ms (environ puisque le Leapmotion envoie 100 images/sec).

Si le réseau de neurones répond positivement, on déclenche alors le début de l'évènement (début de swipe, début de cercle), et on déclenche sa fin lorsque le réseau répond négativement. On aurait donc un décalage de 30ms par rapport au mouvement réel de l'utilisateur, qui serait donc imperceptible et donnerait une impression de temps réel, en plus de permettre la reconnaissance de presque n'importe quel geste après un apprentissage adapté. Même dans le cas où il faudrait plus de données.

Cependant, cette solution étant bien plus difficile à mettre en place (problématique d'apprentissage notamment), nous n'avons pas eu le temps de l'implémenter et laissons aux futurs collaborateurs du projet le soin de décider s'ils souhaitent y consacrer du temps.

IV- Conclusion

Rappelons tout d'abord la problématique du projet : associer un retour sonore aux gestes de l'utilisateur d'une voiture afin de lui permettre une commande efficace de différentes fonctions dans l'habitacle.

Ce que nous avons produit peut se résumer aux points clés suivants :

- Une amélioration de l'API du Leap pour les besoins du projet, mais qui reste suffisamment flexible pour convenir à des projets plus ambitieux
- Une proposition de modulation des différents sons jouant sur différents paramètres : spatialisation, variation de vitesse de lecture et d'amplitude
- Deux thèmes sonores cohérents (école et mécanique) et la possibilité d'en ajouter facilement
- Un prototype fonctionnel permettant de démontrer notre solution

Cependant, nous sommes loin d'un produit fini. Résumons à présent les grands axes d'amélioration et de développement du projet :

- Utilisation d'une modulation supplémentaire (pitch), affinage des différentes lois de modulation pour obtenir un meilleur rendu à l'oreille
- Prise en compte de l'environnement sonore réel de la voiture
- Continuer l'amélioration de la reconnaissance des gestes par le LeapMotion

- Travail éventuellement plus fin des sons proposés avec des outils de design sonore afin encore une fois d'obtenir un meilleur rendu
- Test du prototype sur un panel d'utilisateurs potentiels afin de d'évaluer la facilité de la prise en main et leur ressenti

Comme on peut le voir, ce projet possède de nombreux axes de développement; nous l'avons trouvé intéressant pour notre part et souhaitons bon courage à l'étudiant qui prendra la suite du projet dans le cadre de son stage.