Build and Deploy Documentation

Initially we are going to work on the Raspberry Pi model 4 with 4GB RAM. The programming language of the project is Python.

1. First of all, to set up the Pi board with a 64 Bit OS. If you want to test on an ordinary PC device then skip to the section Project Setup. Currently in the project we are using the release of **raspios-bullseye-arm64** (Download Link) on the Pi board.
2. By using the Raspberry Pi Imager (https://www.raspberrypi.com/software/), the OS files can be flashed into a SD card that is going to be inserted to the Pi board later.
3. Once the Pi board have booted successfully, a Python interpreter (version >= 3.7) will be required to be installed on the Pi board. As the **raspios-bullseye-arm64** comes out with integrated Python 3.9, then we only need to configure the python virtual environment for the project. To do that, just follow the instructions in the section Project Setup.

Some modules must be manually installed for the raspberry pi, find the download links for them below:

- tensorflow-2.8.0-cp39-cp39-linux_aarch64.whl: Download Link (Provided by Q-engineering)
- torch-1.10.0a0+git36449ea-cp39-cp39-linux_aarch64.whl: Download Link (Provided by Q-engineering)

Once the runtime environment setup is done, just follow the instruction in the section Run Program to execute the script, which will start to monitor the panellist in front of the TV all the time unless the power supply (ideally from the TV) is interrupted.

In the productive scenario, the program should be executed automatically when the power supply is provided to the device.

Project Setup

- Clone this project including its submodules to your local disk using `git clone --recurse-submodules https://github.com/amosproj/amos2021ws04-auto-panelist-detection.git`.
- Go to the project directory using `cd amos2021ws04-auto-panelist-detection`
- Create and enable your python virtual enviroment.
- Run `pip install -r requirements.txt`.

Run Program

The Automatic Panelist Detection program can be started using the following commands:

```
cd Implementation
python main.py
```

The logged data can be received either via a REST api or via the MQTT protocol. This can be specified by setting the `API` parameter in the `Implementation/main.py` file.
For MQTT, the `Implementation/api/transmission.py` file needs to be modified and your MQTT broker, username and password should be provided there.
The `CAMERA` parameter in the `gui/gui.py file` can be modified to specify which camera should be used.

Create Performance Reports

Performance reports for using different images and recognition models can be created automatically using the following command:

```
cd Implementation
python benchmark.py
```

This creates two reports: One for the detection benchmark and one for the recognition as well as the age, gender, and emotion detection benchmark.
Both reports are saved as .xlsx (containing the tested images) and .csv files.

The images used for both benchmarks can be specified in the respective JSON files `Implementation/test images/detection_benchmark.json` and `Implementation/test images/recognition_benchmark.json`.
These files also include the corresponding labels.

The benchmarks can be run for different scaled versions of the given images. The image scales can be set by changing the `IMG_SCLAE` parameter in the `Implementation/benchmark.py` file.

Generate Statistics

User statistic can be generated based on the logged data using the following command:

```
cd Implementation/statistics
python calculate_statistics.py
```

This stores various user statistics in the `Implementation/statistics` directory. It also shows plots with the generated data.