Pool-Predictor - a Kalman-Filter Implementation

Marlon Lückert B.Sc. - Julius Neudecker B.Sc.

Feburary 2020

In this article we discuss an implementation of the Kalman-filter [1] to improve measurement quality and predict the movement of balls on a pool table. We are going to point out the reasoning why we implemented the filter as a constant velocity model and its weak points in this situation. Since the performance of the filter deteriorates in cases of a rapid change in direction, we derived two different implementations with adaptive behaviour. The implementations were tested in a simulator and with real world video footage of a pool table. In the end we compared the performaces of each filter implementation.

Keywords— kalman, pool, prediction, filter

1 Introduction

1.1 Game of Pool

At first glance the game of pool is very suitable to examine the behaviour of a kalman-filter enhanced tracking system based on pure visual tracking. The surface of a pool table is made of a thin fabric which covers a hard surface i.e. slate or granite. The balls nowadays are usually made out of resin. This combination of materials creates very small rolling resistance and the balls behave almost fully elastic on collision. Also with only 2 DOF[1] a basic kalman filter implementation is fairly simple in this regard. But when two balls hit each other or a cushion, the velocity vector changes its orientation instantly. If this isn't taken into account, the filter needs some time to adapt to the new direction of movement and will produce wrong estimations during this time.

This behaviour is independent of the type of kalman implementation being CVM or CAM (see section 3.1). A kalman filter will assume the direction of movement on any given sample is about the same as in the last sample. It will therefore create wrong estimations if the directon of movement changes drastically in a short period of time. The time the filter needs to recover depends on the filter gain.

We also use the filter to predict values for any given length into the future by feeding back its estimations as actual state. The quality of this prediction however depends on several factors, which we discuss in detail in 3.

1.2 Visual Recognition of moving elements

In order to create a state-vector input we have to track the balls on the pool table. Since we needed reproductable results we used stock-footage as well as our own videos and processed them with the Python implementation of openCV. These videos fig. 1 are put into an instance of simple video-processing steps to create a black and white mask of the ball contours as displayed in fig. 2, where the center of the white pixel cluster is the ball we're looking for.

Figure 1: color picture of a pool table

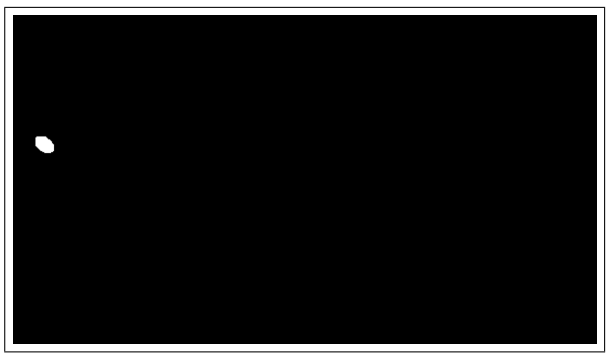

Figure 2: contours of balls after processing

This type of vision based tracking works reliably for our requirements.

[1] Dimensions Of Freedom - determines the possible rotation or translation along each given axis

2 Related Work

In the following, we discuss some previous work that is related to our work but none of these articles targets the problem in the same manner.

Jong-Yun Kim and Tae-Yong Kim [2] developed a method to provide robust tracking of a soccer ball. Their problem is that the soccer ball might be occluded by the player at any given time so an optical tracker could not deliver any results. In this case they used the velocity vector of the player to substitute for the ball presuming that the ball moves in the same direction as the player.

Jia et.al. [3] conducted research in the trajectory of pool balls, which helped us to decide which kalman model is the most suitable.

Shiuh et.al. [4] provided a good starting point how to create a tracking algorithm for pool balls. They also developed an algorithm to track occluded objects using an adaptive kalman filter. In this case they used to threshhold in order to determine wheter the object can still be reliably tracked. If this isn't the case the filter will rely only on predicted values until the object can be tracked reliably again.

Salzmann and Urtasun [5] proposed a more general approach for tracking. They were able to recreate a highly accurate tracking from a noisy picture based on newtons 2nd law and markov models. Using different constraints and presumptions they were could physical parameters like friction and trajectories.

Mohamed and Schwarz [6] are using partly the same approach as we do to improve the results created by INS/GPS[2] Systems. However their approach only targets the 'Q' and 'R' parameters of the filter.

Sarkka and Nummenmaa [7] created an adaptive kalman implementation which adapts itself to time-varying noise parameters. Since our input data is constant in this regard, we decided to simulate for the optimal filter parametrization instead of relying on the filter to adapt itself.

Gabdulkhakova and Kropatsch [8] use a kalman filter to create a video analysis tool for snooker games.

3 Implementation

In this section we're going to discuss the different implementations and their constraints. The filter implementations in section section 3.2 and section 3.3 are derived from the basic implementation in section 3.1. At the end of this section we will provide a comparison of all filter types and their performance in our simulation.

3.1 Kalman Filter

The basic layout and working mechanism of a kalman filter is in described in detail in [1]. However in order to fully understand our improvements we provide a short explanation.

Its purpose is to filter noisy or unreliable sensor data to get a better quality. Its doing this by comparing predictions made based on the model and actual measurements. Depending on the filter gain it will use a bigger portion of either. This estimated value represents the actual state of the measured system and is used to make the next estimation.

This is represented in eq. (1)

$$\hat{x}[n] = \tilde{x}[n] + K * (y[n] - C * \tilde{x}[n] - D * u[n]) \quad (1)$$

Where K represents a factor, which determines the amount of measured values applied to the prediction to create a new estimation. It is defined as in eq. (2), where $\tilde{P}[n]$ is the prediction error from the current prediction, C^T is a matrix to map our state vector to our system model and S represents the combined error of estimation error and measurement noise.

$$K = \tilde{P}[n] * C^T + S^{-1} \quad (2)$$

Every prediction which is used in eq. (1), was previously calculated using the state transition model A_d:

$$\tilde{x}[n] = A_d * \hat{x}[n-1] + B_d * u[n-1] \quad (3)$$

The important parts for our further development are A_d in eq. (3) and $\tilde{P}[n]$ in eq. (2). The state transition matrix Ad defines the motion model of our filter. There are two principal models which we are going to examine: the constant velocity model (CVM) and constant acceleration model (CAM), which describes which parameter can be regarded as constant by the filter. The estimation error $\tilde{P}[n]$ is defined in eq. (4):

$$\tilde{P}[n] = A_d * \hat{P}[n-1] * A_d^T + G_d * Q * G_d^T \quad (4)$$

The element Q is the important part here. It defines the amount of systemnoise in the whole filter process, which we discuss in section 3.2.

3.2 Dynamic Q-parameters

As we already discussed in section 1.1, the filter can't adapt to sudden changes in direction of the moving object. The case where the ball collides with the cushion can be mitigated by choosing different values of Q in eq. (4). Thus the filter puts less trust in its own estimation and relies more on the measurement. By doing so it can adapt momentarily. We implemented a look-up table, where the current position is checked for proximity to a cushion.

3.3 Smart Filter

In addition to the previous section we developed another improvement. In this case it can be assumend that the angle of incidence is equal to angle of reflection. This assumption can be useful since this can be implemented in a single vector-multiplication expression.

[2] Inertial Navigation System / Global Positioning System

Assuming that we have a velocity vector pointing in the direction of x and y, the operation to mirror on the x-axis is as in eq. (5):

$$\vec{v}_{mirror} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} * \begin{pmatrix} -1 \\ 1 \end{pmatrix} \tag{5}$$

As in the previous section this operation can be performed, when the ball hits the cushion.

3.4 Filter performance comparison

We developed a simulation, where we could test different parameters and scenarios to get consistent results. We also used this simulation to determine the optimal value for the process noise in section 3.2. The results are shown in fig. 3:

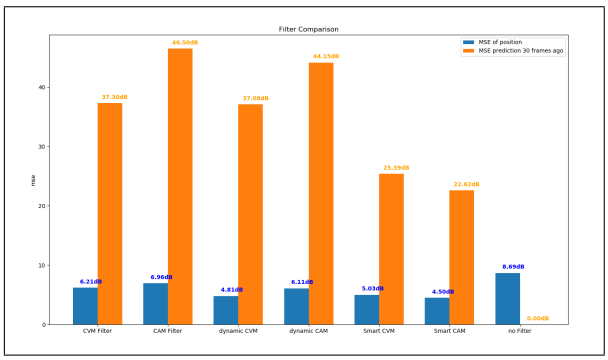

Figure 3: Performance of different filter implementations for CVM and CAM

The blue column represents the filtered values, which were generated from noisy measurements compared the ground truth. The orange column shows how much the prediction 30 frames in the future deviated from the ground truth. The effect of the dynamic process noise is negligable but the vector mirroring has a large impact on the accuracy on either motion models. It is especially interesting that the CAM model shows a better performance in this case. This is due to the fact this model can be very accurate but is also very sensitive for error.

Another impact the vector mirroring has on the time scale is show in fig. 4:

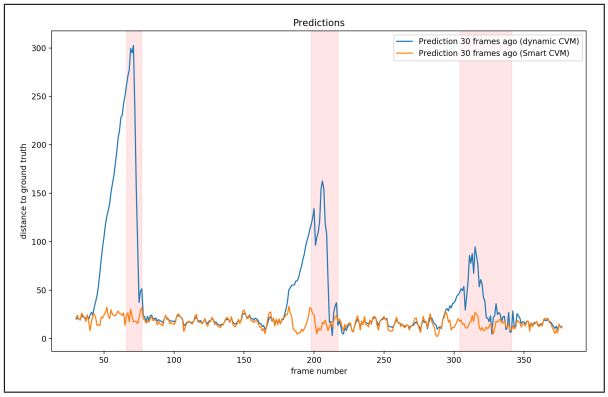

Figure 4: Performance of different filter implementations

This figure represents the distance to ground truth over time for the prediction 30 frames in the future. The red areas are where the ball is close to the cushion. Although the dynamic-q implementation is able to mitigate the overshoot problem, the prediction error rises quickly since the accuracy declines also with a higher process noise. The smart implementation however delivers constant accuracy regardless of proximity to the cushion.

4 Simulations

We simulated this smart implementation of the CVM with different technical parameters to examine its performance for predicting values 15, 30 or 60 frames in the future. Displayed are each distances from the prediction to the ground truth over the entire simulation.

4.1 Sampling

The first simulation altered the sampling-rate. In a real-world scenarion this would be the frame rate of the camera.

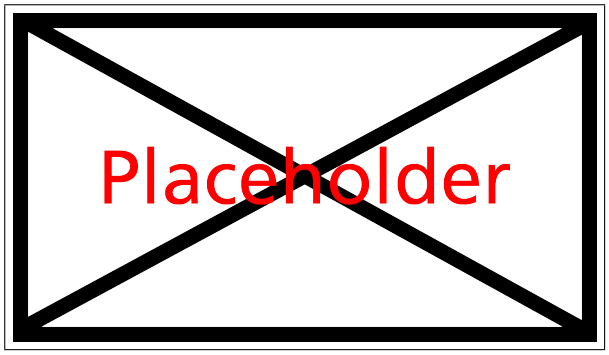

Figure 5: Impact of framerates on prediction accuracy

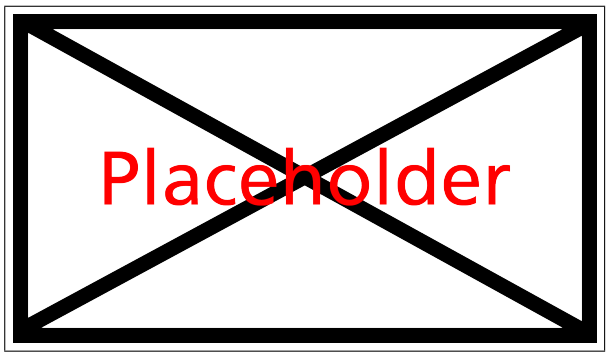

Figure 6: Impact of framerates on prediction accuracy

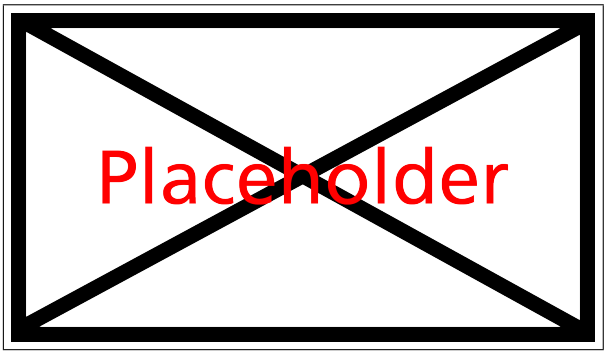

Figure 7: Impact of framerates on prediction accuracy

The sample rate has a certain impact on the quality of the prediction. However the the difference between 30 and 60 frames is not as big as between 10 and 30. This depends greatly on the initial velocity of the ball. The anomaly at 30 frames with a sample rate of 10 fps is caused by the fact that due to the low sample rate the filter just misses the point where the ball hits the cusion and isn't able to predict this.

4.2 Noise

The second simulation altered the amount of noise on the measurement and how the filter is able to cope with bad input data.

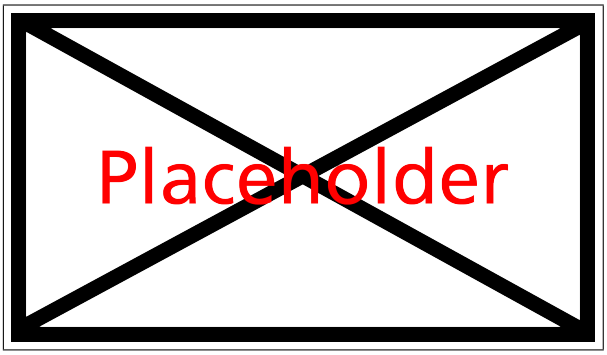

Figure 8: Impact of framerates on prediction accuracy

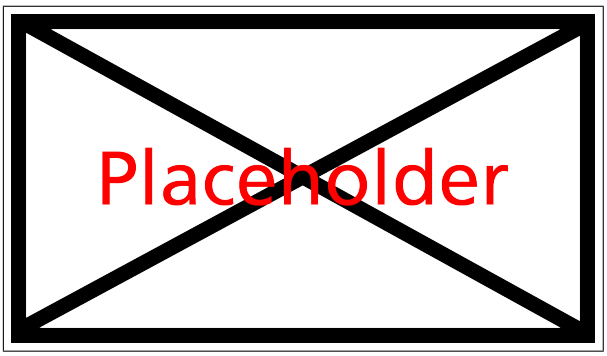

Figure 9: Impact of framerates on prediction accuracy

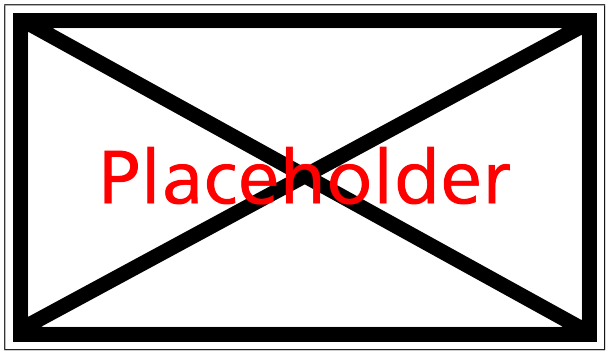

Figure 10: Impact of framerates on prediction accuracy

Although the amount of noise increased fivefold, the median stays almost constant compared to less noisy samples. The min and max values of deviation however change greatly on magnitude. This is a good indicator that the filter is actually able to work with worse measurements than we used in our simulation and is still able to produce reasonably accurate predictions well into the future.

4.3 Velocity

Lastly we tested the impact of different starting velocities, which has an influence of the distance, which the ball rolls between certain sample periods.

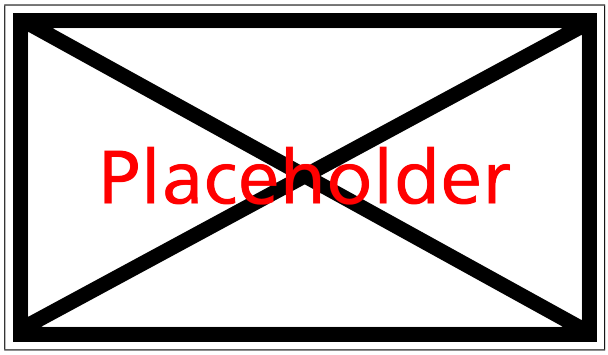

Figure 11: Impact of framerates on prediction accuracy

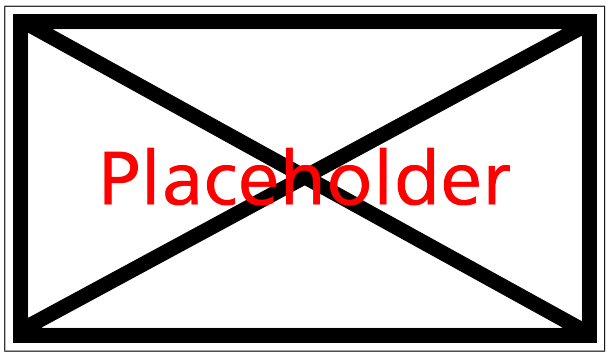

Figure 12: Impact of framerates on prediction accuracy

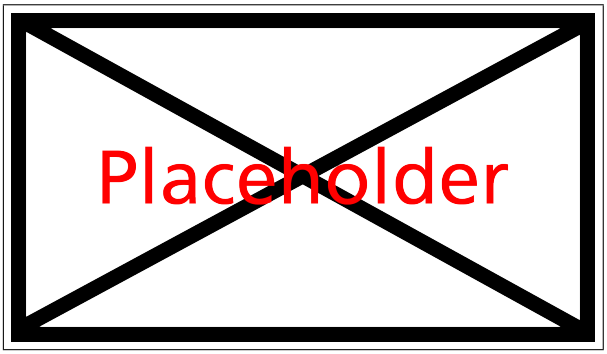

Figure 13: Impact of framerates on prediction accuracy

We can see here that the starting velocity has barely any effect. This is what we expected since the sample rate is high enough to give us sufficient data and we simulated different konfigurations in order to achieve the highest accuary possible.

5 Results with real footage

After evaluating that our filter works well within the simulation we tested it on real footage. It performed as expected in the footage and gave reasonable good predictions.

However because of imperfections in the tracking alorithm such as deviation what the center of the ball is and changing noise levels in the measurement, the filter can't perform as good as in the simulation. Also the reflection-angle on the cushion was mathematically perfect in the simulation but can vary by small margins in a real scenario.

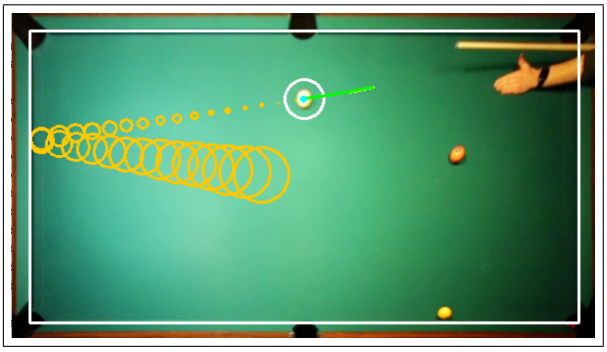

Figure 14: Filter test on real footage

Apart from the simulation we did not have data for the ground truth in this case. The quality of the filter can solely be measured by the quality of its prediction.

As in section 3.4 mentioned, the CAM model is very sensitive to error. This gets very apparend in real world footage with the MSE (for the prediction of the next 30 frames) of the CVM is only 15.39dB compared to CAM with 21.01dB.

6 Improvements

An inherent problem with the nature of sampled values is that there is no prediction or estimation of the behaviour in between to sample points. It is reasonable to assume that the behavior between two sample points can be interpolated. But since this problem has edge cases, this more complicated.

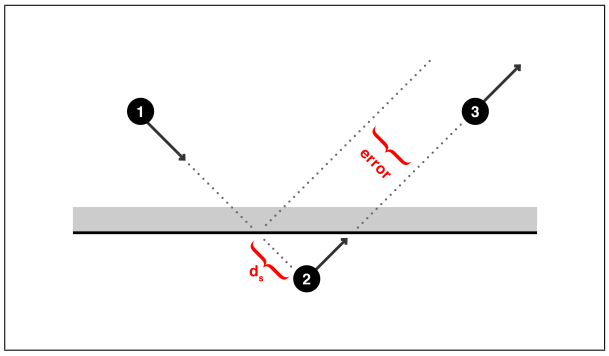

Figure 15: Ball hits cushion between two sample points

As seen on fig. 15 the collision between the ball and the cushion is happens between sample 1 and 2. The code detects this and turns the velocity vector as described in section 3.3. By the time of sample 3 the velocity vector points in the right direction but with a small offset error. The correct solution however is shown in fig. 16: The overshoot of sample 2 is corrected and mirrored on the point of collision, which affects the following samples. This has a large positive impact on prediction accuracy as well as filter error.

Our current version of the implementation does not has this correction yet but this could be implemented in the future.

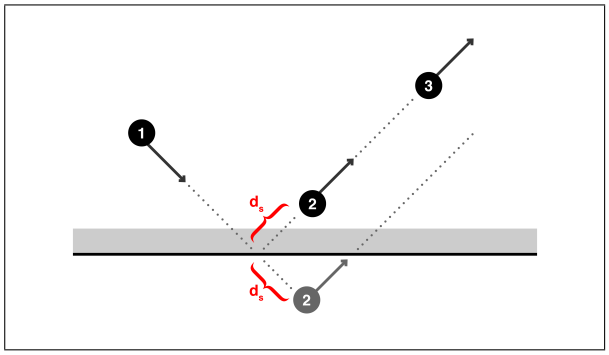

Figure 16: Collision correction

7 Acknowledgement

We would like to thank Prof. Edeler for his input during the lecture which really helped to further develop the idea.

References

[1] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, 1960.

[2] J.-Y. Kim and T.-Y. Kim, "Soccer ball tracking using dynamic kalman filter with velocity control," *IEEE, Tianjin, China*, 2009.

[3] Y.-B. J. et. al., "Trajectory of a billiard ball and recovery of its initial velocities," *Department of Computer Science Iowa State University*, 2011.

[4] S.-K. et. al., "Video object tracking using adaptive kalman filter," *Journal of Visual Communication and Image Representation*, 2006.

[5] M. Salzmann and R. Urtasun, "Physically-based motion models for 3d tracking: A convex formulation," *International Conference on Computer Vision*, 2011.

[6] A. H. Mohamed and K. P. Schwarz, "Adaptive kalman filtering for ins/gps," *Journal of Geodesy*, 1999.

[7] S. Sarkka and A. Nummenmaa, "Recursive noise adaptive kalman filtering by variational bayesian approximations," *IEEE Transactions on Automatic Control*, 2009.

[8] A. Gabdulkhakova and W. G. Kropatsch, "Video analysis of a snooker footage based on a kinematic model," *Vienna University of Technology*, 2012.