Project Demonstration: Automated Jersey Number Recognition

CREATED BY: NGUYEN NGUYEN, JEFF REIDY, AND NOAH WAGNON

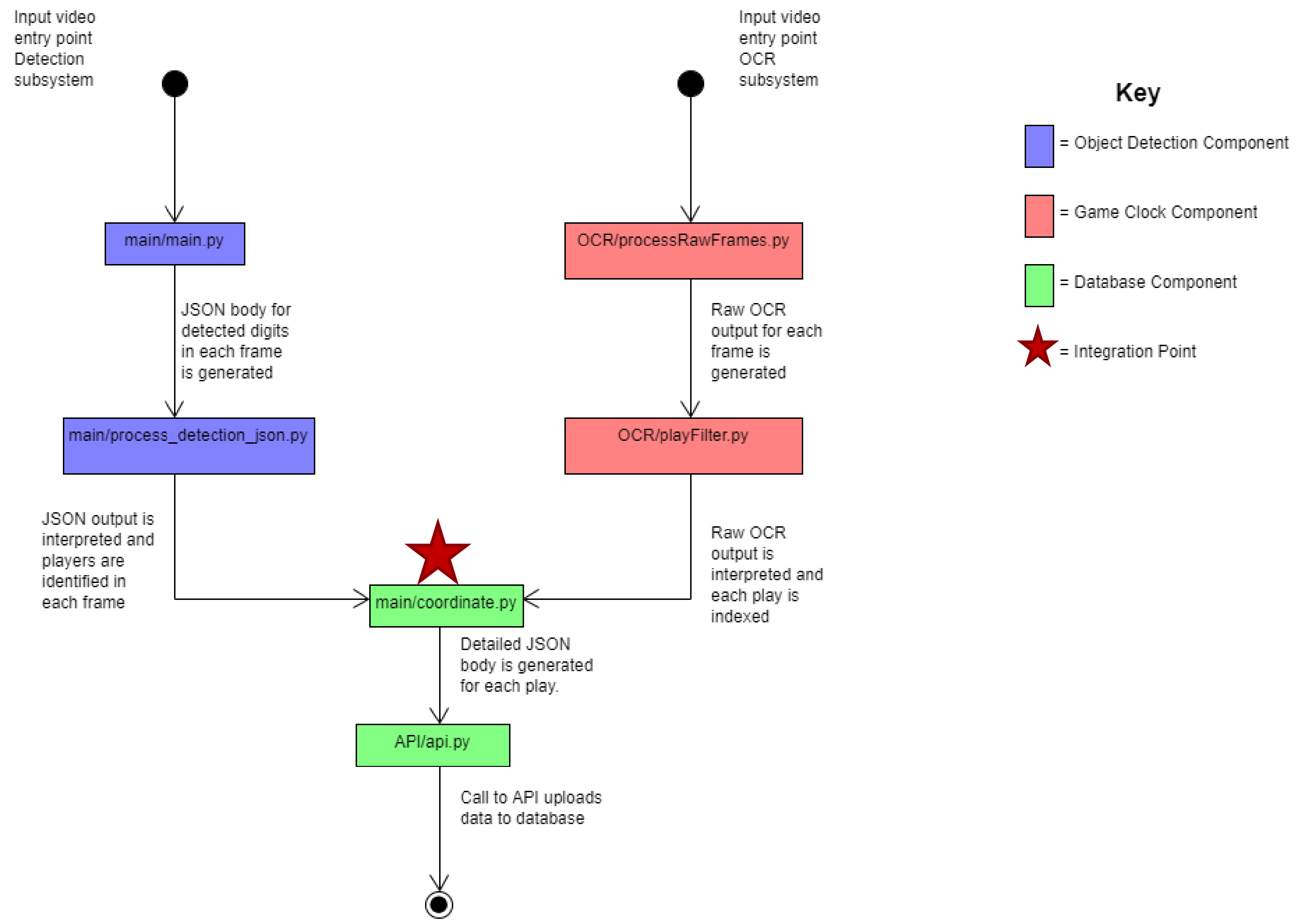
THIS PRESENTATION ALSO SERVES AS A GUIDELINE FOR EXECUTING THE SOFTWARE FOR ANY FUTURE USERS.

Project Setup

- Clone from repository: https://github.com/ndwagnon/Automated_Jersey_Number_Recognition
- Recommended OS: Linux (MacOS if Linux is not available)
- Prerequisites: (Python3, Pip3, Anaconda)
- Setup Environment: from root, run “conda env create –file footballEnv.yml”
  - Next run “conda activate football” to enter your newly created environment
  - Note: you may have to manually install some packages with pip via “pip3 install <package>”
- You should now be setup to run the projects scripts.
  - Download any test videos to the project’s root folder
  - To train a model, run “python Mask_RCNN_Scripts/train.py <dataset>” where “dataset” is either svhn or football. Include an optional command line argument specifying a weights file if you are retraining.

System-as-a-Whole Execution Flow

- Each listed component is a python script.
- Input for each process serves as output to next script.
- Object Detection and Game Clock components initially run in parallel.


Input video entry point Detection subsystem
Input video entry point OCR subsystem
main/main.py
OCR/processRawFrames.py
JSON body for detected digits in each frame is generated
Raw OCR output for each frame is generated
main/process_detection_json.py
OCR/playFilter.py
JSON output is interpreted and players are identified in each frame
Raw OCR output is interpreted and each play is indexed
main/coordinate.py
Detailed JSON body is generated for each play.
API/api.py
Call to API uploads data to database
Key
= Object Detection Component
= Game Clock Component
= Database Component
= Integration Point

Step 1A: Start the Detection Software

- Command: “python main/main.py <pathToVideo> <color>
  - pathToVideo -> video name such as “LSU.mp4”
  - Color -> either “crimson” or “white”. Case sensitive
  - Typically takes the real video length to process.
  - Can end early with “ctrl + c” and you will not lose results.
- Output: navigate to the “debug” folder and examine “JSON” and “frames”
  - JSON folder: contains the JSON structure for each play. Stores absolute frame index, ROI’s of digits seen, numbers of digits seen, and confidence values of digits seen
  - Frames folder: contains the original frame, each person proposal, and the labelled result frame for every processed frame.
  - The JSON folder will be interpreted during step 2.
- Two OpenCV windows will display. One contains the live video being played, the other contains the real-time detection output.

STEP 1B: Start the OCR Software

- Note: Use a separate terminal tab to run OCR software simultaneously while the detection software is running in the original terminal tab.
- Command: python OCR/processRawFrames.py <pathToVideo>
  - pathToVideo -> video name such as “LSU.mp4”
  - Terminal will show live OCR readings
  - Typically takes 10 minutes for a 40 minute video
  - Can use “ctrl + C” to end processing early and you will not lose results.
- Output: “RawOCRFrames” folder and “ocr_raw_output.txt”
  - RawOCRFrames folder: contains raw input frames and cropped frames. Can be viewed for debugging if adjusting crop window
  - ocr_raw_output.txt: contains absolute frame #, game clock, and snap clock for each frame
  - The .txt file will be interpreted as part of step 2.

Step 2A: Process raw detection results

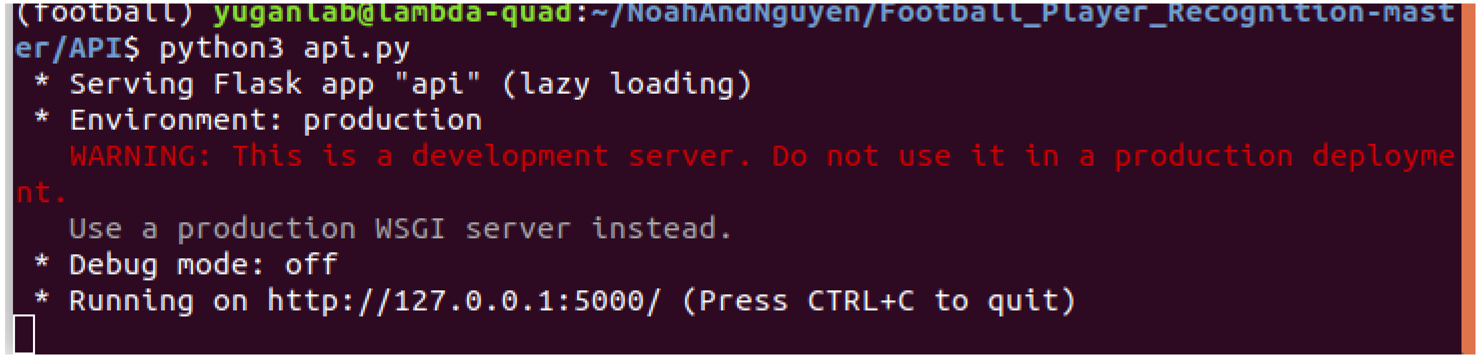
- Command: python main/process_detection_json.py
  - Typically executes instantly
- Output: detection_output.txt
  - Contains absolute frame number for each frame and a list of jerseys seen in that frame
  - This .txt file will be combined with the results from step 2b during analysis in step 3.

Step 2B: Process raw OCR results via a play filter.

- Command: python OCR/playFilter.py
  - Typically executes instantly.
- Output: ocr_filtered_output.txt
  - Contains the serial play number, quarter number, start/end time for each play seen
  - .txt file will be combined with results from step 2A to be analyzed and integrated in step 3.

Step 3A: Open the API on a local server

- Command: “cd API” followed by “python3 api.py”
  - Simply opens the local http server running the API
  - Must be performed on a separate terminal window.
  - If terminal looks like screenshot below, http server is operating correctly.



```
(football) yuganlab@lambda-quad:~/NoahAndNguyen/Football_Player_Recognition-mast
er/API$ python3 api.py
 * Serving Flask app "api" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Step 3B: Combine OCR and Detection Results and post to API

- Command: “python main/coordinate.py”
  - Terminal displays response code (200 indicates a success) for every attempted post.
  - In python3, perform the following to load api contents within external software
    - url = f"""http://127.0.0.1:5000/plays"""“
    - Response = requests.get(url)
    - Response.json() allows you to import the data as a json body
- Output:
  - Generates “plays” folder containing JSON structure for each play.
  - Communicates with open API to post play data to API/plays.csv

Step 4: Clean intermediate files before running on next game

- Files that need to be cleared or deleted:
  - Debug folder
  - RawOCRFrames folder
  - Ocr_raw_output.txt
  - Ocr_filtered_output.txt folder
  - Plays folder
  - Detection_output.txt
- If not cleared, could impact the results of next iteration (this could be fixed in the future)

Acknowledgements

- Dr. Gan
- Alex Ramey
- Shengting Cao
- Dr. Kung
- UA ECE Department
- UA School of Library and Information Studies
- Dr. MacCall

Questions