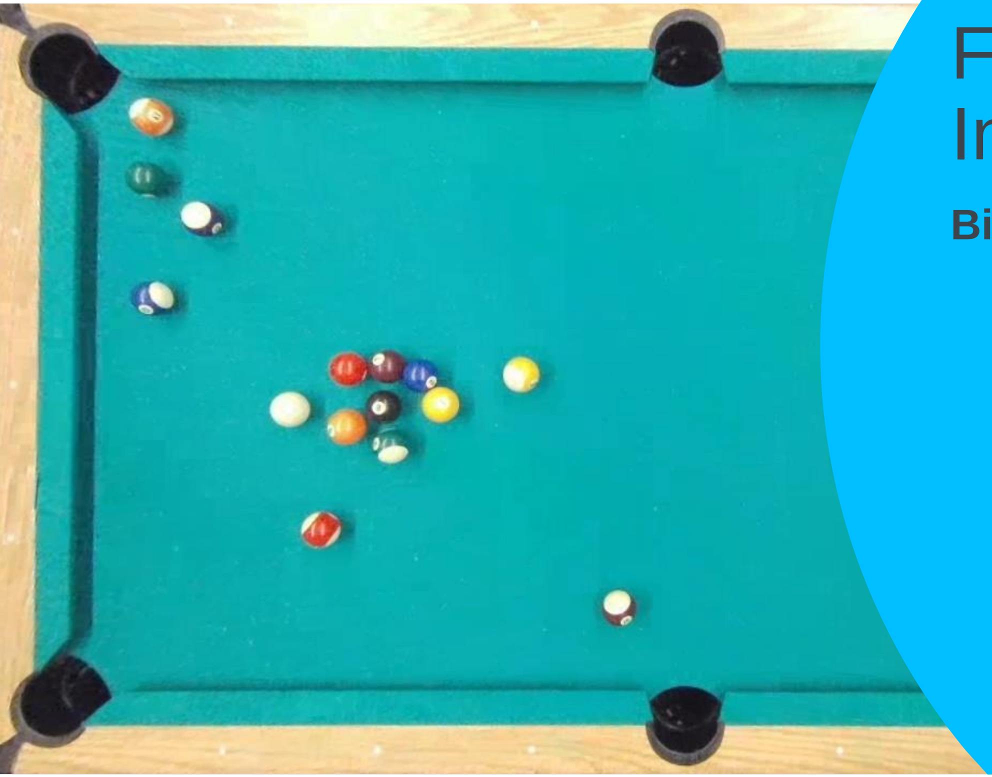
University of Stuttgart
IPVS

Analytic Computing



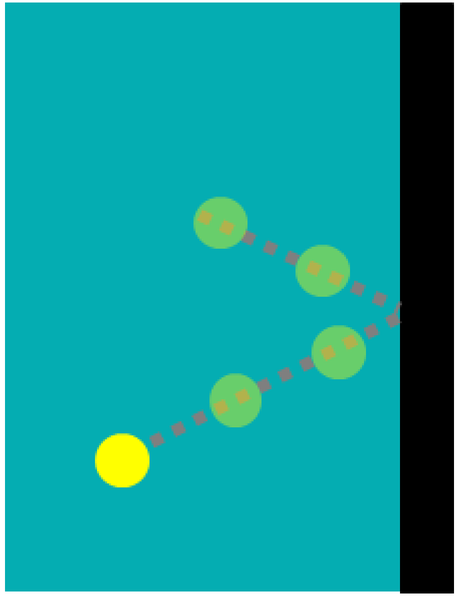
Fachpraktikum Artificial Intelligence

Billiard Motion Prediction

Motivation & Problem Statement

Motivation

- Use Artificial Intelligence by learning from data
- Implement whole data mining cycle
- Application: Motion Prediction of multiple objects



Problem Statement

- Billiard balls motion prediction
- Input: Short video starting with strike
- Output: Ball positions and velocities over time
- Regression Goal: Minimize difference between predicted and ground truth trajectory

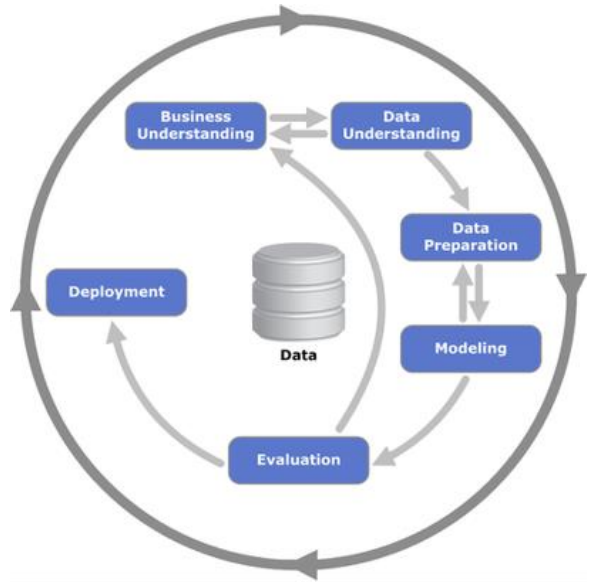
Methodology & Setup

Methodology

- CRISP-DM Model
- Requirements analyisis

Setup

- Tracking of tasks with Trello
- Communication via Discord
- Scrum with weekly sprint review
- Git as collaborative version control system
- Modular software architecture
- Formal documentation with UML



https://statistik-dresden.de/archives/1128

Table of contents

- Data Preparation
- Modeling
- Evaluation
- Conclusion

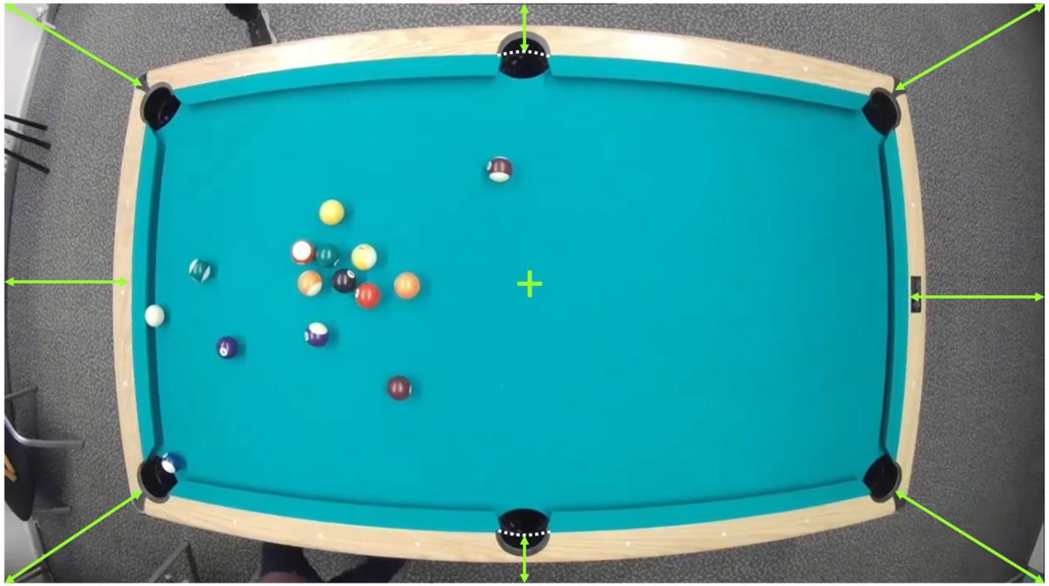
Data Preparation

Overview

1. Video Recording
2. Clips Generation
3. Camera Calibration
4. Distortion Correction
5. Data Augmentation
6. Alignment
7. Preprocessing
8. Dataset Cleaning

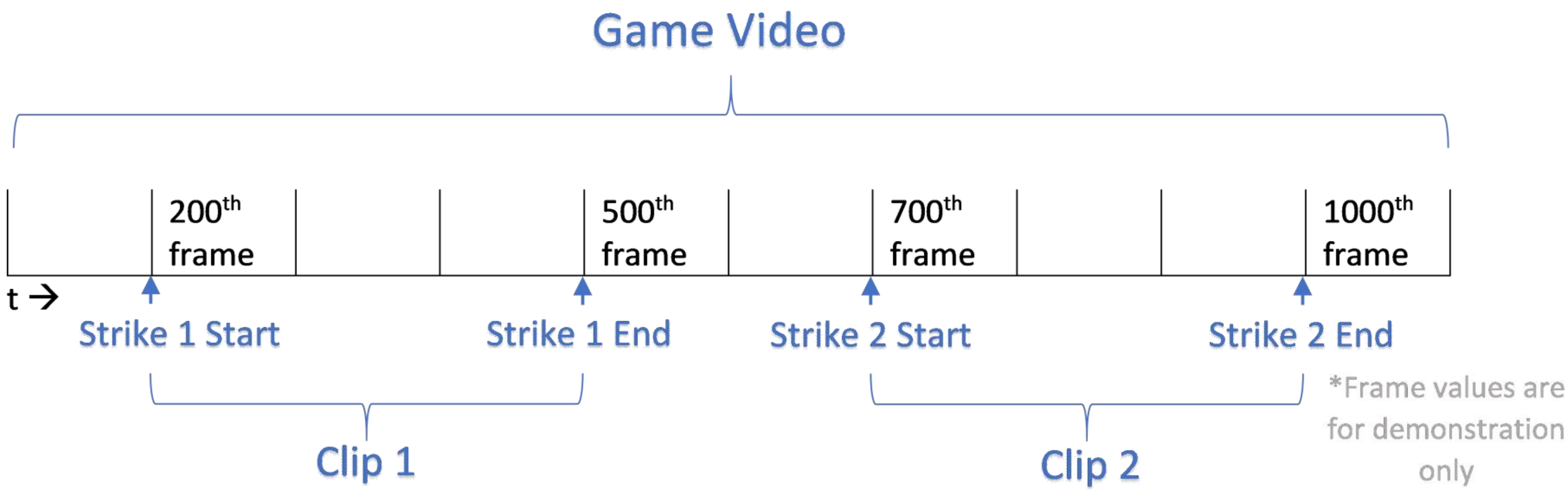
Video Recording

- No occlusions
- Uniform lighting
- Table centered
  - borders equidistant from image edges
  - corners equidistant from image corners



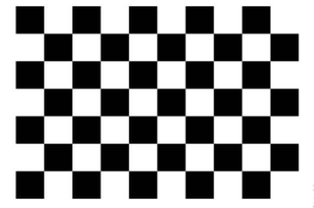
Clips Generation

- Timestamps noted manually
- **Automated video clip generation** using **python script** that accepts timestamps

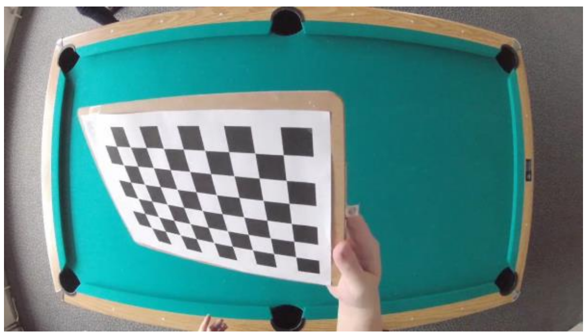


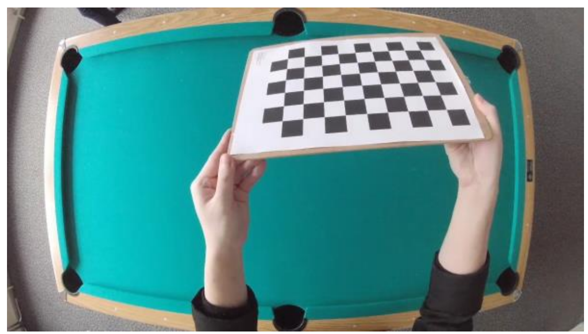
Camera Calibration

- Moving chessboard pattern in different angles
- Snapshots (10-20) as source
- Finding the chessboard pattern positions
- Calculating intrinsic of camera: camera matrix & distortion matrix



Chessboard pattern
(via opencv.org)



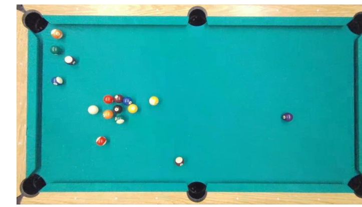


- Chessboard patterns images (cut from video)
- Found corners

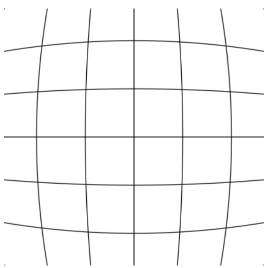
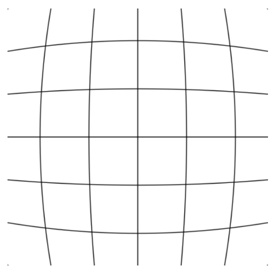
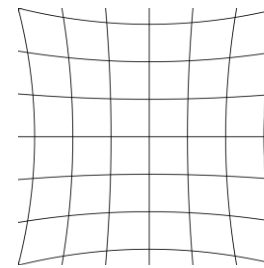
Distortion Correction

- Using these two matrices to un-distort the source images



- original
- Based on 7 images
- Based on 14 images
- Based on 21 images

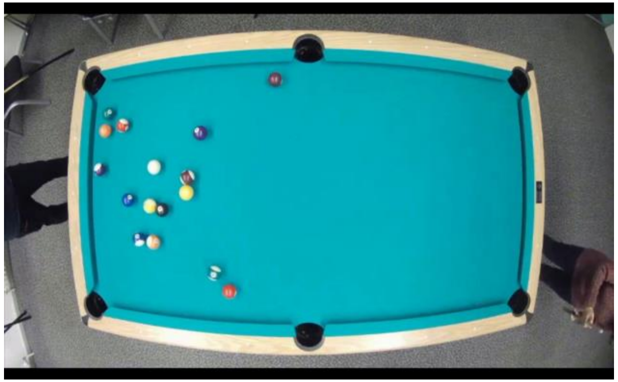




→ Increasing number of images improves result

Data Augmentation

- original

- vertical

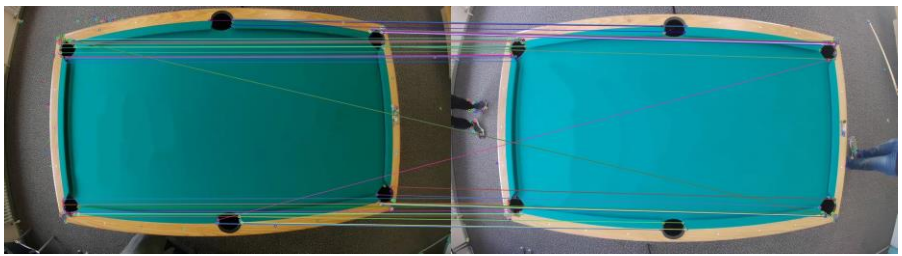


- horizontal

- vertical & horizontal

Alignment

- Different size
- Different angle



- Test video
- Training video

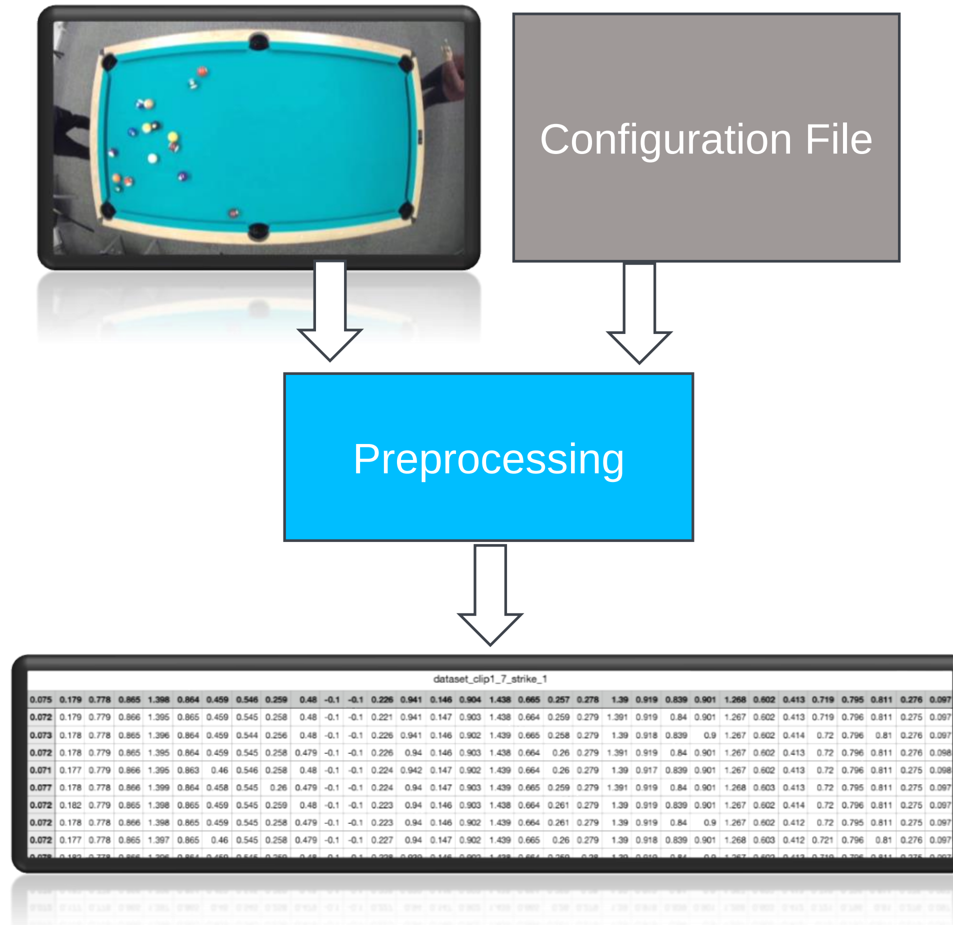
- Feature matching

- Alignment result
- Correction result

- Calculating the homograph matrix
- Applying it on the source image

Preprocessing – Overview

- Inputs : Clips, Configuration File
- Image Processing
  - Image preprocessing & transformation
  - Circle detection + Color classification
- Bidirectional temporal tracking
- Outputs : CSV Files



Preprocessing – Inputs

Configuration File Parsing

- YAML file containing clip names as entries
- Each entry is marked with at-least one:
  - Start frame
    - Frame where the strike/s occurs
  - Valid flag
    - True : Balls are tracked successfully during the corresponding strike duration, hence csv file can be generated
    - False : Unsuccessful tracking
- Parser extracts the clip name and its metadata
- Feeds it to image processing pipeline

```yaml
randomly_clip_7.mp4:
    1:
        start_frame: 13
        valid: False
    2:
        start_frame: 200
        valid: True
```

Preprocessing – Pipeline

Render frames

- Read video file stream
- Capture frame in target frame rate

Preprocess image

- Compensate lense distortion
- Transform to HSV color space

Transformation

- Extract corners of billiard table
- Calculate affine transformation matrix

Segmentation

- Find circular shapes
- Calculate centroids

Classification

- Classify color
- Classify pattern

Extract positions

- Project into table coordinate system
- Calculate distance to table boundaries

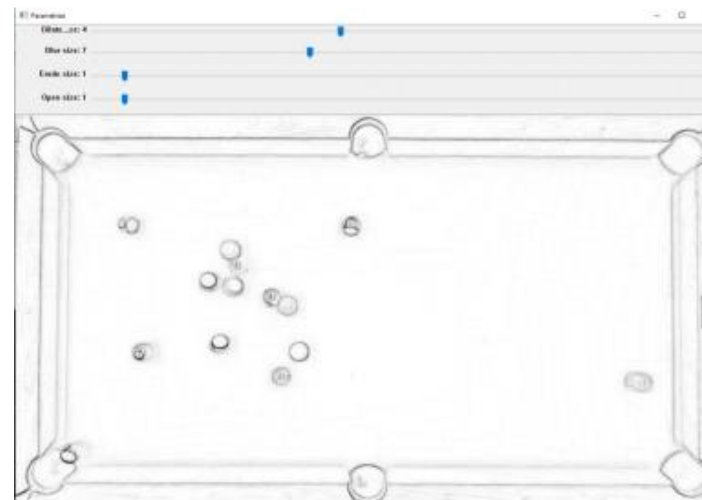
Preprocessing – Image Processing

Preprocess Image :

- Camera calibration and distortion correction
- Parameterization
  - Shadow removal
  - Morphological transformations
    - Dilation, Blurring, Erosion, Opening
    - Real time trackbar for adjustment of values
- Gamma Correction
  - Adjustment of contrast in different lightning condition

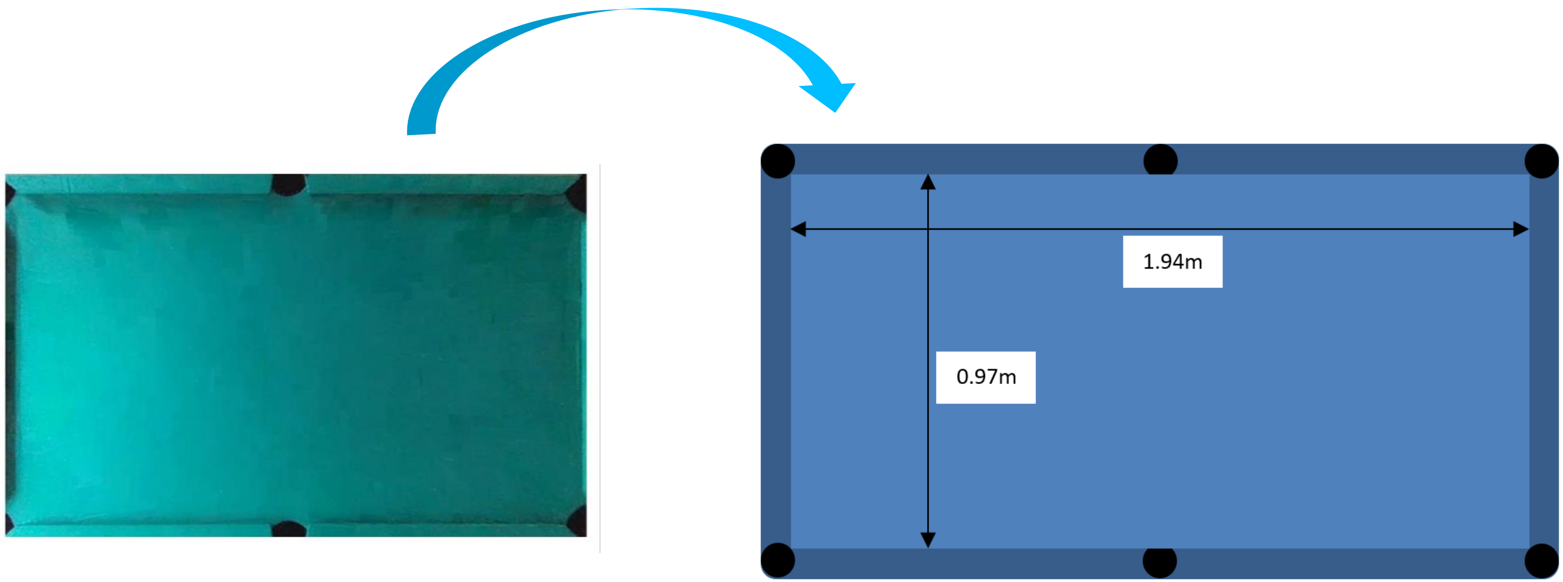
```
##camera matrix
cammtx = np.array([[661.36745612, 0, 626.09553138],[0, 663.35892418, 354.8585088 ],[0, 0, 1]])

##distortion coefficient
distco = np.array([-3.52903264e-01, 1.65315869e-01, -3.39131620e-04, -1.64084879e-06,-4.30240406e-02])
```



Preprocessing – Image Processing

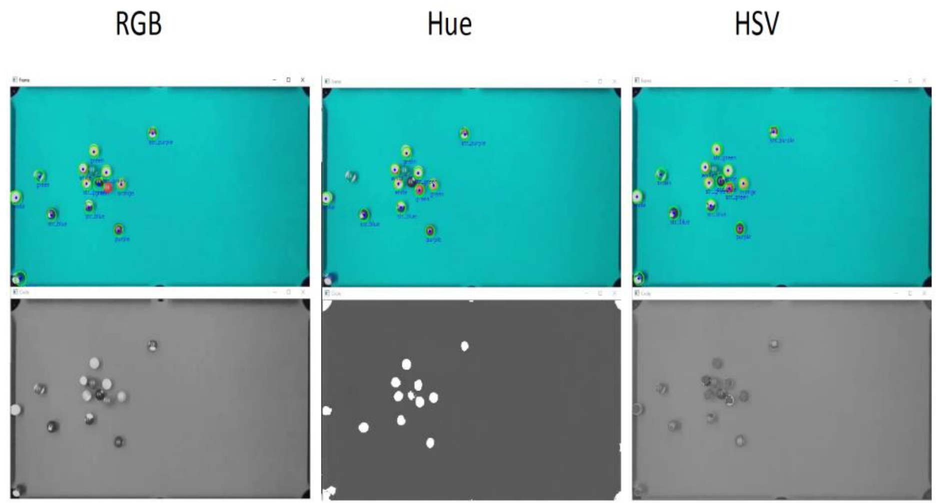
- Find affine transformation matrix to table dimension
- Project from image coordinate system to table coordinate system



Preprocessing – Image Processing

Circle detection :

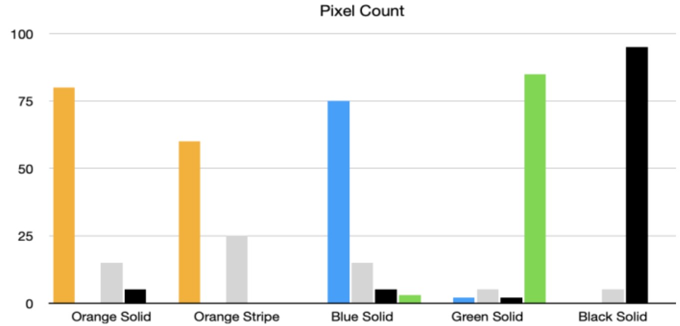
- Three strategies :
  1. Red-Green-Blue (RGB)
  2. Hue
  3. Hue-Saturation-Value (HSV)
  - Combine to get noise free frame
- Apply 3*3 kernel for blurring
- Hough Circle transformation
  - Parameter tuning
- Detect circles
  - Return centroids and radii



Preprocessing – Image Processing

Color classification :

- Mark detected circles as region of interests (ROI)
- For each ROI in a frame :
  1. Determine pixels per color using defined Hue, Saturation, Value (HSV) color ranges
  2. Find dominant color and compute color-to-white ratio
  3. Use (2) to classify ball as ball-type
- Store ball positions with corresponding ball-type for each frame in a sequence



Orange	Blue	White	Black	Green	Dominant color	Color to White ratio	Ball
80	0	15	5	0	Orange	5	**Orange Solid**
60	0	25	0	0	Orange	2	**Orange Stripe**
0	75	15	5	3	Blue	5	**Blue Solid**
0	2	5	2	85	Green	17	**Green Solid**
0	0	5	95	0	Black	19	**Black Solid**

Preprocessing – Bidirectional Temporal Tracking

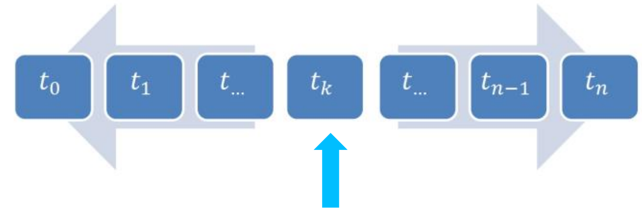
Need :

- To avoid color flickering due to non-robustness of color classification in some of the frames

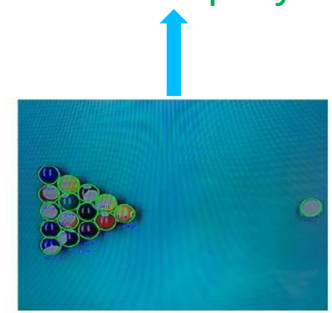
Steps :

1. Search across input sequence for best detection frame t_k
2. Initiate backward tracking
   - Store sequence t_0 to t_{k-1}
3. Initiate forward tracking
   - Store sequence t_k to t_n
4. Combine both sequences to form entire strike sequence

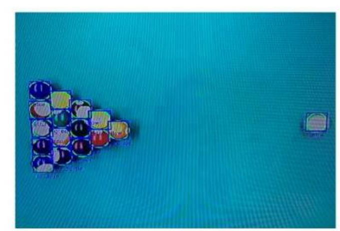
Tracker used : Channel and Spatial Reliability Tracking (CSRT)



All detected balls are uniquely color-classified



Perfectly detected frame



Tracking triggered

Tracking continues in subsequent frames

Dataset Cleaning

Problems with the generated dataset :

Hand occlusions (resulting in false positives)

- Workaround : Avoid hand occlusions during data collection

Lost trackers (fast movement of balls)

- Workaround : Data collected at a higher frame rate

No white ball (White ball misclassified as yellow)

- Workaround : Python script which eliminates data with absent white ball

Balls disappearing & reappearing (coming out of holes)

- Workaround : Python script which eliminates such data

No movement till 1 sec (30 to 60 frames)

- Workaround : Python script which eliminates initial rows with no movement

Modeling

Overview

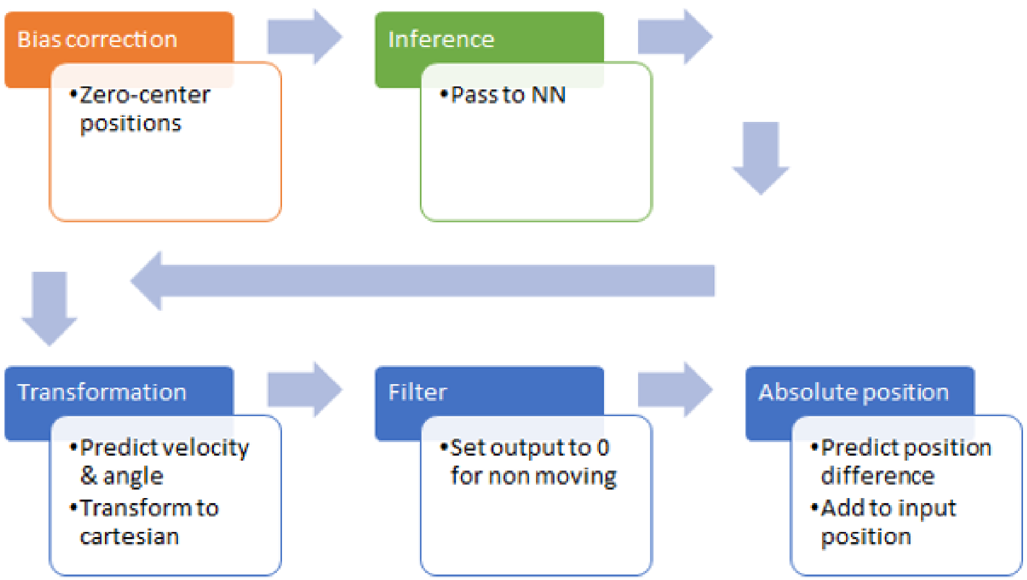
- BaseNet serves as template
- Baseline: Naive Model
- Baseline: Linear Model
- LSTM Recurrent Neural Network Model

Model Inputs & Outputs

- Inputs:
  - Position [5x32]: 5 timesteps, absolute position in x and y for 16 balls
- Outputs:
  - Position [1x32]: 1 timestep, difference in x and y for 16 balls
  - Is_Moving [1x16]: 1 timestep, moving probability for 16 balls
- Prediction "in the loop" → Output used as new input

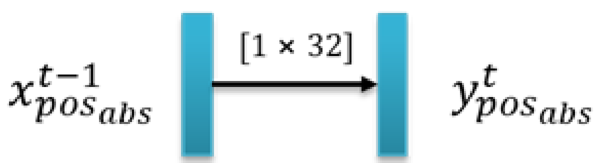
BaseNet

- Template for NN implementation → shared across models
- Realize preparation of input
- Realize transformation of output



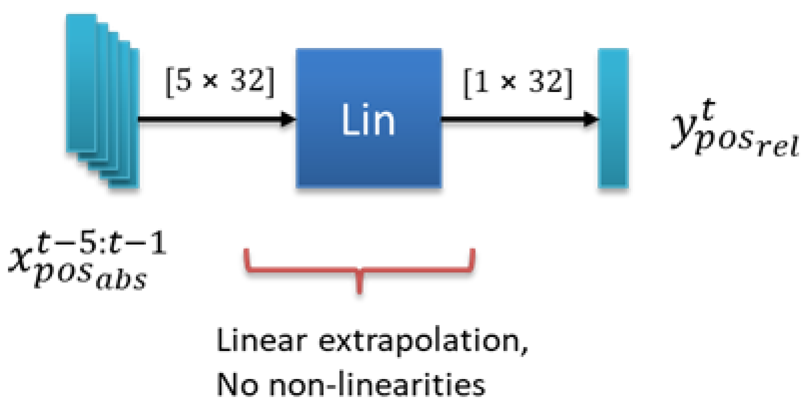
Baseline: Naive Model

- In billiard most balls are steady during strike
- Constant position is good prior
- Naive model gives input as output
- Used as reference to quantify



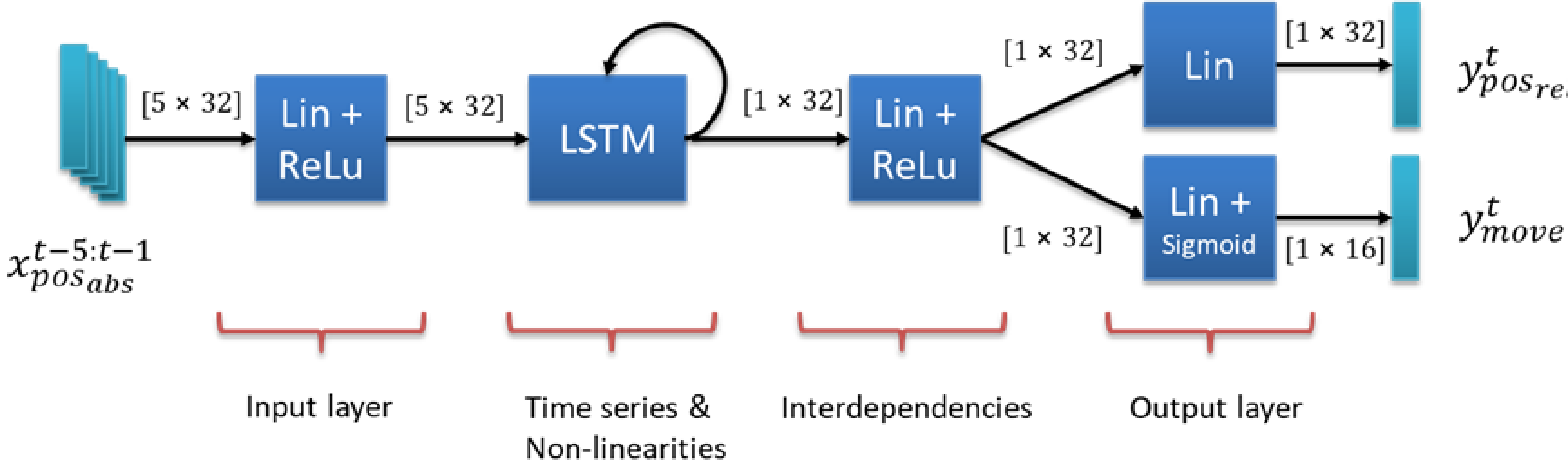
Baseline: Linear Model

- Simple linear layer without activation function
- Learns extrapolation of position
- Underestimates position difference in linear motion case
- Disregards non-linearities (collisions, boundaries)



LSTM Model

- Recurrent cell to capture non-linearities across time series
- Linear layers before and after LSTM to prepare and inter-combine features
- No activation function applied to position output to learn linear mapping
- Output includes is_moving per ball as probabilistic estimate



Loss Function

- Evaluation metric: Avg. Euclidean position error
  - Position: RMSE
- Training loss: Joint loss accumulated over N training steps and K balls
  - Position: Mean Squared Error
  - Moving: Binary Cross Entropy
  - Physical constraints: Potential map

$$loss_{train} = \sum_{n \in steps}^{N} \sum_{i \in balls}^{K} loss_{pos_{n,i}} + loss_{move_{n,i}} + loss_{physics_{n,i}}$$

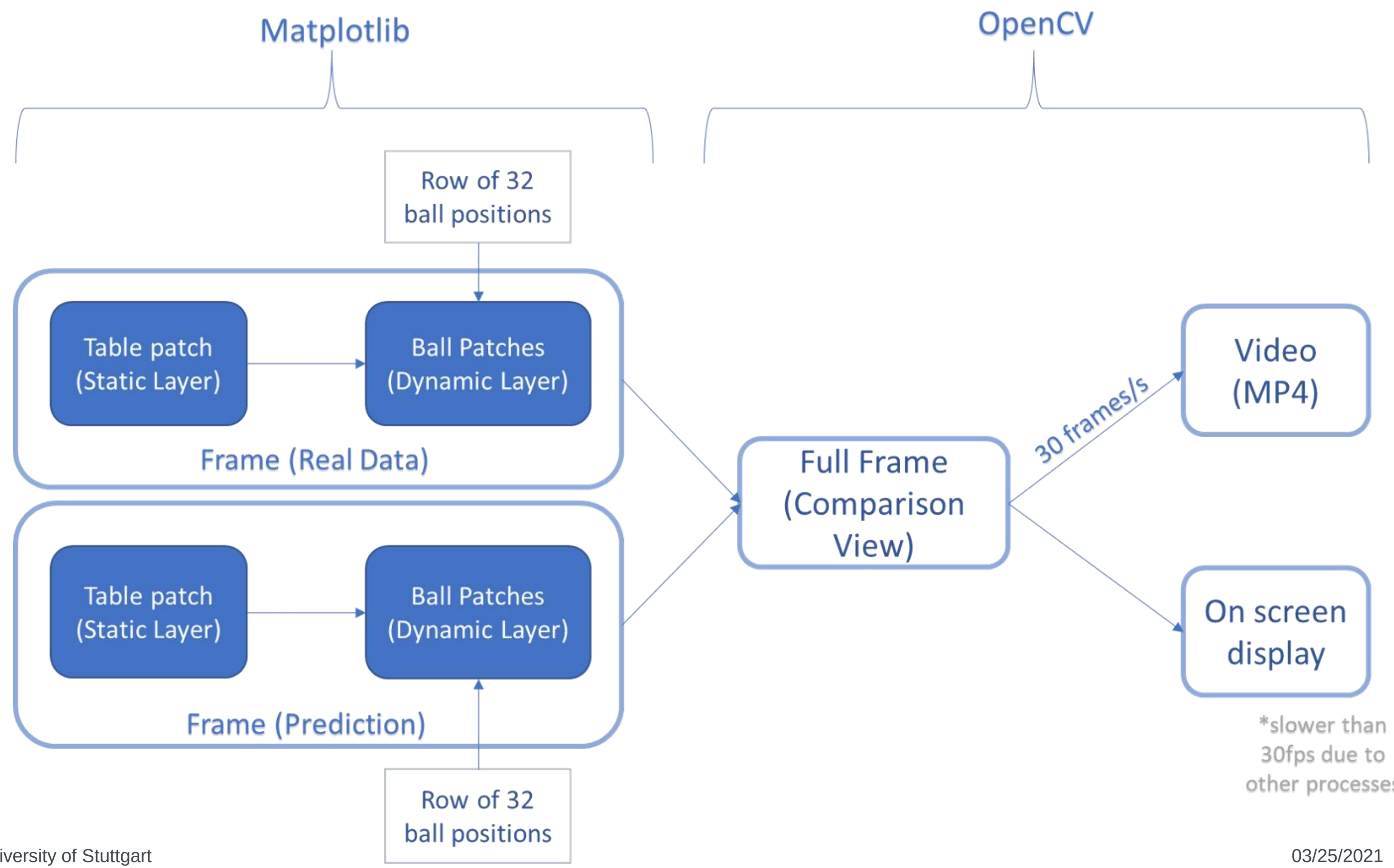
$$= \sum_{n \in steps}^{N} \sum_{i \in balls}^{K} MSE\left(pos_{d_{n,i}}, pos_{p_{n,i}}\right) + BCE\left(mov_{d_{n,i}}, mov_{p_{n,i}}\right) + Potential\left(pos_{p_{n,i}}\right)$$

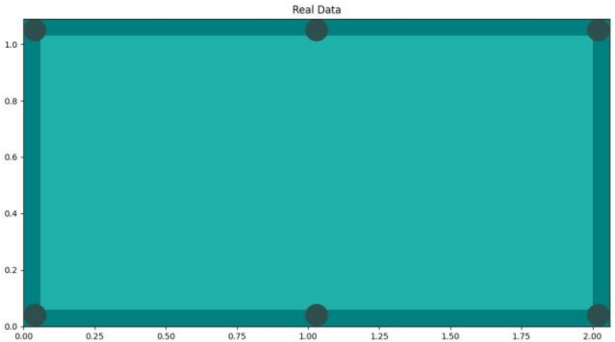
Evaluation

Overview

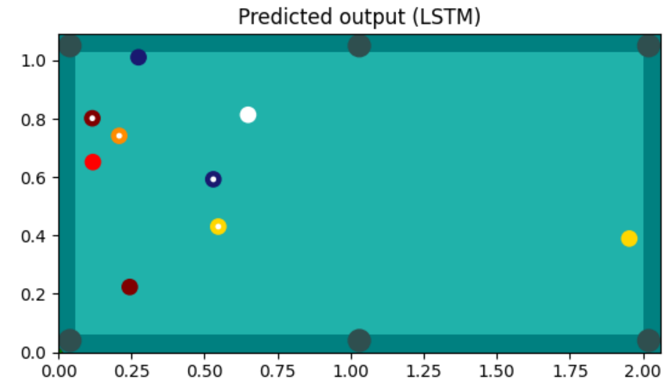
- Model performance is evaluated:
  - Quantitatively: Compare evaluation metric to baselines
  - Qualitatively: Subjective plausibility of predicted videos
- Predicted videos instrumental to figure out current state and weaknesses of the model
  - Rendering toolbox developed to compare model to ground truth
  - Easily visible:
    - Linear movement
    - Collisions with boundaries/balls
    - Stationary balls
  - Prediction flaws in video guide adjustments of training

Rendering

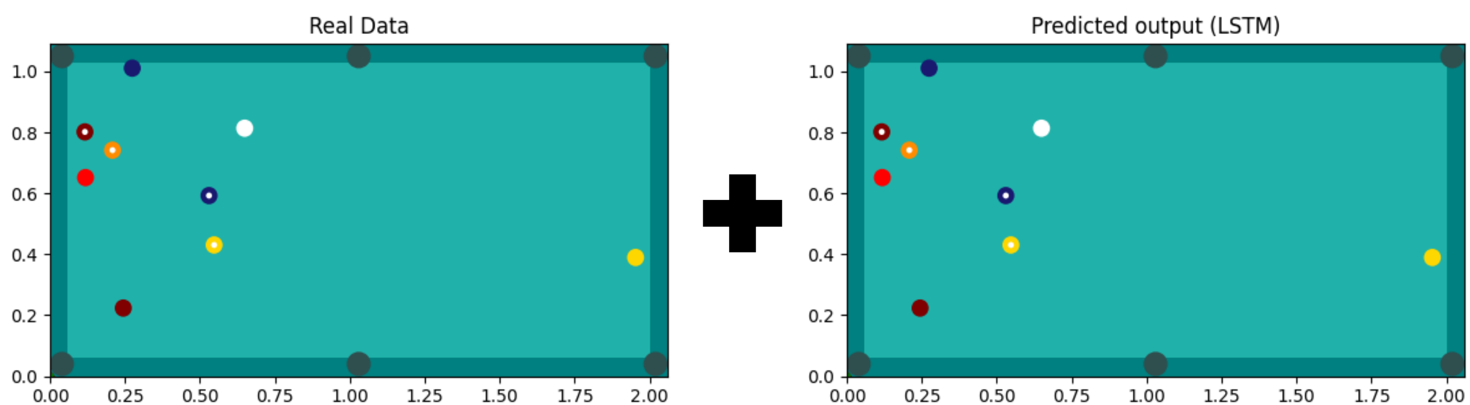

Matplotlib
OpenCV
Row of 32
ball positions
Table patch
(Static Layer)
Ball Patches
(Dynamic Layer)
Frame (Real Data)
Table patch
(Static Layer)
Ball Patches
(Dynamic Layer)
Frame (Prediction)
Row of 32
ball positions
Full Frame
(Comparison
View)
30 frames/s
Video
(MP4)
On screen
display
*slower than
30fps due to
other processes



Static Layer



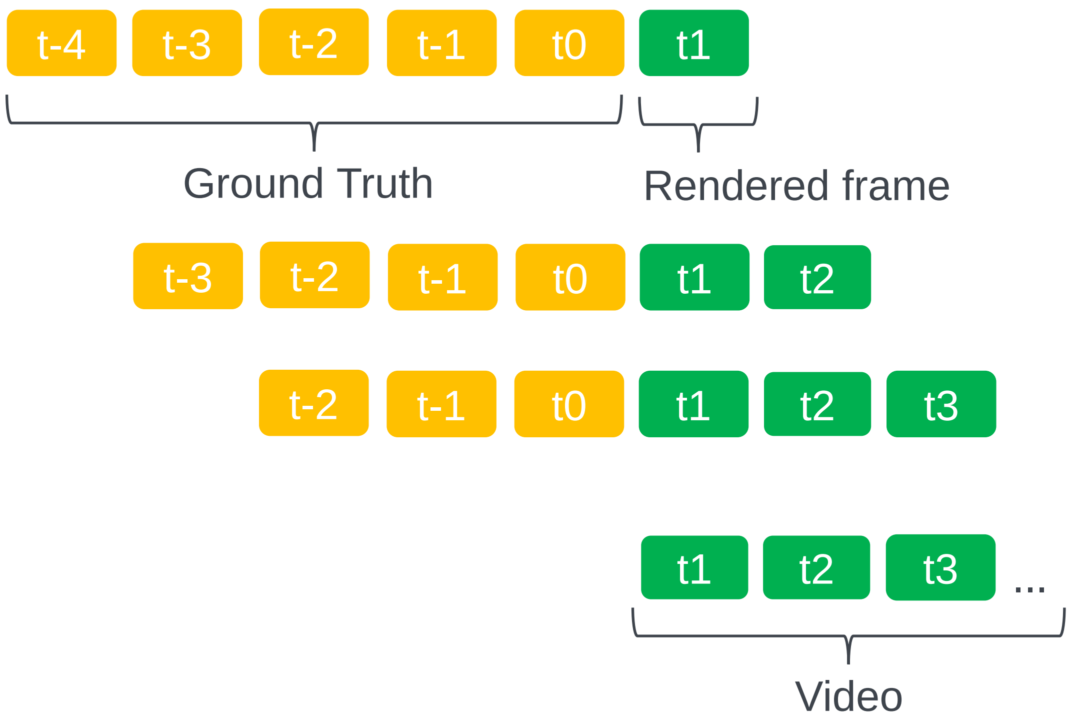
Dynamic Layer



Comparison View

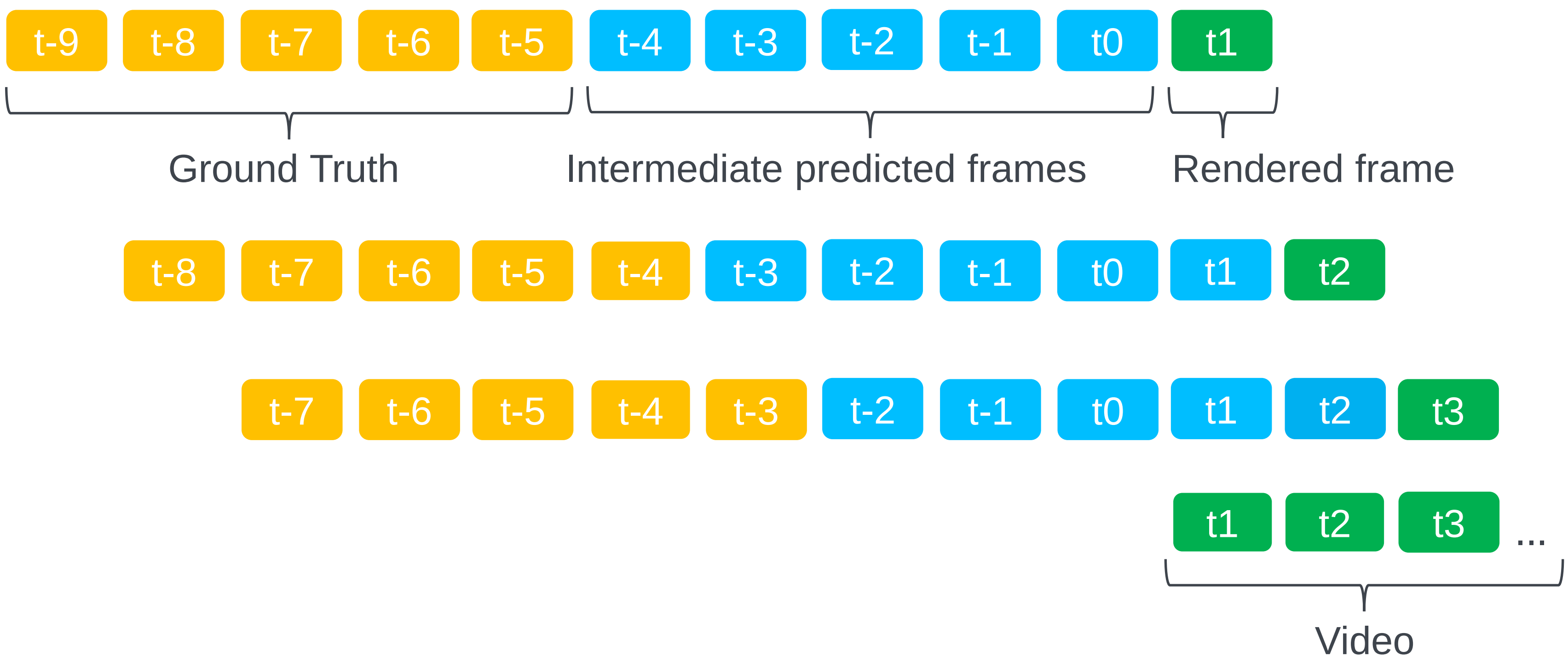
Multi-step prediction

- Two prediction modes for evaluation:
  - "In-the-loop" prediction of entire strike after initializing with the first 5 frames:



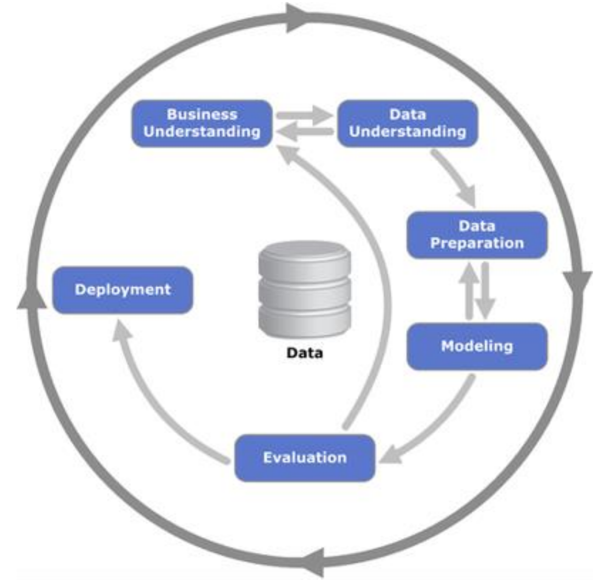
Multi-step prediction

- Two prediction modes for evaluation:
  - N-step prediction with ground truth as initialization for each predicted frame:



Modeling-Validation Cycle

- Multiple cycles in CRISP-DM model:
  - Initially not enough data
  - Modeling of non-existing balls with large position
    → set very small position
  - Inconsistencies in data found
    → clean data with scripts
  - Insufficient performance of tracker
    → use higher recording framerate
  - Model instable
    → predict relative instead of absolute positions
  - Disregards non-linearities
    → extend loss function with physical regularization



https://statistik-dresden.de/archives/1128

Comparison to Baseline

- Baseline naive→ steady state, actually good
- Baseline linear → learns linear motion
- LSTM recurrent → learns non-linear collisions

- No detailed quantitative analysis available yet due to:
  - Delayed/restricted data collection (COVID)
  - Complex data preparation
  - Build generic model pipeline
  - Limited dataset size

  → We created starting point for future work

Conclusion

Conclusion

Lessons learned

- Pandemic adds difficulty (delayed data collection, pure virtual collaboration)
- Data creation and preparation takes time
- Lack of data is limitation for sophisticated NN architectures

Future work

- Incorporate more physical constraints (e.g., conservation of momentum)
- Further DL architectures: Transformer, Graph Neural Network

Thank you!

Demo

Backup

Demo

- Show data preparation pipeline
  - Code walkthrough
  - Recorded videos
  - Preprocessing -> Ashish to prepare

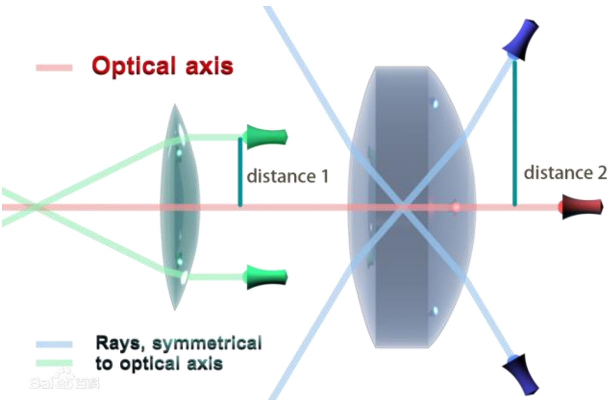
- Show Rendering
- Show video of early model
- Show LSTM with nonlinearity
- Show constant baseline
- Show Linear baseline

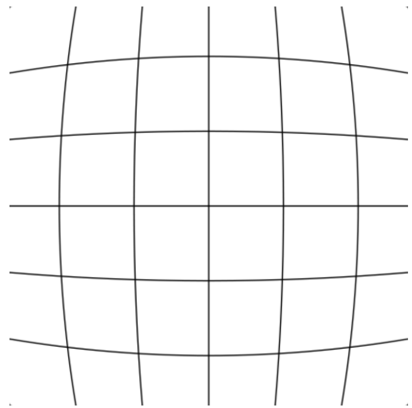
Data Preparation - Video Recording

- **Camera Device:** GoXtreme Black Hawk 4k+
- **Software:** OBS Studio
- **Table measurements:** 194cm x 97cm
- **Ball diameter:** 5.7 cm
- Several games were played with 600+ strikes
- **Camera orientation:** Points that were considered while video recording:
  - Table should be free of occlusions
  - Uniform lighting conditions all over the table
  - Table placed at the center of the field of camera vision
  - All 4 table borders should be equidistant from 4 edges of the camera field of vision.
- **Frame rate:** Initially videos were recorded at 30fps, which wasn’t suitable for preprocessing so later videos were recorded in 60 fps
- **Video Resolution:** 1280 x 720

Data Preparation – Lens Distortion

- Distortion is one of the 5 basic optical aberrations.
- Distortion ∝ d^3
- The camera we used causes Barrel Distortion.





- Optical axis
- Barrel distortion(source:wiki)

dataset_clip1_7_strike_1

0.075	0.179	0.778	0.865	1.398	0.864	0.459	0.546	0.259	0.48	-0.1	-0.1	0.226	0.941	0.146	0.904	1.438	0.665	0.257	0.278	1.39	0.919	0.839	0.901	1.268	0.602	0.413	0.719	0.795	0.811	0.276	0.097
0.072	0.179	0.779	0.866	1.395	0.865	0.459	0.545	0.258	0.48	-0.1	-0.1	0.221	0.941	0.147	0.903	1.438	0.664	0.259	0.279	1.391	0.919	0.84	0.901	1.267	0.602	0.413	0.719	0.796	0.811	0.275	0.097
0.073	0.178	0.778	0.865	1.396	0.864	0.459	0.544	0.256	0.48	-0.1	-0.1	0.226	0.941	0.146	0.902	1.439	0.665	0.258	0.279	1.39	0.918	0.839	0.9	1.267	0.602	0.414	0.72	0.796	0.81	0.276	0.097
0.072	0.178	0.779	0.865	1.395	0.864	0.459	0.545	0.258	0.479	-0.1	-0.1	0.226	0.94	0.146	0.903	1.438	0.664	0.26	0.279	1.391	0.919	0.84	0.901	1.267	0.602	0.413	0.72	0.796	0.811	0.276	0.098
0.071	0.177	0.779	0.866	1.395	0.863	0.46	0.546	0.258	0.48	-0.1	-0.1	0.224	0.942	0.147	0.902	1.439	0.664	0.26	0.279	1.39	0.917	0.839	0.901	1.267	0.602	0.413	0.72	0.796	0.811	0.275	0.098
0.077	0.178	0.778	0.866	1.399	0.864	0.458	0.545	0.26	0.479	-0.1	-0.1	0.224	0.94	0.147	0.903	1.439	0.665	0.259	0.279	1.391	0.919	0.84	0.901	1.268	0.603	0.413	0.72	0.795	0.811	0.275	0.097
0.072	0.182	0.779	0.865	1.398	0.865	0.459	0.545	0.259	0.48	-0.1	-0.1	0.223	0.94	0.146	0.903	1.438	0.664	0.261	0.279	1.39	0.919	0.839	0.901	1.267	0.602	0.414	0.72	0.796	0.811	0.275	0.097
0.072	0.178	0.778	0.866	1.398	0.865	0.459	0.545	0.258	0.479	-0.1	-0.1	0.223	0.94	0.146	0.902	1.439	0.664	0.261	0.279	1.39	0.919	0.84	0.9	1.267	0.602	0.412	0.72	0.795	0.811	0.275	0.097
0.072	0.177	0.778	0.865	1.397	0.865	0.46	0.545	0.258	0.479	-0.1	-0.1	0.227	0.94	0.147	0.902	1.439	0.665	0.26	0.279	1.39	0.918	0.839	0.901	1.268	0.603	0.412	0.721	0.796	0.81	0.276	0.097
0.078	0.182	0.778	0.866	1.396	0.864	0.459	0.545	0.259	0.48	-0.1	-0.1	0.228	0.939	0.146	0.902	1.438	0.664	0.259	0.28	1.39	0.919	0.84	0.9	1.267	0.602	0.413	0.719	0.796	0.811	0.275	0.097

CSV Generation

- Traverse through entire strike sequence obtained from tracking
- Each csv row corresponds to a point/frame in the strike sequence
- Row consists of 16*2 coordinates of balls ordered in a predefined way
- Mirror/Augment data (see next slides)

Modeling

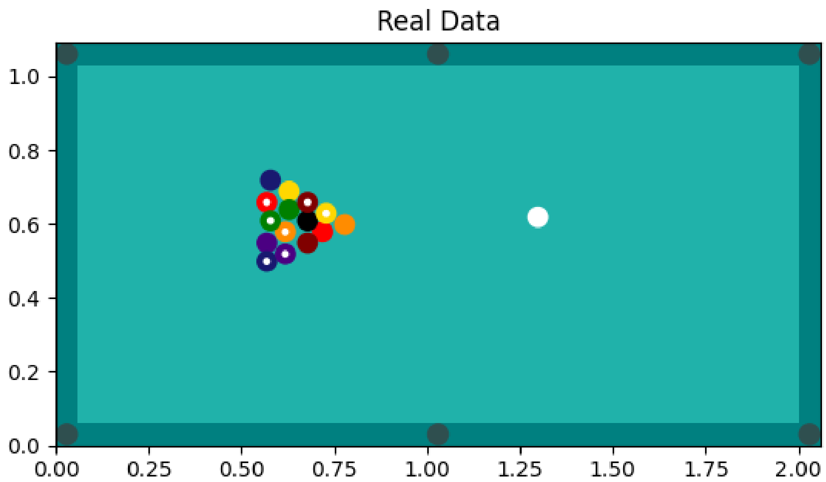
Synthetic Dataset

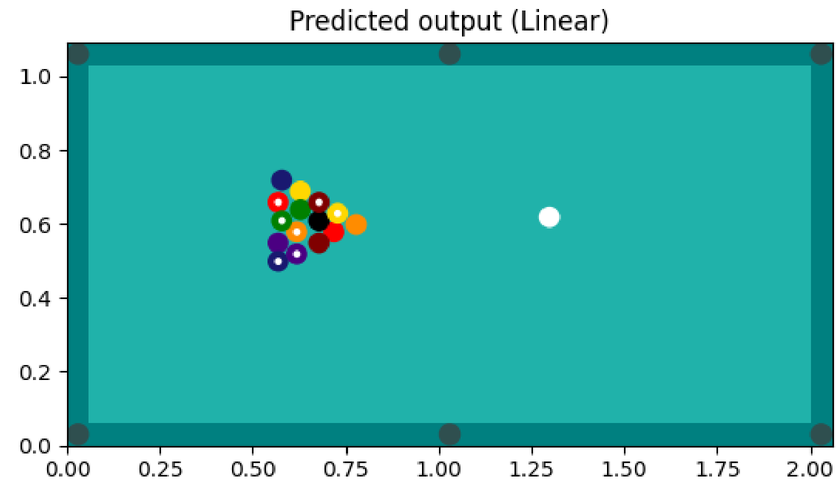
- Obvious bottleneck for model early into development: training data
- Create intermediary synthetic dataset:
  - Balls at random positions on table
  - Random linear motion direction of balls
  - No collisions, but learn linear motion
  - Keep improving the model until more real data is available
- Later in development switch back to real (and augmented) data

Evaluation

Rendering

- Object plotting on layers using Matplotlib
- Frame Generation
- Video generation using OpenCV
- Comparison view





Rendering

Features

- Dimensions are an exact scaled version of the original. Colours are as close as possible to the original table.
- Solid balls represented as solid circles and striped balls are represented as a solid circle with a white inner circle.
- **Comparison view** of real data and predicted output
- Option for **video generation** as well as **on screen display**
- Support for different representation of different model outputs
- Video generation at the rate of 30 frames/s

Rendering

Idea

- Prediction output of the model is in the form of a row of 32 float values corresponding to the (x,y) coordinate of each ball with respect to the table.
- Difficult to have a cohesive and coherent understanding of the model behavior through plain numeric values.
- To get a better idea of the behaviour of the model, with respect to real life ball movement on a billiards table, a visual representation of the numeric values was needed which was close to the original imagery.
- **Matplotib** library was used for **frame** generation and **OpenCV** was used for **video** generation.

Evaluation

Multi-step prediction

- N-step prediction easier than full prediction
  - Can be as simple as singlestep prediction with n=1
  - Shows short-term mistakes, but no cumulative long-term inaccuracies
  - Useful to spot mistakes in earlier models
- Later switch to full prediction for more advanced models