Target Tracking in Video Sequences

Eduardo Daniel Bravo Solis, Santiago Herrero Melo

I. INTRODUCTION

The main goal of this assignment is to implement some of the different tracking filters studied during the theoretical lessons. We will develop the two main tracking filters studied, which are the Kalman filter and the Meanshift filter.
Differently to the previous assignment, this time we will face the challenge of building the entire algorithm from scratch, by only using the knowledge acquired during the lessons and the papers corresponding to those filters.

II. METHOD

Contrary to the cases of the previous laboratory assignments, this time no code template was provided, and we had to develop the full code from scratch.

A. Kalman Filter Tracker

In order to understand and develop the Kalman filter, we have studied [1].
Kalman filter is a recursive tracking algorithm based on inference. This means that we have two different measures, the prediction and the measurement. These two values are Gaussian functions with different mean and variation values, being the variation higher when we are more uncertain about the measurement/prediction. The final goal is, depending on how much we trust both measurement and prediction, combine these two functions to find the real state of the element or particle that we are tracking. The variance of the final combination between the measurement and the prediction should be smaller than the variance of the two functions that we are combining.
The prediction is calculated considering the previous state and assuming a constant velocity model, this is, that the tracked element doesn't vary its velocity during the tracking. As we are working with a constant velocity model, the only variables to be considered are the position of the element as coordinates in both x and y axes, and the speed of the element along these axes.

B. MeanShift Tracker

MeanShift is a different approach for tracking. Its base is to define a certain region of interest (window), that will be shifted in the direction of its centre of mass, till no further shiftings can be performed. We have developed our code following the steps proposed in [2].
In the first place, we have to select a target model and represent it by its Probability Density Function in the feature space. Then, a candidate will be selected, and also represented by its Probability Density Function. The goal of the algorithm is to maximize the similarity between both PDFs. In order to measure the similarity, we will calculate the Bhattacharyya coefficient.
For localiazing the target, we will start from the position of the model in the current frame, and search in the model's neighborhood in the consecutive frame. The following steps are to approximate and maximize the similarity function, for which we have to choose an appropriate kernel and calculate the weights in consequence.
The window of the candidate will move following the direction of the centre of mass shown by the PDF, and will stop shifting and pass to the following frame when the distance is below a predefined threshold (when it has converged).

C. CBWH Improvement

According to [3], this algorithm is an improvement from the Background-Weighted Histogram algorithm (BWH). The improvement has been achieved following two main and simple changes.
In the first place, the neighbourhood of the original target has been expanded, allowing to know more information about the background. It only modifies the target model, but not the candidate model.
Secondly, weights are calculated differently from the original MeanShift algorithm, as it also exploits the background information. Finally, if necessary, the background is updated for a more precise tracking.

III. IMPLEMENTATION

A. For Kalman Filter Tracker

First of all, we will initialize the Kalman filter variables, this is, all the matrices that play a role when performing the tracking. For this purpose, the **KalmanFilter** OpenCV class has been used. The matrix created are the state transition matrix (1), the measurement matrix (2), the process noise covariance matrix (3), the measurement noise covariance matrix (4) and the posteriori error estimate covariance matrix (5).

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2}$$

$$Q = \begin{pmatrix} 10^{-5} & 0 & 0 & 0 \\ 0 & 10^{-5} & 0 & 0 \\ 0 & 0 & 10^{-5} & 0 \\ 0 & 0 & 0 & 10^{-5} \end{pmatrix} \tag{3}$$

$$R = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix} \tag{4}$$

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

The following thing to do will be to substract the background of the original video sequence. To this purpose, we will create a background substractor with the *createBackgroundSubtractorMOG2* function. We will apply the Kalman filter on this foreground mask.

Once we have the foreground mask, we will use the *extractBlobs* function, based on the *Grass-Fire* algorithm, to determine where the ball is. As we are tracking a single ball, the code will return an error if more than one element are detected.

Now that the blob, which is the object to be tracked, is detected, we start to apply the Kalman filter. The first detection will be the initialization of the target state. The centre coordinates of the detected blob will be stored as the previous state coordinates, which will be used for the prediction. Then, those values will be called by the *predict* function to compute the prediction value using the matrix (1), following the equations (6) and (7).

$$x_k^- = A * x_{k-1} \tag{6}$$

$$P_k^- = A * P_k * A^T + Q \tag{7}$$

After this, the central x and y values of the ball will be assigned to the *mesurement* value. This value will be the input of the *correct*, which updates the prediction and the measurement, obtaining the final state value via the equations (8), (9) and (10)

$$K = P_k^- * H^T * (H * P_k^- * H^T + R)^{-1} \tag{8}$$

$$x_k = x_k^- + K * (z_k - H * x_k^-) \tag{9}$$

$$P_k = (I - K * H) * P_k^- \tag{10}$$

Where x_k represents a four element vector whose elements are the position and speed, in axes x and y, respectively, and x_k^- is that same vector in the previous state, and K is a value that relates the belief on the measurements with the belief on the predictions. The higher this value is, the more we trust the measurements.

Finally, we will visually check the performance of our algorithm. Three crosses will be painted in the frame, representing the measurement (red), the prediction (green) and the estimation (white).

B. For MeanShift Tracker

For this section, the use of the OpenCV functions **calcBackProject** and **menShift** was forbidden.

In the first place, we have to select the target model. The target and candidate models are represented by the equations (11) and (12).

$$\hat{q}_u = C \sum_{i=1}^{n} k(||x_i^*||^2)\delta[b(x_i^*) - u] \tag{11}$$

$$\hat{p}_u = C_h \sum_{i=1}^{n_h} k(||(y - x_i)/h||^2)\delta[b(x_i) - u] \tag{12}$$

In the case of our code, a .txt file was provided with the correct positions (groundtruth) of the object to track in every frame. The first line of this file corresponds to the initial position of this object, this is, the coordinates in this line define the target model that we will use all along the task. For this purpose, we developed the *Read_First_Line_Ground_Truth* function, which is located in the **helper_functions** file (the rest of the created functions are located in the **mean_shift** file). This value will be assigned to the *ROI* element, which is our region of interest. We will work with gryscale images.

Once the region of interest is defined, we have to calculate its Probability Density Function with the *Compute_PDF* function which calculates the histogram (with *myHist*) of the region of interest and normalizes it (*normalize* function). This process is applied to the region of interest only in the first frame of each video.

For the candidate models, a loop is created, that is only left if convergence is achieved. We create a matrix of the same size as the region of interest, in which the weights are stored, and compute them in the *Compute_Weights* function following the equation (13). Then, the next location of the target candidate is calculated using the equation (14) in the *Compute_New_Center* function. We calculate the euclidean distance between the new and old centres, and if this distance is below a predefined threshold, we consider that convergence has been achieved and stop computing, passing to the following frame. Otherwise, we repeat the procedure, being the region of interest the region centered in the new centre.

$$w_i = \sum_{u=1}^{m} \sqrt{\frac{\hat{q}_u}{\hat{p}_u(\hat{y}_0)}}\delta[b(x_i) - u] \tag{13}$$

$$\hat{y}_1 = \frac{\sum_{i=1}^{n_h} x_i w_i g(||(\hat{y}_0 - x_i)/h||^2)}{\sum_{i=1}^{n_h} w_i g(||(\hat{y}_0 - x_i)/h||^2)} \tag{14}$$

C. For CBWH Improvement

The code for this task was based on the code developed in section III-B adding minimal modifications that are consistent to what was explained in section II-C, so only the modifications will be explained in this section.

The first step is to create a bigger window for the target model, for which we will expand the original window one time the original width and height it had. This area that represents the difference between the original and the new window will be our background model.
We will work with the Probability Density Function of this background model, and compute it with the *Compute_Bkg_PDF* function, using the OpenCV *calcHist* and *normalize* functions.
Once this Probability Density Function has been calculated ($\hat{o}_u$), we will modify this histogram by giving a higher weight to the bin with smaller size in the original histogram. To do so, for every bin, we will take the value of the shortest one and divide it by the original value, obtaining a value of 1 in the case on the shortest bin, and below 1 in the rest, but maintaining the size relation between them. For computing this modified histogram (v_u), the *Get_V_Min_Value* function has been developed.
The following step is computing the PDF of the target candidate, using the already mentioned *Compute_PDF* function. The weights are computed following the same procedure as MeanShift, but now their final value depend on the value of the modified histogram, according to equation (16), and then the new location is computed.

$$w_i^{''} = \sqrt{v_{u'}} * w_i \tag{15}$$

The final aim is to achieve convergence. For doing so, there are two options. The first one is that the euclidean distance between original an new centre reaches a value below a determined threshold. On the other hand, we can force the convergence by determining a maximum number of possible iterations. When this convergence is achieved (or forced), the background PDF of the new region of interest is computed and therefore also the weighted background histogram. If the Bhattacharyy distance of the background PDF is below a predefined value ϵ, then the last computed histograms are updated to be the new $\hat{o}_u$ and v_u for the next frame.

$$w_i^{''} = \sum_{u=1}^{m} \sqrt{\hat{o}_u * \hat{o}'_u} \tag{16}$$

D. Running the code

The code is very simple to run. To use the finally set values for the different parameters, the user just needs to insert the right path of the video sequences he/she wants to use. In the case the user wants to modify the parameters, he/she has to be careful on how to proceed, as the use of the matrices in Kalman tracking is not so intuitive for users who ignore the functioning of the algorithm, and the modification of certain values in the MeanShift algorithms can lead to a very bad performance of the filter.

IV. DATA

There are two different datasets of videos in this assignment. One of them will be used for testing the Kalman tracker, and the other one for the Meanshift tracker.

A. Videos for Kalman Filter

The dataset for the Kalman filter consists in seven videos. The first video, *singleball* is used to validate how accurately our algorithm works. In this video, the ball passes behind a box, so for certain frames the ball is not seen, therefore there are no measurements of the position of the ball. The main task is to check how good the algorithm predicts the position of the ball, and how near it is from the ball when it appears again from behind the box.
The other videos of this dataset show a tennis ball that either disappears of the scene, behind some object or bounces off a wall.

B. Videos for Meanshift Tracking

This dataset is formed by six video sets, from which we have individual frames instead of the continuous video sequence. Additionally, for testing the performance, each set has a .txt file with the position of the centre of the object to be tracked in every frame.
In the *ball1* video set, there is a man kicking a ball, and the task is to follow the ball while it is being kicked. The camera is fixed.
In the *ball2* video set, someone throws a ball to a football goal, and the task is to follow the ball's trajectory. The camera is fixed.
In the *basketball* video set, we have to track an specific basketball player, who can be confused by some other players wearing the same suit. The camera moves along the court, and sometimes we lose sight of the player.
In the *bolt1* video set, we have to track a specific runner during a 100m race. The element to track is always in scene and the camera moves along the court following it.
In the *glove* video set, someone is holding a glove that we have to track and moving it around a chair with a similar color. Additionally, there is another glove in that chair in order to make the tracking more challenging. The glove to track is always in the scene and the camera is fixed.
In the *road* video set we have to track a motorcycle. It is always in the scene as the camera moves following it, and more or less keeps a constant angle of vision over it.

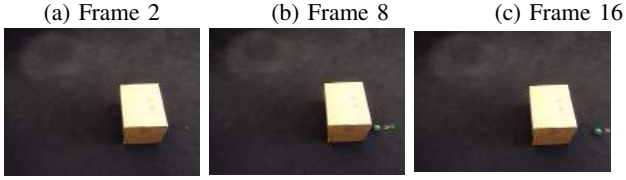
V. RESULTS AND ANALYSIS

A. Kalman Filter Tracking

As explained in the IV-A section, we will start to test our tracker with the matrices (1), (2), (3), (4) and (5) with the values that were settled in the III-A section. For following tests, the values of matrices (1) and (2) will always remain the same.

In the images in Figure 1 we can see the evolution of the prediction and the correction of the tracking. The red cross corresponds to the measurement given by the blob

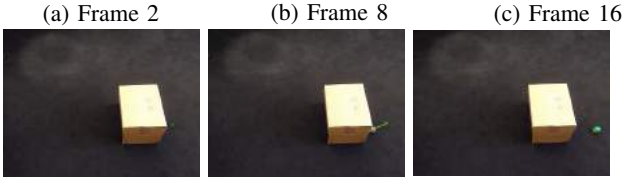
(a) Frame 2 (b) Frame 8 (c) Frame 16



(d) Frame 20 (e) Frame 23 (f) Frame 26

Fig. 1: First test of Kalman filter

(a) Frame 2 (b) Frame 8 (c) Frame 16



(d) Frame 20 (e) Frame 23 (f) Frame 26

Fig. 2: Second test of Kalman filter

extraction, the green cross is the prediction, and the white cross is the correction. Helpful lines have been also painted to check the "distances" between all three values.

We can see that the first prediction (Figure 1a) is very bad, but after very few frames (Figure 1b) it is very precise and almost matches the measurement. Once the ball disappears behind the box (Figure 1c), only the prediction is present. As we are working with constant velocity, we don't take into account how the ball slows down, so the prediction will go on too fast in a straight line trajectory (Figure 1d), finding itself far from the real measurement when the ball reappears (Figure 1e). At the end of the sequence, the prediction is corrected so that it is very close to the real measurement again (Figure 1f).

For example, we can check the good precision of the corrected state (white cross) in Figure 1a. Being only in the second frame, and having a very bad prediction (which is not even inside the frame), the corrected state is quite similar to the measurement, which is an indicator of good selected initial values in the matrices.

Now, we will modify the matrices (3), (4) and (5) setting the same values that are shown in the slide 35 of the lectures' slides. The new matrices are (17), (18) and (19).

$$Q = \begin{pmatrix} 25 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 10 \end{pmatrix} \tag{17}$$

$$R = \begin{pmatrix} 25 & 0 \\ 0 & 25 \end{pmatrix} \tag{18}$$

$$P = \begin{pmatrix} 25 & 0 & 0 & 0 \\ 0 & 10^5 & 0 & 0 \\ 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 10^5 \end{pmatrix} \tag{19}$$

In matrix (19) we are giving much more importance to the measurement than to the prediction. This leads to an improvement in performance, as we can see in the sequence in Figure 2, which shows the same frames as Figure 1.

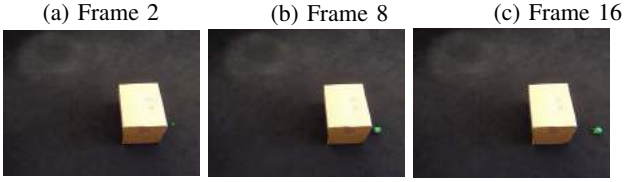
We can see that the initialization (Figure 2a) is very similar to the one of the previous case, but the matching of measurement, prediction and correction gets almost perfect after a few frames (Figure 2b). Also, in (Figure 2d) we can see how the cross is much nearer to the box than before, which means that the prediction is much better. Also, it takes less frames to match more precisely with the measurement (Figure 2f).

Now, we will modify matrix P giving more confidence to the speed and less to the position, obtaining matrix (20).

$$P = \begin{pmatrix} 10^5 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10^5 & 0 \\ 0 & 0 & 0 & 10 \end{pmatrix} \tag{20}$$

The results in Figure 3 show a much more precise prediction both at the beginning (Figure 3a) and after the ball reappears from behind the box (Figures 3e and 3f). Also, the very good accuracy between measurement, prediction and correction last for the rest of the sequence.

(a) Frame 2 (b) Frame 8 (c) Frame 16



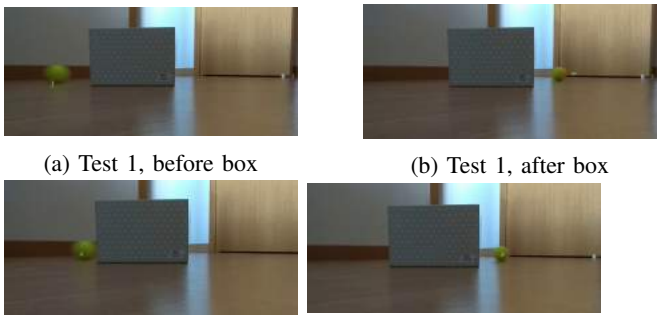
(d) Frame 20 (e) Frame 23 (f) Frame 26

Fig. 3: Third test of Kalman filter

Now that we have found a combination that seems to be quite precise, we will test with other video sequences the performance of the first and the third evaluations.

In *video6* there is a similar sequence than the one of the *singleball* as the ball disappears behind a box. The images

shown in Figure 4 show the differences on predicting the state of the ball before (Figures 4a and 4c) and after (Figures 4b and 4d) being hidden by the box. A detail to take into account in this video is that, at the end, when the ball stops on the door, it becomes background, so the blob is no longer detected, there is no measurement value, and the prediction understands that the ball has "trespassed" the door.

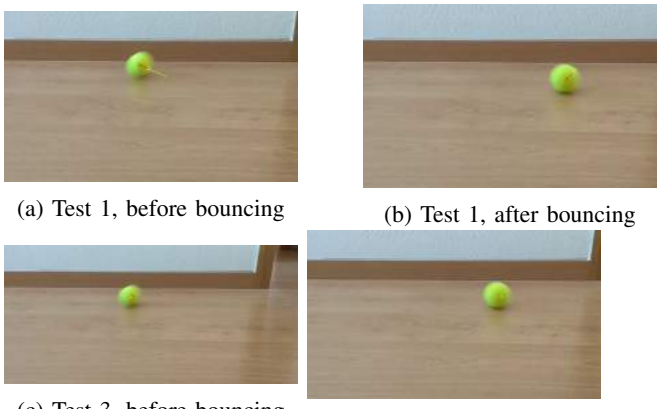


(a) Test 1, before box (b) Test 1, after box

(c) Test 3, before box (d) Test 3, after box

Fig. 4: Test of Kalman filter with *video6*

As we can see, the third test provides better results also in this case, being more precise when making predictions and being able to detect with a certain accuracy where the ball is several frames before the first test does.
In *video3* we find a sequence in which the ball just bounces off a wall, so the filter can be challenged because of this abrupt direction change. In Figure 5 we show the performance over this video with the first and third test settings. The performance before the ball bounces (Figures 5a and 5c) and after this bouncing (Figures 5b and 5d) are studied.



(a) Test 1, before bouncing (b) Test 1, after bouncing

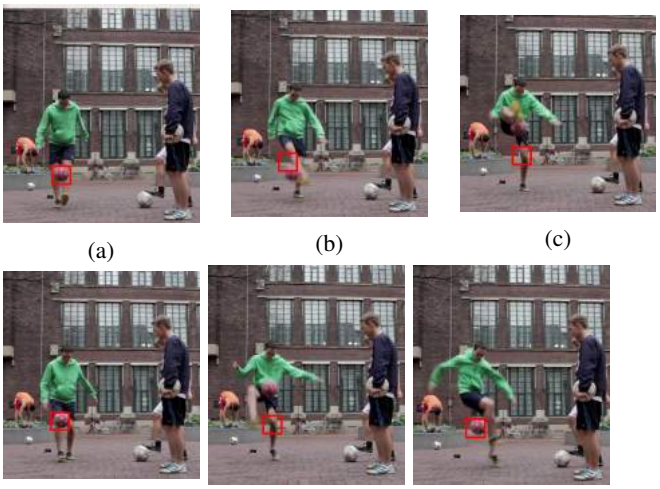
(c) Test 3, before bouncing (d) Test 3, after bouncing

Fig. 5: Test of Kalman filter with *video3*

In this sequence we are facing problems like bad detection of the centre of the ball, due to its high speed, and confusion by the filter when the ball goes out of the scene at the end of the sequence. We can see again how the third setting of parameters predicts the ball better and faster than the first set does.

B. MeanShift Tracking

The performance of the algorithm differs quite much from one video sequence to the other. The tracking performance is shown by a red rectangle, which shows where the object has been detected with the tracker. Different aspects of the performances for every video sequence are shown in figures 6, 7, 8, 9, 10 and 11. Both in this section and in section V-C, the user can also see a window with the iterations that the code is making for achieving the convergence and finding the new centre.
Figure 6 shows the performance in the *ball1* video sequence. We can see that the initialization is good in image 6a. There are several problems to track the ball due to its speed, especially when the ball is far from the original position (images 6c and 6e), but it relocates very accurately when the ball passes near the original position (images 6b, 6d and 6f). One explanation for this bad tracking may be that the algorithm gets confused with the PDF of the knee of the player, as can be seen in image 6c, thinking wrongly that it has captured the ball.



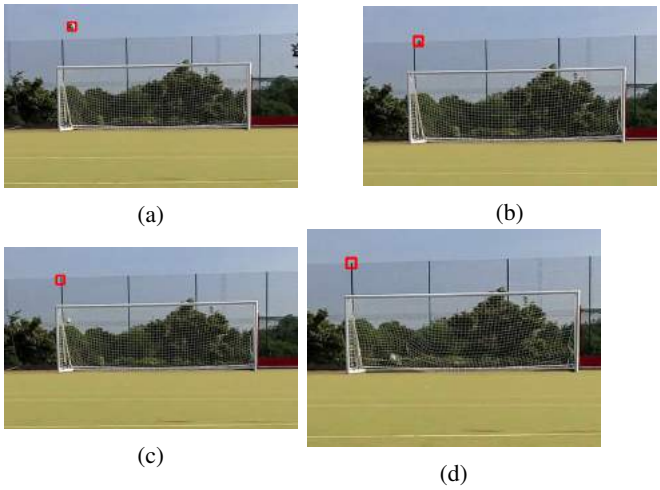
(a) (b) (c)

(d) (e) (f)

Fig. 6: MeanShift tracker on *ball1* video

Figure 7 shows the performance in the *ball2* video sequence. In the image 7a we can see that the initialization is good, and the tracker follows the ball while it flies through the air (image 7b). The problem arrives when the ball reaches the height of the back net. The tracker gets confuse by the post and detects it as the ball, so stays in that position for the rest of the video sequence, being unable to follow where the ball really is (balls 7c and 7d).

Figure 8 shows the performance in the *basketball* video sequence. The initialization is good (image 8a) and follows the player during the camera panning (image 8b). Afterwards, the box will be stuck in the background publicity, and only recover tracking a player when that player invades the space of the box, even in the cases in which the players wear different equipment (image 8c).



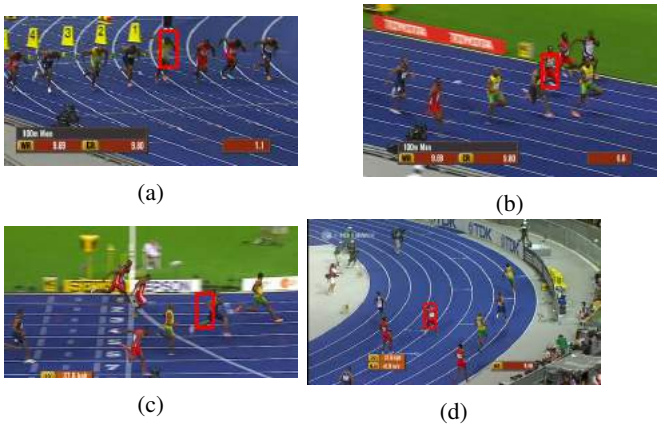
(a) (b) (c) (d)

Fig. 7: MeanShift tracker on *ball2* video

(a) (b) (c)

Fig. 8: MeanShift tracker on *basketball* video

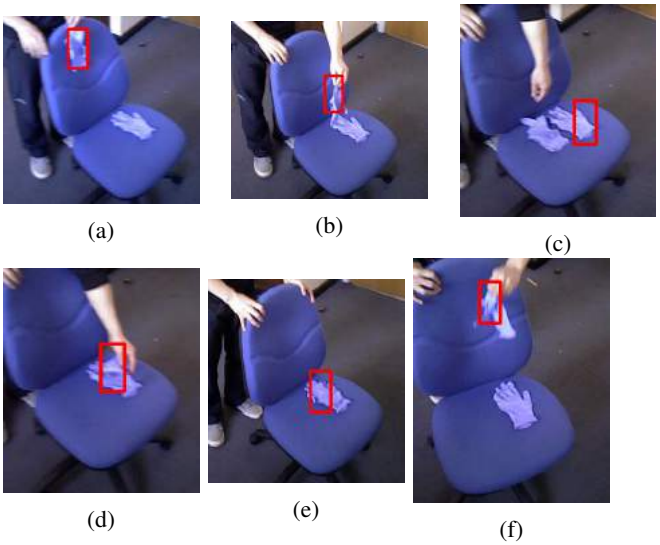
Figure 9 shows the performance in the *bolt1* video sequence. In this sequence, the initialization of the runner is good (image 9b), and the tracker follows it very well till the point in which our target overpasses the runner in red (image 9c). Then the tracker stays with that runner, and keeps on changing to runners that move close to the one that has already selected (image 9d).



(a) (b) (c) (d)

Fig. 9: MeanShift tracker on *bolt1* video

Figure 10 shows the performance in the *glove* video sequence. As shown in image 10a, initialization is good. The tracker follows the glove (image 10b), but as it passes very near another glove, the tracker stays with this second one (image 10c). The tracker will be maintained in the position of the glove in the chair (images 10d and 10e), and when one of the gloves is picked up again, the tracker will follow it (image 10f). From all the video sequences, this is the one that has worked better with this algorithm.



(a) (b) (c) (d) (e) (f)

Fig. 10: MeanShift tracker on *glove* video

Figure 11 shows the performance in the *road* video sequence. The initialization is good (image 11a) and the box tracks the motorbike even after passing behind some tree that covers it completely (image 11b). The sequence stops prematurely because of another bigger tree, which challenges our tracker to follow the motorbike, and makes the box disappear from the image (image 11c).

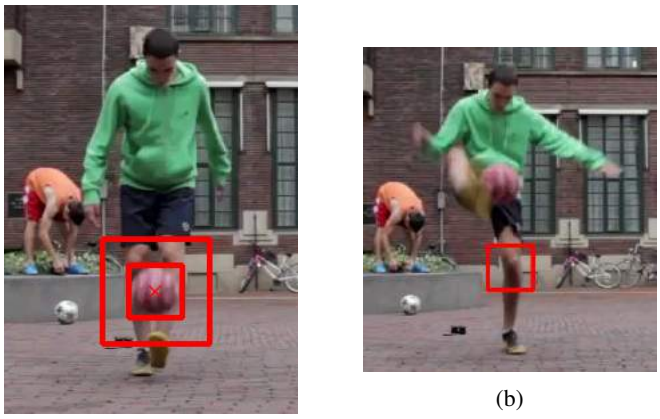
(a) (b) (c)

Fig. 11: MeanShift tracker on *road* video

C. CBWH Improvement

The goal of this evaluation is to measure the difference in performance between this method and the MeanShift filter. The tracking performance is shown by a red rectangle, which shows where the object has been detected with the tracker. Different aspects of the performances for every video sequence are shown in figures 12, 13, 14, 15, 16 and 17.

In image 12a we show the initialization with both the old and the expanded target model. We can see that the performance is still deficient when it detects the knee, forgetting about the ball (image 12b). At some point that the ball moves near the window it tries to recapture but the tracker gets lost again in different textures (images 12c and 12d).

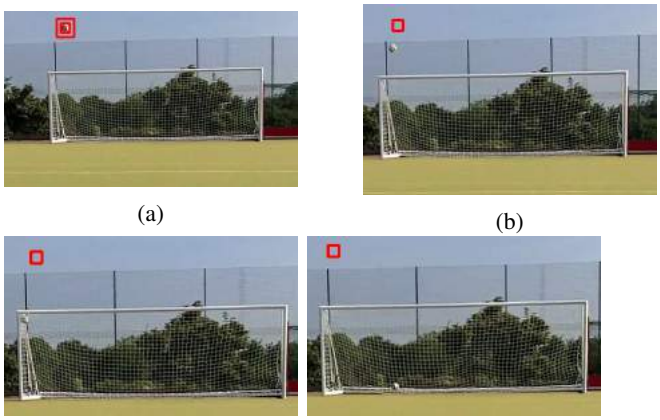


(a) (b)

(c) (d)

Fig. 12: CBHW tracker on *ball1* video

In image 13a we show the initialization with both the old and the expanded target model. The code is unable to converge by itself, so the forced convergence makes the tracker lose the ball very quickly (image 13b), and stays like that for the rest of the sequence (image 13c and 13d).

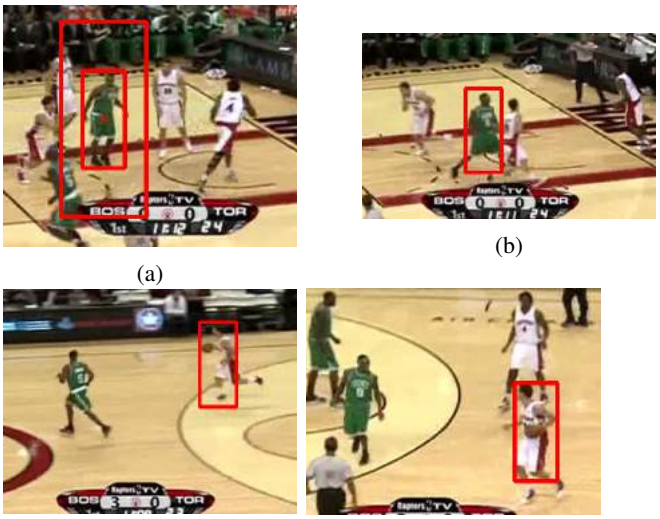


(a) (b)

(c) (d)

Fig. 13: CBHW tracker on *ball2* video

In image 14a we show the initialization with both the old and the expanded target model. The code works a bit better than the MeanShift in the sense that it follows the target for more frames (image 14b), but it misses it when another player crosses in front of the target and, after a few frames of doubt, the tracker picks this second player (image 14c) and stays with it for the whole sequence (image 14d).

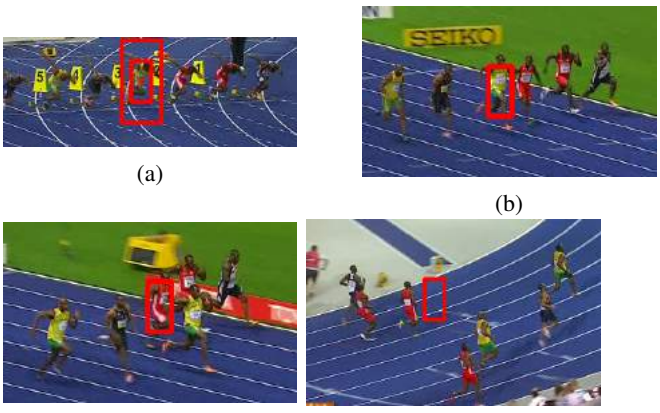


(a) (b)

(c) (d)

Fig. 14: CBHW tracker on *basketball* video

In image 15a we show the initialization with both the old and the expanded target model. The tracker follows the right runner for several frames (image 15b), but at a point in which the runners pass in front of a yellow element, the tracker loses the tracking object and returns with a red runner (image 15c). In the end, the tracker is unable to reach again the original tracking element, and even loses the new runner (image 15d).



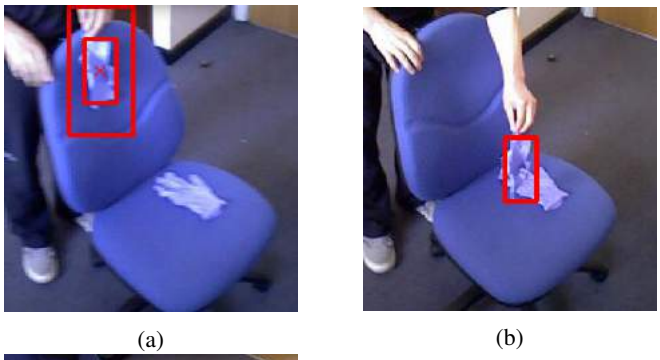
(a) (b)

(c) (d)

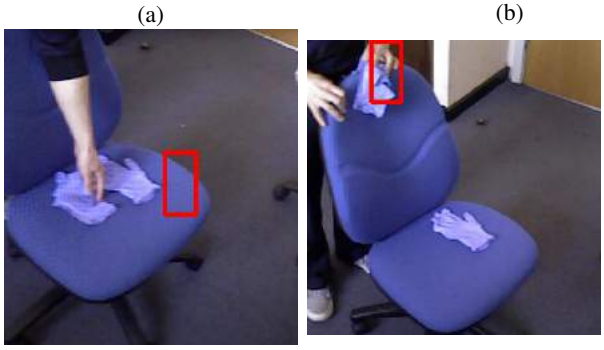
Fig. 15: CBHW tracker on *bolt1* video

In image 16a we show the initialization with both the old and the expanded target model. The tracker follows the glove (image 16b) till the moment in which it is left on the chair. Then it loses the track of the gloves around the chair (image 16c), but in the end it is able to recapture the glove again (image 16d). This is the sequence in which our

implementation works best.



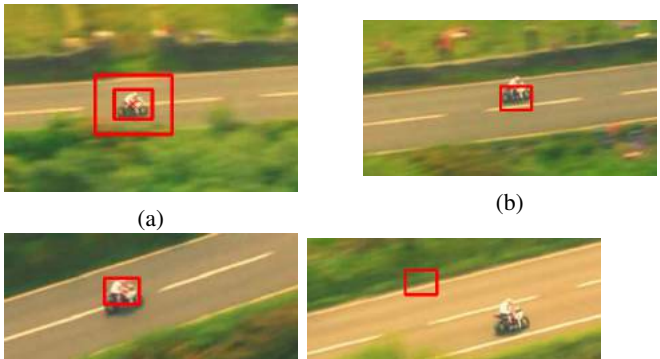
(a) (b)



(c) (d)

Fig. 16: CBHW tracker on *glove* video

In image 17a we show the initialization with both the old and the expanded target model. The tracker follows the motorbike pretty well (image 17b), even after it is occluded behindthe tree that made our algorithm fail in the MeanShift task (image 17c). The main issue is that afterwards, the motorbike is occluded by an even bigger tree, so the tracker loses the object completely (image 17d) and makes the sequence finish abruptly. This is the sequence in which our implementation shows a higher improvement, even if it only lasts for a few frames.



(a) (b)

(c) (d)

Fig. 17: CBHW tracker on *road* video

VI. CONCLUSIONS

In Kalman filter tracking, the constant velocity approach can lead to errors as the velocity of the balls in the videos is not constant, but increases or decreases with time. Only by highly taking into account the speed of the ball, the prediction can be more precise, but as the acceleration is not taken into account, there is always room to certain error. Also, tracking objects that remain still (as we said for *video6*) become part of the background so a learning rate for the background substraction should be diminished. It is advisable to train the model for several frames before making a prediction without correction.

In MeanShift the main challenge we faced were objects with similar Probability Density Functions, which made our tracker to be confused and stay closer to the position of the tracked object in the previous frame, as that centre of density was closer than the new one.

In the CBWH method, we haven't been able to improve significantly the MeanShift algorithm. One possibility to solve this issues may have been using a higher number of iterations to force convergence, but we already checked that the center was not improving and stuck in a constant distance, which means that "more patience" wouldn't have meant a better result.

VII. TIME LOG

A. Kalman Filter Tracking

Read the paper and understand the values of the input matrix, took us around two hours. The implementation and testing were two extra hours. Latter modifications were added in the second review of code and required around one hour.

B. MeanShift Tracking

For the second section of the assignment, the most time consuming task was the ground truth text file interpreter. It took us close to three hours to make a robust version. Reading the documentation and creating the first scratch of the code took us close to four hours. Debug and code logic modifications required three more hours. Testing with several videos to add robustness in the code took us one extra hour.

Ṫesting with several subsectionhoursCBWH Improvement Reading the documentation and comprehension of the algorithm required close to two hours. Basing CBHW implementation on the previously developed Meanshift code, the required modifications took us around three hours.

C. Documentation and Quality Control

The creation of this report and all the needed tests for getting the images shown in it, took us around two and a half hours.

REFERENCES

[1] R.E. Kalman; (1960). "A New Approach to Linear Filtering and Prediction Problems", Research Institute for Advanced Study, Baltimore, Md; Transactions of the ASME - Journal of Basic Engineering (Series D): 35-45.

[2] D. Comaniciu; V. Ramesh; P. Meer (2003). "Kernel-Based Object Tracking", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 25, no 5.

[3] J. Ning; L. Zhang; D. Zhang; C. Wu; (2010). "Robust Mean Shift Tracking with Corrected Background-Weighted Histogram", IET Computer Vision.