Single Audio to Inferred Animation

PROGRESS REPORT 2

COMP 400

Kevin Junyang Cui (260984715)

supervised by

Joseph Vybihal

McGill University

School of Computer Science

July 5, 2023

REPORT ON CURRENT PROJECT PROGRESS

0.1 PROJECT

0.1.1 VMC Protocol

An interface was developed in Unity built from based on the VMC protocol for the capture and playback of blendshape and bone positional/rotational data. VMC protocol, or **Virtual Motion Capture Protocol**, is a protocol built from Open Sound Control (OSC) protocol and VRM for the communication of motion capture data, especially for avatar or virtual motion capture.

VMC data is received via Unity OSC server from an existing motion capture client such as VSeeFace (which uses OpenSeeFace) as such:

- **Bone Transform**
  /VMC/Ext/Bone/Pos (string)name (float)p.x (float)p.y (float)p.z (float)q.x (float)q.y (float)q.z (float)q.w

- **VRM BlendShapeProxyValue**
  /VMC/Ext/Blend/Val (string)name (float)value
  /VMC/Ext/Blend/Apply

and subsequently exported to CSV as a 396-dimension dataset (18 blendshapes and 54 bones with a 3-tuple positional vector and 4-tuple rotational quaternion each, or $18 + 54(3+4)$) mapped to timestamps of a recorded audio at each call to */VMC/Ext/Blend/Apply*.

An OSC client is also implemented in Unity to convert the dataset back to scheduled VMC calls to replicate playback of captured or predicted values, synchronised to an audio player.

0.1.2 Torch Dataset

The data is imported from CSV to a Torch dataset where each VMC window is loaded with the corresponding time frame from the previous time of capture to the current time of capture (to be adjusted). Mel-frequency cepstral coefficients (MFCC) are computed from the time frames.

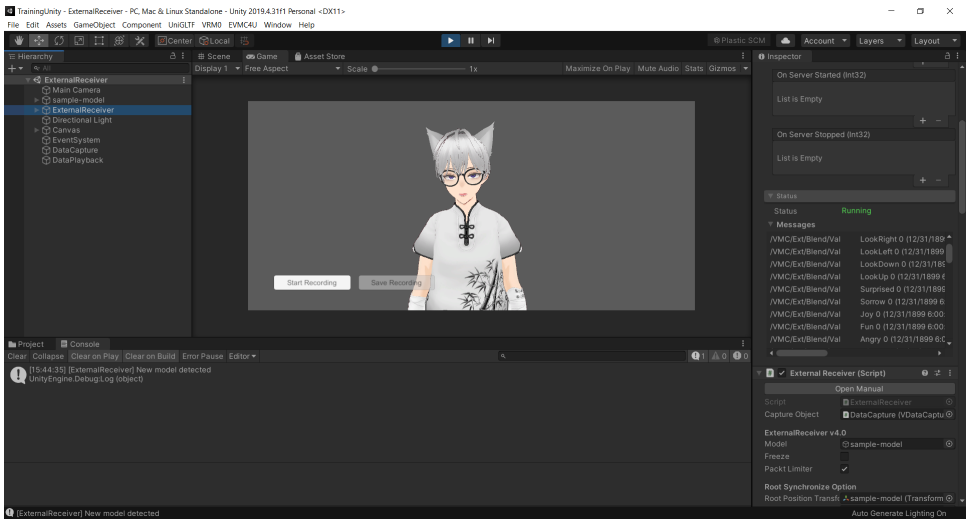

Figure 1: Interface using stock VRM model for VMC capture/playback

	A	B	C	D	E	F	G
1	0.027765	8.59E-18	0	0	0	0	3.60E-18
2	0.03335	7.23E-18	0	0	0	0	3.03E-18
3	0.034348	6.75E-18	0	0	0	0	2.83E-18
4	0.035214	5.95E-18	0	0	0	0	2.49E-18
5	0.036156	5.37E-18	0	0	0	0	2.25E-18
6	0.037216	4.72E-18	0	0	0	0	1.98E-18

Figure 2: Sample snippet from captured dataset

0.1.3 Training & Testing

Currently attempting to implement loss functions/architectures of the Bi-directional LSTM. Pydub is used to perform basic noise reduction filtration. The model has been implemented in PyTorch but has yet to be successfully trained.

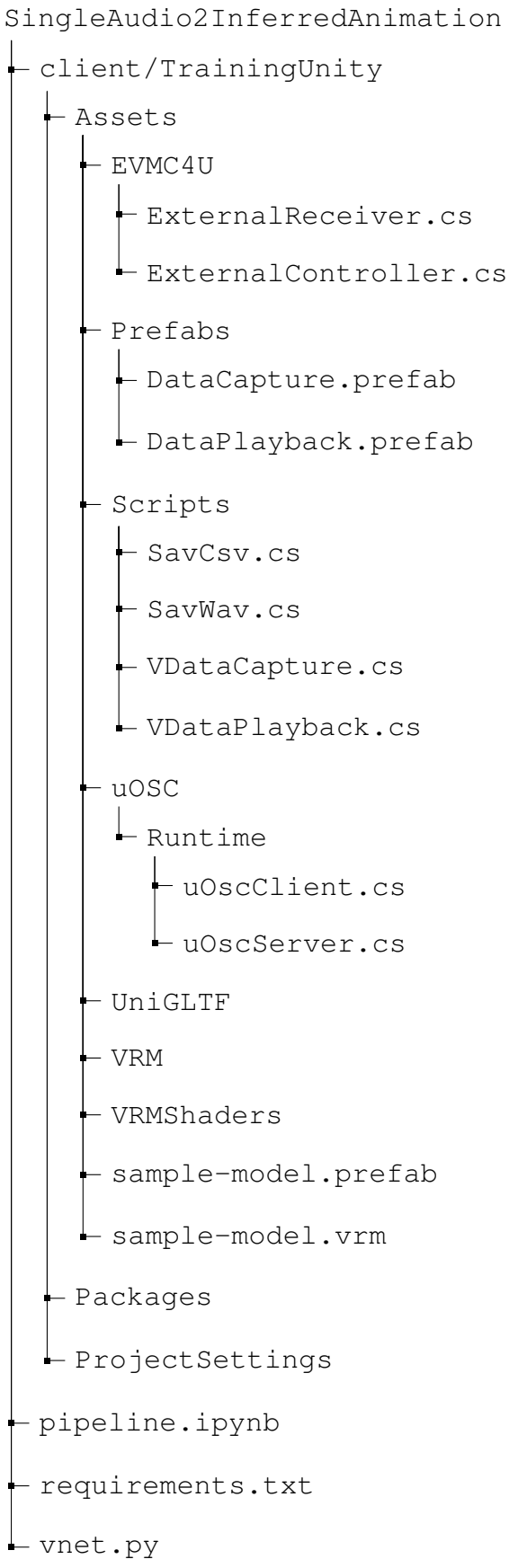

Figure 3: Directory structure of important files

PROJECT DEVELOPMENT PLAN

0.2 NEW ARCHITECTURE

0.2.1 Audio Feature Extraction

It can be hypothesised that the most relevant features in audio for animation (assuming speech in an arbitrary language) will be phonetic and tonal. That is, all features would be derivable from the MFCC of the target audio.[2][3] It can be noted that LPC is another possible avenue of feature extraction, but is observably inferior.[3]

Furthermore, as the model may be working with live production audio, it may be beneficial to implement some noise reduction filtration to isolate vocals. Each frame of MFCC can be mapped to the corresponding mean VMC values, which are the variables by which the loss function is calculated.

0.2.2 Bidirectional LSTM

A bidirectional LSTM with an embedded attention mechanism has been shown to be effective in training for audio to facial animation, as it gives the network memory of past input audio features.[3] Therefore, it would follow that a similar model may be effective in inferring upper-body animations. The LSTM would have an input layer corresponding to the MFCC of a select frame of isolated vocals and a 396-dimensional output layer of 396 VMC parameters which can be played back via the Unity interface. Adam optimiser may be employed.

The loss function, borrowed from Audio2Face's approach[3] taking into consideration unit selection in concatenative speech[1], will be some variation of the loss function containing two loss functions:

$$L = \frac{1}{n}\sum_{i=1}^{n} w_1 L_t(y_i, y_i^p) + \sum_{i=2}^{n} w_2 L_s(y_{i-1}^p, y_i^p)$$

where y is the desired output and y^p is the output produced by the network, w_1 and w_2 are the weights of the loss functions, and L_t and L_s are the target Huber loss function and smooth "concatenative cost" loss function respectively, where

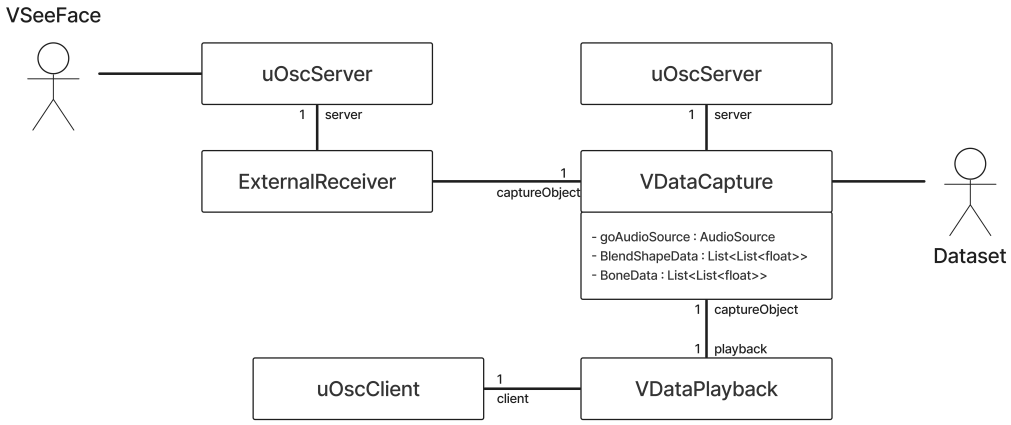

Figure 4: Domain model of interface

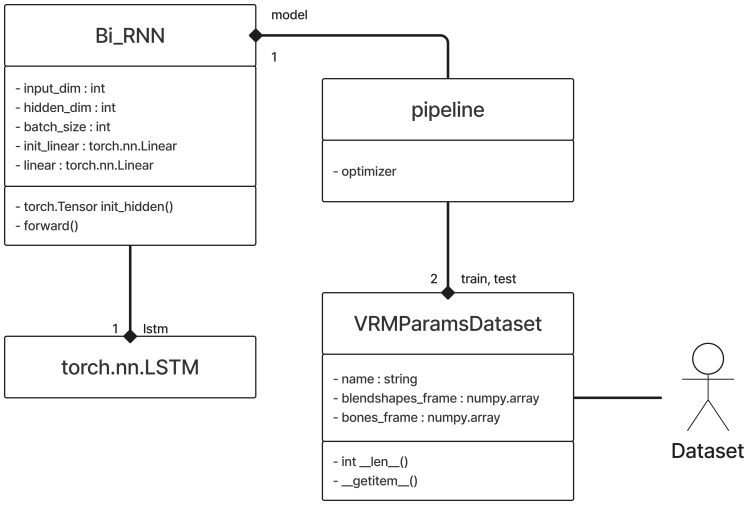

Figure 5: Tentative domain model of RNN training pipeline

$$L_t(y_i, y_i^p) = L_\delta(y_i, y_i^p) = \begin{cases} \frac{1}{2}(y_i - y_i^p)^2 & \text{for } |y_i - y_i^p| \leq \delta \\ \delta|y_i - y_i^p| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

$$L_s(y_i^p, y_j^p) = cos(y_i^p, y_j^p)$$

where δ is an adjustable hyperparameter initialised to 1. A Huber loss function is less sensitive to outliers in a dataset which may contain noise and converges faster than mean square error (MSE) loss and mean absolute error (MAE) loss to the minimum. L_s is a "concatenative cost" that ensures temporal smoothness between concatenated frames.[3][1]

The hyperparameters of the model can be adjusted empirically, quantitatively evaluated using a function such as the root mean squared error (RMSE) of parameters over all frames in a held-out test set.[3]

$$\varepsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^p)^2}$$

0.2.3 Animation

Once trained, the output of the model given some target audio input would be a 396-dimensional set of VMC motion parameters that can be played back on a VRM model via a Unity interface that converts the data back to VMC. Additional random blink patterns (on top of inferred blinking) may be optionally applied. Additional physics and shading is applied automatically by the Unity game engine.

0.3 REPORT ON DELIVERABLES

Table 1: Status of deliverables from report 1

Planned	Actual	Description of Task
15 June	6 July	Configure runnable base MFCC extraction and LSTM
22 June		Modify network to apply gesture parameters to loss function
29 June	15 June	Develop interface and recorder for training dataset
6 July	30 June	Collect data and create dataset of audio and parameters
13 July	22 June	Develop animation engine and base lip-sync
3 August		Train usable model for audio to animation inference
17 August		Train model which passes established test case(s)
24 August		Final project and report

0.4 NEW DELIVERABLES

Note on End of Semester Statement 1 *In report 1, it was stated that the project aims to achieve a certain degree of "naturalness" in motion. As discussed with the AlterEcho team, it is indicated by user survey that "naturalness" in a virtual production may be actually undesirable in contrast to "engagement". This can be taken into consideration for the overall objective when evaluating the results qualitatively.*

Table 2: New deliverable dates

Date	Description of Task
15 June	Develop interface and recorder for training dataset
22 June	Develop animation engine and base lip-sync
30 June	Collect data and create dataset of audio and parameters
6 July	Configure runnable base MFCC extraction and data loader
20 July	Configure trainable LSTM using loss function and optimiser
3 August	Train usable model for audio to animation inference
17 August	Train model which passes established test case(s)
24 August	Final project and report

0.5 NEW WORKFLOW & TECHNOLOGY STACK

The environment for the client is the Unity game engine using C# with Visual Studio Code. Unity plugins used include VRM4U, UniGLTF, uOSC, and a reverse engineered version of the Unity plugin EVMC4U. The neural network is configured in PyTorch with the Adam optimiser using Python and Jupyter Notebook. Version control is handled by Git, at the repository `https://github.com/kevinjycui/SingleAudio2InferredAnimation`.

0.6 MID-TERM DEMO

Expected date to demo TBD.

REFERENCES

[1] Andrew J. Hunt and Alan W. Black. Unit selection in a concatenative speech synthesis system using a large speech database, May 1996.

[2] R Raja Subramanian, Yalla Sireesha, Yalla Satya Praveen Kumar Reddy, Tavva Bindamrutha, Mekala Harika, and R. Raja Sudharsan. Audio emotion recognition by deep neural networks and machine learning algorithms, October 2021.

[3] Guanzhong Tian, Yi Yuan, and Yong Liu. Audio2face: Generating speech/face animation from single audio with attention-based bidirectional lstm networks, July 2019.