Lab 3

Sebestyén Németh, Bendegúz H. Zováthi and Jaime Cortón González

I. INTRODUCTION

This laboratory report presents the implementation of an end-to-end single-object tracker based on background subtraction and Kalman filtering, written in C++ using the OpenCV library and specifically the Kalman-Filter class. The project consisted on the merging of 2 different parts, the initial background subtraction + blob detection method for extracting measurements (Task 1.a) and the final part of estimate and predict the object state by using a Kalman filter pipeline jointly with the initial part computed measurements (Task 1.b).

II. METHOD

This lab consisted on 3 different tasks. Task 1 consisted on the merging of Task 1.a and Task 1.b in a single pipeline, Task 2 consisted on the testing of different Kalman Filter parameters in irder to evaluate performance on toy data video sequences. Task 3 consisted on analyzing the whole Kalman filter pipeline parameters in a real world scenario in order to asses which parameters works better on certain scenarios.

A. ***Task 1:*** *Integration of individual works*

Initially each pipeline was design to work independently from each other. Task 1.b will take the measurement of each frame from a pre-computed txt file. This txt file was supposed to simulate the output of Task 1.b so integration the results of the first tasks as the input of the second one the pipeline works well on the the given video sequences. We made small changes in the code to display all the important information about the pipeline and there were some errors in the Kalman matrices which needed to be fixed.

B. ***Task 2:*** *Analysis of toy data*

At this task it was essential to analyze the outputs of the segmentation part. The opening morphological transformation is an erosion followed by dilation, which is useful in removing noise. Since we had too much false positive detection at video 5, we increased the morphology kernel size for the opening function, which results in less blobs with larger sizes. Despite the suggestions written in the slides we kept the original learning rate at the value 0.005, which leads to better performance because the algorithm was able to learn faster the background noise and decreased the ratio of the foreground segmentation. Apart from that modifications, the parameters related with task 1.a were maintained constant for the whole Task 2 performance comparison.

The comparative approach that we followed for the Kalman filter parameter analysis consisted on run the default parameters model for either the constant speed or the constant acceleration model and after the analysis of those results we started to modify parameters until obtained results are optimal.

Default settings for both models were the following:

1) `MEAS_NOISE_COV` (R matrix diagonal) = 25
2) `errorCovPost diag.` (P matrix diagonal) = 10^5

C. ***Task 3:*** *Analysis of real data*

In this task the tracked object changed on every video sequence, that's why we have to readjust not only the parameters of the Kalman filter, but also the opening, blob detection and shadow detection parameters on every video sequence.

The pipeline for the performance analysis consisted on having an initial guess over Task 1.a parameters and then just keep the default Kalman filter parameters. After that, by doing a visual analysis of the initial performance the parameters were optimized to improve results in tracking and in measurement computation. The most challenging part of this task was to adjust the parameters so that the measurements are placed were they are needed in a robust manner for the whole sequence, since tracking objects vary from boats, to bikes, or even cars. Also the illumination and shadows conditions vary a lot between sequences.

Task 1.a Parameters analyzed were the following

1) Shadow detection module (On / Off)
2) Minimum Blob size (width x height)
3) Opening kernel size
4) Learning rate
5) History size

III. IMPLEMENTATION

The implementation challenge of this assignment was to merge the solutions of the colleague working on the segmentation and blob detection and the others implementing the KALMAN tracking. For the final code, we used the blob detection code as baseline and moved the appropriate codes to this project. We created a `Kalman` class to store and run the tracking algorithm. In the constructor it initializes the filter, `Kalman::run()` is responsible for running `cv::KalmanFilter::correct()` after `cv::KalmanFilter::predict()`, and `Kalman::print()` displays the detection and tracking history. The class is implemented in `Kalman.h` and `Kalman.cpp` files.

IV. DATA

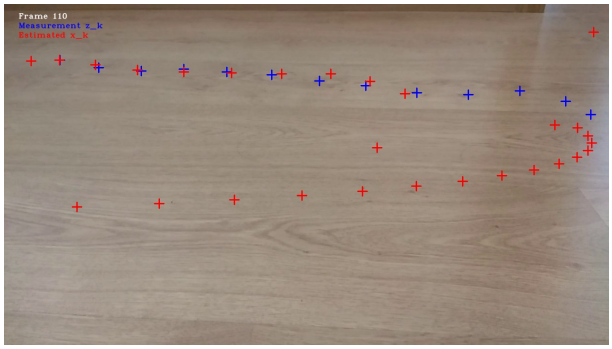
The data used this assignment composed by different data sets, split into 2 categories:

1) The first one is the toy data which were used at the Task 3.2, containing 4 simple video sequences about a movement of the ball. The main challenging scenarios were the occlusion and the bouncing of the ball.
2) The second one is the real data which were used at the Task 3.3, composed by 4 sequences captured on the streets. The main challenging scenarios were the occlusion, shadow, body-part detection because of motion and appearance changes.

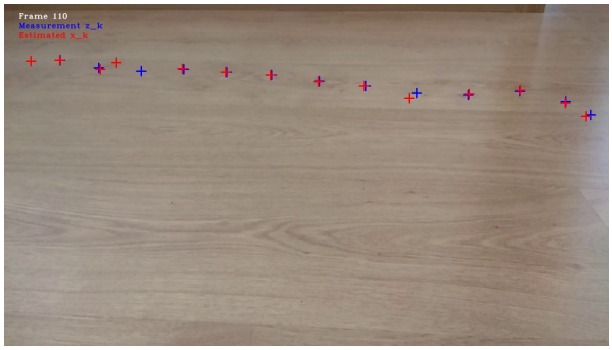
V. RESULTS AND ANALYSIS

A. *Task 2 results*

*1) **Video 2** (Straight line):* Default parameters usage, no big change since it follows a straight line without any distortion (constant velocity works better than the constant acceleration since the video follows a constant velocity motion and the acceleration consideration adds noise). We updated the default value `MEAS_NOISE_COV` = 25 to be equal to 5 which means that we trust more in the measurements. After analyzing the new results we could observe that there are less false estimations with modified parameters, as you can see at the Figure 1.



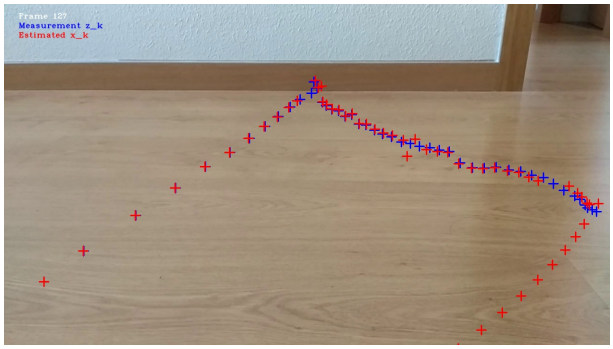
(a) Estimations and detection with default Kalman filter parameters



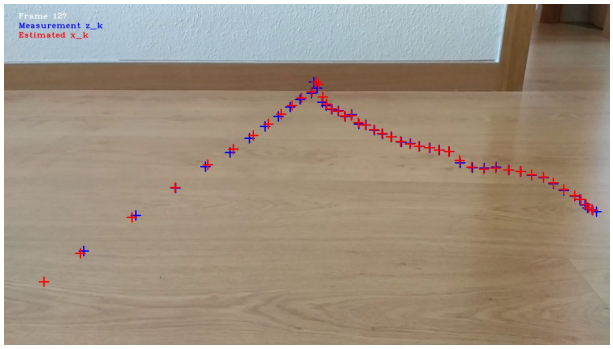
(b) Estimations and detection with `MEAS_NOISE_COV` = 5

Fig. 1: Constant acceleration Kalman filter model results at video sequence 2

*2) **Video 3** (1 bounce on the wall):* At constant velocity model the algorithm works well, but in case of constant acceleration the estimations were inaccurate so we changed the `MEAS_NOISE_COV` to be 5. After changing our parameters from defaults we got rid of the false estimated values after the ball left the scene, as you can see at the Figure 2. After many trials we could not find a good parameter settings for the constant acceleration model.



(a) Estimations and detection with default Kalman filter parameters



(b) Estimations and detection with `MEAS_NOISE_COV` = 5

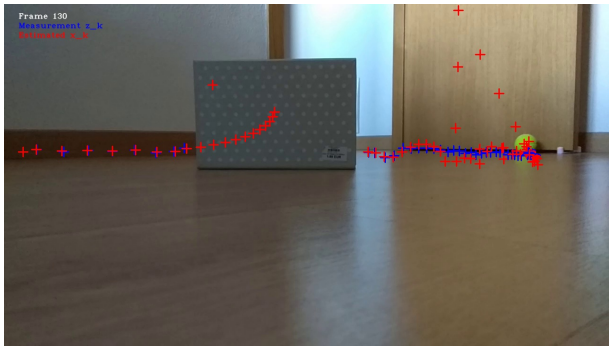
Fig. 2: Constant velocity Kalman filter model results at video sequence 3

*3) **Video 5** (Bouncing like a basketball (with the ground)):* At this video we discovered that the algorithm works better with the constant velocity model settings than with the constant acceleration model, if we are using the default parameters. Both of them are providing us promising results.

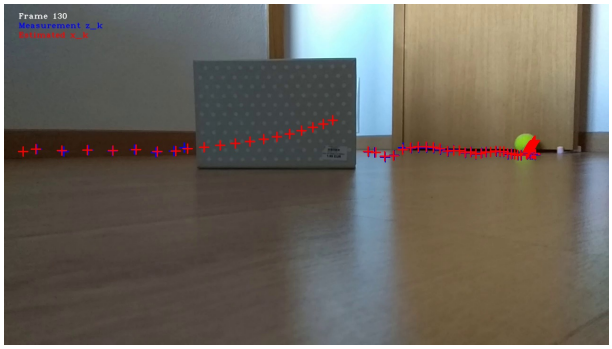
*4) **Video 6** (Occlusion and little door bounce):* This sequence was the most challenging one of these data because the ball was occluded and the ball was moving to the opposite direction at the same time. As we analyzed the result, although the Kalman filter works well with constant velocity model, in case of constant acceleration model it provides inaccurate results with the default parameters (Figure 3a). As we increased the `MEAS_NOISE_COV` value we got better results, but it was still calculating not real estimations as the ball stops. After we also changed the value of the `P_MAT_DIAG` correspondent to the x and y positions to 25 and the other diagonal values to 15 we achieved the our final results (Figure 3b).

B. *Task 3 results*

*1) **Boats**:* On this sequence the intuition lead us towards keeping a minimum blob size of 15x65 since the boat is a thin and tall object because of the sail. For the initial trial the learning rate was set to 0.01, the history size to 100, the opening kernel size to 5x5 and we turn on the shadow



(a) Estimations and detection with default Kalman filter parameters



(b) Estimations and detection with `MEAS_NOISE_COV` = 50, `ERROR_COV_POST` = 15 and `P_MAT_DIAG` = 10 except the x-y positonal values, which were modified to 15.

Fig. 3: Constant acceleration Kalman filter model results at video sequence 6

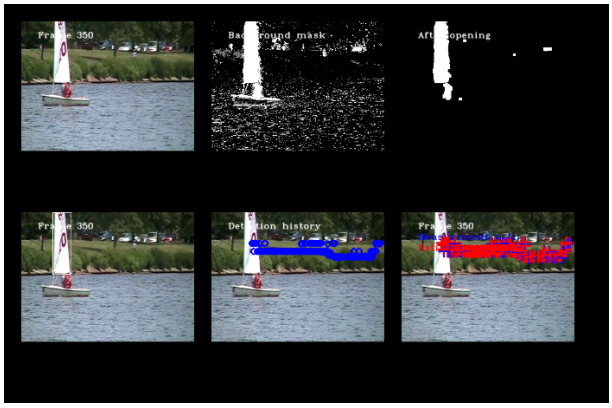


Fig. 4: Initial trial for boat sequence (Constant speed model.)

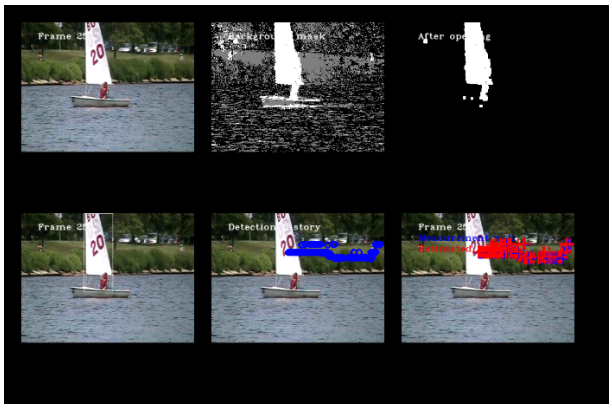


Fig. 5: Second trial for boat sequence (Constant acceleration model.)

detection. Model choosen was the constant speed model.You can see the result on Figure 4. We can see that since the tracker was placed on the blob center and the center of that blob was constantly changing that overall result is noisy. Despite that, the trajectory can be assessed in general terms by looking at the image

Looking at the initial results it can be seen that the problem stands with the obtained measurements. For the second trial (You can see it on Figure 5) chosen model was constant acceleration model.

*2) **Pedestrians**:* On the pedestrian video, we experimented with the different parametrization of the MOG detector and the constant acceleration Kalman filter.

As the initial trial, we used the following parameters for the blob extraction:

- Histrory: 100
- Learning rate: 0.01
- Kernel size: 3x3
- Min blob size: 15x50px
- Variance threshold: 16

The Kalman filter used the default parameters with a constant acceleration model. The result of the tracking can be seen on Figure 6. While the detection results seem reasonable, we can notice that the blob center is oscillating between two paths. This is because the legs of the person are many times not segmented properly. To tackle this, we changed the parameters and ran a second trial, with the following changes:

- Turned off shadow detection
- Increased the kernel size to 5x5
- Lower VarThreshold to 8

With these changes, we obtain a significantly clearer path for the detection, therefore for the tracking as well. However, we have some false detections for the shadow of the person. Results can be consulted on Figure 7.

*3) **Abandoned box**:* We tried out our algorithm with the default parameters, which raises inaccurate segmentation results. In order to improve the performance of the segmentation, especially to get bigger foreground mask and detect it longer when the cyclist is not moving to avoid false estimations (Figure 8a), me made the following changes:

- Turned off shadow detection
- Increased the history to 100
- Changed the morphological kernel size to 3x3
- Changed the minimum blob size to be 30x30 pixels large
- Reduced the learning rate to 0.0001

This segmentation algorithm with the defaults Kalman filter results feasible estimations (Figure 8b).

*4) **Street corner at night**:* Although the results what we got with the default parameters of the segmentation

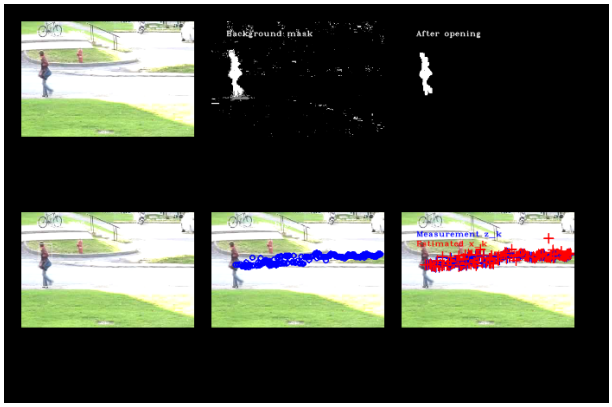


Fig. 6: Initial trial for the pedestrian video sequence

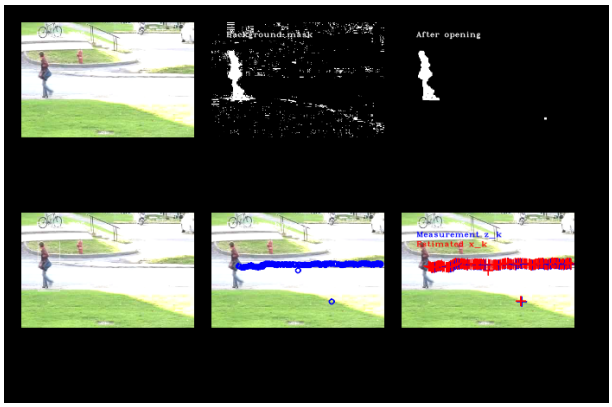
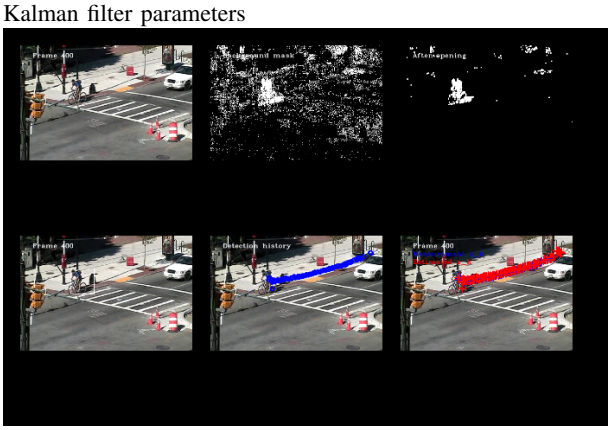


Fig. 7: Second trial for the pedestrian video sequence

(a) Estimations and detection with default segmentation and Kalman filter parameters



(b) Estimations and detection with modified segmentation parameters and default Kalman filter parameters

Fig. 8: Constant acceleration Kalman filter model results at the **Abandoned box** video sequence

and the Kalman filter were quite premising (Figure 9a, we made some changes in order to improve the performance. As we discovered, the inaccurate results came from the foreground segmentation of the lights. We doubled the value of the threshold, which narrows the size of the blobs. With this modification the algorithm performs slightly better with default Kalman filter parameters, as you can see at the Figure 9.

VI. CONCLUSIONS

As the conclusion of our work we would like to emphasize the significance of data pre-processing in order to get accurate estimations from the Kalman filtering algorithm. While in case of toy data the algorithms work well with initial segmentation parameters, at real world scenarios (as we discovered in the Lab 3.3) the problem became more difficult. Once we faced challenges like illumination changes, partial occlusion, shadows, motion and appearance changes we needed to change the segmentation parameters according to the image conditions. After we were satisfied with the visual results of the foreground masks we obtained better results even on the challenging sequences. The Kalman filtering algorithm is a powerful tool to estimate the positions of the objects but it is highly dependant on the quality of the measurements in which the algorithm bases on their tracking estimations. Bad measurements can get non-sense tracking predictions even if we are giving low confidence on measurement values.

VII. TIME LOG

We have worked on this project on a equally distributed time schedule of about 16 hours / person including the labs and the programming tasks.

Studying 2 hours, including the research about the topic and the lecture presentation.

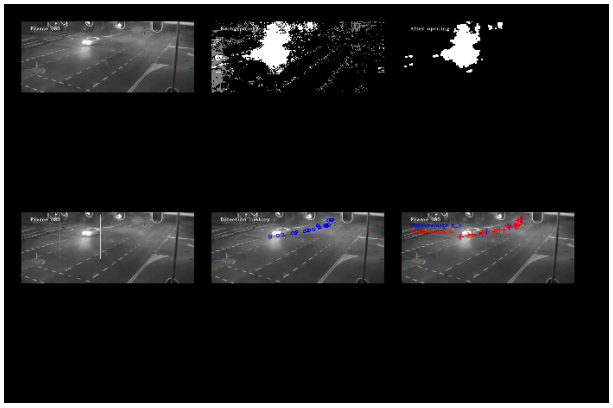
Coding 2 hours, constructing the code which bounds the Lab 3.a and b parts, integration the individual works.

Writing the report and testing the parameters 12 hours, testing the code with different parameters and discussing the conclusions about the different settings at the same time.

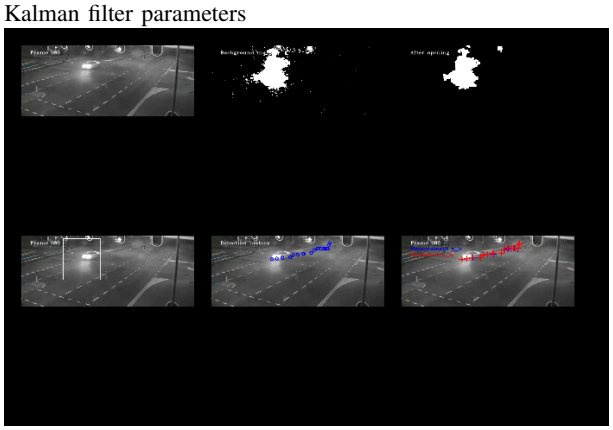
VIII. REFLECTION ON THE COLLABORATION AND INDIVIDUAL WORK

The main challenge being a group of 3 students (2 for task 1.b and 1 for task 1.a) is that we had the extra difficulty of not only dealing with the merging of both tasks developed by different students but also the conflict of selecting the pieces of code that work better on the task 1.b.

It was really instructive to be 3 students involved in ”pair programming”, there were no guidelines about it but we



(a) Estimations and detection with default segmentation and Kalman filter parameters



(b) Estimations and detection with modified segmentation parameters and default Kalman filter parameters

Fig. 9: Constant acceleration Kalman filter model results at the **Street corner at night** video sequence

managed to create a pipeline for that. Every 45 minutes we rotated over 3 positions:

1) **Coding student** This student will be in charge of the main computer, coding and running the tasks of the lab.
2) **Supervision student** This student will be in charge of looking at the coding student code, thinking in other ways of doing things for its improvement or just detect solutions for problems that might appear.
3) **Document/Internet/Bibliography checking student** This student will be going through external documentation (using a secondary laptop) for the problematic questions answering. From error on the compile to tracker optimization or class slides.