1 Hypothesis

The tracking and prediction of billiard balls by a Kalman-Filter improves by explicitly taking the physical conditions of a billiards table into account when implementing the filter.

2 Operationalization

In the following section we are going to operationalize the different terms in our hypthesis.

2.1 Tracking

The tracking describes the deviation from the filtered position of the billiard ball (result of Kalman-Filter) to the real position of the billiard ball. The position divides into three dimensions, the x-coordinate, the y-coordinate, the z-coordinate. Since we are investigating a billiard game we disregard the up-direction (z-coordinate), because the ball moves on a flat surface and will not leave the plane. We are measuring the coordinates in pixels. The range of the pixels is defined by the resolution of the input video on which the Kalman-Filter gets applied to. The origin, coordinate (0,0), is always the top-left corner. The final deviation from the two position is calculated with the Euclidean distance, which results in a length in pixels.

The position of the billiard ball changes over time, because we are investigating video sources. To measure the deviation of the whole video and to get a key figure, we are calculating the mean square error (mse) in which the error is the pixel distance between the two positions.

2.2 Prediction

The prediction describes the deviation from the calculated future position of the billiard ball to the real position at this specific time. The positions use the same measurements as described in the previous chapter. The time is measured in frames. The specific number of frames we will look in the future will vary in our experiments. To make the frame count comparable with different input video source we align them with the framerate (frames per second) of the video.

2.3 Improvement

The improvement compares a standard Kalman-Filter to our implementation of the Kalman-Filter. The improvement is measured as the difference between the different mean square errors of the tracking and the prediction. The smaller the mse of our Kalman-Filter compared to the standard implementation the greater the improvement.

3 Data Acquisition

We are going to acquire our data from billiard simulations and real videos.

3.1 Simulation

We construct a simulation to fully control every parameter and the state of the billiard game. This helps us to accomplish a better conclusion to the performance in real video. Our examination units are the framerate, the ball velocity, the sensor noise.

Framerate The framerate describes the framerate of the video, which is equal to the sampling rate of the Kalman-Filter. A standard Kalman-Filter performs much worse on low framerate because it gets less data to process. We test different framerate to inspect the performance of our implementation for different environments. For realtime applications on mobile devices our implementation has to perform well on low framerates. We are going to test the standard framerates 30 and 60 FPS and a very low framerate at 10 FPS.

Ball Velocity The ball velocity describes the speed of the ball, which depends on how hard the player hits the ball. The velocity is measured in pixel per frame. The cover a lot of cases, we choose a low velocity of 300 pixel per frame, an average velocity of 500 pixel per frame and a high velocity of 700 pixel per frame. We have to keep in mind that a high velocity also leads to more collisions because the ball will move a longer distance.

Sensor Noise When we analyze realtime videos we have detect the balls in the video frame. The detection is not always perfect and heavily relies on the light conditions and resolution of the camera. To simulate the quality of detection we added a noise value which changes the detected ball position by random value in a certain range. We test a deviation of 15 pixel, 30 pixel and 60 pixel to inspect how well the Kalman-Filter will perform.

Change of parameters To test the parameters individually we only change one parameter at the time. Three variations on each parameter lead to 9 simulation environments in total.

3.2 Real Videos

After testing the filter in the simulation we are also doing some test on random billiard videos. Since we are relying on a ball detection algorithm to identify the billiard balls, we cannot obtain the real position of the ball like we could in the simulation. That's why we cannot make statements about the tracking quality, but we can evaluate the performance of the tracking.

4 Data samples

For the data sampling we must take into account our two approaches in 3.1 and 3.2.

Simluation Since the simulation is not very computational intensive we are able to select any sample size and resolution in terms of delta time. This is especially useful in edge cases as we already outlined in our research proposal. However in regard to our evaluation with real world footage we also consider a less dense temporal resolition to match with these. The way we are doing this in our research is outlined in 3.1.

Real Videos Since the real videos have a fixed framerate, our data sampling relies on the temporal resolution of the cameras - see 3.1. Every videoframe represents one data sample in this regard. We can derive all the necessary parameters outlined in 3 off these.

5 Data Evaluation

The key figure of evaluation is the mean square error. If the means square error of our implementation is lower compared to the standard implementation in every experiment we can prove that our implementation improved the tracking and prediction of billiard balls.

5.1 optimizing

Since we can create a big dataset with our simulation, we're able to optimize certain parameters in our filter design. There two approaches to solve this optimization problem.

1. Iteration
2. Modeling

Iteration We simply iterate over all parameters of our simulation which have an effect on the outcome of the filter quality. This approach might take a long time to simulate considering every iteration of one parameter is a nested loop. This will result in an O^n runtime behavior, where n denotes the number of parameters involved. This is very undesireable, since runtime grows very rapidly.

Modeling This approach is more complex to implement but more effective for a big sample size. It works as follows:

1. Create a sparse dataset by iterating over a few hundred parameter variations
2. Use a regression algorithm to fit a model to our filter behavior
3. Use this model to find the optimal konfiguration by stochastic gradient descent

Which regression algorithm is yet to be determined, since we have to do further research to evaluate the complexity of our behavior. However we aim to use a regularized least squares approach to get the sparsest solution possible. This will further enhance the quality and effieciency of our model.