system documentation

ELEC-C9820 - Design Thinking and Advanced Prototyping

Table of contents:

1. System's overview

System solves the problem of hosts wanting to know how many people visit their exhibition, collect more fine-grained data to understand which parts of the exhibition are the most popular and how visitor numbers change with time.
The system consists of:

- Computer vision device capturing the exhibition and tracking visitors behavior
- Backend used to store data, perform the analysis and collect statistics. A backend server can handle many devices: every user can have several, there can be multiple users
- Web app for displaying results to the user

2. Computer vision device

We built a Raspberry Pi-based prototype with core functionality: it captures video and uses computer vision for counting the daily number of visitors, which are regularly pushed to the storage via data API. Once booted up, the Pi automatically connects to "aalto open" wifi network and starts the python script responsible for capturing and processing video.
Script constantly fetches new frames and uses OpenVINO-based object detection model (person-detection-0202) to find people on them, detections are filtered based on a threshold and the information about every detection is reduced to a single point (center of a rectangle).
System assigns incremental ID to every new point in order to keep track of them through several frames. We match points from two consecutive frames based on the minimal euclidean distance and copy the ID of the previous point for the new one. Only if no corresponding old point was found a new ID is generated.
Script keeps a backlog of points for several previous frames to form a track of a person's movement. Every frame it checks all the tracks that are long enough for intersection with the line, and, if it does intersect, its ID is added to a set of ID which have been counted as entered. Set is used to count each ID only once.
The script also includes two additional threads: one that periodically sends the number of people counted to the server; another waits until midnight, after which it resets the counter by emptying the set and sleeps until the next midnight.
The user can edit the following parameters:

DEVICE_ID - numeric ID of the device used for the submission to the server (allows to have multiple devices)
SERVER_URL - URL of the server hosting data API
MODEL_LOCATION - location of the model description file
DIST_THRESHOLD - threshold distance for the points on two consecutive frames to be treated as belonging to the same person
VIDEO_SOURCE - either 0 to use camera, or name of the video file (for testing purposes)
CONFIDENCE_THRESHOLD = threshold for person detection model confidence
ID_AGE_THRESHOLD - how long a track crossing a line should be to count
LINE - two points defining a line as a tuple of tuples: ((x1, y1), (x2, y2))
SHOW_IMG - show image with the current frame, line and counter value (for testing purposes)

SENDING_FREQ = how frequently to push the data to the server (in seconds)

3. Web App

We built a simple and user-friendly web application with vanilla JavaScript and Node.js to support the Back-end.

- Front-end

The web application has a calendar, which shows the current day when the user accesses it, and by clicking on a date, the information about the number of visitors on that day will be shown next to the calendar. There is also a small pop-down calendar so that the user can choose a range of time and click the button “Get data” then a chart that summarizes the data and will be shown below the main calendar. The front-end is built by vanilla Javascript, and basic HTML & CSS.

In the first case, by clicking any date on the calendar, the front-end will call the method GET to contact the back-end for the data, and the back-end side will transfer the date that will be rendered immediately. In the second case when users may want to choose a range of dates, the method is similar to the first case, and it will render the chart which shows the statistic.

- Back-end

In the back-end, we implemented the REST API to communicate with the front-end. Backend server is running on Node JS and features two endpoints: /api/stat and /api/device implementing following methods:

- POST to /api/stat?id=*{id}* adds/updates a record for a device identified by {id}. The record data should be in the body of http request, in JSON format with following structure: {“date”: {YYYY-MM-DD}, “visitors”: {visitor count}}
- GET to /api/stat?id=*{id}*&startdate=*{YYYY-MM-DD}*&enddate=*{YYYY-MM-DD}* gets records for the specific device - either all of them if only id is present, or for the specified period if startdate and enddate are specified. Data is returned in the following format: [{'date': 'YYYY-MM-DD', 'visitors': n}, {'date': 'YYYY-MM-DD', 'visitors': n}, …]
- DELETE to /api/deviceid?=*{id}* deletes all the records for specified device

Server uses a JSON file to store data in the following format:

```
{
   "id1": [ {'date': 'YYYY-MM-DD', 'visitors': n}, {'date': 'YYYY-MM-DD', 'visitors': n} … ],
   "id2”: [ {'date': 'YYYY-MM-DD', 'visitors': n}, {'date': 'YYYY-MM-DD', 'visitors': n} … ],
   …
}
```