Barbichette

(en cours)

Installation interactive pour rire et causer des algorithmes de reconnaissance faciale, de reconnaissance vocale et de synthèse vocale

Prototype v0 / juin 2023

Mise en route de l'installation

Tout brancher, relier l'ordi et le rpi au réseau local

Démarrer le raspberry pi

- Chercher l'adresse IP locale `hostname -I` dans un terminal et la noter
- Démarrer Programmation/Geany
- Lancer le script osc_reception_tracking_visage.py **dans le terminal** de Geany

Démarrer l'ordi

- Régler les paramètres webcam : `guvcview` dans un terminal
  - → changer au moins la fréquence de rafraîchissement sur 50Hz (si éclairage électrique)
- Fermer le terminal
- Vérifier qu'il n'y pas de terminal ouvert (cela perturberait l'étape suivante)
- Lancer qt creator : fichier → projet récent → dlibFaceTrackerTEST.qvs
- Cliquer sur la grosse flèche verte ou utiliser le raccourci clavier ctrl+r
  - Choisir la webcam à utiliser → 0 si 1 seule webcam
  - Choisir le périphérique audio à utiliser → regarder dans la liste mais sûrement 6 ou 8
  - Entrer l'adresse réseau du rpi

Fonctionnement de l'application

Raccourcis clavier

- F fullscreen
- I indicateur de performance et valeurs des différents types de sourire.
- U fait apparaître la souris et plusieurs options
  - radius : largeur des landmarks
  - smile threshold : seuil de détection d'un sourire
  - minimum head size : défini la taille minimum pour qu'un visage soit capté.
  - detection method :
    - 1 capte le plus grand visage
    - 2 capte le visage le plus au centre
  - debug mode : fait apparaître l'image de la webcam en arrière plan

===== Automate =====#####

Construction d'un automate d'après le projet de Rolf Jethon, voir
https://lesporteslogiques.net/wiki/openatelier/projet/tete_animatronique
L'automate est commandé par un raspberry pi et actionné par des servomoteurs

Installation d'un ordi pour la reconnaissance faciale

Installation de l'ensemble de logiciels nécessaires sur un client léger (CL28 : i3-4170T @ 3.2 GHz x 2, 4GB RAM, Debian 11 bullseye)

Installation de Qt Creator

installation de QTCreator par apt (cf. https://openframeworks.cc/setup/qtcreator/)

```
sudo apt install qtcreator #version : Qt Creator 4.14.1 based on Qt 5.15.2
sudo apt install qbs
```

Installation openframeworks

téléchargement du paquet OF v.0.11.2 https://openframeworks.cc/download/
installation en suivant https://openframeworks.cc/setup/linux-install/

Ci-dessous, OF est à remplacer par le nom du répertoire choisi pour les fichiers d'openframeworks

installer les bibliothèques dépendances

```
cd OF/scripts/linux/ubuntu
sudo ./install_dependencies.sh
```

compiler openframeworks

```
cd OF/scripts/linux
./compileOF.sh -j2 #2 = nombre de cores
```

ensuite, test d'un exemple avec make

```
cd OF/examples/graphics/polygonExample
make
make run
```

compiler PG (le Project Generator pour Qt Creator)

```
cd OF/scripts/linux
./compilePG.sh
```

J'installe aussi le Project Generator en ligne de commande

```
projectGenerator --help
```

installer QT Creator plugin for openframeworks

```
cd OF/scripts/qtcreator/
./install_template.sh
```

J'essaie d'ouvrir le projet polygonExample dans Qt Creator

- fichier / ouvrir un projet
- compiler / compiler le projet
- compiler / exécuter (CTRL-R)

→ ça fonctionne

Installation caméra

Premiers test avec une caméra PS3 eye

```
sudo apt install v4l-utils
sudo apt install guvcview
sudo apt install webcamoid
```

(?) La caméra plante très vite avec guvcview ou webcamoid, et n'est plus visible par `lsusb`

Installation d'addons dans openframeworks cf.
https://openframeworks.cc/learning/01_basics/how_to_add_addon_to_project/

- https://github.com/kylemcdonald/ofxCv/tags
- (A INSTALLER) https://github.com/HalfdanJ/ofxFaceTracker2

Mais un premier exemple d'ofxCv ne compile pas ...

:(à suivre ...

Préparation d'un prototype de code

Info matériel capture vidéo

On utilise la webcam de la playstation 3, le “Playstation eye”, avec une résolution de 640×480 pixels à 60 Hz. Il est également possible de sélectionner une résolution de 320×240 px à 120 Hz.

Créer un projet avec le project generator

Pour créer un projet et y rajouter des addons, il est plus simple d’utiliser le project generator inclus avec openframeworks : `openframeworks_v0.11.2/projectGenerator-linux64/projectGenerator`

> **BUG** : le project generator ne peut pas retirer un add-on déjà présent dans un projet (du moins sur cette version).

Une fois le projet créé dans le dossier `openframeworks_v0.11.2/apps/myApps`, ouvrir le fichier .qbs dans qtcreator et cliquer sur Configure Project.

Détection de visage

Test avec la méthode Haar Cascade :

2 solutions :

- avec l’addon ofxCvHaarFinder : inclus dans les addons de base d’openframeworks.
- avec l’addon ofxCv : télécharger la derniere version de ofxCv master branch (https://github.com/kylemcdonald/ofxCv/) dans le dossier addons, dezipper et renommer ofxCv si le nom est différent.

Pour utiliser la méthode Haar Cascade, il faut rajouter un fichier de modèle pré-entraîné dans le dossier data de notre application (par exemple haarcascade_frontalface_default.xml) puis créer un projet OF avec ofxCv en addon et faire un test avec l’exemple « exemple-face » d’ofxCv.

> **Observations** : performance et stabilité insuffisantes pour le projet barbichette mais simple à mettre en place.

Test avec la méthode des face landmarks

Commencer par télécharger la dernière version de ofxCv master branch qui est nécessaire au fonctionnement des addons que l’on va rajouter par la suite.

Premier test avec l’addon ofxFaceTracker2

Le repo officiel de l’addon fonctionne, par défaut, uniquement sur les os android, osx et windows 64 bit.

étape 1

Installer l’intel® oneAPI Math Kernel Library si l’ordinateur utilisé n’a pas de carte graphique dédiée et possède un processeur intel :
« https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-download.html?operatingsystem=linux » ou sinon la commande :

```
sudo pacman -S intel-mkl
```

Si l’ordinateur est équipé d’une carte graphique nvidia CUDA, installer ça à la place de l’intel mkl:
https://developer.nvidia.com/cudnn et https://developer.nvidia.com/cuda-zone

étape 2

Télécharger ofxDlib et ofxFaceTracker2 dans le dossier addons.

> :warning: la bibliothèque ofxFaceTracker2 que l'on utilise ici n'est pas l'addon "officiel" mais la fork de bakercp qui fonctionne avec les distrib UBUNTU : « https://github.com/bakercp/ofxFaceTracker2 »

Attention : utiliser la commande `GIT CLONE` pour récupérer `ofxDlib` sinon le script shell de l'étape suivante ne fonctionnera pas (besoin du .git)

```
cd of_v0.10.1_linuxarmv6l_release/addons
git clone https://github.com/bakercp/ofxDlib.git
```

Une fois les deux addons téléchargés et décompressés, lancer le script `bootstrap.sh` dans le dossier `ofxDlib/scripts` :

```
cd scripts
./bootstrap.sh
```

étape 3

Dans le fichier `ofxDlib/addon_config.mk` : - Si on utilise la intel mkl : décommenter la ligne 56 :

```
ADDON_INCLUDES += /opt/intel/mkl/include
```

- Si on utilise une carte graphique nvidia sous CUDA décommenter la ligne 53 :

```
ADDON_LDFLAGS += -L/usr/local/cuda/lib64 -lcuda -lcudart -lcudnn -lcublas -lcurand -lcusolver
```

Commenter les lignes 740 à 743 du fichier `openframeworks_v0.11.2/addons/ofxDlib/libs/dlib/include/dlib/matrix/kiss_fft.h` à cause d'un conflit entre la lib ofxDlib et une lib openframeworks :

```
// inline int kiss_fftr_next_fast_size_real(int n)
// {
//    return kiss_fft_next_fast_size((n+1)>>1) << 1;
// }
```

étape 4

Avec le project manager, création d'un nouveau un projet avec les addons `ofxDlib`, `ofxFaceTracker2`, `ofxCv` et `ofxOpenCv`. Rajout du fichier shape_predictor_68_face_landmarks.dat dans le dossier bin/data du projet.

> Aide pour les systèmes sous ArchLinux : https://roosnaflak.com/tech-and-research/set-up-ofxfacetracker2-on-arch-linux/

étape 5 – détection de sourire

Télécharger l'addon ofxBiquadFilter dans le dossier addon de OF. Le rajouter au projet avec le project generator. Copier les 4 fichiers de `openframeworks_v0.11.2/addons/ofxFaceTracker2/example-svm/bin/data` vers le dossier data du projet. Reprendre l'exemple inclus avec ofxFaceTracker2 `exemple-svm` qui devrait fonctionner.

> **Observations** : bonne performance, léger problème de stabilité (l'image sautille), installation un peu compliquée mais nous allons retenir cette méthode.

version d'OpenGl

Si besoin, pour connaître la version d'OpenGL qu'utilise notre machine

```
cd openframeworks_v0.11.2/examples/gl/glInfoExample
make
make run
```

Pour le client léger du projet barbichette on obtient ce résultat :

```
version=3.0 Mesa 20.3.5
vendor=Intel Open Source Technology Center
renderer=Mesa DRI Intel(R) HD Graphics 4400 (HSW GT2)
```

On peut préciser la version opengl du projet dans `main.cpp` pour être sûr de la compatibilité avec le matériel utilisé :

```
ofGLFWWindowSettings settings;
settings.setGLVersion(3, 0);
settings.setSize(webcamW, webcamH);
settings.windowMode = OF_WINDOW;
ofCreateWindow(settings);
ofRunApp(new ofApp());
```

Développement du logiciel de détection de visage pour le projet barbichette

Les landmarks

- 68 repères (points avec un index) qui vont définir la structure / position d'un visage. L'index des points sera toujours le même peu importe le visage (ex. les points 0 à 16 représentent la mâchoire).
- à partir de l'exemple-svm de ofxFaceTracker, mettre en place une détection du sourire d'un des visages détectés selon les paramètres small smile (sourire léger), big smile (sourire extasié, bouche ouverte), O smile (bouche en forme de O) et neutral smile (pas de sourire).

> **Observations** : fonctionne bien sur un visage orienté parfaitement vers la webcam. Les points trembles et il y a parfois des sauts dans la captation avec la détection du visage qui cesse pendant quelques images. Captation moins précise sur un visage légèrement tourné. Dans de mauvaises conditions d'éclairage, la captation ne fonctionne pas bien. Un peu le bazar quand plusieurs utilisateurs.

Captation des visages

- Pour que les landmarks sautillent moins, j'utilise un filtre passe bas pour lisser les positions par rapport aux précédentes.

```
smooth = 0.8; // btw 0.0 and 0.999999, closer to 1.0 = smoother
head[i] = head[i] * smooth + tracker.getInstances()[index].getLandmarks().getImagePoints()[i] * (1.0 - smooth);
```

L'addon propose de base deux systèmes d'identification des visages. Premièrement un système de labellisation des visages. Chaque nouveau visage détecté possède un nouveau label et si un visage disparaît et réapparaît sous x secondes dans la même zone, il garde le même label. Et deuxièmement le tracker créé une liste où chaque index représente les landmarks d'un visage capté. Plus un visage est capté depuis longtemps sans interruption, plus l'index de la liste contenant la position de ses landmarks a une valeur proche de 0. Problème : quand on perd et recapte un visage, le label du visage reste le même mais son index dans la liste des landmarks captés change,. Ce qui peut poser problème quand on veut savoir quel label correspond à quelle liste de landmark.

- On peut définir la taille de l'image qui va être analysée pour la détection des visages avec la fonction `tracker.setFaceDetectorImageSize`. Plus la valeur est grande plus le calcul sera long et précis. La valeur MAX est la résolution de la webcam qui est de 640 x 480 px dans notre cas. Avec cette valeur le background thread est environ à 15 fps. Après plusieurs tests j'ai choisi la valeur 480 x 360 qui tourne en 30 fps sur le background thread.

```
tracker.setup();
tracker.setFaceDetectorImageSize(480 * 360);
```

Choix de restreindre la détection à un seul visage pour le projet. Pour cela :

1. Détecter si la bounding box d'un visage est suffisamment grande pour être capté. C.-à-d. détecter si un visage est suffisamment proche de la webcam, dans le but de supprimer les visages parasites des spectateurs qui se trouveraient dans le champs de vision de la webcam mais qui serait trop loin pour en être un utilisateur.
2. Sélectionner le visage principal si plusieurs sont détectés. Deux méthodes misent en place :
    - Sélectionner le visage avec la plus grande bounding box → le visage qui occupe la plus grande surface (aire) sur l'écran et qui est donc normalement le visage le plus proche de la webcam (sauf cas particulier comme les enfants ou les petites têtes).
    - Sélectionner le visage le plus au centre de la caméra.

```
void ofApp::findMainFaceID(int method){
    // find the main face ID and store it in the mainFaceID variable
    isCloseEnough = false;
    float boundingBoxSize = 0;
    float closestToCenter = 10000.0;
```

```
    for (int i = 0, len = tracker.size(); i < len; i++){
        float bsize = tracker.getInstances()[i].getBoundingBox().getArea();
        //TO DO ---- add a method to check if the face is visible on screen
        switch (method){
            case 1: { //Method 1 - biggest bounding box
                if (bsize > boundingBoxSize && bsize > minFaceArea){
                    boundingBoxSize = bsize;
                    mainFaceID = i;
                    isCloseEnough = true;
                }
                break;
            }
            case 2: { //Method 2 - closest to center
                ofVec2f toCenter = ofVec2f(ofGetWidth() * 0.5, ofGetHeight() * 0.5) -
tracker.getInstances()[i].getBoundingBox().getCenter();
                float dToCenter = toCenter.length();
                if (bsize > minFaceArea && dToCenter < closestToCenter){
                    closestToCenter = dToCenter;
                    mainFaceID = i;
                    isCloseEnough = true;
                }
                break;
            }
            default:
                break;
        }
    }
}
```

> rajouter la possibilité de passer d'une méthode à l'autre dans l'UI

- Par défaut, dans l'exemple qui sert à détecter les sourires, seul le premier visage de la liste des visages traqués est évalué, donc seulement le visage qui était traqué depuis le plus longtemps sans interruption (cf. problème vu précédemment entre différence entre index et label). Pour résoudre ce problème, il faut modifier la fonction makeSample (qui détecte les sourires) pour prendre en argument l'id du visage que l'on veut détecter, retourné par notre fonction `findMainFaceID`.

```
sample_type ofApp::makeSample(int id){
    auto outer = tracker.getInstances()[id].getLandmarks().getImageFeature(ofxFaceTracker2Landmarks::OUTER_MOUTH);
    auto inner = tracker.getInstances()[id].getLandmarks().getImageFeature(ofxFaceTracker2Landmarks::INNER_MOUTH);
    auto lEye = tracker.getInstances()[id].getLandmarks().getImageFeature(ofxFaceTracker2Landmarks::LEFT_EYE);
    auto rEye = tracker.getInstances()[id].getLandmarks().getImageFeature(ofxFaceTracker2Landmarks::RIGHT_EYE);
    ....
}
```

- Si par la suite il y a un besoin de détecter plusieurs utilisateurs, il faudrait exécuter la function make_sample sur tous les visages captés pour détecter leur sourire. Il faudrait également revoir la méthode de lissage des données du face landmark qui ne fonctionne que pour un seul visage.

Quelques ajustements :

- rajout d'une fonction qui vérifie si il y a un utilisateur devant la webcam

```
bool ofApp::isFaceCaptured(){
    if (tracker.size()>0 && isCloseEnough){
        return true;
    }
    else{
        return false;
    }
}
```

- mettre les valeurs de "sourire" à 0 quand aucun utilisateur n'est présent.

```
tracker.update(webcam);
findMainFaceID(detectionMethod);
if(isFaceCaptured()){
    // Run the classifiers and update the filters on the face
    bigSmileValue.update(learned_functions[0](makeSample(mainFaceID)));
    smallSmileValue.update(learned_functions[1](makeSample(mainFaceID)));
    oValue.update(learned_functions[2](makeSample(mainFaceID)));
    neutralValue.update(learned_functions[3](makeSample(mainFaceID)));
}
else{
    bigSmileValue.clear(0.0);
    smallSmileValue.clear(0.0);
    oValue.clear(0.0);
    neutralValue.clear(0.0);
}
```

- adaptation du programme à plusieurs résolution et ratio d'écran. En plus de cela il faut inverser l'image sur l'axe horizontal pour créer un effet "miroir". Pour cela j'effectue une inversion de l'axe horizontal + une mise à l'échelle par rapport à la hauteur de la fenêtre + un offset horizontal pour garder le centre de la captation au centre de la

fenêtre.

```
void ofApp::drawHead(vector<ofVec2f>& head, vector<ofVec2f>& mappedHead, int index){
    float scaleY = ofGetHeight() * 1.0/webcamH;
    float offsetX = (ofGetWidth() * 0.5) - (scaleY * webcamW * 0.5);
    for (int i = 0, len = head.size(); i<len; i++){
        head[i] = head[i] * smooth + tracker.getInstances()[index].getLandmarks().getImagePoints()[i] * (1.0 - smooth);
        //scale to screen size (based on screen height) + offset to center the faces
        mappedHead[i] = ofVec2f(head[i].x * scaleY * -1.0 + ofGetWidth() - offsetX, head[i].y * scaleY);
        ofDrawCircle(mappedHead[i].x, mappedHead[i].y, radius * scaleY);
    }
}
```

- Quand un visage n'est plus capté, les landmarks disparaissent brutalement. Pour résoudre ce problème je teste plusieurs solution et finalement je décide de réunir les points au centre de l'écran quand aucun visage n'est capté. En plus de cela, avec l'effet du filtre de lissage, les clignotements d'image quand le traqueur perd le visage qq frames ne sont plus présents.

```
void ofApp::drawWait(vector<ofVec2f>& head, vector<ofVec2f>& mappedHead){
    float scaleY = ofGetHeight() * 1.0/webcamH;
    float offsetX = (ofGetWidth() * 0.5) - (scaleY * webcamW * 0.5);
    float extraSmooth = 0.99;
    for (int i = 0, len = mappedHead.size(); i<len; i++){
        head[i] = ofVec2f(head[i].x * extraSmooth + 320.0 * (1.0 - extraSmooth), head[i].y * extraSmooth + 240.0 * (1.0 - extraSmooth)); //
temp -> need to be change based on the webcam resolution
        mappedHead[i] = ofVec2f(head[i].x * scaleY * -1.0 + ofGetWindowWidth() - offsetX, head[i].y * scaleY);
        ofDrawCircle(mappedHead[i].x, mappedHead[i].y, radius * scaleY);
    }
}
```

- Rajout d'un debug mode. Pour le moment, le debug mode affiche l'image captée par la webcam en arrière-plan.

```
if (debugMode){
    // Draw webcam
    float scaleY = ofGetHeight() * 1.0 / webcamH;
    float offsetX = (ofGetWidth() * 0.5) - (scaleY * webcamW * 0.5);
    webcam.draw(webcam.getWidth() * scaleY + offsetX , 0, -webcam.getWidth() * scaleY, webcam.getHeight() * scaleY);
    // Draw debug tracker landmarks
    //tracker.drawDebug();
    // Draw estimated 3d pose
    //tracker.drawDebugPose();
}
```

Scénario et comportement de l'automate

Définition de plusieurs états possibles pour l'automate à l'aide de switch case :

- qq1 arrive alors que le robot est inactif –> il se réveille quand il détecte un visage
- qq1 part → au bout de X secondes sans détecter de visage le robot devient inactif
- inaction → il réalise des actions tous les x secondes tant qu'il ne détecte pas de visage
- joue au jeu → il détecte les sourires (big, small, O, neutral)

Échange de données entre le robot et l'application de détection de visage.

Premier test avec le protocole OSC : échange entre le programme et une application processing qui tourne sur une autre machine. Envoi et réception de donnée ok → dans le programme besoin de préciser l'adresse ipv4 de la machine recevant les données ainsi qu'un port sur lequel envoyer et recevoir ces données. Pour le moment les données OSC sont envoyées en continu. Pour la suite, il est possible d'envisager que les données soient envoyées uniquement si leur valeur a changé depuis le dernier envoi.

```
void ofApp::setup(){
    ...
    sender.setup(host, PORT);
    // listen on the given port
    ofLog() << "listening for osc messages on port " << PORT;
    receiver.setup(PORT);
    ...
}

void ofApp::update(){
    ...
    if (isFaceCaptured()){
        float posX = tracker.getInstances()[mainFaceID].getBoundingBox().getCenter().x;
        float posY = tracker.getInstances()[mainFaceID].getBoundingBox().getCenter().y;
        int isSmiling = 0;
        if (playerSmile() != neutralSmile){
            isSmiling = 1;
        }
```

```
        float scaleY = ofGetHeight() * 1.0/webcamH;
        float offsetX = (ofGetWidth() * 0.5) - (scaleY * webcamW * 0.5);
        if (offsetX > 0){
         posX = ((posX * scaleY * -1.0 + ofGetWidth() - offsetX) - offsetX) / (ofGetWidth() - (2.0 * offsetX));
        }
        else{
          posX = (posX * scaleY * -1.0 + ofGetWidth() - offsetX) / ofGetWidth();
        }
        posX = ofClamp(posX, 0.0, 1.0);
        posY = posY * scaleY / ofGetHeight();
        ofxOscMessage m;
        m.setAddress("/posx");
        m.addFloatArg(posX);
        sender.sendMessage(m, false);
        ofxOscMessage n;
        n.setAddress("/posy");
        n.addFloatArg(posY);
        sender.sendMessage(n, false);
        ofxOscMessage o;
        o.setAddress("/smile");
        o.addFloatArg(isSmiling);
        sender.sendMessage(o, false);
    }
    else{
        ofxOscMessage m;
        m.setAddress("/posx");
        m.addFloatArg(-1.0);
        sender.sendMessage(m, false);
        ofxOscMessage n;
        n.setAddress("/posy");
        n.addFloatArg(-1.0);
        sender.sendMessage(n, false);
        ofxOscMessage o;
        o.setAddress("/smile");
        o.addFloatArg(0.0);
        sender.sendMessage(o, false);
    }
    int isSpeaking = 0;
    if(robotVoice.isThreadRunning()){
        isSpeaking = 1;
    }
    ofxOscMessage m;
    m.setAddress("/speak");
    m.addFloatArg(isSpeaking);
    sender.sendMessage(m, false);
}
```

Au lancement du programme ajout d'une demande de l'adresse ipv4 du robot.

```
void ofApp::setup(){
    ...
    if(askIP){
        cout << "enter robot ipv4 adress : ";
        cin >> host;
        cout << "My Robot ip is: " << host << endl;
    }
    ...
}
```

Interface utilisateur

Rajouter l'addon `ofxGui` avec le project generator, qui permet d'avoir une interface utilisateur pour modifier des variables pdt que l'application fonctionne. Il est possible d'enregistrer les paramètres modifiés.

```
// ofApp::setup()
gui.setup();
gui.add(radius.setup("radius", 1.6, 0.2, 10.0));
gui.add(smileThreshold.setup("Smile Threshold", 0.42, 0.01, 1.0));
gui.add(minFaceArea.setup("Minimum head size", 4000.0, 0.0, 40000.0));
gui.add(detectionMethod.setup("Detection Method", 1, 1, 2));
gui.add(debugMode.setup("Debug Mode", false));
// ofApp::draw()
if (displayUI){
        gui.draw();
}
```

Ajout de raccourci clavier pour afficher l'ui qui servira au débogage.

- 'i' pour afficher les performances et les valeurs de 'sourire'
- 'u' pour afficher les variables ajustables (taille minimum de tête détecté, ...) et d'une option pour afficher le mode débogage (afficher l'image de la webcam en arrière plan, ...)
- 'f' pour passer l'application en mode fullscreen.

```
void ofApp::keyReleased(int key){
```

```
    switch(key){
    case 'u': case 'U':
        displayUI = !displayUI;
        (displayUI) ? ofShowCursor() : ofHideCursor();
        break;
    case 'i': case 'I':
        displayInfo = !displayInfo;
        break;
    case 'f': case 'F':
        ofToggleFullscreen();
        break;
    default:
        break;
    }
}
```

Audio

Lire des fichiers sons en fonction des différents états du robot. Pour cela il y a la possibilité d'utiliser un ou plusieurs objets `ofSoundPlayer`. Si on en utilise qu'un seul, il faut charger en mémoire à la volée chaque fichier son et de ce fait les sons ne peuvent pas se superposer. Si on utilise plusieurs objets `ofSoundPlayer`, les sons peuvent se superposer et l'on a besoin de moins de puissance machine mais cela requiert une utilisation plus élevé de la mémoire.

```
// ofApp.h
ofSoundPlayer botSound;
map<string, string> soundFile;

// ofApp.cpp
void ofApp::setup(){
    soundFile.insert(make_pair("welcome", "1085.mp3"));
    soundFile.insert(make_pair("bored", "Violet.mp3"));
    soundFile.insert(make_pair("bye", "synth.wav"));
}

void ofApp::playSound(string name){
    botSound.unload();
    botSound.load(soundFile.find(name)->second);
    botSound.play();
}
```

> Premier test avec un seul `ofSoundPlayer` → résultat satisfaisant.

Réglage de la webcam

Si jamais il y a plusieurs webcam sur la machine il faut pouvoir sélectionner celle que l'on veut utiliser (par exemple une webcam interne d'un laptop + une webcam usb). Pour cela, au lancement du programme, il faudrait lister toutes les caméras connectées à la machine et demander laquelle utiliser. Le plus simple pour cela et d'avoir un terminal externe au lancement du programme et de faire des demandes avec `cin` et `cout`. Quand on utilise Visual Studio le lancement d'un terminal ce fait par default à l'ouverture du programme. Dans notre cas, on utilise Qtcreator et il faut aller dans l'onglet `Projects → Build and run` cliquer sur `Run → Exécuter` et cocher la case `Run in terminal`.

```
// Webcam and face tracker setup
int deviceId = 0;
while (!webcam.isInitialized())
{
    webcam.listDevices();
    cout << "Enter device ID for webcam (0 if only 1 webcam) :";
    cin >> deviceId;
    webcam.setDeviceID(deviceId);
    webcam.initGrabber(webcamW, webcamH);
}
```

Problème : la webcam freeze de temps en temps. Comme solution je propose de réinitialiser la webcam si pdt 120 cycles le programme ne reçoit plus d'images en provenance de la webcam.

```
webcamReinitTimer++;
if (webcamReinitTimer > 120.0){ // arbitrary value (120.0 is ~2s at 60 fps)
    webcam.close();
    webcam.setDeviceID(0);
    webcam.initGrabber(webcamW, webcamH);
    cout<<"webcam reinitialized ";
}
```

Nouveau problème : je n'ai pas pu tester la solution parce que je n'ai pas réussi à reproduire le bug.

Problème : clignotement de l'image captée par la webcam due à la fréquence des éclairages (effet de flicker, éclairage 50hz et webcam 60hz par défaut). Solution : régler la caméra sur une fréquence de 50hz. Plusieurs solutions, avec une interface

type guvcview ou en ligne de commande avec v4l2.

```
v4l2-ctl –set-ctrl=power_line_frequency=1
```

Test avec guvcview pour régler l'image webcam. Problèmes de captation liés à la capture vidéo. Image trop sombre, clignotement de l'image. guvcview permet de modifier les réglages de la webcam, qui restent actifs quand on lance l'application. Problème : les paramètres se réinitialisent quand on redémarre l'ordinateur ou quand la webcam est débranchée/rebranchée.

> Principaux réglages à modifier avec guvcview avant de lancer le programme : dans le panneau video control changer la sortie caméra en YUY puis dans les réglages de l'image restreindre la fréquence de rafraîchissement de la webcam à 50 hz et gérer le gain et l'exposition.

Automatisation du réglage des paramètres de la webcam quand on la branche

> Pas vraiment sûr de ce que je fait dans cette section, à ne pas reproduire chez vous !

```
which bash
```

/usr/bin/bash Créer un script pour régler la caméra.

```
#!usr/bin/bash
v4l2-ctl \
--set-ctrl=power_line_frequency=1
```

Rendre le script executable

```
sudo chmod +x /home/linuxquimper/openframeworks_v0.11.2/apps/myApps/dlibFaceTrackerTEST/webcam_startup_param.sh
```

Lancer le script quand on branche la caméra. Commande `lusb` pour trouver l'ID de notre périphérique

```
Bus 001 Device 008: ID 1415:2000 Nam Tai E&E Products Ltd. or OmniVision Technologies, Inc. Sony Playstation Eye
```

Aller dans `/etc/udev/rules.d` et créer un fichier rule

```
sudo touch ps3_webcam_rule.rules
```

Y rajouter

```
ACTION=="add", SUBSYSTEM=="usb", ATTR{idVendor}=="1415", ATTR{idProduct}=="2000",
RUN+="/home/linuxquimper/openframeworks_v0.11.2/apps/myApps/dlibFaceTrackerTEST/webcam_startup_param.sh"
```

puis

```
sudo service udeV restart

tail -f /var/log/syslog
```

> Le script fonctionne seul mais ne se déclenche pas quand la webcam se connecte. Je laisse tomber ce problème pour le moment.

Créer un exécutable et régler les paramètres webcam au lancement

Idée : créer un script qui fait les réglages caméra et lance le programme. Premièrement il faut rendre notre programme exécutable

```
sudo chmod +x /home/linuxquimper/openframeworks_v0.11.2/apps/myApps/dlibFaceTrackerTEST/bin/dlibFaceTrackerTEST
```

Créer un script et le rendre executable avec sudo chmod +x

```
#!/usr/bin/bash
v4l2-ctl --set-ctrl=power_line_frequency=1
/home/linuxquimper/openframeworks_v0.11.2/apps/myApps/dlibFaceTrackerTEST/bin/dlibFaceTrackerTEST start
```

> Ca ne fonctionne pas ! A voir pour plus tard.

Synthèse vocale

Test avec espeak (l'installation de svoxpico ne fonctionne pas)

```
sudo apt install espeak
```

Dans l'application, création d'une fonction qui lance espeak. Pour le moment j'utilise espeak en python plutôt que directement en C++ → plus d'exemples et plus simple à mettre en place pour le moment.

Problème : le programme stoppe son execution pdt la synthèse vocal.

Solution : lancer l'execution de eSpeak sur un nouveau thread. Creation d'une classe qui va hériter de la classe `ofThread` d'openframeworks qui permet d'executer des fonctions sur un nouveau thread.

```
// textToSpeech.h
#pragma once
#include "ofMain.h"

class textToSpeech: public ofThread
{
public:
    textToSpeech();
    void say(string text);
    void custom(string pitch, string speed, string amplitude, string lg, string tone);
    void restore();

private:
    void threadedFunction();
     string text;
     string initialCommand = "espeak -p 60 -v mb/mb-fr4";
     string customCommand;
};

// textToSpeech.cpp
#include "textToSpeech.h"

textToSpeech::textToSpeech(){
    this->initialCommand = "espeak -p 99 -s 80 -a 100 -v mb/mb-fr1+f4";
    this->customCommand = this->initialCommand;
}

void textToSpeech::say(string t){
    // Play text as speech with the eSpeak software
    this->text = t;
    this->startThread();
}

void textToSpeech::restore(){
    this->customCommand = this->initialCommand;
}

void textToSpeech::custom(string pitch, string speed, string amplitude, string lg, string tone){
    this->customCommand = "espeak -p " + pitch + " -s " + speed + " -a " + amplitude+ " -v "+ lg + "+" + tone;
}

void textToSpeech::threadedFunction(){
    // Defines what's going to be executed while the thread is running
    string command = this->customCommand;
    command = command + " \"" + text + "\"";
    system(command.c_str());
}
```

Installation de MBROLA pour une synthèse vocale plus réaliste.

```
sudo apt-get install git make gcc
git clone https://github.com/numediart/MBROLA.git
cd MBROLA
make
sudo cp Bin/mbrola /usr/bin/mbrola
```

Télécharger les modèles de voix sur https://github.com/numediart/MRBOLA-voices et les ajouter dans le dossier `usr/share/mbrola/xxN` où xxN est le nom du fichier (ex : pour le fichier fr1 le placer dans `usr/share/mbrola/fr1`)

```
sudo mkdir
sudo cp -R /home/linuxquimper/Téléchargements/mbrola_voices/fr4     /usr/share/mbrola/fr4
```

Tous les fichiers ne fonctionnent pas sur debian.

> Ok → fr1 fr4 en1 pt1 Ne fonctionnent pas → fr2 fr3 fr5 bz1 cn1

Reconnaissance vocale

L'idée est de capter des phrases pour que le robot les répète. Pour cela je vais essayé d'enregistrer les voix dans un fichier texte qui sera ensuite lu par le programme avec espeak.

Utilisation de vosk (pour l'installation voir le tuto)

Téléchargement et dézippage du modèle `vosk-model-small-fr-pguyot-0.3` dans un dossier `models`.

Premier test avec `test_microphone.py`, dispo sur le wiki, pour transcrire de l'audio dans un fichier texte.

Lister les périphériques

```
Python3 test_microphone.py -l
```

Puis pour exécuter le fichier test_microphone.py:

```
python3 test_microphone.py -f transcription.txt -m /home/linuxquimper/models/vosk-model-small-fr-pguyot-0.3 -d 6 -s 10.0
```

Ca fonctionne.

Modification du script :

- pour tout écrire sur une seule ligne
- pour effacer le contenu du fichier texte à chaque enregistrement
- pour que le script se termine automatiquement au bout de x secondes. (pour enregistrer 5 secondes de texte par exemple).

`pavucontrol` pour accéder aux paramètres de contrôle de l'entrée audio.

```
#!/usr/bin/env python3

import argparse
import os
import queue
import sounddevice as sd
import vosk
import sys
import time

q = queue.Queue()

def int_or_str(text):
    """Helper function for argument parsing."""
    try:
        return int(text)
    except ValueError:
        return text

def callback(indata, frames, time, status):
    """This is called (from a separate thread) for each audio block."""
    if status:
        print(status, file=sys.stderr)
    q.put(bytes(indata))

parser = argparse.ArgumentParser(add_help=False)
parser.add_argument(
    '-l', '--list-devices', action='store_true',
    help='show list of audio devices and exit')
args, remaining = parser.parse_known_args()
if args.list_devices:
    print(sd.query_devices())
    parser.exit(0)
parser = argparse.ArgumentParser(
    description=__doc__,
    formatter_class=argparse.RawDescriptionHelpFormatter,
    parents=[parser])
parser.add_argument(
    '-f', '--filename', type=str, metavar='FILENAME',
    help='text file to store transcriptions')
parser.add_argument(
    '-m', '--model', type=str, metavar='MODEL_PATH',
    help='Path to the model')
parser.add_argument(
    '-d', '--device', type=int_or_str,
    help='input device (numeric ID or substring)')
parser.add_argument(
    '-s', '--seconds', type=float,
    help='number of seconds to record')
parser.add_argument(
```

```
    '-r', '--samplerate', type=int, help='sampling rate')
args = parser.parse_args(remaining)

try:
    if args.model is None:
        args.model = "model"
    if not os.path.exists(args.model):
        print("Please download a model for your language from https://alphacephei.com/vosk/models")
        print("and unpack as 'model' in the current folder.")
        parser.exit(0)
    if args.samplerate is None:
        device_info = sd.query_devices(args.device, 'input')
        # soundfile expects an int, sounddevice provides a float:
        args.samplerate = int(device_info['default_samplerate'])

    if args.seconds is None:
        time_limit = 6.0
    else:
        time_limit = args.seconds

    model = vosk.Model(args.model)

    if args.filename:
        dump_fn = open(args.filename, "a+")
        dump_fn.truncate(0)
    else:
        dump_fn = None

    start_time = time.time()

    with sd.RawInputStream(samplerate=args.samplerate, blocksize = 1024, device=args.device, dtype='int16',
    channels=1, latency='high', callback=callback):
            print('#' * 80)
            print('Press Ctrl+C to stop the recording')
            print('#' * 80)

            rec = vosk.KaldiRecognizer(model, args.samplerate)

            while True:
                data = q.get()
                if rec.AcceptWaveform(data):
                    r = eval(rec.Result())
                    t = r["text"]
                    if t:
                        print(t)
                        if dump_fn is not None and len(t) > 5:
                            dump_fn.write(t+' ')
                timer = time.time() - start_time
                if timer > time_limit:
                    # save the sentence spoke while exiting
                    r = eval(rec.Result())
                    t = r["text"]
                    if t:
                        print(t)
                        if dump_fn is not None and len(t) > 5:
                            dump_fn.write(t+' ')
                    # exit
                    print('\nDone')
                    parser.exit(0)
                    sys.exit()

except KeyboardInterrupt:
    print('\nDone')
    parser.exit(0)

except Exception as e:
    parser.exit(type(e).__name__ + ': ' + str(e))
```

Le robot enregistre constamment ce qu'il entend par bloc de x secondes et va répéter des phrases qu'il a entendu de manière aléatoire. Quelques ajustements à faire mais le concept fonctionne.

```
// speechToText.h
#pragma once
#include "ofMain.h"

class speechToText: public ofThread
{
public:
    speechToText();
    void listen();
    string getText();
    string deviceID = "6";
    string duration = "5.0"; //in seconds

private:
    void threadedFunction();
    string recordedLine;
    vector<string> recordedLines;
    int iterateur = 0;
    int maxNb = 10;
    bool isRandom = false;
};
```

```
// speechToText.cpp
#include "speechToText.h"

speechToText::speechToText()
{
    this->recordedLines.push_back("Blablablablabla");
}

void speechToText::listen(){
    this->startThread();
}

string speechToText::getText(){
    if (!isRandom){
        isRandom = true;
        return this->recordedLines[this->iterateur];
    }
    else{
        int rand = (int)ofRandom(this->recordedLines.size()); //value btw 0 and maxNb-1
        return this->recordedLines[rand];
    }
}

void speechToText::threadedFunction(){
  // defines what's going to be executed while the thread is running
    cout << "Recording user : START" << endl;
    string command = "python3 /home/linuxquimper/openframeworks_v0.11.2/apps/myApps/dlibFaceTrackerTEST/bin/data/test_microphone.py -f
/home/linuxquimper/openframeworks_v0.11.2/apps/myApps/dlibFaceTrackerTEST/bin/data/transcription.txt -m /home/linuxquimper/models/vosk-
model-small-fr-pguyot-0.3 -d " + this->deviceID + " -s " + this->duration;
    system(command.c_str());
    cout << "Recording user : FINISH" << endl;
    ifstream in;
    in.open("/home/linuxquimper/openframeworks_v0.11.2/apps/myApps/dlibFaceTrackerTEST/bin/data/transcription.txt");
    while(!in.eof()){
        getline(in, recordedLine);
    }
    in.close();
    if (this->recordedLine != ""){
        if (recordedLines.size() < maxNb){
            recordedLines.push_back(recordedLine);
        }
        else{
            this->recordedLines[this->iterateur] = recordedLine;
        }
        this->iterateur = (this->iterateur + 1) % this->maxNb;
        isRandom = false;
    }
    cout << "RecordedLine = "<< recordedLine << endl;
}
```

Pour selectionner le micro qu'on utilise on rajoute dans le setup de `ofApp.cpp` :

```
string command = "python3 data/test_microphone.py -l";
system(command.c_str());
cout << "Choose audio input device: ";
cin >> robotEars.deviceID;
```

Quelques réglages pour le futur

- Refonte du système d'états pour le bot
- Interaction avec l'utilisateur
- Améliorer la captation du visage
- Phrases qui s'enchaînent
- Amélioration de la classe `textToSpeech` pour la rendre plus modulable.

Journal

Le journal est tenu sur la page tête animatronique