Kalman filtering for object tracking

Baglan Aitu and Alberto Ruiz

I. ABSTRACT

The aim of this lab is to implement a single object tracker by using Kalman filter. To track the object in a scene, first of all it was performed the blob extraction method from the previous lab assignment. After that, we implemented two versions of Kalman filter (constant velocity and constant acceleration) and checked their behaviours in toy data at first, and real objects in the end.
During the lab, the effect of Kalman filter components were analyzed with visual examples and parameters were tuned for each video frame of provided datasets in order to obtain the correct tracking performance.

II. METHOD

In this section the theory of the Kalman filter will be briefly explained. Kalman filter dates back to 1960, very distant from now. Despite that, it is an optimal estimator for linear systems. It assumes gaussian distributions within its framework. x_k and P_k represents the object and the covariance of the state, respectively. Higher P_k values mean lower precision, and vice versa. Kalman filter can be split into two steps:

1) **Prediction step.** It has two equations:

$$x_{\bar{a}} = A_k X_{k-1} \tag{1}$$

The first equation projects the state of the previous time into the current time, by using a transition matrix A_k that defines the relations between variables of the previous object state and the ones of the current one.

$$P_{\bar{k}} = A_k P_{k-1} A_k^t + Q_k \tag{2}$$

The second equation is the prediction of the error covariance, which is in charge of projecting the previous error covariance (P_{k-1}) to the new error covariance plus a term Q_k that determines the uncertainty of our prediction.

2) **Correction step.** Then, we project the object state and its covariance are fed to this step. The aim of this is to modify our prediction based on what we see in the image according to the measurements that we observe. We compute the Kalman gain with next two equations:

$$S_k = H_k P_{\bar{k}} H_k^t + R_k \tag{3}$$

The first equation determines the uncertainty of the measurement using the observation matrix H_k, which defines the relationship between x_k and our measurements; and the covariance of the observation noise R_k which defines the uncertainty associated to our measurement and feature extraction.

$$K_k = P_{\bar{k}} H_k^t (S_k^{-1}) \tag{4}$$

Based on overall uncertainty S_k, we can estimate the Kalman gain K_k. Kalman gain accounts for how much rely on the measurement given. Here we combine the overall uncertainty of the measurement and the prior uncertainty that we have about the state. In Kalman gain, the higher the value, the more we believe in our measurement.

$$x_k = x_{\bar{k}} + K_k(z_k - H_k x_{\bar{k}}) \tag{5}$$

With this equation we update the predicted object state. This update consists in adding a quantity to the predictions, which is the distance between our measurement and the projection of our prediction. This quantity is controled by the Kalman gain factor.

$$P_k = (I - K_k H_k) P_{\bar{k}} \tag{6}$$

In the last equation we update the error covariance of the state. It also uses the kalman gain.

The process is restarted again, for the next processed frames in time.

III. IMPLEMENTATION

The methodology of the implemented Kalman filter can be described in the pipeline shown below in Fig. 1:

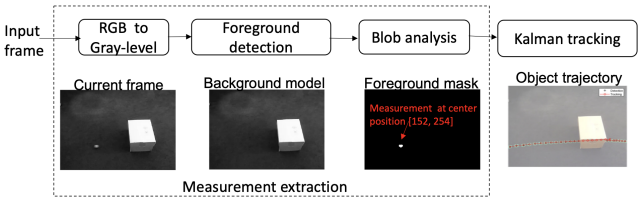


Fig. 1. Assignment steps

In order to measure the trajectory of the blob, we first need to detect it, that is, to extract it from the background. So, we apply our knowledge from previous assignments to extract the background using a Mixture of Gaussians (here and henceforth referenced as *MOG*). We are suggested to use the built-in OpenCV MOG function called *BackgroundSubtractorMOG2*. The morphological operator known as opening has been used to get rid of noise in the mask image. Its effect can be observed in the following figure, where kernels of (3x3) and (7x7) have been used, respectively:

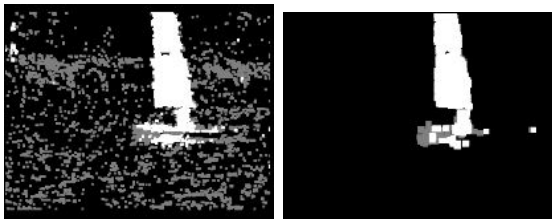


Fig. 2. Effect of the size of the structuring element

ExtractBlobs function (from previous lab) was responsible of extracting all the possible candidates to be a blob, applying the floodfill algorithm. Nevertheless, it will be *removeSmallBlobs* function the one that returns the blob with the biggest area. Adjustable thresholds for width and height of the blob are available as well.

The last, and the one we focused on, is the tracking step. We could use either built-in OpenCV Kalman filter or code it from scratch. We chose the second option, implementing the two variants: constant velocity and constant acceleration Kalman filters. The Kalman Filter produces estimates of hidden variables based on inaccurate and uncertain measurements. As well, the Kalman Filter provides a prediction of the future system state, based on the past estimations.

IV. TASKS

A. Task3.1

In this task we are required to implement a single-object tracker based on background subtraction and Kalman filtering like the one described in section II. For this task we have the *singleball.mp4* video.

From the video it was observed 2 main challenges:

1. Occlusion of ball with a box in some points.

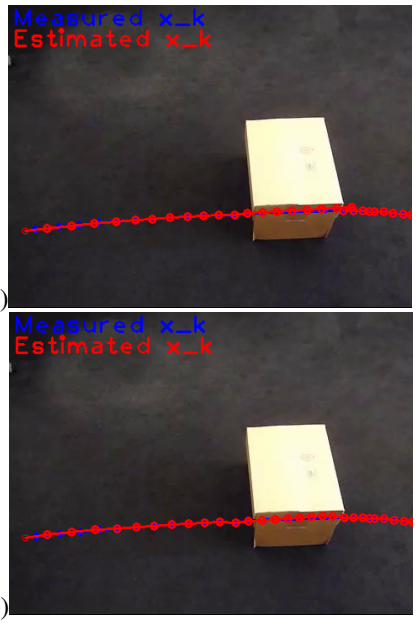
2. Velocity difference (the velocity of ball is higher before the box than when it is after).

Nevertheless, the good prediction performance was achieved as it can be seen in Fig. 2. Parameters used are those given as a recommendation (**default**), and are described in the following table:

- OpenCV MOG for foreground detection:
  - η = 0.001.
  - varThresh = 16.
  - history = 50.
- Size of structuring element in opening = 3
- Blob extraction
  - Width = 10
  - Height = 10

In 3 (a), the tracking of the ball was addressed by Constant Velocity (CV) Kalman filter. It gives more or less good trajectory prediction except the part where the ball comes out from behind of the box. As it was mentioned above, it happened because of velocity difference which CV Kalman filter is not able to handle.

To solve the issue, we need a model of Kalman filter which considers also an acceleration component, the Constant Acceleration (CA) Kalman filter. It uses the same principle as CV Kalman. The difference is introducing an acceleration variable, which makes model matrices bigger and complex. For example, here the object state matrix x_k has 6 values instead of 4. Its result of correction is visualized in 3 (b).



(a)

(b)

Fig. 3. (a) Constant velocity (b) Constant acceleration

B. Task3.2

In this task, based on task 3.1, we are required to analyze the effect of Kalman components for toy video sequences while keeping fixed measurement extraction.

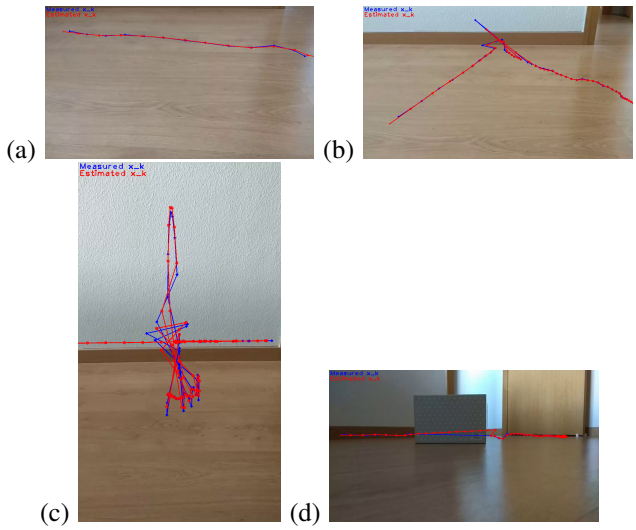
1) Experimental methodology: Our goal was to analyze the effect of both Kalman implementations in toy data, with suggested parameters. Once we see the strengths or weaknesses, we think about possible modifications that might improve the performance.

This dataset has a total of four videos:

- *Video2.mp4*. Here we see a tennis ball that moves from left to right side part of the image. No apparently challenges here. Plotted in subfigure (a).
- *Video3.mp4*. In this case the ball is thrown against the skirting, describing a trajectory similar to an inverse "v". Plotted in subfigure (b).
- *Video5.mp4*. The main challenge here is that the ball comes from the top of the image falling to the ground and bouncing, until it stops. Plotted in subfigure (c).
- *Video6.mp4*. Here we have the same example as in *Singleball.mp4*, but now with the tennis ball, and in a difference camera position. Plotted in subfigure (d).

2) Results and analysis: As obtained results from both CV and CA filters with default parameters were almost the same, not to good overally speaking. So only CV results will be shown in the following figure:

Tracking trajectory in video (a) seems to be correct. Same happens in (d). Once the ball is occluded, it returns the prediction, but fastly corrects the prediction. However, the bouncings in (b) and (c) are not correclty tracked. In this



case, checking the foreground mask we find out that the skirting is detected as foreground. Moreover, in (c), the background wall appears with a lot of noise. This makes our tracker detect the skirting as a prediction, when it is not. Due to that, we decided to increase the size of the structuring element of the opening up to 7. With this change, we obtained the following results:

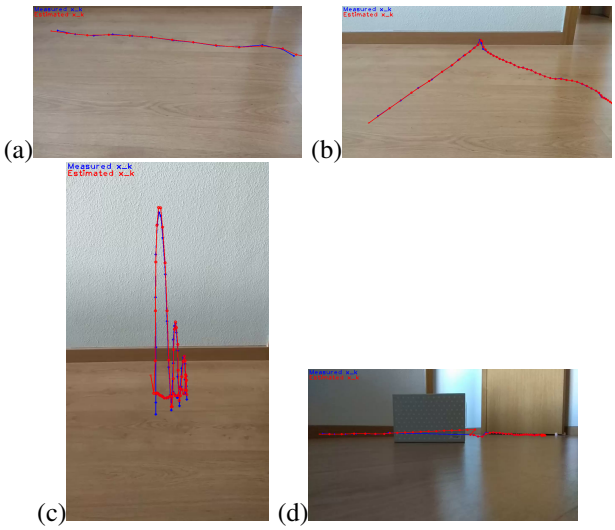


Fig. 4. Modified tracking trajectories in dataset 2

As IV-B.2 shows, increasing the kernel size solves our problem.

C. Task3.3

In this task, based on a mix of task 3.1 and mainly 3.2, we are required to analyze and tune the tracker implemented in task 3.1 for real video sequences from *changedetection.net*.

1) Experimental methodology: Here we follow the same steps as in task 3.2.

This dataset has a total of four videos:

- *abandonedBox-600-1000-clip.mp4*. Here we see man riding a bicycle that stops in front of a traffic light. Plotted in subfigure (a).
- *boats-6950-7900-clip.mp4*. In this case we mainly see a boat that is moving all over the lake.The main challenge here is that cars in motion can be observed in the background, and might affect to the tracking. Plotted in subfigure (b).
- *pedestrians-800-1025-clip.mp4*. In this video, a person is walking. Here, the main challenge is the excessive brightness, that makes us see the person strongly black-lit. In addition, a big shadow is created, that moves along with the person. Plotted in subfigure (c).
- *streetCornerAtNight-0-100-clip.mp4*. Here we see a car filmed by a traffic camera at night. Plotted in subfigure (d).

2) Results and analysis: In this dataset, some videos such as *boats-6950-7900-clip.mp4*, the CA Kalman filter display worse results, as when it changes its direction, the prediction gets out of the image. Same happens with *abandonedBox-600-1000-clip.mp4*. That reason made us show CV results only. Following the same steps as in previous task, results of tracking with suggested parameters are shown below:

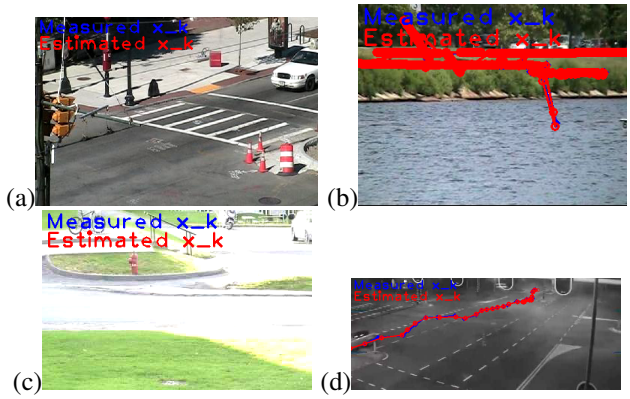


Fig. 5. Initial tracking trajectories in dataset 3

According to the obtained results, we though about why in (a) and (c) no results were obtained. Also, the tracking in (b) was sometimes miscalculated due to the wrongly predictions of the background cars.

Knowing that now we do not have restrictions according to parameter modifications, we made some tuning with:

- Size of blob (H,W): (**10**,50;**10**,50)
- Learning rate η: [-1, **1e-3**:1e-4:1e-2]
- Opening kernel size: [3,5,**7**,9]
- varThreshold: [**8**,16]
- history: [50,**500**]. The reason of using 500 was because it is the default built-in value in the function of foreground detection.

Best results have been obtained with the **bold digits**, and are shown in the following subplot:

Now we can see a quite good performance in our tracker in (a) and (c). Despite (d) has been badly affected, we still wanted to show it in report, proving that using these parameters, at least two trackings are optimized.

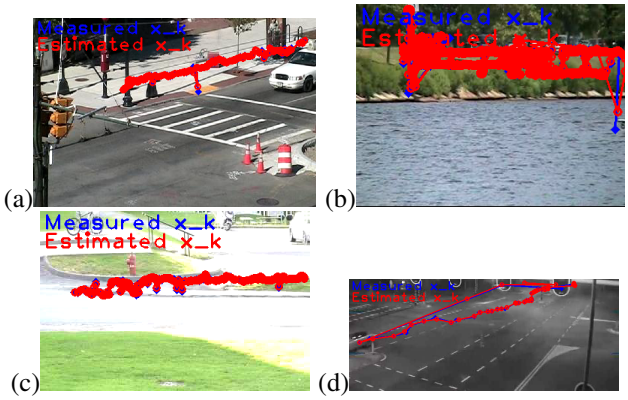


Fig. 6. Modified tracking trajectories in dataset 3

V. CONCLUSIONS

To sum up, both constant velocity and constant acceleration Kalman filters show a good performance. Nevertheless, it seems that the constant velocity one applied to real data works slightly better than the constant acceleration. As each video is different, neither the CV or CA should fit for all.

The size of the structuring element has developed an important role during the performance improvement. The higher it is, the more spikes in image are erased. However, it has the secondary effect of ”blunting” the image.

The minimun size of the blob in order to make the prediction has played a vithal role as well. If it is too big, there will be no detections in results.

VI. LOG

Task 3.1: Understand given example of Kalman filter in OpenCV. Adaptation to our needs. Solving code errors. Saving results. 10 hours.

Task 3.2: Apply concept to toy data. Understand results. Think about how to improve them. Solving code errors. 4 hours.

Task 3.3: Apply concept to real data. Fine tuning. Trying to get best results. Saving results. 15 hours.

Report: importing images. Write report itself. Make evaluation and explanations. 20 hours.

Contributions:

- Baglan: Half of the code, draft for beginning task 3.1. Writing conclusions.
- Alberto: Correction of draft errors, setting correct values to A_k, H_k, Q_k, R_k and P_0 matrixes for both CV and CA Kalman filters. Implementation of task 3.2 and 3.3. Fine tuning, write report and give format.

REFERENCES

[1] https://docs.opencv.org/3.4.1/d1/da2/kalman_8cpp-example.html
[2] https://docs.opencv.org/3.4.1/d2/de8/group__core__array.html#ga388d7575224a4a277ceb98ccaa327c99
[3] https://docs.opencv.org/3.4.1/dd/d6a/classcv_1_1KalmanFilter.html