Assignment 6: Gesture Recognition (20P)

Goals

You know different methods for recognizing unistroke gestures and can implement them. You can apply advanced computer vision libraries to implement pointing techniques. You are familiar with LSTM neural networks. You can evaluate and compare approaches and report the results accurately. You can include robust gesture detection into simple applications.

1 Implementing the $1 Gesture Recognizer 4P

Implement Wobbrock's $1 gesture recognizer. You can find a description on Wobbrock's webiste[1]. Train the recognizer with at least five gestures including **rectangle**, **circle**, **check**, **delete**, and **pigtail**. Build a little user interface with *pyglet* where users can enter gestures to test the gesture recognizer.

Score

(**2P**) $1 gesture recognizer works and is efficiently implemented
(**1P**) Five gestures can be distinguished
(**1P**) Gesture entry user interface

2 Mid-Air Gestures with $1 Recognizer 6P

Create a new Python program called *pointing_input.py*. Use the Google MediaPipe library to detect users' hand and/or body pose. It might be necessary to counteract low camera frame rates by interpolating between frames! It should map suitable body landmarks to a pointer position. Then, use the *pynput* library to move the mouse pointer. Integrate your pointing system into your $1 gesture recognizer application. Now, you should be able to detect unistroke gestures entered with the touchpad/mouse and by mid-air pointing.

Adapt this program in way that allows it to record a gesture. Recorded data should be stored in *.xml* files with a format compatible to Wobbrock et al.'s unistroke gesture logs (see Website). Record at least **ten gestures for each of the five gestures** mentioned above in the description of task 1.

Score

(**1P**) Pose detection works reliably and with low latency
(**1P**) Mapping of landmarks to a pointer position works
(**2P**) Application controls the mouse pointer
(**1P**) Mid-air gesture interaction is integrated into $1 recognizer
(**1P**) Recording works and recorded data is stored appropriately

[1] `https://depts.washington.edu/acelab/proj/dollar/index.html`

3 Comparing Gesture Recognizers 5P

Test Datasets. Create a first test dataset based on the mid-air gestures recorded by you, and a second one from the unistroke gestures using Wobbrock et al.'s unistroke gesture logs. The two test datasets should have the same dimensions.

Train an LSTM Classifier. Train an LSTM classifier with the remaining data of Wobbrock et al.'s gesture logs only. Then, try to systematically reduce the LSTM's parameter count. Create at least three different versions.

Evaluation and Comparison of LSTM Classifier. Compare accuracy and prediction times of the LSTM with different parameter counts, as well as the $1 gesture recognizer. Do this for both datasets, the mid-air dataset based on your recordings and the touchpad/mouse dataset based on Wobbrock et al. Report method and results briefly in a Jupyter Notebook called *unistroke_gestures.ipynb*: Which one performs better for touchpad/mouse input, and which one for mid-air input? Is there at least any (significant) difference? Interprete your results briefly: What could be the reasons for better performance or no difference at all, respectively. Which one would you choose for a practical application, and why?

Score

(**1P**) Both datasets have the same format and dimension, they are loaded correctly
(**1P**) LSTM gesture classifier works and is trained on Wobbrock et al.'s gesture logs
(**1P**) Systematical comparison of different parameter counts and the $1 recognizer
(**1P**) Proper visualization and description of results
(**1P**) Conclusive reasoning when comparing the LSTM vs. $1 and touchpad/mouse input vs. mid-air input
(**1P Bonus**) Correct evaluation and description with significance tests

4 Gesture Detection Game 5P

Create a Python program called *gesture_application.py*. The program should be a very simple 2D application, such as a game, a media controller, an application quick start menu – you name it. The program can be controlled with gestures (e.g. rectangle, circle, check, delete, ...) drawn by a user.

The application should support at least three distinct gestures. Ideas for a game would be something like rock-paper-scissors, or the minigame to learn new spells from *Harry Potter* PC games[2], where players had to follow a specified shape. Hint: Of course you can copy your code of the gesture input application of task 1 and build up on it!

Score

(**1P**) Gesture input works with touchpad/mouse and mid-air
(**2P**) Functionality and aesthetics of the application
(**2P**) Three gestures are distinguished robustly

[2] `https://gamefabrique.com/storage/screenshots/pc/harry-potter-and-the-philosophers-st`
`png`