Thermal Tracking of Sports Players

- Vision, Graphics and Interactive Systems -

Project Report

Péter Rohoska, Louise Abela, Cedric Magnan

March 2017

Aalborg University
Electronics and IT

Contents

Chapter 1

Introduction

In the following work the authors will introduce you to the re-implementation of the project done by Gade & Moeslund [8], with a motivation to utilize a tracking algorithm based on Kalman filters and the CEM tracker to create trajectory maps based on the movement of humans during sport activities, captured by a thermal camera. It is hypothesized that the trajectory maps created by our algorithm and the produced results (discussed in the related chapter), will be similar to the results of the work of Gade & Moeslund. To be able to match the results, the testing data and the implementation details have been provided by the original authors.

The report is divided as follows: In Chapter 2 the background technical information required for the sensor used in this project (namely a thermal camera) can be found, in Chapter 3 the re-implementation of the trackers along with detailed description of the utilized image processing methods are presented. Finally the results are shown, reflected and discussed on in Chapters 4 and 5.

Chapter 2

Technical Background

In this chapter the technical background information required for this report is explained.

2.1 Thermal Camera 2.1

Every object that has a temperature above absolute zero emits electromagnetic radiation. Most of this thermal radiation emitted by objects near room temperature is in the IR domain. Cameras used to detect the temperature of a surface detect photons with wavelength in the range of 9 µm to 14 µm (illustrated on Figure 2.2). This type of camera, unlike others, do not need external light sources, as the objects/surfaces are the sources themselves.

2.1.1 Common Uses and Advantages

These kind of cameras have several uses in everyday work and life. Such as surveillance, law enforcement and maintenance [9].

There are several advantages with using Thermal Cameras over other cameras. The following are some of which:

- **Data Protection** - As these cameras just records the temperatures and movements of a person not an actual picture. Therefore these cameras are widely utilized at public locations such as airports, without the problem of data protection.

Figure 2.1: Thermal Camera Example

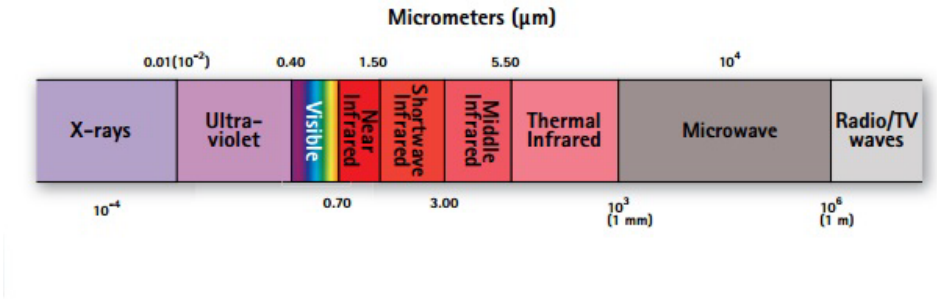

Figure 2.2: The figure represents the Electro Magnetic Spectrum, separated to nine different domains based on wavelength intervals. From the shown wavelengths, only a small interval is visible to the human eye, however the rest can be captured by different sensors, such as a thermal camera. [4]

- **Day and Night Usage** - Due to these kind of cameras work by heat there is no problem to work during the night as light conditions doesn't affect the image of these cameras.
- **Occlusion Proof (to a certain extent)** - As these cameras detects heat if there's a person behind a tree or behind smoke these can still be detected. Therefore it also makes it difficult to hide from, and would increase the security.
- **Weather Conditions** - Works under practically all weather conditions
- **Reflection and Shadow Proof** - thermal light rays do not create shadows or reflections, therefore these elements can not be captured by segmentation and background subtraction.

Chapter 3

Implementation

In this chapter the re-implementation process of the above mention paper is going to be discussed. Due to the paper using two trackers this section is divided into two sections.

3.1 Kalman Tracker

In the following section the authors introduce you to the implementation of the tracking algorithm based on Kalman Filters. The relevant image processing theory and methods are presented with implementation details in C++ and OpenCV.

3.1.1 Image and Background Acquisition

The computer stores the image from the camera into a 2D matrix, every coordinate representing a pixel value. The values of the pixel's variables, such as RGB or grayscale values, are stored. The amount of bits necessary depends on the image size and format. As humans move around, and the scenery stands still, it is possible to store images to be used as reference images. The program can use these images to distinguish elements in the scene that maintain their position from frame to frame. It does so by comparing a reference image (which can be updated in real-time) with the newly captured image, and comparing every pixel value in the new image with the average pixel values of the previous ones. Cumulative moving average can be used to calculate an average background image based on the last n frames $(x_1, ..., x_n)$

$$CMA_n = \frac{(x_1 + ... + x_n)}{n} \tag{3.1}$$

Calculating the cumulative moving average is memory efficient and can easily be implemented to acquire a well estimated background based on the n previous frames.

Implementation

- **getBG(int skip = 0)** is the method that starts the background accumulation process. The argument skip decides how many frames the program should skip from the camera. The *vector<Mat> backs* stores the previous frames. After every 150th frame the calculates the cumulative moving average of the image. This resulting matrix is converted to 8-bit by the *convertScaleAbs(Mat in, Mat out)* function.
- **returnBG()** returns the Matrix object containing the background model.
- **getNextFrame(int skip=0)** returns the Matrix object containing the next frame.
- **subtractBG(Mat frame)** returns the Matrix object that contains the difference of the argument frame and the background model.
- **getDiff(Mat frame)** returns the Matrix object that contains the difference of the argument frame and the background model.

3.1.2 Binarization and Morphology

Thresholding is a method to remove noise and to extract elements from an image scene. The method uses a threshold value between 0-255, which sets all pixels above the value to 255 and everything below to 0, thereby producing a binary image of black and white pixels. Since the image is acquired through a thermal camera, it is easy to differentiate the foreground pixel intensity, since it tends to be more hot compared to the background 3.1.

Further improvements to the image can be achieved by applying morphology in order to remove excess noise from each frame and connect binary objects on the frame. When thresholding the image in order to remove noise and create a binary image, the binary image created still contains noise that is unwanted. To remove the noise, several morphological methods can be utilized that can remove noise while maintaining everything else. There are three different morphological operations that can be executed; hit/fit, dilation/erosion and closing/opening. Another

Figure 3.1: On thermal video footage (left) human silhouettes are easy to differentiate since the human body tends to be warmer than the background. Therefore, to produce a binarized image (right) a simple thresholding is applied.

use of morphological operations are to connect separate BLOBs which we consider to be the same object.

To be able to suppress the amount of pixels our Grassfire algorithm has to go through, a Canny edge detector is run through the image. The edge detector finds the direction of gradient vectors in the image. Then each pixel is compared with its two neighbours in the gradient direction, to suppress two pixels with the smallest pixel value. In other words, the two closest pixels intensity is set to 0.

Implementation

Binarized grayscale images are stored in the *BinaryImage* class. This class is responsible for converting grayscale images into binary images. The constructor of the class, *BinaryImage(Mat im, int thresh)*, converts the image *im* into grayscale if it was not already. It then applies a threshold of *thresh*. After the binary image is obtained, it dilates the image by a 17 x 19 pixel kernel. A more vertical kernel was chosen as humans tend to be more vertical than horizontal. The image is then eroded by a 11 x 11 pixel kernel. As a final step a canny edge detector is run through the image to minimize the number of pixels the Grassfire algorithm has to process.

- **check(int x, int y)** returns true if the pixel (x,y) exists and it has a value of 255. Otherwise it returns false.
- **check(int x, int y, Direction d)** returns true if the pixel to the d of (x,y) exists and it has a value of 255. Otherwise it returns false.

3.1.3 BLOB Separation

When dealing with binary images with no depth a common problem is that, most of the time, when two BLOBs overlap, the result will merge into only one BLOB. Our goal is to reduce the number of merged blobs by splitting every merged blob into two when it is possible.

Height/width ratio is solely used to distinguish, which BLOB might need to be separated. The human width is often considered as one third of its height, so we chose a height/width ratio above 4 to select the blobs to be split vertically.

A problem remains with this method. When two blobs merge, the initial blobs might be above one another and slightly different on the horizontal position. In this case, the merged blob width will be higher than a merged blob from two blobs with no horizontal drift. Thus, the ratio will be below 4 and most probably similar to the average human ratio. In order to prevent that, we use the pixel ratio determined by the number of blob points inside the blob box. We still take a height/width ratio above 2 in order to avoid splitting human blobs with a diagonal setting and if the pixel ratio is below 0.5, the blob will also be split vertically.

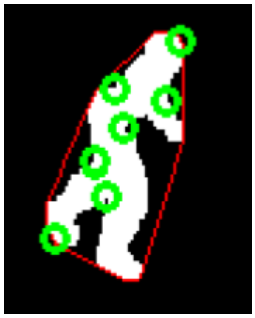

Figure 3.2: The point where the depth is the highest from all convexity defects points and above a certain amount is used in order to only produce a separation when it is significant. We compute the horizontal absolute gradient using the Sobel Filter so we use only convexity defect points coming from the sides of the blob.

We use the convexity defects of the blob to determine where to split the merged blob. We compute the contour of the blob and its hull 3.2. The convexity defect points will be where the contour differs the highest from the hull, at every difference between the contour and the hull. Once the point is found, we delete the white pixels on the same horizontal line as the point found until we reach black pixels and we use the grass fire algorithm on the resulting image to get the two new blobs. An example of vertically split BLOBs can be seen on Figure 3.3.

For the horizontal split, we only use the height/width ratio this time and take the merged blobs with a ratio below 1.5. Once it is chosen, the convexity defects are

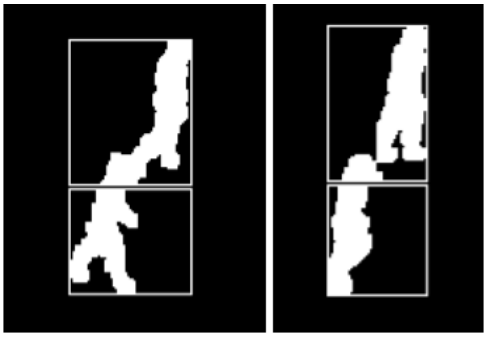

Figure 3.3: A demonstration on how vertically oversized blobs are separated.

also computed as well as the vertical absolute gradient for the same reasons as before except we decide to only take the defect which start point or end point is the highest from all start and end points of all defects. Since the head is always on top, there is at least one out of two start or end point which is located on the upper head of the blob

3.1.4 BLOB Extraction

Binary Large OBject, or BLOB extraction for short, is a method used to isolate and analyze objects in an image so that a program can distinguish between relevant and non-relevant objects such as humans and noise in our case. The procedure is divided into 2 steps called extraction and classification. The Grassfire algorithm is able to extract BLOBs by going through the picture pixel by pixel. When it encounters a pixel that is white, it gives it a number and then looks if there are any neighbouring pixels that are connected to it. Based on the connectivity of the kernel used, the algorithm checks in specific directions. The procedure is either used with a 4-pixel or 8-pixel connectivity, as illustrated in Figure 3.4.

The difference between the 2 kernels is how it detects if a pixel is belonging to the same blob. If the 4-pixel connectivity is used, it checks the pixels up, down, left and right of the current pixel. With the 8-pixel connectivity it also checks diagonally. This means that without a 8-pixel connectivity kernel, the program is unable to consider pixels that are diagonally to each other as part of the same blob, unless they are connected to another part of the blob vertically or horizontally elsewhere.

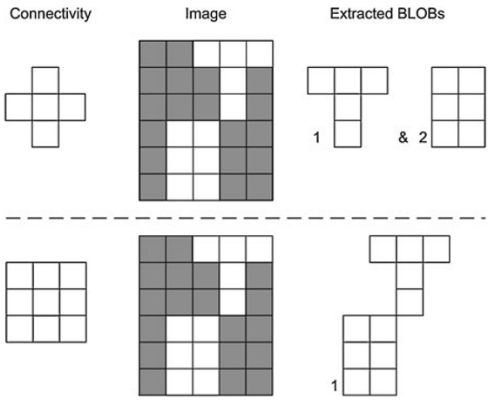

Figure 3.4: A demonstration on how a 4-pixel (top) and 8-pixel (bottom) connectivity is different, in respect to the amount of object it extracts with the Grassfire algorithm.[3]

Implementation

Before the Grassfire algorithm could tag the pixels, a blob class should be built in order to store the relevant information of the BLOBS. The following implementation stores three features and a list of pixels of the BLOB.

The center of mass of an object is a point given by the average of x- and y-values. That means all of pixel coordinate values contained by the individual BLOB are taken as an average.

The bounding box of a BLOB can be defined as the smallest rectangle which contains every pixel of the BLOB. The height and width values of the bounding box are given by the highest and lowest values of the BLOB respectively. The point *BBoxL* and *BBoxH* store one-one opposing corner of the bounding box of the BLOB. Having the two opposite corners of the bounding box makes it possible to obtain the width and height of the BLOB, just by subtracting the two points.

3.1.5 Grassfire

Since the Canny edge detector is utilized, as a result of the non-maximal suppression all the edges are 1 pixel wide. As a consequence, the Grassfire algorithm has to process fewer pixels, thus improving performance. A sequential Grassfire algorithm needs at least two passes and needs to keep track of flag equalities.

The *grassFireHandler* class contains the variables and methods required by the Grassfire. The constructor calls the reset method, which makes sure that variables are initialized. The four point variables are direction vectors.

- **setSeed(Point seed)** sets the seedpoint to seed. The seedpoint is the pixel where the Grassfire starts.
- **setImage(Mat* img)** sets the pointer to the argument *img*.
- **addPxToBl(BLOB blob, Point bp)** checks if pixel *bp* is already part of blob. If it is not, it add the pixel to the blob and add it to the list to be checked.
- **runFromSeed(bool burn = false)**checks if the pixel at the seedpoint has a value of 255. If it does, the *addPxToBl* method is called with a new BLOB object and the seedpoint as the second argument. If the burn argument was given true, it will set the value of the pixel to 0. Using the 8-connectivity kernel it checks if any of the eight neighbours of the just added pixel is white. As long as there is such a pixel, the cycle repeats.

3.1.6 Kalman filter and BLOB-Region of Interest Matching

The Kalman Filter can be used to reduce noise in data and to predict the state of an object in the image. It works on the recursive basis of the simple *predict* $\rightarrow$ *measure* $\rightarrow$ *correct* $\rightarrow$ *predict* model, shifting between predicting the next step from the previous state and correcting the state using a new observed measurement (a graphical explanation can be seen on Figure 3.5 and Figure **??**). The Kalman filter estimates the state x of a discrete-time controlled process controlled by the linear stochastic difference equation

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{3.2}$$

with a measurement z:

$$z_k = Hx_k + v_k \tag{3.3}$$

where w_k and v_k are random variables representing the process and measurement noise, respectively. The matrix A is the transition matrix that relates the state x at the previous time step *k-1* to the state at the current step k. The matrix B relates the control input u_{k-1} to the state x_k (the control input is optional, and this term is often discarded). The matrix H relates the state x_k to the measurement z_k.

For each new detected BLOB, a new Kalman filter assigned, in order to keep track of multiple objects at the same time. For each new frame a list of detections and predictions are created. Then each BLOB is assigned to the nearest detection, where the distance between the BLOB and the estimated state in the next frame is thresholded by a certain distance, in order to avoid matching new BLOBs. Our algorithm

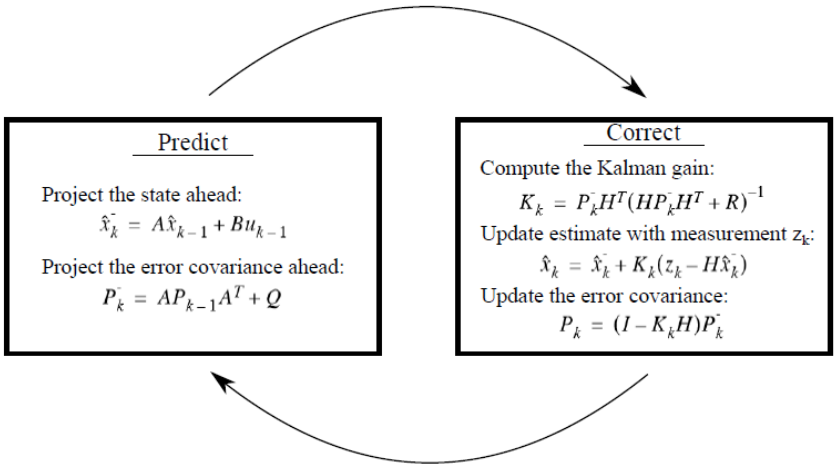

Figure 3.5: A demonstration of the recursive predict → measure → correct → predict model (from left to right) in each frame of the video.

applies region of interests (ROIs) to save the estimated state of each object to utilize the Kalman filter (a graphical representation is shown on Figure **??**). Detected BLOLBs and ROIs are matched based on the distance between the BLOB's center of mass and the ROI's center. The list mentioned previously contains priorities based on the length of the distance between ROIs and BLOBs. Once the priority lists are constructed, the program should check if there are any conflicting first priorities. If there are, the conflicts should be resolved, so that the average distance of the choices are minimized. The program should not start drawing the tracks immediately, it should make sure that the blob has been present in the image for a specific amount of time. This measurement is needed in order to lower the chances of displaying misinterpret data. For each detection that is not assigned to a Kalman filter, a new track is started, by creating a new Kalman filter. Kalman filters that have no assigned detections will be continued based on the predicted new positions. Once no conflicts are present in the priority lists, the program should start to draw and start the algorithm again.

Implementation

The program uses region of interests to estimate the area where a blob might be in the next frame. Once a blob has been detected, it is to be tracked, so there is a need to keep the BLOBs in memory for later identification. A class has been developed just for this reason.

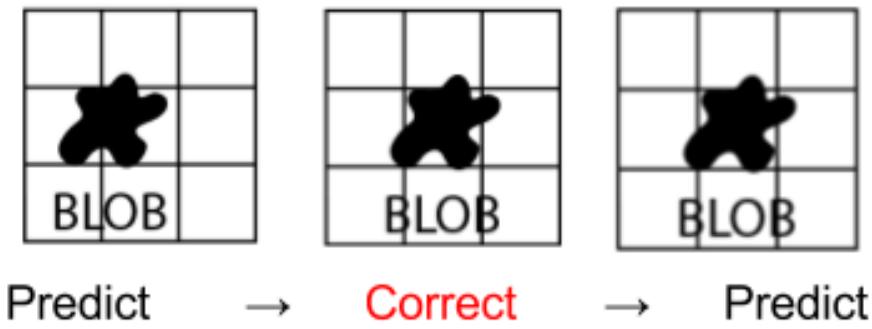

Figure 3.6: The ROI (wireframe) in each state is matched with a BLOB based on the priority list.The state of the ROI is update during the correction phase (middle) to set up a better estimate for the position of the tracked BLOB.

- **vector<vector<ROI> > mem** is a two dimensional vector which stores the estimated ROIs for each past BLOB.
- **vector<int> skips** keeps track of how long the BLOB has been out of the image.
- **vector<vector<Point> > prev** collects the path that the program has to draw.
- **vector<vector<bool> > isDrawn** stores information about whether a point in *prev* has been drawn already.
- **vector<kf> kf** a list of Kalman Filters, one for each BLOB.
- **vector<Scalar> color** a list of colors for the different BLOBs.
- **addMem(Point pt, Point errorp = Point(10, 10), Point errorn = Point(10, 10))** adds a new ROI at point *pt*.
- **getMem(int id)** returns the ROI in question.
- **removeMem(int id)** Removes the ROI in question from the memory.
- **addToMem(Point pt, int id, Point errorp = Point(10, 10), Point errorn = Point(10, 10))** Updates the ROI's Kalman Filter and the ROI position.
- **disappear(int id)** Sets BLOB as disappeared.
- **reappear(int id)** Sets a BLOB as detected in the picture.
- **cleanUp(int skip)** Remove ROIs from the memory that has not been present for the last *skip* frames.

- **predict()** is the method used to tell the Kalman Filter to get a new prediction of the object's state.
- **estimated(Point measure)** applies correction to the Kalman Filter based on a measurement and updates the estimated state.

3.2 CEMTracker

The CEMTraker [5] is a multi-target tracking algorithm as minimization of a continuous energy function. With finding an optimal set of trajectories within a temporal window. The CEMTracker was downloaded from the author's website `http://www.milanton.de/contracking/index.html` [1] [6] [7] and used with default parameters except some the parameters which were indicated in the research paper [8] by Rikke and Moeslund.

As described in the research papers by Andriyenko and Schindler [2] and Milan, Roth, and Schindler [6]. The following were the steps they presented for Multi-Target Tracking:

3.2.1 Preliminaries and notation

In Figure 3.7 indicates a summary of the notations used throughout the paper.

Symbol	Description
$\mathbf{X}$	world coordinates of all targets in all frames
$\mathbf{X}_i^t$	(X,Y) world coordinates of target i in frame t
$\mathbf{x}_i^t$	(x,y) image coordinates of target i in frame t
F, N	total number of frames and targets, respectively
$F(i)$	number of frames where target i is present
s_i, e_i	first, respectively last frame of trajectory i
$N(t), D(t)$	number of targets, respectively detections in frame t
$\mathbf{D}_g^t$	(X,Y) world coordinates of detection g in frame t

Figure 3.7: The Notations Used for the presented CEMTracker [6]

Furthermore, the following is a more detailed explanation of the notations and structures used:

- The image coordinates are expressed by x, y.
- The X and Y world coordinates of all N targets in a sequence of F frames are expressed by the state vector X.
- It is assumed that all dynamic targets are moving on a single plane.

- Continuous location $X_i^t \in \mathbb{R}^2$ of target i at time t is exactly defined for all frames t ∈ si , . . . , ei within the temporal life span of the trajectory.
- *F(i) := ei - si + 1* denotes the temporal length of trajectory *i*, where the first and final frames are *si* and *ei* respectively.
- The number of targets in frame *t* is denoted as *N(t)*.
- *D(t)* indicates the number of detections in frame *t*.
- The location of detection g in frame t itself is denoted as D_g^t.

3.2.2 Continuous energy

Continuous energy is an algorithm that assign a cost or also known as energy to every possible solution and then finds the state with the lowest cost. Due to the complexity and highly non-convexity of the objective functions, this tracker have a powerful algorithm to capture the complex situations with complex mathematical properties.

The algorithm presented is a linear combination of six individual terms as shown in Equation 3.4.

$$E = E_{det} + \alpha E_{app} + \beta E_{dyn} + \gamma E_{exc} + \delta E_{per} + E_{reg} \tag{3.4}$$

Where:

- E_{det} keeps the solution close to the observations.
- E_{app} captures the appearance of different objects to disambiguate data association.
- E_{dyn}, E_{exc} and E_{per} are the three priors that promote the plausible motion and enforce physical constraints.
- E_{reg} is the requlizer that keeps the solution simple and prevents over-fitting.

The aim is then to find the state X^* in Equation 3.5that minimizes the high dimensional continuous energy from Equation 3.4:

$$X* = arg \min_{X \in R^d} E(X) \tag{3.5}$$

Figure 3.8 exemplifies the five components that are explained in more detail in the following sections:

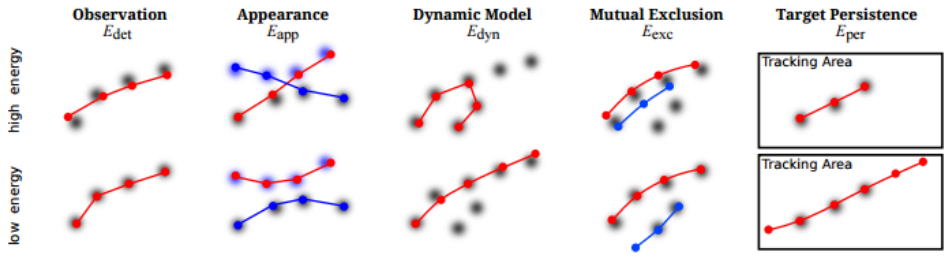

Figure 3.8: The effects of the five different components explained in the next sections of the energy function [6]

The mathematical explanation of the following models go further from the context of this work, therefore we advise you to look for a detailed explanation in the original work of Milan, Roth, and Schindler[6].

Observational Model

The idea is to track by detection, therefore the likelihood of and object presence at every location is calculated. As this shows a reliable tacking even in unconstrained environments and moving cameras.

Dynamic Model

A constant velocity model is used, with "intelligent" smoothing that take into account the energy terms to smooth nodes, which will prevent having identity switches.

Mutual Exclusion

A continuous exclusion constraint is included in the energy function as two object cannot occupy the same space simultaneously. Therefore, this will enforces unique data for the trajectories.

Trajectory persistence

A sigmoid penalty is included as a soft constraint as a target or an object cannot appear or disappear within the tracking area.

Regularization

To better fit the data and have a model with fewer targets and longer trajectories a penalty is added to the number of existing targets.

3.2.3 Global occlusion reasoning

The occlusion scenarios taken into considerations were:

- Crowded scenes when targets frequently occlude each other causing inter-object occlusion as shown in Figure 3.9.
- A target may move behind static objects like trees, pillars, or road signs.
- Orientation changes may cause self-occlusion, extensive articulations, or deformations.

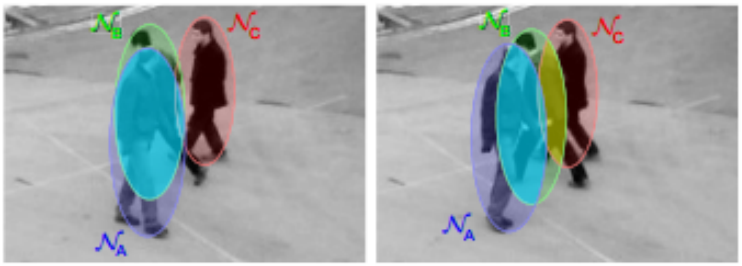

Figure 3.9: People occluding each other while walking, which can be easily mistaken when tracking [6]

Depth ordering is also taken into account for potentially overlapping targets, where a binary indicator variable can be potentially used, to make the energy function non-differentiable. The solution for this is expressed by the use of sigmoid functions shown in 3.10, taking into account that it assumes a ground plane and also a camera at a low point.

3.2.4 Appearance model

Object appearance models provide information to make tracked objects distinguishable from the background and other objects. The appearance term used is continuously differentiable in close form. If there are abrupt changes, a hight penalty is given, while assuming that there is slow light changes and the colour is constant.

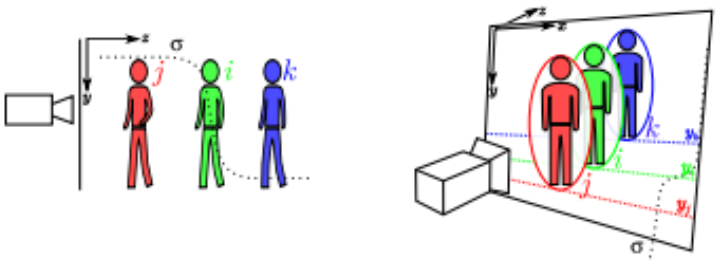

Figure 3.10: Depth Ordering solution in the paper for occlusion by creating ovals around the object [6]

Gaussian weighted regions are used to ensure that the energy remain smooth, as this will ensure differentiability and also a closed-form gradient. This is designed to fit gradient based optimization methods. As this decrease the number of identity switches and track fragmentations.

3.2.5 Transdimensional jumps

The following were optimizations introduced on the tracker [6] over the tracker [1] presented previously.

Six types of jump moves are introduced to escape weak local minima. The current state X_{curr} is altered by jumping to different regions in the search space and also lowering the energy. An example of this is shown in Figure 3.11.

Growing and shrinking

Growing and shrinking trajectory shows when a target is visible in the target area.

Merging and splitting

Improve data association and therefore can eliminate identity switches and track fragmentations. Merging is done by having two paths smoothly connected into one, while splitting is done by breaking the path in two.

Adding and removing

When there are strong detections new trajectories are added, and are started conservatively in three consecutive frames. Which will help also to find missing trajectories not found by the tracking algorithm. On the other hand trajectories are removed then the contribution to the energy is above some threshold.

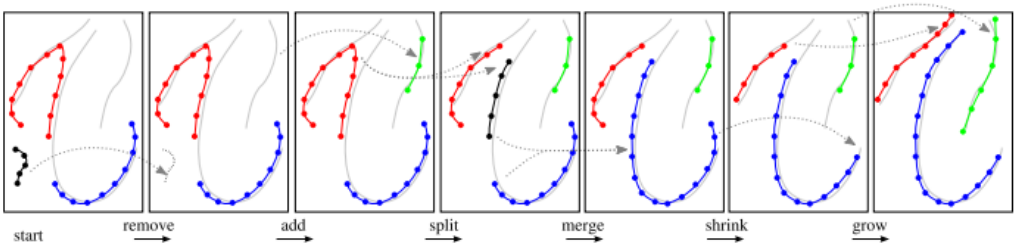

Figure 3.11: The six proposed jump moves [6]

3.2.6 Initialization

As all the others non-convex optimization algorithms, the results depend on the initial values where the iteration is started. The author make use of a per-target extended Kalman Filter (EKF) as a qualified initial value and associate the data in a greedy manner using a maximum overlap criterion. These are used to generate a variety of starting values, and in Figure 3.12, we can see that having different starting values converge to similar but not identical solutions.

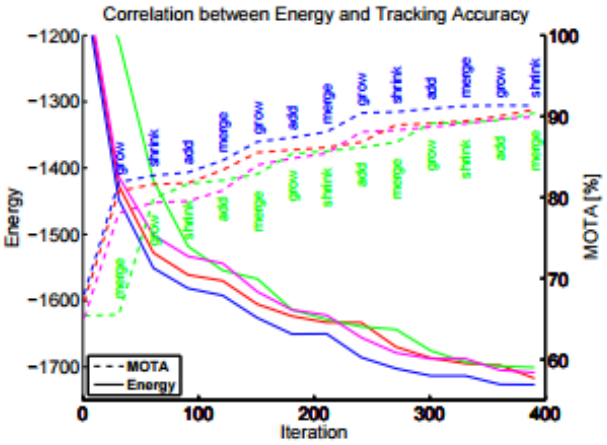

Figure 3.12: Four optimization runs started from four different initializations on the sequence. [6]

Chapter 4

Results

In this chapter the results of our implementation and the results presented on the paper are shown and compared, since the same experiments were conducted.

4.1 Experiments

The tracking algorithm has been tested on the same two minutes (3019 frames) video from an indoor soccer game, as in the original paper.

4.1.1 Kalman Tracker

Our Kalman tracker was implemented in C++ and OpenCV, as it was previously mentioned under the implementation details. Just as in the original paper, the measurement noise covariance R has been tuned to 0.1 and the process noise covariance Q is tuned to 0.002 for position and 0.003 for velocity.

4.1.2 CEM Tracker

Since the re-implementation of the CEM tracker was not done by the original authors, it was not needed for the same purpose in our work. The CEM tracker has been downloaded from the author's website (http://www.milanton.de/ contracking/index.html). Although in this work we investigated the implementation behind the CEM tracker in order to give an in-depth representation of our research and knowledge in the topic, due to limitations and time constraints the CEM tracker has not been used for the experiments. However, we hypothesize that

using the same tracker under similar circumstances with the same data, would produce the same results as those were presented in the original work.

4.2 Results

The trajectories found by our Kalman tracker are plotted against the trajectories of the original Kalman filter and CEM tracker in Figure x.x. Each new identity found by the tracker is plotted in a new color assigned randomly. Even though our trajectories are plotted in image coordinate system, the figure shows that our tracker has found less trajectories compared to the original trackers. That qualitative comparison already indicates, that our trajectories give a less similar representation of the ground truth than the original Kalman tracker and the CEM tracker's. The mass amount of color changes of less trajectories in our implementation indicate the ID changes and high amount of false positive matches during the experiment.

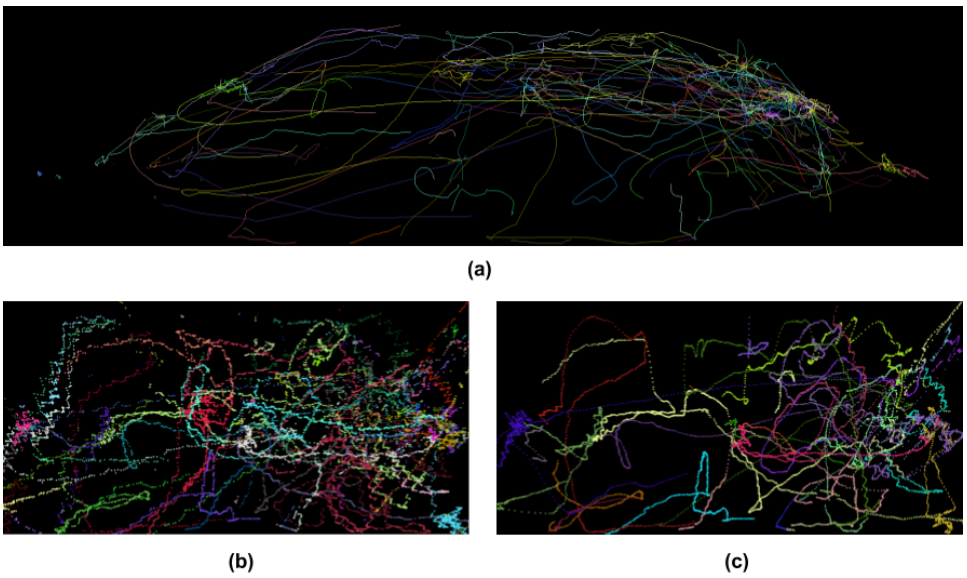

Figure 4.1: Our trajectories plot in image coordinates (a) and the trajectories from the original Kalman tracker (b) and CEM tracker after 60 epochs (c) plot in world coordinates.

We evaluate the tracking results using CLEAR MOT metrics, calculated by the available online MATLAB code (https://github.com/glisanti/CLEAR-MOT). The results are measured by true positives (TP), false positives (FP), false negatives (FN), ID switches and the two combined quality measures: multiple object tracking precision (MOTP) and multiple object tracking accuracy (MOTA). MOTP is the total error in estimated position for matched object–hypothesis pairs over all frames, averaged by the total number of matches made.

$$MOTP = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{4.1}$$

where d_t^i is the distance between the object o_i and its corresponding hypothesis. c_t is the number of matches found for time t.

$$MOTA = 1 - \frac{\sum_t FN_t + FP_t + IDS_t}{\sum_t g_t} \tag{4.2}$$

where FN_t, FP_t and IDS_t are the number of false negatives, false positives and ID switches, respectively, for time t, while g_t is the true number of objects at time t.

Experiment Results							
Tracker	TP	FP	FN	IDSwitch	MOTP	MOTA	#ID's
KF	86,84%	725,01%	0,19%	391	0,73	-638%	392
KF(O)	80,22%	9,86%	18,86%	219	0,75	70,36%	218
CEM(60)	18,14%	38,06%	81,60%	60	0,60	-19,91%	33

The results from our Kalman tracker, the original Kalman tracker and the CEM tracker after 60 epochs are presented in the table above. Our results for Kalman filtering fall short compared to the original and the CEM tracker. The true positive rate of 86% in our case exceeded the original tracker's 80%, however the false positive rate of our tracker reached above 700%, making the accuracy (MOTA) -638%, meaning that our tracker performed seven times less accurate than the original. The above false positive rate represents more than twenty one thousand false positive matches out of twenty nine thousand overall matches during the three thousand frames. This is due to our tracker predicting a different position for an individual for several frames, making the tracker vulnerable for accuracy. This could be due to the unprecise *predict* $\rightarrow$ *measure* $\rightarrow$ *correct* $\rightarrow$ *predict* method in the Kalman filter. Our tracker is less likely to connect broken trajectories, giving one hundred more ID switches compared to the original tracker.

The processing time for the C++ implementation of our Kalman filter have been 4.97 s per frame slower than the original implementation. This is due to using a laptop with inferior hardware specifications (Intel i7 -4700HQ 2.40 GHz with 6GM of RAM) for testing purposes and limitations in the optimization of our tracker algorithm, making it only suitable for offline tracking.

Chapter 5

Conclusion

We presented an offline tracking algorithm based on the work of Gade & Moeslund, which implements Kalman filtering. Although the tracker has been successfully implemented, some key features such as ROI and BLOB matching throughout the predictive method fall short compared to the original tracking algorithm. The authors are confident in, that improving these features would increase the accuracy of the tracker for multi target tracking. We can conclude that our tracker is precise enough to be able to track multiple targets at the same time (according to the MOTP metric), but this result could be different for more than eight tracked objects. In addition, optimizing the code could make it possible to perform better for online tracking purposes under better hardware specifications.

Bibliography

[1] A. Andriyenko, S. Roth, and K. Schindler. "An Analytical Formulation of Global Occlusion Reasoning for Multi-Target Tracking". In: *IEEE International Workshop on Visual Surveillance (in conjunction with ICCV), Barcelona, Spain* 1 (2011).

[2] Anton Andriyenko and Konrad Schindler. "Multi-target Tracking by Continuous Energy Minimization". In: *CVPR*. 2011.

[3] *BLOB Analysis (Introduction to Video and Image Processing) Part 1*. URL: http://what-when-how.com/introduction-to-video-and-image-processing/blob-analysis-introduction-to-video-and-image-processing-part-1/ (visited on 03/21/2017).

[4] Axis Communications. "Some like it hot – Thermal cameras in surveillance". In: (2009). URL: http://www.axis.com/files/whitepaper/wp_axis_thermal_cameras_en_37661_0912_lo.pdf..

[5] Anton Milan. *Anton Milan - Continuous Energy Minimization for Multi-Target Tracking*. 2017. URL: http://www.milanton.de/contracking/index.html.

[6] Anton Milan, Stefan Roth, and Konrad Schindler. "Continuous Energy Minimization for Multitarget Tracking". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.1 (2014), pp. 58–72. DOI: 10.1109/tpami.2013.103.

[7] Anton Milan, Konrad Schindler, and Stefan Roth. "Multi-Target Tracking by Discrete-Continuous Energy Minimization". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.10 (2016), pp. 2054–2068. DOI: 10.1109/tpami.2015.2505309.

[8] Gade Rikke and Thomas B Moeslund. "Thermal Tracking of Sports Players". In: *Sensors (Basel, Switzerland)* 14.8 (Aug. 2014), pp. 13679–13691. ISSN: 1424-8220. DOI: 10.3390/s140813679. JSTOR: {PMC}4179012. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4179012/.

[9] *Top uses and applications of Thermal Imaging Cameras*. 2015. URL: https://www.grainger.com/content/qt-thermal-imaging-applications-uses-features-345.