Kalman filtering for object tracking

Jingwen Yang and Peter Szabo

I. INTRODUCTION

In this laboratory work we present a background subtraction and Kalman filter based single object tracker. We implement and analyze the behavior of the tracker, first on toy data, and afterwards, on real data. We experimented the effect of different parameter setup, and explained the possible strengths and weaknesses of the tracker. 2 different models were implemented: constant velocity and constant acceleration.

The tracker program of two parts: blob extractor and tracking. The blob extractor works on gray-scale frames. The background model is created with the help of foreground detection, and the filtered blobs are feed to the tracker.

During implementation, we used C++ with OpenCV library in eclipse in Ubuntu OS. For evaluation and analysis the provided datasets were used.

II. IMPLEMETHOD

In this section I am going to briefly summarize the Kálman algorithm, we implemented during our work (since we only had to used the learned things). The single object tracker consists of two parts: Measurement extraction and Kalman tracking.

The measurement extraction get gets the RGB frames from a video file, as input, and converts them to gray-level frames. Gaussian Mixture-based Background and Foreground Segmentation algorithm provides the background model (it was the subject of our first work, where a more detailed analysis is available about that algorithm). We used the OpenCV algorithm called BackgroundSubtractorMOG2. The exact algorithm is described in [1] and [2]. Afterwards simple morphological opening is applied in order to get rid of the noisy detection. Opening is an another name of erosion followed by dilation. We do this, in order to remove the smaller elements from the frame. To create and apply this filter we used the OpenCV built in functions, which created a rectangular kernel, and applied the opening on the image. The $extractBlobs$ function is responsible for extracting all the possible blob candidates. It returns the blob list with all the possible blobs on the given frame stored in vector containing $cvBlob$ objects. It applies an extended version of the flood fill algorithm, that we implemented for the previous assignment. The extension consists of counting the number of pixels each blob contains. Also cvBlob were defined in a way, that it is able to store the area of that object, among its x,y, width and height information.

$removeSmallBlobs$ function is responsible for extracting the single object, which we aim to track. It gets the possible blob candidates as its input, and returns a single point, the center of the blob. It calculated the blob with the biggest area, which width and height is also bigger than a given threshold.

Once the measurement was extracted, the Kalman filtering was the following step. In our work, we implemented two KalmanFilter model, one relying on constant velocity, one on constant acceleration. These models are generated in $createConstantVelocityKalmanFilter$ and $createConstantAccelerationKalmanFilter$ functions respectively, and return the same KalmanFilter object, with different parameters. These function define the state transition, the process and measure noise, and the measurement matrix. We defined state transition in a slightly different way from the class, it can be seen on the following equation:

$$x_k = [x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}]$$

(the same way for the constant velocity model). All the matrices were changed accordingly. The reason behind this decision was, that this way our measurement were always the first two values, independently from the model we used. This allowed us greater generality for handling our model

The filtering is executed in $doKalmanFiltering$ function, which takes the Point we measured (the center of the traceable object), which is NULL as long as there are no measurement. Also takes which KalmanFilter model to use, and further vectors, where we store the predictions, and measurements for the sake of creating the trajectory. As long as there were no measuremnet the function is inactive. This is important since we want to reduce the amount of false detection accumulation. Once the first measurement arrives, the State Posterior, with the measurements being the current position and initial velocity and acceleration with random (but probable) values. Once the model is initialized, it created the prediction, and if there is an observation it corrects the prediction, or if there is none, it it uses the predicted value as the assumption of the current state.

After the tracking we visualize the blob for the object at each frame with the $paintBlobImage$ function. It fits an ellipse on the object with its observed height and width. If there is no observation, a base circle is displayed. Moreover a label shows if the object was only predicted or also corrected.

Finally once a sequence finishes, in $showTrajectory$ we draw the final trajectory on the background image. The blue dots, connected with a line shows the measurement, and the red ones are showing the development of the trajectory.

Also our sequence can be seen image-by-image or as a video, and the frames of the tracked object can be saved.

Different parameters of our program can be set under the PARAMETRS section (at the 107.th line).

III. RESULTS AND ANALYSIS

A. Task 3.2: Toy Data

In this task we analyze the effect of kalman components for toy video sequences while keeping fixed measurement extraction parameters. Four sequences with different scenarios are used to test the performance of Kalman filter.

TABLE I: Videos for Testing Purposes

Name of the Video	Number of Frames	Challenges
video2.mp4	110	fast movement
video3.mp4	127	speed and direction change
video5.mp4	113	lost of measurement
video6.mp4	130	occlusions

The first video 'Video2' is ball rolling fast. The second video 'video3' is ball rolling and hit the wall and change its direction. The third video 'video5' is ball rapidly bouncing on the floor. The fourth video 'video6' is ball rolling through a box where the model lost measurement.

B. Task 3.2: Analysis for toy data

The objective of this task in our report, is to analyze the effect of Kalman components for toy video sequences with fixed measurement extraction parameters. During this section, we used the recommended parameters.
Parameters of the measurement extraction:

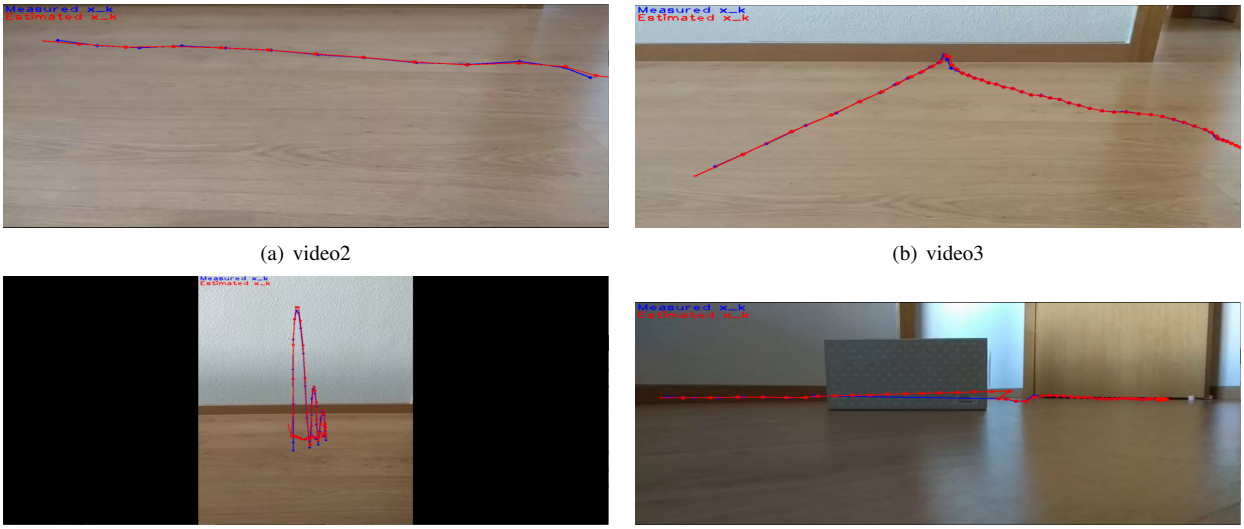
- MOG $learningrate$ = 0.001
- $varThreshold$ = 16 and $history$ = 50
- Opening parameters: morphological kernel size = 7 with rectangular type (*)
- minimal with and height for blob extraction = 50

(*) thats the only point where we deviated from the recommended values, since with kernel size = 3, there were some at the Extraction of the object Measurement and in this section we only wanted to experiment with the effect of the parameters of the Gaussian. We decided to use the best possible parameter setup, and the detailed discussion of the effect of different kernel sizes will be the scope of the next subsection.

In addition,to find the biggest blob, at the beginning we searched for the one, that has the biggest width * height (and is also bigger than a given threshold). This area definition proved to result some miss-classification, as in some cases our foreground detector detected L shaped objects, which had remarkable width and height, but did not contained numerous pixels, and was a false positive detection. The morphological opening aimed to overcome these kind of errors, but the kernel size proved to be heavily context dependent. In order to provide a more general solution we were seeking for objects, which contains the largest number of pixels. That is the reason why we modified our blob class and flood fill algorithm.
The solution could possibly result further error, that lot of connecting pixels may get prioritized over real objects, but this scenario is less likely to happen due to the morphological operator, so we were happy to pay this price.

Our goal was to explore the differences between the constant velocity and constant acceleration model. In order to describe their strengths and weaknesses I compare their behavior or each toy video. As we can see both of the models produced a reasonably good accuracy, but On video2, the velocity was approximately constant, both of the model did a very accurate tracking. On video3, we can observe a ball bouncing back from a wall, which results a sudden change in direction, and afterwards the speed start to decay. In this case we can see, that both model need a some time at the beginning to catch up with the fast movement. This can be solved by better initialization of the parameters. In my model, I set the initial velocity to be a random number to be between -2 and 2, and the initial acceleration to be between -1 and 1. By increasing the initial velocity we will observe that our model follows the initially fast moving objects silhouette better.
Both of them handles the bouncing well, although the velocity model need a slightly more time to catch up with the real movement of the ball.
One thing worth noticing, that while constant velocity model correctly follows the object leaving the scene, the acceleration model turns around at the edge of the image and returns. We can see it as a possible source of error, when the model doesn't handle well, when the object leaves the scene. The cause of this effect is that the model incorporates the negative acceleration, and when the object leaves the frame lacking the measurement the acceleration still remains negative, changing the direction of the movement.
In the video5, we can again observe that both model handles the change of the direction very well, there are no overshoots. Also they adapt accurately the decrease in velocity, and finally the stopping of the object.
Finally, the last video was the most challenging since at some point our object disappears, and an reappears, while also changing its velocity. As we can see both model follows the imaginary path, but slightly loses their speed the longer they develop without the correction of the measurement. The constant velocity is better at being closer to the original speed, but once the new measurements arrive they correct their mistake.

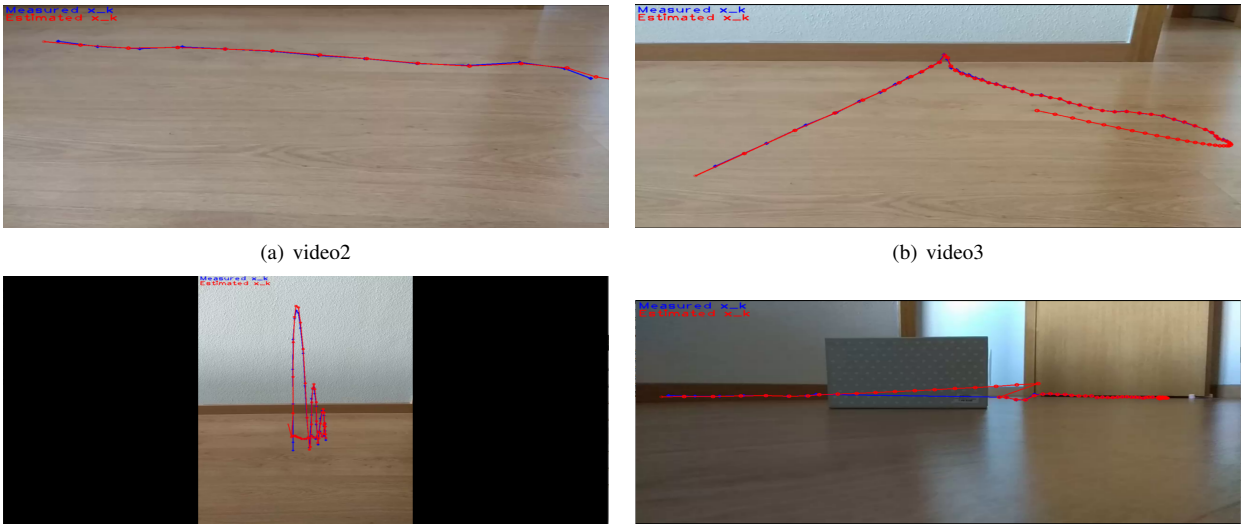
Finding the correct values for Kalman filter we should take multiple things into consideration. By setting up P_k we should decide how sure we are initially in our model. In most of the cases we are not, since we don't know, how fast our ball is going to move, with what acceleration. So out initial goal was to set this parameter a bit higher, so that at the beginning our observation should have a greater weight at forming our model, we can observe this, that or tracker catches up faster with the object's speed.
The next value to consider was the covariance of the observation noise (R). This parameter tells use how much we rely on the accuracy of our feature extraction. In our case we should heavily rely on that, since we put a lot of emphasis



(a) video2 (b) video3

(c) video5 (d) video6

Fig. 1: Results of constant velocity model on toy dataset



(a) video2 (b) video3

(c) video5 (d) video6

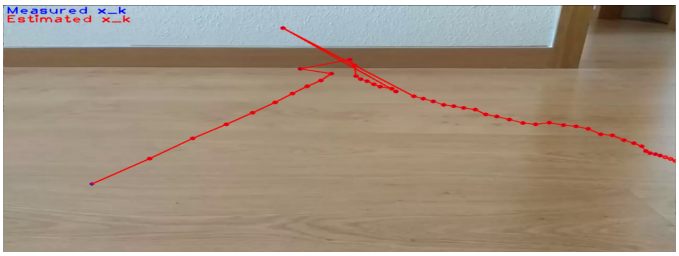
Fig. 2: Results of constant acceleration model on toy dataset

using a feature extractor with good parameters. But if we would have a noisy foreground model, we should be less sure about our observations.
P and R together make the overall uncertainty.

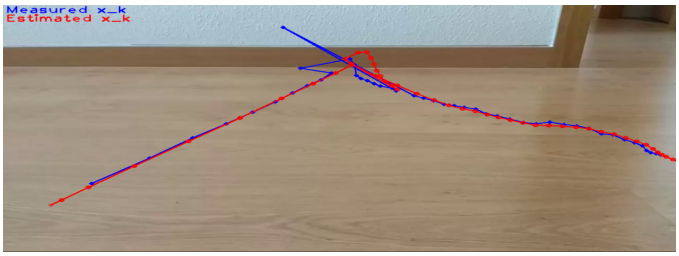
To completely explore the effect of R value we went back to use the kernel size = 3, in order to introduce some noise. Its effect can be seen on Figure 3. On the first image we can see when the R = 0, and we completly rely on the observations. This could be dangerous, since as we can see, we also follow the noise of the observations. On the other hand as we increase R, we see that we get more robust to the errors on the measurement. Also on the video6, we can observe greater accuracy with higher R value, and we will be able to estimate the speed of the object, while its hidden.

Finally on Figure 4 we can see that the value of R has a bigger impact on the acceleration model.

To conclude, both model works accurately, and have advantages and disadvantages in given scenarios. The acceleration model adopts better to speed changes, but is also more exposed to errors in measurement, since the acceleration is the second derivative of the location(what we could measure directly), and if there is a small error in the acceleration, that results a bigger error in the location compared to the velocity. On the other hand in this sense the velocity model is less stable, but also follows the changes in acceleration worse. This is supported by the observation form Figure 4, looking at the behavior of the different models under the same conditions.

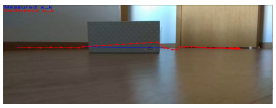


(a) The value of R is very small



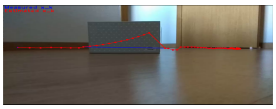
(b) The value of R is greater

Fig. 3: Investigating the effect of R matrix on the behavior of our model

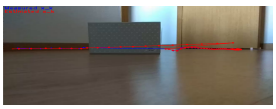


(a) The value of R is close to 0

(b) The value of R around 100



(c) The value of R is close to 0



(d) The value of R is around 100

Fig. 4: Investigating the effect of R matrix on different models

C. TASK3.3: Real data

The first video 'abandonedBox' is taken by a monitor at a crossroads. There is a man riding bike waiting for the traffic light. The difficulty of this video is the lost of measurement when object stops. The second video 'boats' is taken at a boat driving on the lake, which turns around the opposite direction. The third video 'pedestrians' is taken under strong illuminance condition and cause shadow of the walking woman. The fourth video 'streetCornerAtNight' is taken in the night without much color difference or contrast but strong light from the fast-moving car.

TABLE II: Videos for Testing Purposes

Name of the Video	Number of Frames	Challenges
abandonedBox.mp4	401	lost of measurement
boats.mp4	951	angle change
pedestrians.mp4	226	shadow
streetCornerAtNight.mp4	90	fast movement and strong car light

D. TASK3.3: Analysis of real data

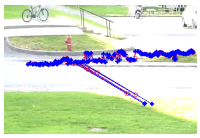
In this task we should use a set of fixed parameters for Kalman filter for all the sequences, either the default setting or the one with the best performance overall(should be justified). The task here is to adjust the EOM parameters for each sequence.

TABLE III: EOM parameters setting

Type of parameters	Name of parameters
MOG	learning$_rate$
MOG	varThreshold
MOG	history
Opening	Kernel Size
Opening	Type
Blob extraction	min$_width$
Blob extraction	min$_height$

(a) abandonedBox

(b) boats



(c) pedestrians

(d) streetCornerAtNight

Fig. 5: Constant Velocity Model with default parameters setting, blue dot-prediction, red dot-measurement.

1) Tracking problems that exist for all sequences: For the first sequence 'abandonedBox', at the beginning the trajectory is very good although there are a little bit drifting due to partial occlusions. But at the end, after the man riding a bike stops, the measurements keep going a little bit.
For the second sequence 'boats', there are a large amount of repeat overlapped predictions in vertical direction due to the turning around of boat in the middle of lake and unstable incoherent detections led by angle change.
For the third sequence 'pedestrians', there are several serious outliers because of the wrong measurements led by shadow. The path of predictions is a little bit drifting.
For the fourth sequence 'streetCornerAtNight', there are part of unreasonable predictions caused by the severe shift from incorrect measurement to correct measurement. After the first real measurement detected, the performance is quite satisfying except for the incoherent drifting led by low frequent measurements.

Since the constant acceleration model has either the same performance or worse, and here we mainly discuss the influence of EOM parameters, so the outputs of Constant Acceleration Model are not discussed here.

2) Adjust the parameters of the system: MOG parameters: learning rate The value between 0 and 1 that indicates how fast the background model is learnt. Negative parameter value makes the algorithm to use some automatically chosen

learning rate. 0 means that the background model is not updated at all, 1 means that the background model is completely reinitialized from the last frame.

(a) abandonedBox, learning rate=0.0005 (b) abandonedBox,learning rate=0.001

Fig. 6: Constant Velocity Model with various learning rate setting, blue dot-measurement, red dot-prediction.

The improvement of predictions here, facing the problem of stopping of object, can be dealt with a smaller learning rate. A small learning rate can make still good detections for stationary foreground object..

MOG parameters: varThreshold $varThreshold$ is the threshold on the squared Mahalanobis distance between the pixel and the model to decide whether a pixel is well described by the background model. This parameter does not affect the background update.

(a) abandonedBox, varThres=10 (b) abandonedBox,varThres=25

Fig. 7: Constant Velocity Model with various varThresholds setting, blue dot-measurement, red dot-prediction.

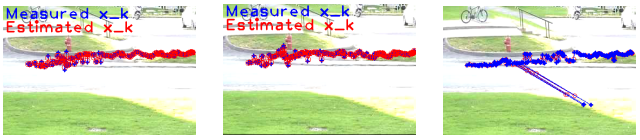
As we can see here for the abandonedBox video, the tuning of $varThreshold$ successfully cope with the problem of predictions after object stops. Since the object stops with a low learning rate, we certainly want a higher variance threshold to tolerate the precision of background model describing a pixel.

Openning parameters: morphological kernel size This parameter is the size of the structuring element for morphological transformation. As we can see there for the boat video,

(a) boats, morpho kernel size =2 (b) boats,morpho kernel size =4 (c) boats,morpho kernel size =3

Fig. 8: Constant Velocity Model with various morphological kernel size setting, blue dot-measurement, red dot-prediction.

the decreasing of $morphologicalkernelsize$ can achieve better performance than default setting. In this case we need less erosion to help preserve more information from the boat since when it turns around the sail almost be orthogonal to the camera and cannot be detected well.



(a) pedestrians, morpho kernel size =5 (b) pedestrians,morpho kernel size =6 (c) pedestrians,morpho kernel size =3

Fig. 9: Constant Velocity Model with various morphological kernel size setting, blue dot-measurement, red dot-prediction.

In this case to cope with the shadow we need larger morphological kernel size to get stronger erosion effect. Therefore both size = 5 and size = 6 have a decent performance. But I would say size=5 is slightly better than size=6 because the coherency.

(a) StreetCornerAtNight, morpho kernel size=8 (b) StreetCornerAtNight,morpho kernel size=7 (c) StreetCornerAtNight,morpho kernel size =3

Fig. 10: Constant Velocity Model with various morphological kernel size setting, blue dot-measurement, red dot-prediction.

In this case we need a even higher erosion to deal with the strong car light detected as moving object and therefore the wrong localization of the right object.

IV. CONCLUSIONS

To conclude we can say that the both constant velocity and constant acceleration model are working quite impressive with the good parameter setup. Finding the correct parameters is very task-specific especially in various scenarios with all the possible difficulties, like occlusions, multiple type of objects,shadows, the possible detection problem caused by angle change and so on. Also accurately defining a model is really experiment-oriented. For the task3.3, the $higtory$, min_width and min_height in EOM parameters seem not contribute to the performance for these sequences, plus the other parameters have already achieved good results, therefore they are not discussed in the report. A further improvement could be the use of more objects, since this task is only dealing with one single object.

V. TIME LOG

code implementation: 10 hours. 3 hours initially understanding the models, 7 hours further improve our models based on the feedback given from toy data and real data.
analysis:
playing around the effects of different parameters, and analyze their contribution. **writing the report:** 3 hours, writing the report itself, collecting the data and the images, evaluation and explanation