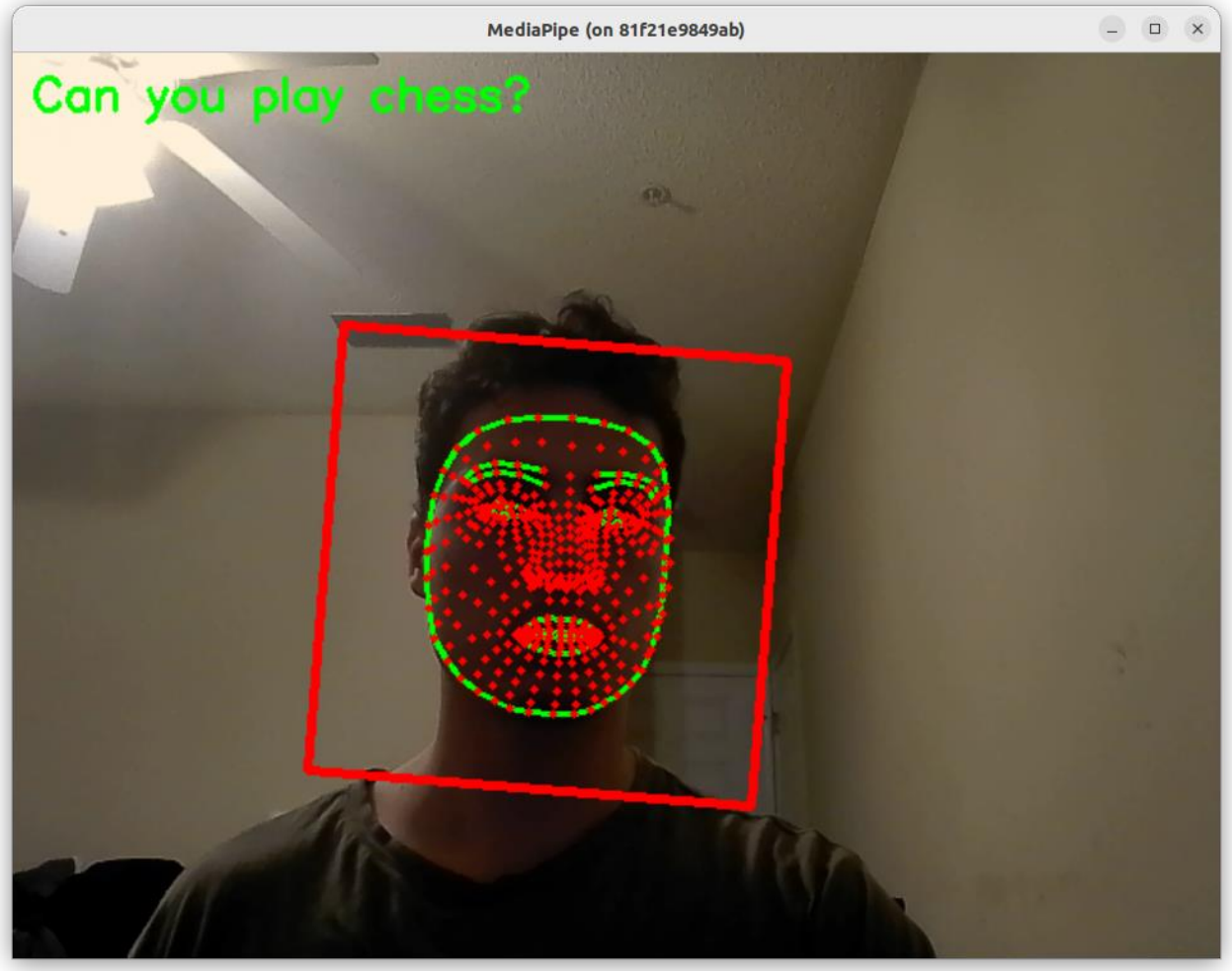
Neoklis Vaindirlis

VirtuSense Internship code documentation

Exercise 1



Introduction

I approached this problem by first reviewing current literature for top performing CV face mesh models that are open sourced. Mediapipe stood out, promising very high accuracy as well as availability in both Python and C++.

Usually when I approach a new prototyping problem I always starting working on it in Python, because of its simplicity and aboundance of libraries for working on any problem.

In this case I designed the basic structure of the program in Python then transitioned to C++ for achieving also the bonus points.

We will use Mediapipe's facemesh pipeline. This pipeline has a graph that takes as input an image, detects face(s) and calcualtes a total of 470 (x,y,z) points covering the whole face and returns an array of this points with their x,y location normalized as a percentage of the image width/height (0.0-1.0). z represents the depth of the point. The smaller the value the closer the landmark is to the camera. (A second graph also takes as input the result of the first graph and the input image and returns an annotated image tracking the user's face shown on the application window).

NOTE: the graph has been setup to detect only one face.

Tracking algorithm explanation

In our case I designed a simple algorithm for detecting nodding and shaking gestures working with the following steps:

1. Capture points of interest for tracking head movements. I chose one point on the chin and one on the sidehead. This points are pushed in a list holding the numFramestoAnalyze more recent points. The meaning and purpose of this value will be explained below.
2. Capture two points, one on top of the head and another one on the bottom of the head. The distance of this two points is calculated and serves as an indicator of the proximity of the head relative to the camera for readjusting the sensitivity of the algorithm based on this distance. The closer the user , the less sensitive the algorithm, because their moves will look more drastic on the camera
3. The function scanDirection checks for abrupt depth changes on the chin/sidehead list for detecting nodding/shaking respectively by monitoring their z value. When it detects an inflection point it returns a positive value, indicating movement. In the case of nodding however, it only considers an inflection positive when moving only on one direction, more specifically nodding down. That also explains the requirement of the nod boolean
4. The if clauses afterwards validate only specific movements.

   -More specifically the first if clause checks for nodding by requiring a positive scanDirection value at the nodding axis but zero scanDirectionvalue on the shaking axis as well as a limit on the excessive movement of the head by checking translation on y axis by finding minimum maximum values.

-The second if clause does the same but for the shaking movements.
-The third if clause simply reports invalid movements, by tracking excessive movement or simultenous depth changes on both x,y.

5. FramesToAnalyze explanation: This variable changes in real-time according to processing speed in oder to adjust for slower/faster computers as well as any performance flactuations that may occur in real-time. The reason this variable cannot be static will be explained with an example. Using the same framesToAnalyze on a slower computer can misclassify a user slowly moving his head as nodding/shaking because the aanlysis took place in a bigger time interval. This value is adjusted by measuring in realtime the time it takes to process 10 frames using the std::chrono library

NOTE: Comments are also placed on the code itself explaining the function of every line.

Other functions

- A total of 100 general yes/no questions were generated using ChatGPT and saved on a text file. When the program starts it randomly select a random index and begins asking questions.
- The questions,user answer and timestamp are saved on a csv file

Porting from Python to C++

- This was a quite difficult process due to very limited avaiable documentation from Google for the Mediapipe API in C++ as well as limited and incorrect instructions for setting up the environment. Installing the correct dependencies and configuring build files was a very long proces. However, I managed to successfully setup the environment in my Ubuntu 22.04 system.
- The mandatory use of the bazelisk builder requires placing the c++ source files in the examples directory

Dockerization of the Application

- The above reasons make it a good idea to dockerize the c++ application. Thankfully google provides a dockerfile, although still problematic. After some modifications I successfully made it work. When launched it automatically runs the application.

Instructions for running application

The log files will be saved in docker-volume. A camera is required to run the application.

1. `docker build --tag=mediapipe .`
2. `xhost +local:docker`
3. `mkdir -p docker-logs && docker run --device /dev/video0:/dev/video0 -v $PWD/docker-logs:/facemesh_files/logs -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix -it --name mediapipe mediapipe:latest`

`To close the application simply hit Escape`

Source code

The source file is located in
mediapipe/mediapipe/examples/desktop/ demo_run_graph_main.cc

`I have also made a copy (ex1.cc) in the root directory for your convenience`