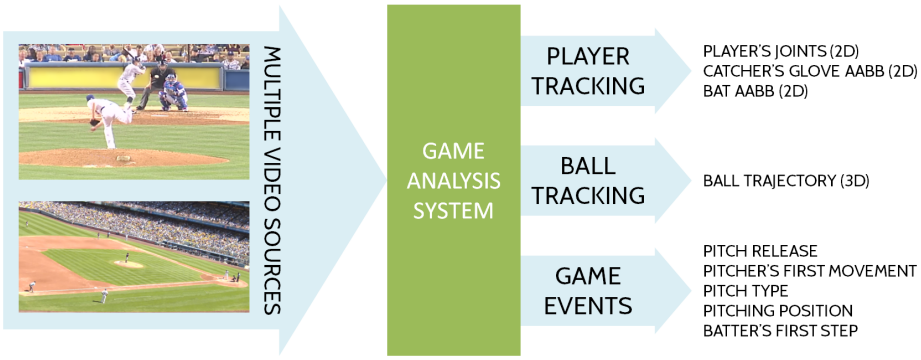
Computer Vision Methods for Baseball Analysis - Documentation

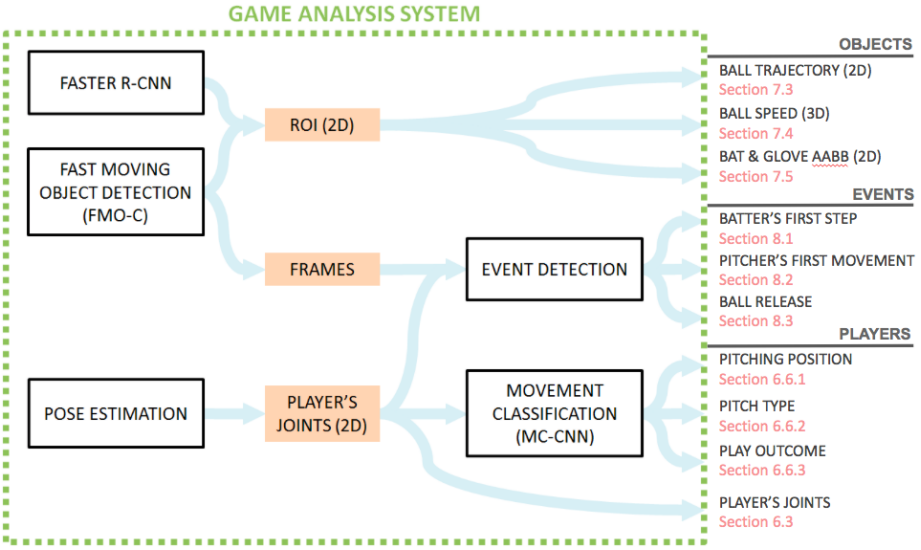
Nina Wiedemann

09.06.2018

Contents



{#fig: system_overview}



{#fig: game_analysis_system}

1 Overview (Main directory)

The proposed framework consists of three main modules: player tracking, object tracking and event detection. Different methods are combined for each module. The codes is also sorted into these three modules, where player tracking is further divided into pose estimation and movement classification.

1.1 Dependencies:

To create a temporary environment in anaconda:

```
conda env create -f environment.yml
source activate baseball_analysis
```

Otherwise see requirements.txt file - can be installed in anaconda with

```
conda install --yes --file requirements.txt
```

or

```
while read requirement; do conda install --yes requirement; done < requirements.txt
```

1.2 Train data

All data required for training is contained in the train_data folder:

1.2.1 Video data:

- The major video database consists of videos of a center-field and a side-view videos, one for each play (6 seconds length). The videos are sorted by data are stored in train_data/atl.
- For bat detection, high quality videos from YouTube are used. They are stored in train_data/high_quality_videos.
- For ball speed, shorter videos that show only the ball trajectory (<1 second)from a public database are used. There are two side view and one center field camera. Stored in: train_data/pitchfx.

1.2.2 Pose estimation output:

- Joint trajectories for pitcher and batter in center field and side view videos are saved as csv files, one for each view (cf for center-field, sv for side-view) and each player (pitcher and batter): cf_pitcher.csv, cf_batter.csv, sv_pitcher.csv and sv_batter.csv
- Joint trajectories for high quality videos can be found in the folder train_data/batter_hq_joints

1.2.3 Metadata

- Speed labels for side view videos (for ball release frame evaluation) as json file
- Manually labeled (with gradient approach) data for the batters first step: train_data/labels_first_batter_test and train_data/labels_first_batter_train

1.3 FMO-detection

Fast Moving Object Detection finds moving objects by thresholding difference images and searching for connecting components. Here, it is used to track ball and bat and to find the pitcher's first movement. The algorithm is build on the work in "The World of Fast Moving Objects"(Rozumnyi et al. 2017).

As FMO is used for both object tracking and event detection, the script and relevant functions are stored in the main folder, in fmo_detection.py. The script takes a video as input, and outputs pitcher's first movement frame index (if joint trajectories were inputted as well), ball trajectory and the motion candidates for each frame.

In order to run FMO detection on a video directly, run

fmo_detection [-h] [-min_area MIN_AREA] [video_path VIDEO_PATH]

1.3.1 Parameters

Hyperparameters can be set in the config_fmo.py file.

1.4 Neural Network related files

1.4.1 Training

For both movement classification and for event detection ANNs are trained. Since the general code for training an ANN is the same, scripts are contained here in the main folder. The model file contains different ANN models, from LSTMs to one-dimensional CNNs, while the run-files are used to start training.

- run_thread.py is used for classification tasks, where classes are represented as one-hot-vectors
- run_10fold.py is used for ten fold cross validation in the experiments
- run_events.py is used to train a network to find the frame index of an event, such as the moment of the batter's first step

Runner classes can be executed as Threads.

1.4.2 Testing

In the test.py file, any model can be loaded and data (saved as a numpy array) can be loaded and the labels are predicted. If labels are available, they can also be loaded and the accuracy is displayed. The function is mainly used in the the different modules (movement classification and event detection), but can also be executed directly with:

test.py [-h] [-labels LABELS] data_path model_path

data_path: Path to a numpy array of the data (joint trajectories) model_path: Path to a saved model of a Neural Network to restore the model -labels: Path to a numpy array with same length as the data, with labels for each data point

For example:

```
python test.py train_data/data_array.npy saved_models/pitch_type
-labels=train_data/labels_array.npy
```

1.5 Utils:

1.5.1 Utils for data processing

In utils.py functions for all kind of tasks can be found, for example calculating the accuracy per class, getting data from the csv files, shifting joint trajectories by a range of frames, etc. The functions are saved in a class Tools, so it is clearly visible if a function is saved in utils.

1.5.2 Other utils of previous versions or used for filtering:

Helper files and filtering files are saved in the folder utils_filtering. For example, filtering functions used to smooth the joint trajectories are saved here.

Conversions between csv to json files, saving videos as jumpy arrays, converting numpy arrays into a json file and similar can be found as well. In the experiments, only filtering is used though.

2 Pose estimation

In this folder, files for pose estimation, player localization and filtering of the joint trajectories are provided. For pose estimation itself, a pre-trained model from the work of (Cao et al. 2017) is used. All files in this directory that are not explained in this documentation belong to the code of (Cao et al. 2017).

2.1 Script for pose estimation

To test pose estimation (No own code!), use the script pose_estimation_script.py

Input one video and the script outputs images with plotted skeletons (without player localization, filtering etc), and a json file with the joint coordinates output, which is a list of shape #frames x #detected_persons x #joints x 2 (x and y coordinate)

Usage:

python pose_estimation_script.py [input_file] [output_folder] [-number_frames]

- input_file: path to the input video, must be a video file (.m4v, .mp4, .avi, etc.)
- output_folder: does not need to exist yet
- number_frames: if only a certain number of frames should be processed, specify the number

2.2 Process videos to joint trajectories

In my framework, the scripts read a video frame by frame and use the pose estimation network to yield the skeletons of all detected persons. Then the target person is localized and the region of interest for the next frame is calculated. After processing all frames, the output trajectories are smoothed and interpolated.

More in detail, the following steps are executed:

- Pose estimation on each frame –> list of detected people
- Localize the target player (start position required) –> joint coordinates for each frame for the target person
- swap right and left joints because sometimes in unusual positions they are suddenly swapped
- interpolate missing values
- smooth the trajectories with a lowpass filter
- save the output in a standardized json file for each video, containing a dictionary with 2D coordinates per joint per frame

Usage:

joint_trajectories.py [-h] input_dir output_did center

Pose Estimation Baseball

positional arguments:

- input_dir: folder with video files to be processed
- output_dir: folder where to store the json files with the output coordinates (does not need to exist before)
- center: specify what kind of file is used for specifying the center of the target person: either path_to_json_dictionary.json, or datPitcher, or datBatter

Examples:

```
python joint_trajectories.py demo_data/ out_demo/ datPitcher
python joint_trajectories.py hq_videos/ out_hq/ center_dics.json
```

Note: The "center" parameter might be confusing: It refers to the center of the hips of the target person in the first frame, which is required for localizing the target from all detected persons. However, I have tested pose estimation with two different kind of input videos: The database of MLBAM, containing ten thousands of videos of plays, and 30 high quality videos on the other hand. For MLBAM videos, the starting position is giving in a .dat file for each video. If these videos are used, then dependent on whether the target player is the Pitcher or the Batter, the center argument must be datPitcher or datBatter. Then for each video the belonging dat file is used. If the high quality videos are used though, no .dat files are available, so I just manually made a json file containing the starting point for each video file in a dictionary. In this case, the argument must be the file to the json path, which can be found in train_data/center_dic.

2.2.1 Files that are imported in the joint trajectories script:

- The pose estimation is done using the function handle_one(img) in the file pose_estimation_script.py
- Localization, smoothing and interpolating functions can be found in data_processing.py

2.2.2 Run for all videos:

To process all files from the Atlanta stadium (by date), use from_video_to_joints. The script iterates through all folders of the video data and processes each video. The outputs are saved as json files per video separate folders for center field and side view videos.

Finally, the json_to_csv file is used to take all json files of one folder and save them in a csv file instead (better for training models later than loading json files individually every time)

2.3 Player localization (visualization)

A notebook called player_localization demonstrates the approach for player localization, comparing the results on a video in which the approach works to one in which the target player is los.

2.4 Filtering (visualization

A notebook called visualization_pose_estimation is used to color videos compare different methods for filling in missing values and smoothing/filtering the joint trajectories.

3 Movement classification

The 2_Movement_classifictaion folder contains files to recognize actions on the field. Basically, it is the last stage of the processing pipeline from videos to motion classes. So first, a video must be processed with the files in the folder 1_Pose_estimation, such that joint trajectories for ONE player are outputted. For movement classification, these trajectories are input to a CNN called MC-CNN that can solve different classification problems.

Here, pose estimation was already run on all available videos and saved in csv files. Thus, for training and testing MC-CNN, the cf_pitcher.csv and cf_batter.csv files in the train_data folder are used. 5% are used as validation data, while a network is trained (again splitting into train and test set) on the other 95%. For the saved models, the saved indices test_indices.npy were used as test data.

3.1 Training

To train a CNN (or other networks) to classify motion, run the file classify_movement.py. It uses the run-, model-, and utils-files from the main directory to create a tensor flow graph and start training. Tasks and data can be specified as arguments.

Usage: classify_movement.py [-h] [-training TRAINING] [-label LABEL][-view VIEW] save_path

Arguments:

- save_path: indicates path to save the model if training, or path to the model for restoring if testing
- – training: if training, set True, if testing, set False (default: True)
- – label: “Pitch Type”, “Play Outcome” or “Pitching Position (P)” are possible so far
- – view: either “cf” (center field) or “sv” (side view)

Examples:

```
python classify_movement.py saved_models/pitch_type
 -label="Pitch Type" -view="cf"

python classify_movement.py saved_models/pitching_position
 -label="Pitching Position (P)" -view="sv"
```

3.1.1 Ten-fold-cross validation:

To train in 10 fold cross validation, change line 18 in the classify movement file to "from run_10fold import Runner" instead of "from run_thread import Runner"

3.1.2 Parameters:

All parameters are set in the config file.

- For other ANN architectures, change the required field
- Specify if the number of classes for the pitch type (3 superclasses) should be reduced, or if the data should be restricted to 5 players or to one pitching position
- Hyperparameters for the CNN architecture can be set (number of layers, learning rate, number of epochs etc.)

3.2 Testing

3.2.1 Validation data:

Test directly on data from the same csv file (only 95% of this file is taken for training, the other 5% serve as validation data). Make sure that the same configuration (number of players and position included, labels as super classes etc) as used in training is set in the config file.

Run

```
python classify_movement.py saved_models/pitch_type
 -label="Pitch Type" -view="cf" -training=False
```

to test a trained model on the rows of the csv corresponding to the saved test_indices.

3.2.2 Own data:

Use the test file in the main directory to input your own data as a numpy array (no labels required, but can be added for comparison). Use test file with the approriate model from saved_models in main folder with your input data.

3.2.3 Available models:

- pitch_type: All players and positions included, 10 classes, should have around 55% test accuracy
- pitching_position: All included, should have around 96% accuracy

- play_outcome: Must be used with cf_batter.csv file (batter trajectories), should have around 98% test accuracy

4 Event detection

This folder contains test and train files for event detection. Four events can be distinguished: The pitcher's first movement, ball release, the batter's lifting of his leg and the batter's first step towards first base.

Test functions for a single video for each event are in *detect_event.py*. These functions are demonstrated in a demo notebook, that visualizes the results for all events for videos in the demo_data folder.

4.1 Pitcher's first movement

Experiments are conducted on one folder of ATL videos with different configurations (artificially changing the frame rate). See the related notebook Pitcher's first movement.ipynb for further explanations. Outputs of the experiments are saved in outputs/first_move_result_dic.

4.2 Ball release frame

Three different methods to find the ball release frame are presented: Using the joint trajectories to find the moment the arm is highest above the shoulders, train a CNN to learn to recognize the pitcher's stance in images, and using FMO-C to detect the ball and calculate back to ball release.

4.2.1 Higher - Shoulders Release:

See explanations and visualizations in the related notebook HS_release.ipynb.

4.2.2 Stance recognition in images:

Training and testing can be done by running release_from_image_train.py.

Usage:

release_from_image_train.py [-h] [-training][-model_save_path]

Optional arguments:

- -training: if training, set True, if testing, set False
- -model_save_path: if training, path to save model, it testing, path to restore model

In this file, the training and testing data is directly split, since some game-dates are specified that serve as test data and some serve as train data.

4.2.3 Release from ball detection:

The ball is detected and the release frame is calculated with a rough 2D approximation of the ball speed and the distance of the ball to the pitcher. See explanations in the related notebook release_ball_detection.ipynb.

4.3 Batter's lifting of the leg:

The code is a function in the detect_event.py file. It is demonstrated in the Batter's movement notebook, where experiments are run and results are visualized.

4.4 Batter's first step

For the batter's first step, a LSTM is trained. It learns to find the frame index of the batter's first step, given the joint trajectories of the batter as input.

4.4.1 Training

Use the batter_first_move_train.py file to train the LSTM.

Usage:

batter_first_move_train.py [-h] [-training][-model_save_path][-data_path]

- – training: if training, set True, if testing, set False
- – model_save_path: if training, path to save model, it testing, path to restore model
- – data_path: path to data for training and testing

For example:

```
python batter_first_move_train.py -training="True"
-model_save_path="../saved_models/batter_first_step_new"
-data_path="../train_data/batter_runs"
```

4.4.2 Testing

Testing is done with the same script as training. In train_data/batter_runs, the data is distinguished between train and test data directly. Thus, if the argument training=False is passed, directly the correct data is processed and outputs are saved in the outputs folder in a json file called batter_first_move_test_outputs.

For example:

```
python batter_first_move_train.py -training="False"
-model_save_path="../saved_models/batter_first_step"
-data_path="../train_data/batter_runs"
```

Experiments and visualization can be found in the Batter's movement notebook.

5 Object tracking

5.1 Ball speed

Ball tracking is tested on videos from a public database, filming just the ball trajectory from pitcher to batter (<1 second).

Using the Ball speed notebook for ball detection and speed, firstly the for each videos the ball trajectory is retrieved, running the script fmo_detection.py.

I saved the ball trajectories of each camera for each one-second-video in json files, that can be found in the same folder as the data: ../train_data/pitchfx

The trajectories were then transferred to the 3D domain and speed was estimated with linear regression CODE EINFUEGEN. The outputs are saved in the folder outputs as reports (in csv format) for both side view cameras.

5.2 Bat and glove detection

The bat is detected by a combination of Faster R-CNN and FMO-C. All relevant code is in the Bat Detection notebook. The notebook uses the Faster R-CNN implementation and models from the Tensorflow Object Detection API (Huang et al. 2017). The code from the API is all found in the folder models.

As data for bat detection I use the high quality videos in the train_data folder in the main directory. The output of bat and glove detection is stored as a json file in outputs and can also be plotted on a video which is saved in the same folder.

Bibliography

Cao, Zhe, Tomas Simon, Shih-En Wei, and Yaser Sheikh. 2017. "Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields." In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, Hi, Usa, July 21-26, 2017*, 1302–10. doi:10.1109/CVPR.2017.143.

Huang, Jonathan, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, et al. 2017. "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors." In *IEEE Cvpr*.

Rozumnyi, Denys, Jan Kotera, Filip Sroubek, Lukás Novotný, and Jiri Matas. 2017. "The World of Fast Moving Objects." In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, Hi, Usa, July 21-26, 2017*, 4838–46. doi:10.1109/CVPR.2017.514.