Laboratoire 5

Rapport

GIF-3004

Léo Clouet : 111 187 232

Pierre-Olivier Denis : 111 186 161

---

1. Présentation générale du projet et objectifs

   Notre projet est d'enregistrer le son capturé par un microphone qui est branché sur un raspberry-pi, de transmettre les échantillons vers un autre raspberry-pi et d'y faire jouer le son. Le premier défi, dans le cadre du projet, est de transmettre le son avec le moins de délais possible. Il faut idéalement que le son soit joué en temps réel entre le microphone et le haut-parleur.
   Une fois cette première étape réalisée avec succès, il serait alors intéressant d'ajouter des effets sur le son. Cela ajouterait une contrainte de temps réel ainsi que de la difficulté au projet. Des exemples d'effets qui pourraient être ajoutés sont : un filtre passe-haut, un filtre passe-bas, un délai, de la distorsion, etc.
2. Cahier de charge

Exigence	Cible	Explication
Fluidité du son	Pas de coupure apparente dans le son	On ne veut pas percevoir de coupure dans la lecture du son, ce qui indiquerait des pertes d'échantillons ou encore une variabilité de latence.
Délais entre l'enregistrement et le playback	< 50 ms	On ne veut pas percevoir de délais notables entre l'enregistrement et l'émission. Puisque l'humain peut entendre jusqu'à 20 Hz, cela représente une période de 50ms.
Bande passante du système	>= 44 100 échantillons/seconde	Il faut que le système soit en mesure de supporter un flux d'échantillonement classique des microphones. Autrement, le système va accumuler du retard et donc des latences notables.

3. Présentation de la solution

   Pour réaliser le système, nous l'avons décomposé en différents processus. Chaque processus communique entre eux par un « pipe », s'ils sont sur le même Raspberry-Pi. Le premier processus se nomme « audioReceiver ». Il s'occupe d'aller chercher le son du micro, à l'aide de la librairie Alsa, et le passe au suivant. Le second processus est « comEmitter ». Il s'occupe d'envoyer les données vers l'autre Pi. La version finale de notre projet communique via un socket « TCP/IP ». La communication via Bluetooth a été testée également. Cependant, celle-ci n'a pas fonctionnée. Le code est présent dans « bluetooth_com ». La compression des données a également été testée avec les librairies « opus » et « vorbis ». Cependant, par manque de temps, la compression n'a pas pu être fonctionnelle à temps.

   Les 2 processus présentés ont également leur homologue, mais pour le second « pi ». L'exécutable « comReceiver » reçoit le son du premier « pi » et le passe à « audioEmitter », qui fait jouer le son, à l'aide de la librairie Alsa.

   Des effets ont été implémentés. Ils ont été testés sommairement, mais ne sont pas fonctionnels. Les effets qui ont été réalisés sont, soit un délai, de la distorsion, un filtre passe-haut et un filtre passe-bas.

   Tous les processus qui ont été faits peuvent recevoir comme argument l'option -s, qui permet de changer le type d'ordonnanceur pour mettre le processus en temps réel. Les processus ne prennent pas des arguments qui spécifie le nom des fichiers créés par les « pipes ». Ils sont « hardcodés ». Les processus des effets prennent également l'option -f pour donner le facteur qui module l'effet. L'effet de délais prend également l'option -t qui permet de définir le temps de délais.

4. Méthodologie expérimentale pour valider la solution

   Pour valider la fluidité du son, il est possible de simplement la valider à l'oreille. En regardant si ce qui est dit est bien répété par le haut-parleur, sans coupure.

   Pour valider le délai d'enregistrement, la première méthode qui sera utilisée, si la latence est notable à l'oreille, est de la chronométrer à l'aide d'un chronomètre. Si la latence n'est pas notable à l'oreille, un logiciel d'analyse du son pourrait être utile pour regarder la latence entre le moment où l'utilisateur parle et le moment où le son est joué sur le haut-parleur. Celui-ci enregistrait le son d'entrée et le son sortant des haut-parleurs, et la latence pourrait être qualifiée en regardant la différence de temps entre le moment où l'on commence à parler et le moment où les haut-parleurs donnent une sortie. Il faudrait faire des sons avec une courte durée, pour pouvoir apercevoir la latence sur le logiciel d'analyse.

   Pour valider la bande passante du système, du code pourrait être rajouté dans « audioEmitter » pour que celui-ci écrive dans un fichier tous les échantillons que celui-ci a envoyés au haut-parleur durant la dernière seconde.

5. Résultats et analyse

Pour ce qui est de la fluidité, il est possible de noter une bonne fluidité dans le son. Il n'y a donc pas de problème de ce côté. Le son semble sans coupure. Ceci est prouvée par la bande passante, présentée dans le deuxième paragraphe ci-dessous.

Pour ce qui est du délai, celui-ci est perceptible à l'oreille. Notre système a donc une lacune de ce côté. Nous estimons qu'en moyenne, la latence du système à environ 800ms. Ceci est probablement causé par le fait que notre système communique avec des sockets « TCP/IP », sans algorithme de compression. Ce type de communication cause de la latence plus grande, puisque les paquets sont plus gros et que ce n'est pas une communication de type point à point. Une communication de type Bluetooth serait sans doute plus appropriée. Par ailleurs, compresser les données serait probablement quelque chose qui permettrait de diminuer la grosseur des paquets et donc de diminuer la latence dans le son.

Pour ce qui est de la bande passante du système, du code a été ajouté dans « audioEmitter » pour voir combien d'échantillons sont traités en 1 seconde. Le résultat est que notre système est en mesure de traiter, en moyenne, 711200 bits par seconde. Cela donne donc 44450 échantillons par seconde. Le cahier des charges est respecté sur ce point.

6. GitHub et compilation

Pour compiler notre projet, il faut « builder » le « CMake » dans « Visual Studio Code ». Un script « sendToPi.sh » est présent pour envoyer les exécutables sur les 2 « pi ». Il faut cependant ajuster l'adresse IP des 2 « pi » dans le fichier « sendToPi.sh » avant de l'exécuter. L'adresse du « pi », servant comme serveur, doit également être ajustée dans le fichier « constant.h » dans le « define » de « IP_ADDRESS », avant de compiler.

Le GitHub est accessible via l'adresse suivante :
https://github.com/MisterDeenis/SETR_Labo5_EQ1

Il est à noter que nous avions commencé le projet sur un autre « repo », qui contient d'autres projets, avant de le migrer vers celui-ci. La contribution est donc peut-être un peu erronée par le gros « commit initial ». Nous vous assurons, toutefois, que nous avons travaillé, à parts égales, sur cette partie du projet.