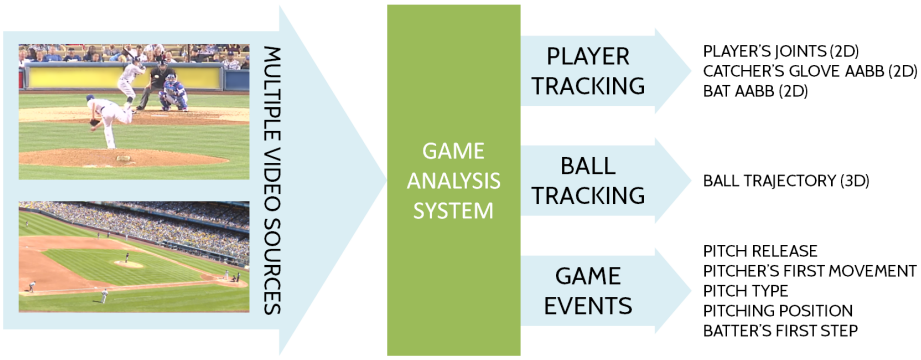
Computer Vision Methods for Baseball Analysis - Documentation

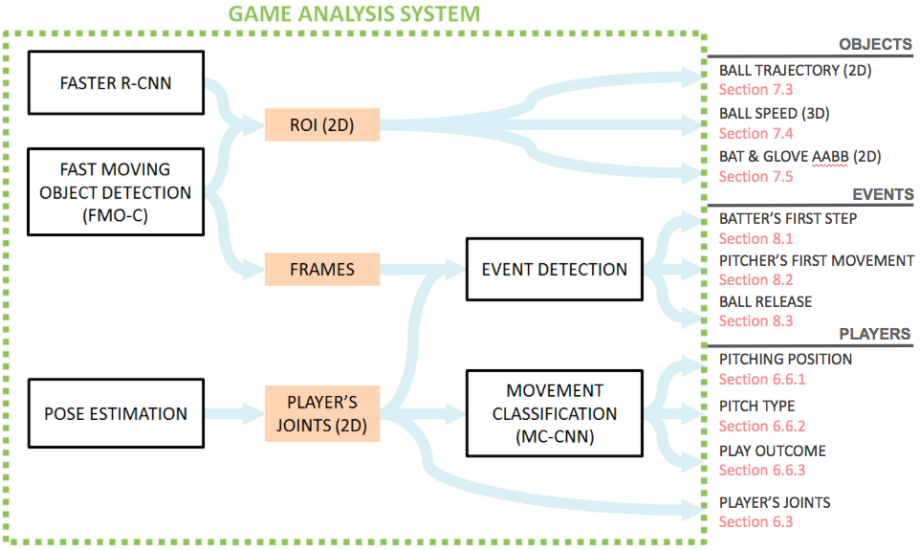
Nina Wiedemann

09.06.2018

Contents



{#fig: system_overview}



{#fig: game_analysis_system}

1 Overview

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 sorted by data are stored in a subfolder called atl.
- High quality videos for pitcher and batter which are used for bat detection are stored in train_data/high_quality_videos
- Low quality videos (side view from a public online database): 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 (cf_pitcher.csv (center field pitcher trajectories), cf_batter.csv, sv_pitcher.csv, sv_batter.csv)
- Joint trajectories for high quality videos 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 Outputs

The outputs folder contains the results of various experiments:

- ten_fold.json contains a dictionary with the test and train accuracies for different tasks of movement classification
- first_move_result_dic contains the predicted frame indices for the pitcher's first movement in different variations
- batter_first_move_test_outputs contains the outputs of testing the LSTM model on the test data for the batter's first step in train_data/labels_first_batter_test

1.4 FMO-detection:

Fast Moving Object 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 Fast Moving Object Detection is used for both object tracking and event detection, the script is in the main folder. In fmo_detection.py the script and relevant functions can be found. It contains a script and function taking a video as input, outputting 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]

1.4.1 Parameters:

Hyperparameters can be set in the config_fmo file.

1.5 Neural Network related files

1.5.1 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 split the data and 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.

1.5.2 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 folders, but can also be executed directly with:

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

1.6 Utils:

1.6.1 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.

1.6.2 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

For all versions, you first need to download the model: `cd model; sh get_model.sh`

Simple test version of pose estimation: input one video and outputs images with plotted skeletons: here - without player localization, filtering etc

usage:

python pose_estimation_script.py path_to_input_video output_directory - number_of_frames

output_directory: does not need to exist yet path_to_input_video: must be a video file (.m4v, .mp4, .avi, etc.) number_of_frames: if only a some frames should be processed

2.0.1 Process videos to joint trajectories

In this folder, most files just belong to the original Pose estimation network. In our framework, the scripts read a video frame by frame and use the pose estimation network in to output 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.

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 * filter 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

Two different files for either testing a few videos or all available videos:

- To process all files from the Atlanta stadium (by date), use folder script
- For some example files in one folder, use file script

usage: joint_trajectories.py [-h] DIR DIR center

Pose Estimation Baseball

positional arguments: DIR folder with video files to be processed DIR folder where to store the json files with the output coordinates 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

optional arguments: -h, –help show this help message and exit

optional arguments: -h, –help show this help message and exit

Examples:

```
python joint_trajectories.py demo_data/ out_demo/ datPitcher
python joint_trajectories.py high_quality_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, we 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 we 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.

2.0.2 Other files in this folder:

- 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
- A notebook is used to color videos compare different methods for filling in missing values and smoothing/filtering the joint trajectories.
- json_to_csv is used to take all output json files of one folder and save them in a csv file instead (better for training models later than loading sons individually every time)

3 Movement classification

This folder is the last stage of the processing pipeline from videos to motion classes. So first with the files in the folder 1_Pose_estimation the video must be processed, 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.

For training and testing, 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%.

3.1 Training

- Data for training is available on Google Drive called for example cf_pitcher.csv, specifying viewpoint of the camera and target player. Here these csv files can be downloaded.
- Download the files and store them in the train_data folder
- Then the classify_movement file here can be be executed. It uses this file for building a tensor flow graph and training. Tasks can be specified as arguments. For usage (arguments) type

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

positional arguments: save_path indicates path to save the model if training, or path to the model for restoring if testing

optional arguments: -h, –help show this help message and exit -training TRAINING if training, set True, if testing, set False -label LABEL Pitch Type, Play Outcome or Pitching Position (P) possible so far -view VIEW either cf (center field) or sv (side view)

Examples:

```
python classify_movement.py models/pitch_type_model -label="Pitch Type" -view="cf"
python classify_movement.py models/position_model -label="Pitching Position (P)" -view="sv"
```

This will train the network for the number of epochs specified in the file.

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.

- 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, 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.

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 approriate model from 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
- 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 just a single video for each event are in detect_event. 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

See the related notebook. 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 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 TRAINING] [-model_save_path MODEL_SAVE_PATH]

Train ANN to find batter first step

optional arguments: -h, –help show this help message and exit -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 by specifying the dates of games from which the videos are selected.

4.2.3 Release from ball detection:

See explanations in the related notebook Release frame from ball detection.ipynb

4.3 Batter's lifting of the leg:

The code is a simple function in the detect_eventfile. It is used 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 TRAINING] [-model_save_path MODEL_SAVE_PATH][-data_path DATA_PATH] optional arguments: -h, –help show this help message and exit -training TRAINING if training, set True, if testing, set False -model_save_path MODEL_SAVE_PATH if training, path to save model, it testing, path to restore model -data_path DATA_PATH path to data for training and testing

Example:

```
python batter_first_move_train.py -model_save_path="../saved_models/batter_first_step_new" -
```

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.

example:

```
python batter_first_move_train.py -model_save_path="../saved_models/batter_first_step" -trai
```

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

Object tracking:

4.5 Ball speed:

Ball tracking can be tested best on 20 second videos from a public database, filming just the ball trajectory from pitcher to batter.

Using the notebook for Ball detection and speed, firstly the for each videos ball trajectory is calculated (from the script fmo_detection.py in the main directory).

Running this on the videos from the database above, I saved the ball trajectories of each camera in json files, that can be found here

The trajectories were than 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.

4.6 Bat and glove detection

The bat is found as 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 in a video which is saved in the same folder.

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.