CSCE 421 Project Proposal

Sibo Min
Texas A&M University
College Station, Texas
minsibo0420@tamu.edu

Stern McGee
Texas A&M University
College Station, Texas
sternmcgee@tamu.edu

Abstract

The ability to interact with high fidelity artificial agents in virtual environments is a critical factor in producing a sense of presence from the user. Most virtual reality applications rely on traditional video game user interfaces for user interaction with virtual agents. We propose a solution involving a trained model that can recognize user voice commands and execute the action corresponding to the command. The proposed system's goal is to facilitate more natural interaction with virtual agents by utilizing conversational interaction that users are familiar with from real-world environments.

1. Introduction

Figure 1. VR Environment

The field of Virtual Reality has undergone extensive research as to the effectiveness of various user input methods for 3D user interfaces. Common approaches for user input in commercial VR applications include gaze-based point and click, hand controllers with 3-6 DOF tracking, and 6 DOF tracked gloves for hand gesture recognition.

User interaction with artificial intelligence in virtual environments can benefit from speech recognition. Conversation is a more intuitive method for users to express their intent.

In the range of audio recognition, there are numerous subtopics which are focusing on different types of sounds such as Animal Sound Recognition and Music classification. In research related to VR, the desired functionality is mainly to understand the user commands which could be under the subject of Speech Recognition.

For instance, in a simple setting of a virtual reality game, the user uses 6 degree-of-freedom tracked controllers to conduct a variety of tasks including locomotion, selection, and manipulation. If the Speech Recognition function is embedded in the system, users will be able to tell the system by voice the movement they desire to carry out. The environment will be able to first capture the user's voice, interpret the user command, and in the end giving feedback to the user by either performing the required action or instructing the users to repeat the command.

In the above example, the system is only expected to pickup the predefined executive commands given by the user, however, with or without speech recognition, an number of additional features can also change VR games enormously such as Voice Recognition in user login systems, Emotion Recognition, Natural Language Processing [10, **?**] in order to enable the system to understand complex meanings from multiple sentences.

In the recent decade, virtual reality games are growing in popularity as the supply of commercial VR systems has risen. An increasing amount of research has been conducted in improving the user experience of stereoscopically rendered virtual environments. It is intuitive for developers to relate VR with 3D audio effects since speech and hearing are crucial in daily situations within real-world environments. Nevertheless, in traditional games such as TRPG, role-playing games, and adventure games, developers resort to more familiar methods for simulating communication by letting the user select predefined questions and responses during dialogue segments. Implementing speech recognition in the VR medium as well as these game genres could greatly enhance the user's immersion in the experience.

The fields of artificial intelligence and virtual reality have traditionally been researched by independent groups,

but the fields are converging due to growing interest of simulating intelligence in virtual environments. The combination of artificial intelligence and virtual reality techniques is referred to as intelligent virtual environments. [4]

Since Automatic Speech Recognition is an important area that can be applied in many commercial products, numerous research has been done over this topic; many APIs and libraries have been produced for companies to embed the ASR features easily in the applications such as Tensorflow and Keras.

Understanding the difference between Natural Language Processing and Automatic Speech Recognition is vital in deciding which approach to use in this research. Natural Language Processing mainly concentrates on understand the underlying meaning or the ”true” meaning of human speech. Automatic Speech Recognition is mostly utilized to convert the audio into text. [2] In practical settings, most of the time ASR and NLP are both used sequentially in application development. Machine Learning has been a common approach in solving the ASR problems.

In search of similar solutions using intelligent agents in virtual environments, we found several papers that serve as previous works to our system. One system focused on direction and locomotion commands for control of a semi-autonomous mobile robot. [5] The commands consisted of natural spoken language and hand gesture input from the user. Another intelligent virtual environment system, The VirBot, facilitates the building of intelligent agents that can be driven around a virtual environment by responding to voice and gesture commands from users. [8] The agent consists of three components: multimodal input, natural language understanding, and planning and context recognition.

In order to implement the Speech Recognition system, the development team decided to interpret the problem as a classification problem. The system will be able to recognize 6-10 commands, each are composed of 1-3 English word. A machine learning model will be trained to match the audio input with predefined labels. In a practical environment, the model will receive a short audio input and output the label. The label, which is the same with command, will be sent to the VR program interface for future processes.

Regarding the simulation of the virtual environment, we want the solution to have the capacity to simulate as high fidelity a virtual environment as possible. Graphical quality, robust physics, and user interaction are important features of the simulation engine in case of continued work on the solution in the future. The Unity platform is an ideal game engine as it supports machine learning APIs and an acceptable level of simulation fidelity. [3]

2. Methods

The project is generally separated into two structures: the model part and the front-end virtual environment part. The machine learning model is developed using the Tensorflow API with python 3 as the programming language; the VR game is created as a Unity project which is based on C# language. The overall flow of the program starts with the VR game side, which captures microphone audio from user. The audio file is then sent as an argument when running the pre-trained model. The model sends back the resulting prediction in one string to the VR game once the prediction is done (fig: Overview). A more in-depth explanation of this can be found in the Communication Between Components section below.

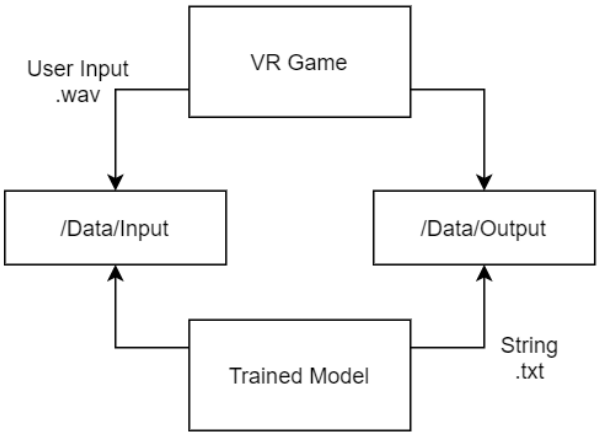

Figure 2. Overview of the Program Flow

2.1. Data Set

Speech recognition usually requires a huge set of data in order to obtain a proper accuracy. In order to produce a game prototype that is deliverable and improvable, the developers decided instead to use a existing data set proved to be high quality so that the model could be trained and performed as expected.

The data set used in this project is collected by Google and published as an open source. [6] This data set is chosen for multiple reasons. First, the data set is composed of over 105000 audio files in WAVE form spoken by different people. [7] Second, each audio file is one second long and contains only one single command which does not need to be further handled as input; the large amount of data is also a main advantage. There are thirty commands included in the data set and only ten of them are used in this project: ”yes”, ”no”, ”up”, ”down”, ”left”, ”right”, ”on”, ”off”, ”stop”, ”go”. Twelve labels are created accordingly with the addition of ”unknown”, and ”silence”. The data set itself is widely used by many researchers; great amount of machine learning tutorials use the data set as input are created for educational purpose. By studying tutorials related to the speech command data set, one example created by the Tensorflow API proves to be a great guideline for the

Figure 3. Overview of the Program Flow

model training of this project. The researchers have studied the tutorial and the provided code thoroughly to understand the procedure in training. [9]

2.2. Tensorflow Speech Commands Example

A convolutional neuron network is used as the main structure of the model in the Tensorflow Speech Command Example. [1] The specific CNN model is subject to keyword spotting tasks. The way that the provided code trains the model is very similar to training a model in classifying photos. The audio files are transformed into spectrograms in order to increase the dimension of the data set.

The entire data set is divided into three sections: training set(80% of the data), validation set(10%), and testing set(10%). The learning rate increases as the training step increases to enable a higher efficiency in training the model. The program is also able to create check points so that it is possible for the user to pause the training and come back to where the training stopped later on. It provides the accuracy of prediction again the validation set which.

2.3. Communication Between Components

As described in the overview of the project, the virtual reality component of the solution is run by the Unity game engine which is driven by C# scripting. The model runs by scripts written in Python. As a result, it was necessary to implement a communication channel between the two components so that the overall system could function seamlessly in real-time for the user.

From a program flow perspective, the first communication of data across components needs to happen when the user speaks a voice command in the VR experience. A C# script records a 1 second WAVE audio clip from the user's default microphone and saves it to disk within Unity's asset directory. The C# script then begins a command line process to execute the model's python script with an argument being the file path of the voice command audio file.

The second cross-component communication of data happens after the model outputs the string that represents the recognized user command. The Unity C# script waits for the command line process to finish execution and reads the resulting output as a string. The string is then compared to strings mapped to the current list of possible commands; when a corresponding command is identified, the avatar controller script is notified to execute the action associated with the respective user command.

3. Experiments and Results

3.1. Challenges with Model Training

Training model took place on the one of the developer's laptop and the entire process lasted for several hours. The training process includes 18000 steps of training in order to reach a high accuracy. Once the model is done training, the model information is saved into a protbuf file(.pb) which contains the graph definition as well as the weight of the model so that the deliverable does not need to include the training data and the scripts for training.

Meanwhile, because of the hardware limitation, only one model is trained successfully while in the earlier stages of the project, it is expected by the developer to train the model multiple times and compare its accuracy in game using actual user input. The developer was only able to compare the trained model with a online pre-trained model to determine the performance and both model act in a similar fashion.

3.2. VR Game

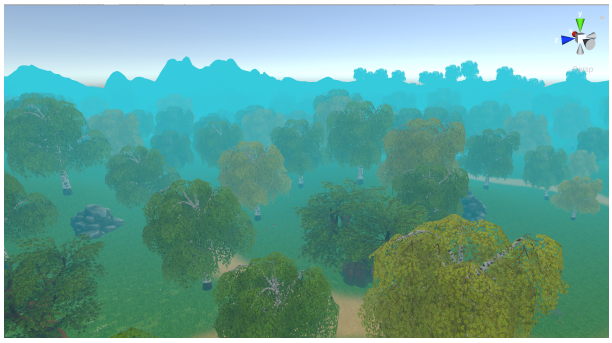

Figure 4. Overview of the Program Flow

The development team initially planned to evaluate results by conducting a user study to measure accuracy of the model's recognition of user voice commands, but due to the COVID-19 pandemic it was not feasible to gather users to collect data. The system was tested informally by the developer that had the VR system, and the model was found to recognize commands accurately a majority of the time. Ideally the system would have been tested with unique voices of many different users; the lack of a formal evaluation is a limitation that should be noted. In the future, the team would like to conduct a formal study to determine the system's robustness if the opportunity presents itself.

Figure 5. Overview of the Program Flow

4. Conclusion and Future Expectation

The completed game prototype has proved that the usage of speech recognition can be applied to VR Gaming easily and produces a better gaming experience. The prototype in this project is relatively short in length due to various limitation.

The project itself is subjected to keyword spotting task, command recognition. It costs not too much to implement due to the fact that the amount of researches over this area is numerous. It brings obvious improvement to the game and free the player from the handler in many situations; the ratio of effect against price is high. After developing the game prototype, the developers realized that it is likely that when it is applied to a fully developed game, the short command recognition could provides even better experience for the players when the agent gives a clear prompt on the options. Consider the conversation below which happened in a dating simulation game:

Agent: Would you marry me?
Player: Yes.

Comparing requesting the player speaking the word "yes" with selecting options from a menu, it is apparent that even a simple interaction between the agent and player through short speech could bring magical effect to the game.

In this project, a frozen model is provided as the recognizer in the game. However, the data set is still growing by size; the trained model in the future is likely to even reach a higher accuracy. When it comes to a published game, the concurrent renewing of the model should be applied.

5. Task Assignment and Acknowledgement

This project was completed by the cooperation of Sibo Min and Stern McGee, under the instruction of Dr. Zhangyang (Atlas) Wang and Zhenyu Wu. Sibo Min took charge of training and testing the model, and Stern McGee took charge of the model implementation within a virtual reality environment. They both collaborated on this report. The data set used was the Speech Commands data set provided by Google. [6]

Figure 6. Overview of the Program Flow

Figure 7. Overview of the Program Flow

Figure 8. Overview of the Program Flow

References

[1] Google. Convolutional neural networks for small-footprint keyword spotting. 2015.

[2] Curry I. Guinn and R. Jorge Montoya. Natural language processing in virtual reality training environments. 1998.

[3] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents, 2018.

[4] Michael Luck and Ruth Aylett. Applying artificial intelligence to virtual reality: Intelligent virtual environments. *Ap-*

plied Artificial Intelligence, 14(1):3–32, 2000.

[5] Dennis Perzanowski, Alan Schultz, and W. Adams. Integrating natural language and gesture in a robotics domain. pages 247 – 252, 10 1998.

[6] Google Brain Team Pete Warden, Software Engineer. Lanching the speech commands dataset.

[7] Mountain View Pete Warden, Google Brain. Speech commands: A dataset for limited-vocabulary speech recognition. 10 2018.

[8] Jesus Savage, Mark Billinghurst, and Alistair Holden. The virbot: A virtual reality robot driven with multimodal commands. *Expert Systems with Applications*, 15:413–419, 10 1998.

[9] Tensorflow. Simple audio recognition.

[10] Dean Weber. Object interactive user interface using speech recognition and natural language processing. 1999.