PFE – ScoutBOT – Exploration technique – Reconnaissance vocale sur STM32MP1

Responsable du document : Arnaud HINCELIN

Documentation réalisée dans le cadre de notre projet de fin d'études. Cette partie concerne comment effectuer de la reconnaissance vocale sur le MPU STM32MP1.

1. Enregistrements Audio sur STM32MP1

a. Enregistrer un signal vocal

Nous allons voir ici comment enregistrer un signal vocal simple sur la carte STM32MP1.

Etape 1 : Microphone

Brancher une paire d'écouteurs sur la **prise jack** de la carte. Ces écouteurs doivent avoir un **microphone intégré**, et ne doivent **pas être de la marque APPLE**.

Les écouteurs APPLE ne fonctionnent pas, nous avons testé inconsciemment avant de trouver pourquoi cela ne marchait pas. C'est un problème de protocole audio (AHJ vs OMTP) : Smartphone Headset Standards: Apple iPhone, AHJ (CTIA), & OMTP – Headset Buddy Help

> Apple's non-standard signalling and control method means many designed for iPhone headsets are incompatible with other devices. In general the audio out and one button control on such headsets will work, but the audio-in (microphone) and volume controls will not. Some headset manufacturers produce

Etape 2 : Enregistrer

Une fois le microphone bien branché, nous allons effectuer un enregistrement audio via la carte.

Voici le wiki ST qui explique comment enregistrer un signal audio : How to record audio - stm32mpu

En ayant testé les différentes options proposées sur le wiki, voici la commande qui fonctionne avec notre application :

MP1$ *arecord -D record_codec -f S16_LE -d 2 /usr/local/SOUND/avance.wav*

MP1$ *arecord -D* **<carte_audio>** *-f* **<type_de_donnees>** *-d* **<duree> <chemin_fichier>.wav**

Nous avons choisi de réaliser des enregistrements mono d'une durée de 2s, et de traiter des fichiers de type WAV. La commande ci-dessus va créer le fichier audio WAV avec le chemin et le nom désiré.

Etape 3 : Vérifier l'enregistrement

Afin de valider l'enregistrement, il faut lire le fichier depuis la carte.

Voici le wiki ST qui explique comment lire un fichier audio : How to play audio - stm32mpu

En ayant testé les différentes options proposées sur le wiki, voici la commande qui fonctionne avec notre application :

MP1$ aplay -D *playback_codec /usr/local/SOUND/avance.wav*

Cette commande va lire dans les écouteurs le signal audio désiré /usr/local/SOUND/avance.wav.

Si le signal est correctement écouté dans les écouteurs, alors l'enregistrement est validé. Dans la cas contraire, penser à vérifier la carte son du STM32MP1 (voir wiki), et vérifier que les écouteurs sont bien compatibles (pas APPLE), et ne pas hésiter à en utilise d'autres paires.

b. Enregistrer des mots cles

L'enregistrement d'un signal est maintenant validé. Il faut maintenant récolter un certain nombre de signaux pour chacun des mots clés à reconnaître.

Afin de largement faciliter le travail d'enregistrement, nous avons créé un script en C qui permet d'enregistrer directement un certain nombre de signaux, qui classe les signaux par répertoire selon le mot-clé, et qui classe les fichiers avec un index (ex : avance/avance3.wav, ou recule/recule345.wav).

Avec ce script, il est facile de programmer le nombre d'enregistrement désirés, les mots clés désirés, et l'utilisateur a juste à suivre les instructions à l'écran et prononcer le mot-clé demandé. Evidemment, il est également possible de se séparer le travail entre plusieurs utilisateurs (ce que nous avons fait), pour cela un utilisateur A va configurer le script pour enregistrer tous les mots clés avec un index de 0 à 99, tandis que l'utilisateur B sera de 100 à 199.

Etape 1 : Configuration du script

Le script est un fichier C (get_audio_data.c).

```
#define RECORD_ON_MP1   1

#define NUM_START_RECORD 0
#define NUM_STOP_RECORD 100


typedef enum{
    AVANCE=0,
    RECULE,
    DROITE,
    GAUCHE,
    STOP
}VocalCommande_e;

char * VocalCommade[5] = {"AVANCE", "RECULE", "DROITE", "GAUCHE", "STOP"};
char * pathCommand[5] = {"avance/avance", "recule/recule", "droite/droite",
"gauche/gauche", "stop/stop"};
```

Il faut commencer par remplacer **les macros NUM_START_RECORD et NUM_STOP_RECORD** afin de définir le nombre d'enregistrement à effectuer pour chaque mot-clé (ici, nous enregistrer 100 signaux pour chaque mot-clé, soit 500 signaux de 2s, ce qui donne environ 15min de temps passé).

Puis il faut renseigner les **mots-clés** en les indiquant dans **l'énumération**, dans la chaine de caractère **VocalCommade**, et dans la chaine de caractère **pathCommand**.

Enfin, il faut renseigner la **macro RECORD_ON_MP1** afin d'indiquer si on enregistre sur un PC (0) ou sur la MP1 (1). Les enregistrements marchent aussi théoriquement sur PC, mais nous avons préféré enregistrer dans les conditions réelles, c'est-à-dire enregistrer sur la MP1. Cette macro définira juste la commande d'enregistrement qui change selon la plateforme PC ou STM32MP1.

La dernière chose est d'ajouter dans la fonction main du script la fonction qui va acquérir le mot-clé, voici ce que nous avons pour nos 5 mots-clés :

```
89      get_audio(AVANCE);
90      get_audio(RECULE);
91      get_audio(DROITE);
92      get_audio(GAUCHE);
93      get_audio(STOP);
94
```

Etape 2 : Répertoires

Maintenant il faut configurer les répertoires sur la carte.

Après être connecté en STLink à la carte (PC$ minicom -D /dev/ttyACM0), on va créer nos différent répertoires nécessaires. On commence par se déplacer dans le dossier où se trouve l'éxécutable ELF, puis on y créé un dossier pour chaque mot-clé, dedans seront créés les fichiers audio enregistrés :

MP1$ cd /usr/local/SOUND/

MP1$ mkdir avance

MP1$ mkdir recule

MP1$ mkdir droite

MP1$ mkdir gauche

MP1$ mkdir stop

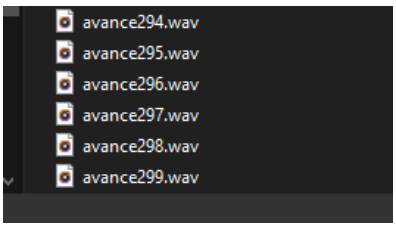
Etape 3 : Enregistrement

Si on exécute le script sur la carte MP1, il faut absolument compiler le fichier C avec le SDK donné par ST (faire le source). Puis on transfère le fichier ELF généré sur la MP1 avec la commande scp.

PC$ scp file.elf root@192.168.7.1:/usr/local/SOUND (Le dossier SOUND a été créé auparavant)

Il suffit maintenant d'exécuter le script et suivre les indication, le script indique à chaque changement de mot-clé et l'utilisateur doit valider avec ENTREE avant de continuer.

A chaque fois que la commande d'enregistrement s'exécute, l'utilisateur doit prononcer le mot-clé.



A la fin, vérifier que nous avons dans chaque répertoire des fichiers ayant des index désirés.

c. Transferer les signaux vers un PC

Nous avons donc des fichiers audios WAV sur la carte MP1, afin de les transférer sur un PC, nous allons utiliser une clé USB.

En branchant le clé USB sur une des interfaces USB de la carte nous observons sur la console des lignes qui vont préciser sur quelle interface est connectée la clé (Sda, sdb, sdc, sdd,…), chercher la ligne en question qui le précise.

Puis il faut monter la clé USB dans la partition /mnt/ de la carte, en précisant l'interface (ex avec sda1)

MP1$ mount -t vfat **/dev/sda1** /mnt/ -v

Après cela, on peut se déplacer dans /mnt/ et observer le contenu de la clé

Puis, nous allons copier chaque répertoire de mot-clé contenant les enregistrements dans la partition /mnt/ (soit sur la clé). Exemple avec avance :

MP1$ cp usr/local/SOUND/avance /mnt/

Enfin ne pas oublier de démonter la clé de la partition /mnt/ à la fin, sinon la copie peut ne pas se terminer correctement.

MP1$ umount /dev/sda1/

Enfin, sur le PC, copier les répertoires des signaux audio.

On a maintenant des données prêtent à être utilisées.

2. Modele de reconnaissance vocale

a. Principe

Ce que nous allons vouloir faire est de réaliser un modèle un modèle d'apprentissage qui va reconnaître les mots-clés prononcés par l'utilisateur. L'apprentissage sera supervisé, c'est-à-dire que le modèle va apprendre le pattern de chaque mot-clé et l'associe à un label (mot-clé), comme nous avons trié les signaux par mot-clé le labelisage des signaux est déjà fait.

Nous avons utilisé un modèle d'apprentissage basé sur un réseaux de neurones.

b. Modele

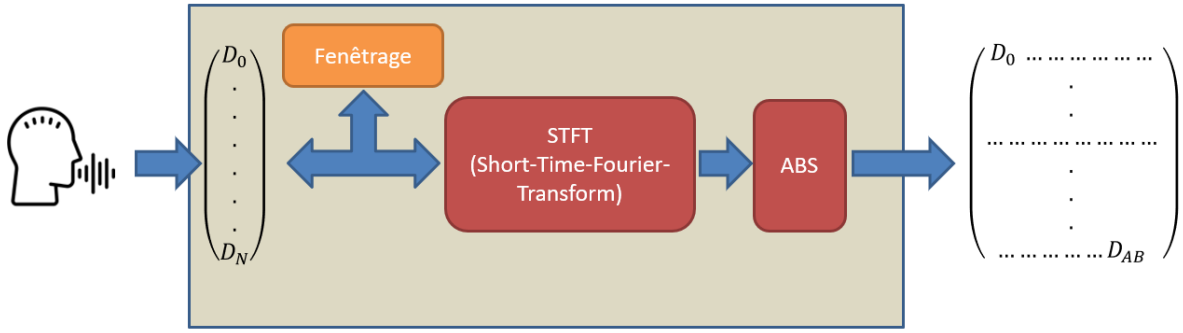
Le modèle a été créé en Python avec le Framework **TensorFlow**.

Ce document ne présente pas les explications techniques et choix du modèle, mais seulement l'utilisation. Il serait complexe et inutile de présenter ici ces aspects.

En résumé le modèle est un **réseau de neurones à convolutions**, un type de réseau adapté au traitement **d'images**. La raison est que nous n'utilisons pas directement les signaux audios bruts, nous allons convertir le signal est une **représentation temps-fréquence** appelé « spectrogramme » grâce à une transformation basée sur Fourier (**STFT** : Short-Time-Fourier-Transform). Ces images sont ensuite utilisées par le modèle qui va associer le pattern de cette représentation au mot-clé (Label).

Notre script se charge de convertir les données audios en images Temps-fréquence.

Voici un schéma de ce que va effectuer la transformation depuis le signal audio vers le spectrogramme.



Le fichier qui créé le modèle, et l'entraine est le fichier Python « **process_sound.py** ».

Voici un rapide aperçu de notre modèle (1,6 millions de paramètres à entraîner) :

```
model = Sequential()
model.add( Input(shape=input_shape) )
model.add( Resizing(height=32, width=32) )
model.add( Conv2D(filters=32, kernel_size=3, activation='relu') )
model.add( Conv2D(filters=64, kernel_size=3, activation='relu') )
model.add( MaxPooling2D() )
model.add( Dropout(rate=0.25) )
model.add( Flatten() )
model.add( Dense(units=128, activation='relu') )
model.add( Dropout(rate=0.5) )
model.add( Dense(units=output_shape, activation='softmax') )

model.summary()
# 1,625,221 parameters
```

Au sein du script, des informations sur les couches utilisées, sur le traitement avec Fourier sont indiquées.

<u>Etape 1 : Répertoire</u>

Créer un dossier « audio_data », et y placer dedans les répertoires des mots-clés contenant les signaux audios.

Si jamais les mots-clés ou le fichier de data sont différents, il y a juste à changer cela dans le script :

```
222  PATH_DIR = "audio_data"
223  CATEGORIES = ["avance", "droite", "gauche", "recule", "stop"]
224  K = 5
225
```

(Avec K le nombre de mots-clés)

Etape 2 : Lancer l'entrainement

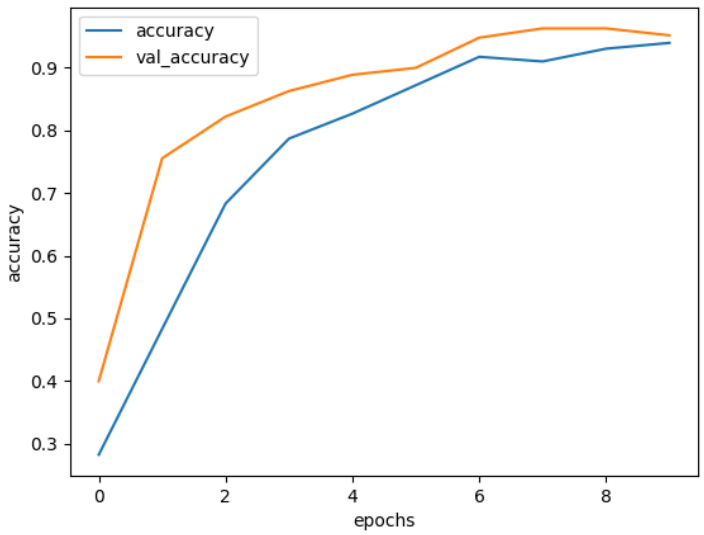
Avant d'exécuter le fichier Python, s'assurer que les librairies suivantes sont bien installées sur le PC : Numpy, Matplotlib, Tensorflow, tqdm. Dans le cas contraire, les installer avec pip : PC$ pip install <lib_nom> (ou pip3)

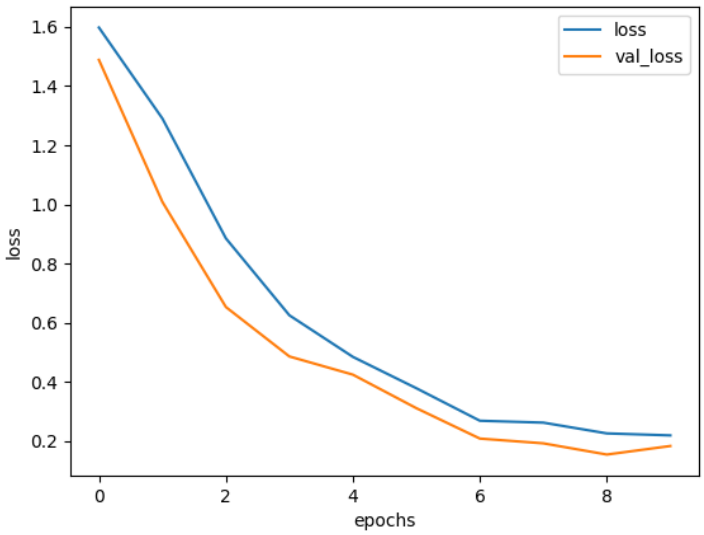
Puis exécuter le script avec : PC$ python process_sound.py (ou python3)

Le script va indiquer brièvement les différentes étapes avec des barres de chargement (librairie tqdm), comme le transformation en images spectrogramme, puis la création du dataset en array Numpy, et enfin l'entrainement du modèle.

A la fin, est affiché les performances du modèle, 2 graphiques qui représentent respectivement l'évolution du Coût (à gauche) et de la précision (à droite) du modèle.

Dans notre cas, nous avons une évolution du coût qui tend vers 0 (0.2) et une précision qui tend vers 1 (0.98), c'est un très bon modèle qui donne avec une bonne performance des résultats justes.





Également, le modèle entrainé va s'enregistrer sous le nom de « model_audio_reco_v1.h5 ». Ce type de fichier contient la topologie du modèle, et les valeurs de tous les paramètres du modèle.

Enfin, la console va afficher le Coût final et la précision finale du modèle.

c. Conversion en modele TFLite

Sur les cibles embarquées comme la STM32MP1, on utilise TensorFLowLite et non TensorFLow. Ce Framework permet de charger un modèle et de l'utiliser. Contrairement à TensorFLow, ce dernier est beaucoup plus léger, utilise des optimisations d'espace mémoire (conversion des paramètres en entiers 8bits,…) et des optimisations de ressources.

TensorFLow n'est de toute façon pas disponible sur la STM32MP1, seul TensorFLowLite est disponible.

Il faut donc convertir le modèle TensorFLow en TensorFLowLite.

C'est le rôle de la fonction « convert_TFModel_to_TFLiteModel() » qui prend en paramètre le modèle TensorFLow « model_audio_reco_v1.h5 », et va générer un modèle TensorFowLite « model_audio_reco_TFLite.tflite ».

```
def convert_TFModel_to_TFLiteModel(tfModel):

    tfLiteModel = None
    converter = tf.lite.TFLiteConverter.from_keras_model(tfModel)
    tfLiteModel = converter.convert()

    file = open( 'model_audio_reco_TFLITE.tflite' , 'wb' )
    file.write( tfLiteModel )
```

Enfin on envoie ce modèle TFLite sur la MP1 avec scp.

3. Inference du modele sur la MP1

a. Enregistrement audio en temps reel

Depuis JumpC, l'application de la télécommande, il est possible de piloter ScoutBOT avec le bouton qui effectue la commande vocale.

Ce bouton va effectuer dans le code C de JumpC un enregistrement de 2s similaire à celui vu précédemment et créer donc un fichier nommé « command_record.wav », puis va exécuter un script Python « inference_mp1.py ».

L'utilisateur a donc 2 secondes pour prononcer le mot-clé désiré.

b. Inference sur la MP1

L'opération d'inférence du fichier audio se réalise avec un script Python « inference_mp1.py » qui est responsable de la lecture du fichier audio, de l'utilisation du modèle TensorFlow et de l'enregistrement de la prédiction.

Il y a beaucoup de difficultés car des librairies comme TensorFlow ou Scipy ne sont pas disponible sur la carte.

Il faut commencer par installer 2 librairies : Numpy (opérations sur listes à N dimensions) et TensorFlowLite (voir wiki : X-LINUX-AI OpenSTLinux Expansion Package - stm32mpu)

La liste des packages AI dispo est sur ce wiki : X-LINUX-AI licenses - stm32mpu

Voici les commandes pour installer (La carte STM32MP1 doit être relié à internet via un câble Ethernet):

- Numpy : MP1$ apt-get install python3-numpy
- TFLite : MP1$ apt-get install python3-tensorflow-lite

Remarque : Pour utiliser TFLite sur la MP1 avec Python il ne faut pas importer la librairie normalement comme c'est le cas avec Nympy (« import numpy as np »). Mais avec une autre commande qui n'est pas indiqué sur le wiki, nous avons dû contacter des personnes de chez ST pour avoir la réponse : « import tflite_runtime.interpreter as tflr ».

➔ Lecture du fichier audio

La lecture du fichier audio enregistré est difficile sur la carte STM32MP1 car lecture d'un fichier audio WAV et sa conversion en une liste Numpy (Array) de flottants normalisés entre 0 et 1 était faite par la librairie TensorFlow.

Nous avons donc dû nous servir d'un projet trouvé sur le WEB qui transforme un fichier WAV en une array : audio2numpy/audio2numpy at c1759798b084f75d4e03739cf4e05d84861c4003 · wiccy46/audio2numpy (github.com)

En nous servant de ces fichiers, nous avons modifié le code pour qu'il donne le bon résultat, et surtout qu'il soit exécutable sur la carte MP1.

Au final, le script « inference_mp1.py » doit être à la même racine que 2 autres fichiers Python :

- Loader.py
- Wavdec.py

La fonction « read_audio_wav_file() » du scipt Python « inference_mp1.py » réalise cette opération de lecture et conversion en appelant le module Loader.

➔ Transformation en spectrogramme

De même la transformation d'une array numpy en spectrogramme était grâce à la librairie TensorFlow, nous avons donc dû refaire cette opération à la main en python.

C'est le rôle de la fonction « transform_into_spectrogram() ».

➔ Inférence avec un Interpréteur TensorFlowLite

TensorFlowLite, qui a été abordé précédemment va être utilisé pour réaliser une simple inférence du modèle ; c'est-à-dire une utilisation du modèle pour faire une prédiction.

Le modèle doit avoir été bien converti en préalable de TensorFlow à TensorFlowLite.

TensorFlowLite permet en réalité d'interpréter ce modèle, c'est pourquoi on va utiliser un interpréteur qui va interpréter le modèle.

Informations : TensorFlow Lite inference

```python
#reshape
spectrogram_input = spectrogram_input.reshape( -1, 62, 65, 1 )

interpreter.allocate_tensors() #before execution
# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Test the model on random input data.
input_shape = input_details[0]['shape']
input_data = np.array( spectrogram_input, dtype=np.float32)
interpreter.set_tensor(input_details[0]['index'], input_data)

interpreter.invoke()

vector_predicted = interpreter.get_tensor(output_details[0]['index'])

proba_max = np.max(vector_predicted)
label_predicted = np.argmax(vector_predicted)
```

Voici un aperçu du code de l'inférence via l'interpréteur.

Voici le résultat affiché et les informations de l'inférence :

```
root@stm32mp1:/usr/local/ai_mp1/scripts# ./process_reco.elf
Recording WAVE '/usr/local/ai_mp1/scripts/command_record.wav' : Sign
shape audio :  (16000,)
spectrogram shape :  (62, 65, 1)

 -- VOICE RECOGNITION --
predicted command :  gauche |probability :  0.9724008

Inference duration : 0.0769 seconds

 -- END --
```

Le résultat est l'index du Label, soit du mot-clé. Donc un chiffre de 0 à 4, 0 correspond à avance, 1 à droite, etc. En suivant l'ordre ci-dessous :

```python
CATEGORIES = ["avance", "droite", "gauche", "recule", "stop"]
```

Ce chiffre est enregistré dans un fichier texte via le script Python, et enfin est lu par le fichier C.

Puis selon le chiffre, JumpC va alors envoyer une commande de direction au Robot.

4. RESUME DES FICHIERS NECESSAIRES

Sur PC : répertoire SOUND

- Process_sound.py : Script qui créé le modèle, qui l'entraine et le sauvegarde
- Audio_data : Dossiuer contenant l'ensemble des données
- Audio_data/<mot-clé> : Dossier (1 par mot-clé) qui va contenir tous les fichiers audios du mot-clé en question
- Convert_TF_TFLITE.py : Script qui permet de convertir le modèle TF en modèle TFLite.
- Model_audio_reco_tf.h5 : Modèle TF créé à l'exécution du script process_sound.py
- model_audio_reco_TFLite.tflite : Modèle TFLite à l'exécution du script convert_TF_TFLITE.py

Sur STM32MP1 :

- model_audio_reco_TFLite.tflite : Modèle TFLite à l'exécution du script convert_TF_TFLITE.py
- loader.py et wavdec.py : Fichiers permettant la lecture de fichiers audio
- inference_mp1.py : Script Python responsable de l'inférence avec l'interpréteur TFLite.
- command_record.wav : fichier audio créé avec le script C de JumpC lorsque l'on demande une reconnaissance vocale.
- voice_reco_predict.txt : Fichier texte qui contient la prédiction du modèle, qui va être créé par le script Python, et lu puis supprimé par le script C de JumpC.