CVLab: Computer Vision Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
http://cvlab.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Real time multi-object tracking using multiple cameras

Semester Project

Michalis Zervos

Supervisor **Professor Pascal Fua**
Teaching Assistant **Horesh Ben Shitrit**

Spring Semester

June 2012

ABSTRACT

This report presents the work conducted during the semester project. We built and present a real-time multi-people tracker, which is based on the Kalman Filter. The input to the software is a Probabilistic Occupancy Map of the observed area. The main goal of the project was to incorporate this tracker to the real-time detection software available on the CVLab demo room. A standalone version was also built. The algorithm exploits appearance cues to prevent identity switches. Instead of computing the appearance difference in a frame-by-frame manner, an appearance model is initially built when an individual enters the scene and is afterwards matched against the detected people. The frame-by-frame spatial tracking of the Kalman Filter makes the algorithm computationally efficient and the appearance model matching increases the robustness. The experiments performed in the demo room show that the method is satisfactory. We also validate our algorithm on a few datasets and the results prove that the method can be used in many scenarios. In certain datasets it even outperforms the state-of-the-art method while it’s one to two orders of magnitude faster.

SUBJECT AREA: Computer Vision

KEYWORDS: People Tracking, Kalman Filter, Appearance Model

CONTENTS

1. INTRODUCTION

One common approach to multiple people tracking is to perform the task in two discrete steps. First an object detector is employed, which provides measurements (sometimes noisy) about the positions of each individual. On the second step, a tracking algorithm is used to link those detections and create continuous trajectories. On this project we concentrate on the second part of this process. The main goal was to integrate a real-time multiple people tracker on the currently existing detection framework of the CVLab. Based on the requirements, a Kalman Filter was used to facilitate the tracking of each person. A multi-person tracker which also exploits appearance cues was built and presented in this report.

1.1 Report structure

The rest of this report is organized as follows. In the following section (1.2), the related work is presented. It is followed by a description of the existing detection framework in 1.3. The basic theory behind the Kalman Filter is analyzed in chapter 2. In chapter 3 we discuss all the aspects of the tracker. Chapter 4 is devoted to the software implementation of the tracker and the architecture of the system. In chapter 5, we present the experiments of our approach. Finally, some conclusions and future work can be found in chapter 6.

1.2 Related work

A lot of work has been done on the field of multi-object tracking for many years, both in monocular and multi-view scenarios. We focus on the multi-view approaches, based on Kalman filter, which are relevant to our project.

In [**1**], a Kalman filter is employed to simultaneously track in 2D image coordinates and 3D world coordinates for each camera. The 2D/3D trackers of each camera share information to improve the performance and trajectory prediction, which is used in case of occlusions.

In [**2**], the authors propose a system where multiple synchronized cameras are used to segment, detect and track multiple people. For each pair of cameras an object location likelihood map is formed. Those maps are combined, taking into account possible occlusions and a presence likelihood map on the ground plane is computed. The ground plane locations are then tracked using a Kalman filter.

In [**3**], both a motion model and an appearance model is used to keep track of each individual. The motion models are obtained using a Kalman filter which predicts the position both in 2D and 3D. The tracking is performed by the maximization of a joint probability model which takes into account both the appearance and motion model.

1.3 Existing framework

Currently, the demo room of the CVLab has a real-time detection application based on the POM [**4**] software[1]. The room has four cameras, one on each corner and a single door. The POM software provides real-time detections of people in the room, presented both in a grid and as boxes around the person on the live camera feed. The goal of this project is to provide not only detections of individuals in the room, but to also track them and present the tracking result in a similar manner (on video and on the grid).

[1] http://cvlab.epfl.ch/software/pom/

2. KALMAN FILTER

In this chapter we present the basic theory behind the Kalman filter [**5**], in the context of object tracking. Given the motion model of a moving object (which contains some kind of dynamic noise), and some noisy observations about its position, the Kalman filter provides an optimal estimate of its position at each time step. The optimality is guaranteed if all noise is Gaussian. Then the filter minimizes the mean square error of the estimated parameters (e.g. position, velocity). The Kalman filter is an online process, meaning that new observations are processed as they arrive.

To formulate a Kalman filter problem, we require a discrete time linear dynamic system with additive white noise that models unpredictable disturbances. The Kalman filter tries to estimate the state $x \in R^n$ of that system which is governed by the vector difference equation:

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

with a measurement:

$$z_k = Hx_k + v_k \tag{2}$$

The random variables w_k, v_k represent the process and measurement noise respectively. They are assumed to be zero mean, white noise with covariance matrixes Q, R respectively.

The matrix A is called the state transition matrix and relates the previous state x_{k-1} to the current state x_k, if no noise was present. The size of A is $n \times n$. Matrix B is optional and relates the control input (if any) $u_k \in R^l$ to the state x_k. In the context of our tracker, there is no control input, thus the factor Bu_k is dropped from the equation. Finally, the $n \times l$ matrix H, relates the measurement z_k to the state x_k.

The Kalman filter maintains the following two estimates of the state:

- $\hat{x}(k|k-1)$, which is an estimate of the state at time-step k, given knowledge of the process up to step $k-1$. It is an a priori state estimate at time-step k.
- $\hat{x}(k|k)$, which is an estimate of the process state at time-step k given the measurement z_k. It is an a posteriori estimate of the state at time-step k.

It also maintains the following two error covariance matrices of the state estimate:

- $P(k|k-1)$, which is the a priori estimate error covariance of $\hat{x}(k|k-1)$
- $P(k|k)$, which is the a posteriori estimate error covariance of $\hat{x}(k|k)$

A recursive minimum mean-square estimator, such as Kalman, operates in two phases on each time-step k. The first one is the prediction of the next state estimate $\hat{x}(k|k-1)$ using the previous one. The second is the correction of that estimate using the measurement, to obtain $\hat{x}(k|k)$. Initially, $\hat{x}(1|1)$ and $P(1|1)$ are considered known. To maintain those estimates, the following operations take place. In the prediction step:

1. State prediction:

$$\hat{x}(k|k-1) = A \cdot \hat{x}(k-1|k-1) \tag{3}$$

2. Error covariance prediction:

$$P(k|k-1) = A \cdot P(k-1|k-1) \cdot A^T + Q \tag{4}$$

In the correction step:

3. Measurement prediction:

$$\hat{z}(k|k-1) = H \cdot \hat{x}(k|k-1) \tag{5}$$

4. Residual:

$$r_k = z_k - \hat{z}(k|k-1) \tag{6}$$

5. Measurement prediction covariance:

$$S_k = H \cdot P(k|k-1) \cdot H^T + R \tag{7}$$

6. Kalman gain:

$$W_k = P(k|k-1) \cdot H^T \cdot S^{-1} \tag{8}$$

7. State update:

$$\hat{x}(k|k) = \hat{x}(k|k-1) + W_k r_k \tag{9}$$

8. Error covariance update:

$$P(k|k) = P(k|k-1) - W_k \cdot S_k \cdot W_k^T \tag{10}$$

So to initialize the Kalman filter, we have to define the state transition matrix A, the state – measurement matrix H, the two noise covariance matrices R, Q and at each time step to feed the filter with a measurement z_k. Those are all defined in the following chapter.

While the Kalman Filter was selected for this project, the design of the system allows easy interchanging of the filter with some other frame-by-frame state estimator, such as a Particle Filter or an Extended Kalman Filter.

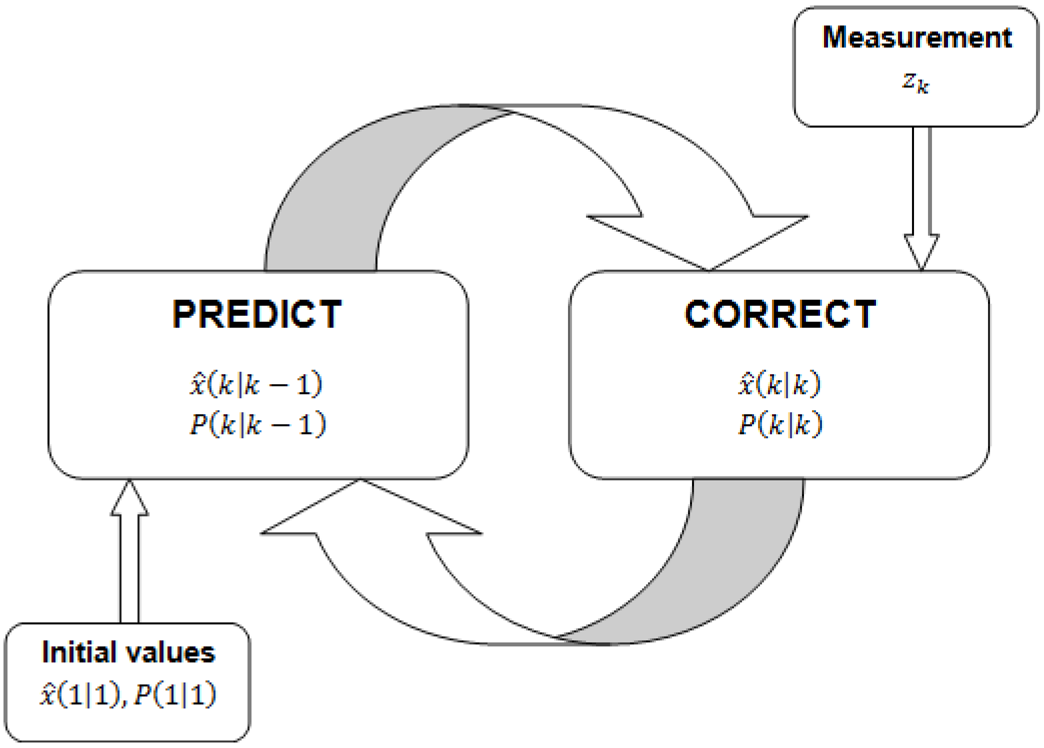


Figure 1. The Kalman filter Predict/Correct model.

For more information about the Kalman filter, the reader is referred to [**6**].

3. TRACKING

Let $POM(x,y)$ be a Probabilistic Occupancy Map [**4**] of the scene with grid dimensions $W \times H$ where $0 \leq x < W, 0 \leq y < H$ and $0 \leq POM(x,y) \leq 1$. We threshold POM and keep only the values above θ. This defines a function/grid G:

$$G(x,y) = \begin{cases} 1, if\ POM(x,y) \geq \theta \\ 0, \quad otherwise \end{cases} \tag{11}$$

Each position of the grid where $G(x,y) = 1$ is considered as a measurement m and M_k is the set of all the available measurements at a given frame k.

Let C be the number of cameras viewing the scene and providing frames at a constant framerate fps. The interval between each frame is:

$$dt = \frac{1}{fps} \tag{12}$$

In what follows, we discuss a single person tracker (section 0), the multi-person tracker (section 3.2) and the appearance model added on top of the multi-person tracker (section 3.3).

The notation used throughout the chapter is summarized on Table 1.

G	Detection grid
dt	Interval between two consecutive frames
x_k	State of the Kalman Filter at time-step k
z_k	Measurement provided to the Kalman Filter at time-step k
o	An object
m	A measurement
O_k	Set of objects at time-step k. Subscript k may be dropped.
M_k	Set of measurements at time-step k. Subscript k may be dropped.
$N = size(O)$	Cardinality of the O set
$L = size(M)$	Cardinality of the M set
$p = (px, py)$	2D coordinates of an object
(mx, my)	2D coordinates of a measurement
(vx, vy)	2D velocity of an object
$D, \dot{D}$	Distance arrays. Original and augmented
c	A camera
C	Set of cameras
$NC = size(C)$	Cardinality of C set
W, H	View width and height

Table 1. Notation table

3.1 Single person tracking

Using a Kalman Filter, tracking of a single individual in the scene is a relatively easy task. Let (px, py) be the real 2D coordinates of the object in the grid, (mx, my) the 2D coordinates of a measurement in the grid so that $G(mx, my) = 1$ and (vx, vy) the velocity in each direction. The state vector x_k and measurement vector z_k of the Kalman filter on frame k are defined as:

$$x_k = (px, py, vx, vy) \tag{13}$$

$$z_k = (mx, my) \tag{14}$$

The State-Measurement matrix H is then defined as:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{15}$$

Assuming that a person moves with constant velocity, the state equations are defined as:

$$\begin{gathered} px_k = px_{k-1} + vx_{k-1} \cdot dt \\ py_k = py_{k-1} + vy_{k-1} \cdot dt \\ vx_k = vx_{k-1} \\ vy_k = vy_{k-1} \end{gathered} \tag{16}$$

And the state transition matrix:

$$A = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

The measurement noise covariance matrix R and the process noise covariance matrix Q are defined as:

$$R = \begin{bmatrix} \sigma_{mx}^2 & 0 \\ 0 & \sigma_{my}^2 \end{bmatrix} \tag{18}$$

$$Q = \begin{bmatrix} \sigma_{px}^2 & 0 & 0 & 0 \\ 0 & \sigma_{px}^2 & 0 & 0 \\ 0 & 0 & \sigma_{vx}^2 & 0 \\ 0 & 0 & 0 & \sigma_{vy}^2 \end{bmatrix} \tag{19}$$

Where σ^2 denotes variance of each quantity. In our case, the measurement noise corresponds to the POM detections noise, and is set in the beginning of the execution. The process noise is defined according to the motion we want to track. For pedestrian tracking, usually there are small variations to their speed.

On the other hand, for sport players, the variance of their velocity vectors is greater.

Having defined those vectors and matrices, at each time step we update the Kalman filter with the new measurement and then predict an estimate $\hat{x}_k$ of its state as described in chapter 0. The 2D spatial coordinates $(\widehat{px}, \widehat{py})$ of the state, is considered as the position p of the object o.

For single person tracking, if at some frame k more than one measurement (due to noise probably) is found in the scene, then the measurement closest to the person's last estimated location $(\widehat{px}_{k-1}, \widehat{py}_{k-1})$ is assigned to that object and used to update the Kalman filter.

When a measurement is not available, then the Kalman Filter is updated by its predicted state and not corrected by any measurement.

3.2 Multiple person tracking

The multi-person tracking is a generalization of the single person tracker. We assume that the motion of each person is independent of the others. For each object in the scene, a separate Kalman Filter is initialized and models its trajectory. The multi-person tracker maintains a set of objects O_k currently being tracked at frame k and a set of measurements M_k available on this frame. Let N_k denote the number of objects and L_k denote the number of measurements.

$$O_k = \{o_1, o_2, \dots, o_{N_k}\} \tag{20}$$

$$M_k = \{m_1, m_2, \dots, m_{L_k}\} \tag{21}$$

For the rest of the section, we drop the subscript k from the notation and discuss for a single frame.

Some caution must be taken, on how to link the measurements with the trajectories. For now, consider the case where there are as many measurements as the objects being tracked, $N = L$. We deal with noisy detections on the following section (3.2.2).

A very simple, yet efficient, algorithm to assign a measurement to each object, would be to iteratively traverse the list of objects and for each one calculate the closest measurement and assign it to it. This measurement would then be invalidated and the next object will be processed. The algorithm is presented below.

Algorithm 1. Greedy assignment

```
1: $N = size(O)$
2: $assigned[N] = \{false\}$
3: $for\ i = 1\ to\ N$
4:   $dists[:] = computeDistances(o_i, M, assigned)$
5:   $(d_{min}, m) = \min(dists)$
6:   $Assign\ M[m]\ to\ o_i$
7:   $assigned[m] = true$
8: $end\ for$
```

Function $computeDistances(o, M, assigned)$ computes and returns an array of distances between object o and every measurement $m \in M$ which has $assigned[m] = false$. The function $\min(dists)$ returns the minimum distance d_{min} and the corresponding measurement m. The distance used throughout this section is the Euclidean distance between the object's position and the measurement's position on the grid.

This obviously is a greedy algorithm which might not lead to the best assignment between objects and measurements, since at each iteration it tries to minimize the "local" distance of this object and the available measurements. Also, depending on the order, which the objects are going to be processed, the output might be different. Consider the situation depicted in Figure 2, where the diamonds represent the measurements and the triangles represent two objects (o_1: red, o_2: blue) being tracked. The red object lies in the middle of the two measurements, slightly closer to m_2. Using Algorithm 1, the red object will be linked to m_2 since it will be processed first, leaving only m_1 available for the blue object. The resulting assignment ($red - m_2, blue - m_1$) is probably not the optimal one.

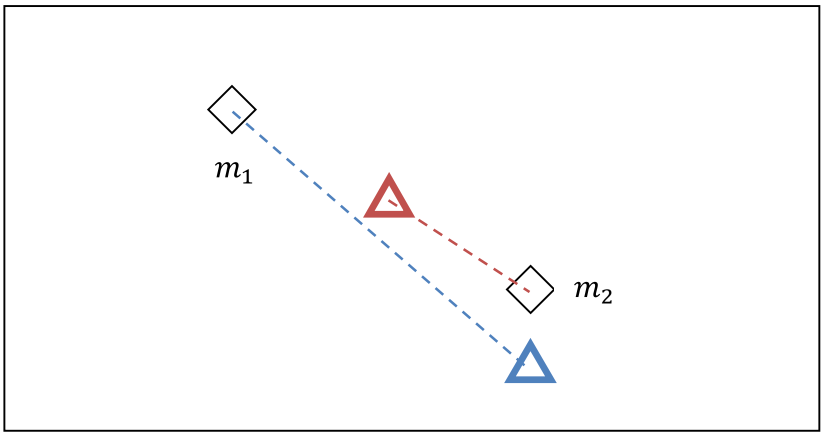


Figure 2. Assignment of the Greedy algorithm

3.2.1 Optimal object – measurement assignment

A better idea is to use an algorithm that minimizes the sum of all distances between every object – measurement linked pair. Using such an algorithm the result would be the globally optimal one, as shown in Figure 3.

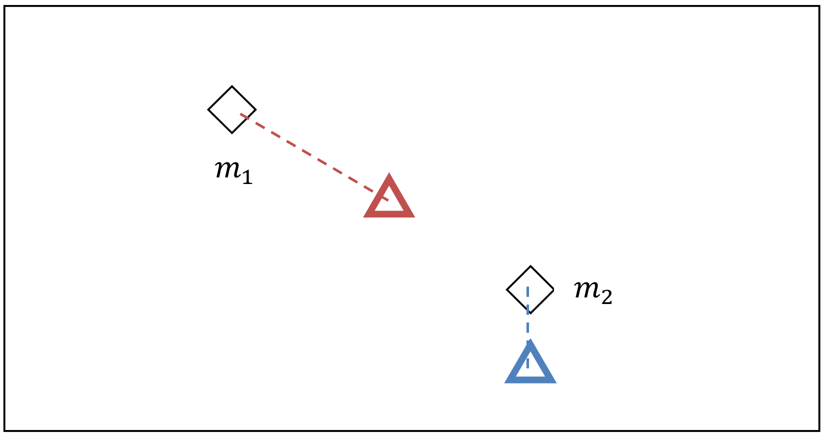


Figure 3. Optimal assignment

Basically we need an assignment that maps every measurement to exactly one object and every object to exactly one assignment such that the total distance is minimized. To find such an assignment, we need to properly formulate the problem. As before, O denotes the set of objects and M the set of measurements. Let $D_{N\times N}$ be a matrix of the distances between every $o \in O$ and $m \in M$ such that $D(i,j) = dist(o_i, m_j)$.

$$D = \begin{bmatrix} d_{1,1} & \cdots & d_{1,N} \\ d_{2,1} & & d_{2,N} \\ \vdots & \ddots & \vdots \\ d_{N,1} & \cdots & d_{N,N} \end{bmatrix} \quad (22)$$

We want to find a bijection (one-to-one correspondence) $f: O \rightarrow M$ such that the quantity below is minimized:

$$\sum_{i=1}^{N} D(i, f(i)) \tag{23}$$

This can be formulated as linear optimization program:

$$\begin{aligned} &\text{Minimize} && \sum_{i \in O} \sum_{j \in M} D(i,j) \cdot a_{ij} \\ &\text{Subject to} && \textstyle\sum_{i \in O} a_{ij} = 1 \ \forall j \in M \\ &&& \textstyle\sum_{j \in M} a_{ij} = 1 \ \forall i \in O \\ &&& a_{ij} \geq 0 \ \forall i \in O, j \in M \end{aligned} \tag{24}$$

This is a well-known optimization problem (the Assignment Problem) and is proved to have an optimal solution [**7**] where the variables a_{ij} take the value 1 if object $o_i$is linked with measurement m_j, or 0 otherwise. The problem can be solved in polynomial time using the Hungarian Method. The analysis of this algorithm is out of the scope of this report and the reader is referred to [**7**], [**8**] for more information.

In our implementation, initially we calculate the pair-wise distances of every object and measurement and then employ an instance of the Hungarian Algorithm solver to find the optimal assignment. Given that the number of objects is usually fairly small, the algorithm is very efficient and can be used in our real-time tracker.

3.2.2 Dealing with detection noise

As is usually the case, the detections are noisy. There are two different types of noise that can arise during the detection phase.

1) Wrongly positioned detections.
2) Different number of detections (L using the previous notation) than the actual number of the objects (N) being tracked.

In the first case, Kalman filter, by its design, can deal with those outliers and provide a smooth trajectory. To handle the second case, though, we need to modify the assignment method described in section 3.2.1, since $L \neq N$ .Two cases can be distinguished: a) The number of measurements is greater than the number of objects being tracked, $N < L$ and b) the number of measurements is less than the number of objects being tracked, $N > L$.

In both cases, we construct the array of pair-wise distances $D_{N \times L}$ as before. Let $r = \max(N, L)$. Next, we augment as many columns/rows needed to D to create an $r \times r$ matrix $\dot{D}$. The distance between an object and an augmented measurement is set to +∞ (same for augmented objects). With then run the Hungarian method on matrix $\dot{D}$, which will still link objects and measurements that provide the global minimum of the objective function.

Below, we describe three examples that demonstrate the linking algorithms under noisy detections.

a) Number of objects greater than the number of measurements, $N > L$. Consider the four objects (triangles) and the two measurements (diamonds) shown in Figure 4. Let the original distance matrix $D_{4 \times 2}$ be:

$$D = \begin{bmatrix} 8 & 4 \\ 1 & 6 \\ 4 & 3 \\ 6 & 1 \end{bmatrix} \tag{25}$$

It is obvious that o_2 should be linked to m_1 and o_4 to m_2, while the remaining two object shouldn't be linked with any measurement. Since the Hungarian method requires the two sets to have equal cardinalities, we augment the matrix with 2 measurements as discussed.

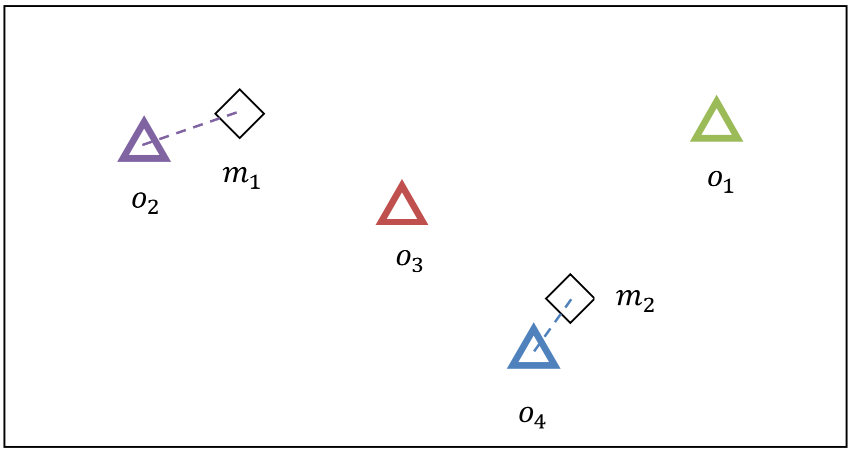


Figure 4. The case where there are more objects than measurements

The augmented matrix $\dot{D}$ will be:

$$\dot{D} = \begin{bmatrix} 8 & 4 & \infty & \infty \\ 1 & 6 & \infty & \infty \\ 4 & 3 & \infty & \infty \\ 6 & 1 & \infty & \infty \end{bmatrix} \tag{26}$$

And the Hungarian method will produce the desired output $(o_2, m_1), (o_4, m_2)$. The two remaining objects will be assigned to one of the two augmented columns, but they is no actual measurement corresponding to them.

b) Consider, now, the case of Figure 5 where $L > N$. There are 4 measurements and 2 objects.

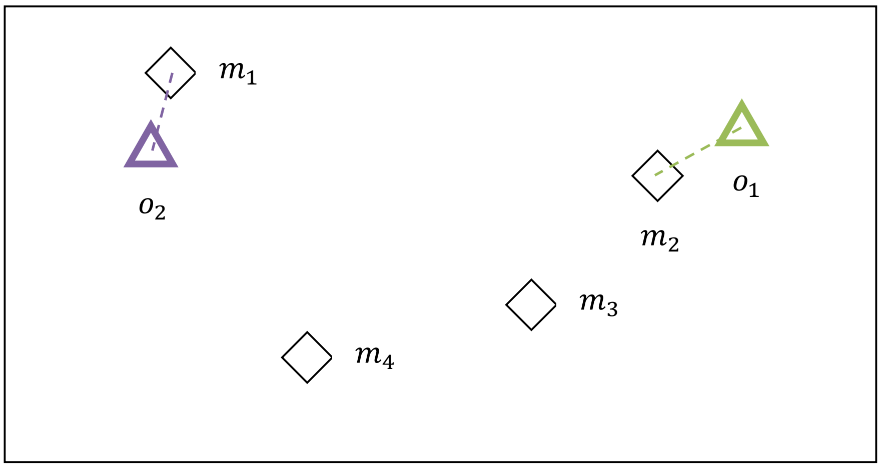


Figure 5. The case where there are more measurements than objects

Let the original distance matrix $D_{4\times2}$ be:

$$D = \begin{bmatrix} 8 & 1 & 3 & 7 \\ 1 & 7 & 6 & 4 \end{bmatrix} \tag{27}$$

We augment the matrix with two more objects and it becomes:

$$\dot{D} = \begin{bmatrix} 8 & 1 & 3 & 7 \\ 1 & 7 & 6 & 4 \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix} \tag{28}$$

Running the Hungarian method on $\dot{D}$ will produce the assignment $\{(o_1, m_2), (o_2, m_1)\}$.

c) Finally, consider the case where $N = L$, but a detection for an object is absent, while two detections appear for another object. This is shown in Figure 6. This configuration will result in a normal execution of the Hungarian algorithm and the assignment $(o_1, m_2), (o_2, m_1)$. While this is actually the optimal solution to the assignment problem, we might want to avoid the linking of o_1 with m_2. For this reason, we set a threshold on the maximum distance between an object and a measurement can be linked.

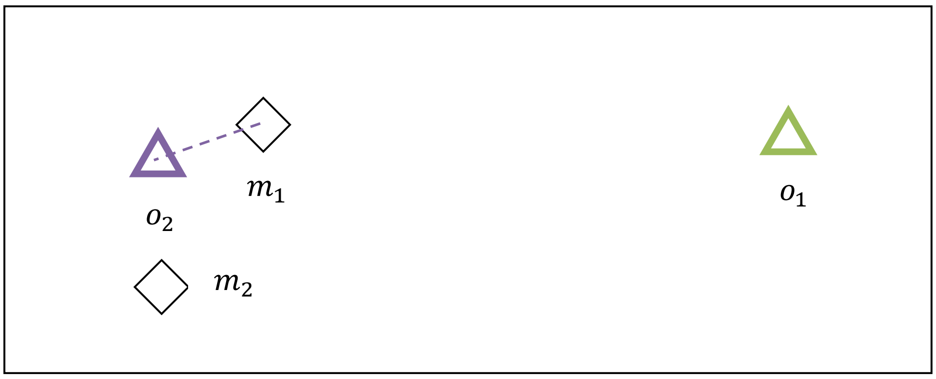


Figure 6. The case where two measurements are detected for an object, while none for the other

The complete assignment algorithm is presented in the following section (3.2.3).

3.2.3 The complete linking algorithm

With those modifications in mind the complete linking algorithm can be summarized as follows.

The function $computDistances(o, M)$ computes the distances between the object o and every measurement $m \in M$. The $Hungarian(D)$ returns an array of size r with the measurement assigned to each object. Only the first N are then processed (not the augmented) to determine if they are below the threshold distance θ.

Algorithm 2. Optimal linking algorithm

```
1:  $N = size(O), L = size(M)$
2:  $r = \max(N, L)$
3:  $D[r \times x] = \infty$
4:  $for\ i = 1\ to\ N$
5:     $D[i, 1:L] = computeDistances(o_i, M)$
6:  $A[:] = Hungarian(D)$
7:  $for\ i = 1\ to\ N$
8:     $m = M[A[i]]$
9:     $if\ dist(o_i, m) < \theta$
10:       $Assign\ measurement\ M[A[i]] to\ o_i$
```

3.2.4 Track management

Each object has a timestamp describing the last frame on which a measurement was assigned to it. When the detection – object linking phase of the current frame is complete, two more steps are performed.

3.2.4.1 Disabled Objects

If for some reason, the algorithm has lost track of an individual, meaning that no measurement was assigned to that specific trajectory, for more than a predefined number of frames, then the specific object is disabled and no output is produced for this individual.

At each frame, all the disabled objects are checked against measurements that are not assigned to an existing enabled object. If the appearance model at a detection grid position matches one of the disabled persons', then the corresponding trajectory is re-enabled and continues as normal.

3.2.4.2 Access points

The tracker allows individuals to enter or exit the scene at specific locations called access points, which are defined in the configuration file. For the demo application in our lab, a single access point is defined at the door. A measurement near the access points, which is not part of an existing trajectory, triggers the creation of a new person being tracked. Similarly, if the last observed detection of the trajectory was near an access point, then the corresponding person is removed from the list of objects being tracked.

In the first few frames, when the application starts, every grid cell is considered as an access point so that people who are already in the scene will be tracked. After that point, only individuals who enter through the access points are tracked.

3.3 Appearance model

Up until now, to link a measurement with an object we considered only the Euclidean distance between the two points. However if two trajectories intersect, there is the possibility of an identity switch. To minimize the number of identity switches, we exploit the color information of an object to build an appearance model for each individual.

Initially, for each camera, we build an occlusion map as described in 3.3.1. Then, for those individuals that are not occluded in at least k views, we create a YCbCr normalized color histogram of their image for each camera. The color histogram has 3×20 bins. For the k non-occluded views, we calculate the average of those histograms, and this serves as the appearance model of the person.

To calculate the color histogram of a person for a camera, we first crop the rectangle defined by the bounding box corresponding to the specific grid position. Then, we use the background subtraction to mask only the part of it that actually contains pixels belonging to the moving object. Finally, we convert those pixels to YCbCr color space and build a normalized histogram on those. The appearance model is only defined if the percentage of pixels that contributed to the histogram (with respect to the number of pixels in the bounding box) is above a threshold. The various steps of this process can be seen in Figure 7.

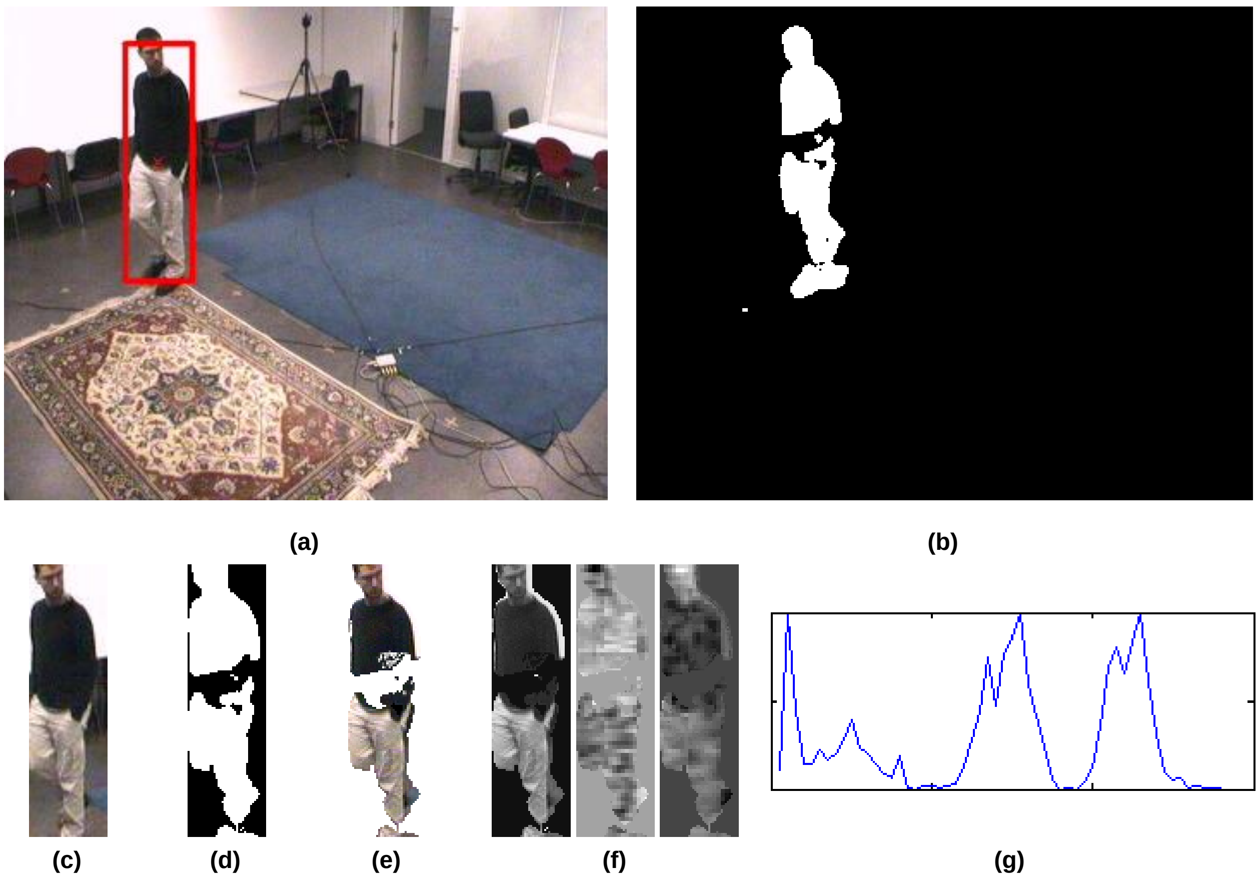


Figure 7. (a) Captured image with track result. (b) Background subtraction of that frame. (c) Region of interest of the original image. (d) ROI of the background subtraction. (e) Masked part of the image. (f) Y, Cb, Cr channels of (e). (g) Concatenated histogram of the 3 normalized histograms of the YCbCr channels with a total of 3x20 bins.

This process is repeated at each frame for all measurements in the grid, so each measurement has an appearance model associated with it. When a new object enters the scene, the appearance model of its corresponding measurement is assigned to it on the first frame that the object is not occluded in a specified number of cameras. From that point on, that appearance model will be used to compute the distance (section 3.3.2) between that person's appearance model and every measurement's appearance model.

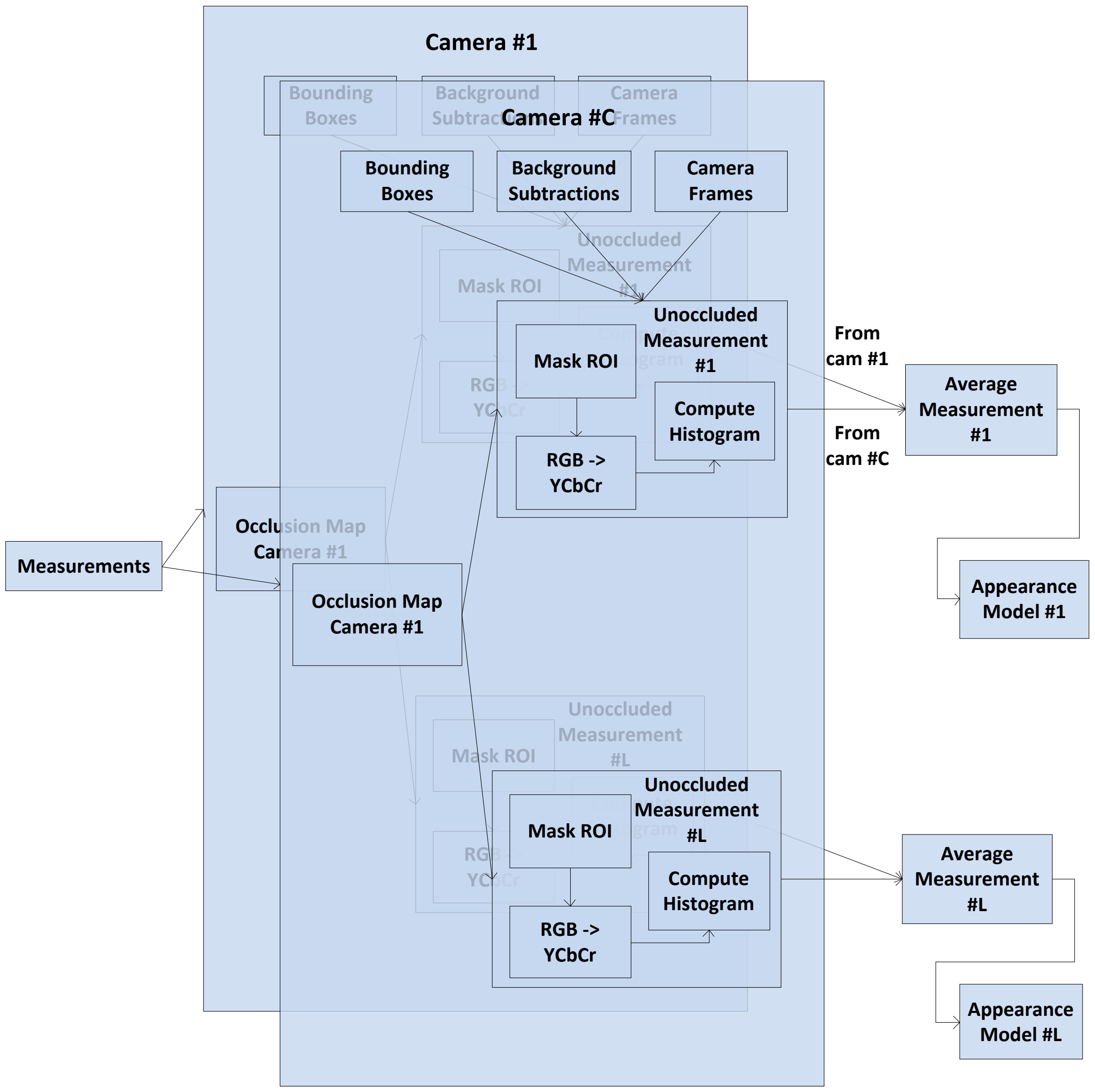


Figure 8. Appearance model computation workflow

As is clear, we don't rely on a frame-by-frame appearance matching as this would still cause problems when two persons' trajectories intersect. By combining the frame-by-frame nature of the Kalman filter spatial tracking and the aforementioned appearance model matching, the algorithm becomes more robust while still is computationally efficient.

The complete workflow of the appearance model computation is presented in Figure 8.

3.3.1 Handling occlusions

On the previous section, we said that in order to calculate the appearance model on a specific location l (3D cylinder of specific height corresponding to a grid position), only the non-occluded views of that location l are used. To find which cameras have a clear view of l, we build an occlusion map for each camera. We describe the process for a camera c.

An image I of equal size to the captured frame is created and every pixel is set to zero. For each measurement m available in that time-step, the bounding box b of that measurement for the camera c is retrieved (since the cameras are calibrated, we know the exact projection of the 3D cylinder in a specific grid position to the camera c). The pixels of image I inside the rectangle b are incremented by 1. We do this for every measurement. In the end the image I looks like the one in Figure 9. We know that a location is not occluded if the sum inside the corresponding bounding box is exactly the same as the number of pixels in that rectangle. In practice, we allow a small percentage (10%) of the box to be occluded. In the image black corresponds to 0, dark grey corresponds to 1 and light grey corresponds to 2. So in (a) we know that no objects are occluding each other, while in (c) the green and blue boxes are occluding each other.

So once the image I is computed, we just have to check if the sum inside each rectangle is equal to the area of that rectangle. To efficiently compute the sum inside each rectangle, we first compute the integral image I_{integ} of I. Then the sum of a rectangle can be computed in constant time.

The algorithm for computing the non-occluded views is the following. The algorithm takes as input the view width W and height H, the number of cameras NC and the measurements M, where $size(M) = L$. It returns a boolean array $occluded[L, NC]$ which denotes if the measurement m is occluded in camera c.

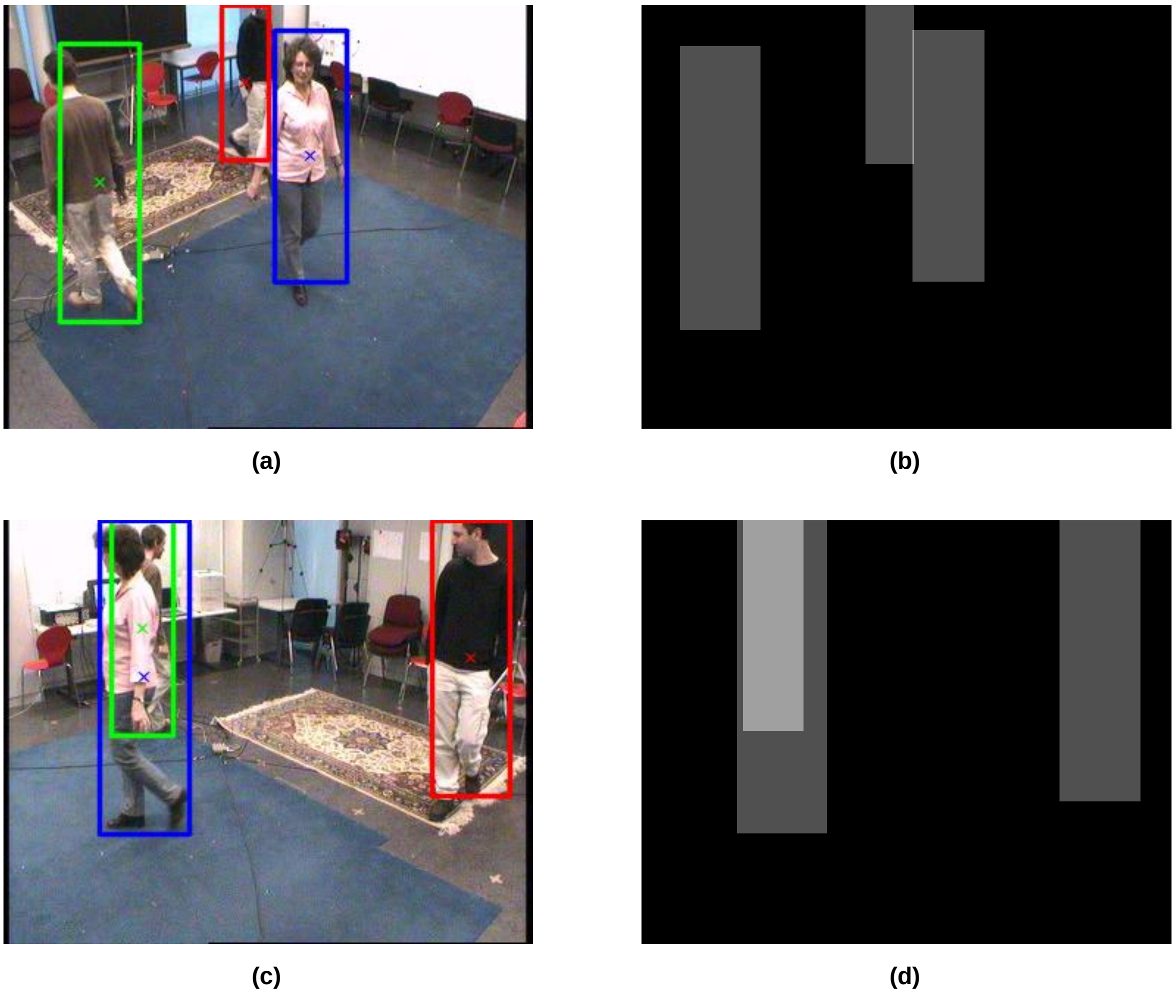


Figure 9. (a), (c) Views from camera #1 and #2. (b) Corresponding occlusion maps

Algorithm 3. Occlusion map computation

```
1:  $occluded[NC \times L] = true$
2:  $Foreach\ camera\ c$
3:    $I[W \times H] = 0$
4:    $Foreach\ measurement\ m \in M$
5:      $b = boundingBox(m.position, c)$
6:      $I(b) = I(b) + 1$
7:    $End\ foreach$
8:    $I_{integ} = integral(I)$
9:    $Foreach\ measurement\ m \in M$
10:     $b = boundingBox(m.position, c)$
11:     $s = sum(I_{integ}, b)$
12:     $if\ (area(b) * (1 + \theta) \geq s)$
13:       $occluded[c, m] = false$
14:   $End\ Foreach$
15: $End\ foreach$
```

The function $boundingBox(p, c)$ returns the bounding rectangle of position p in cameras' c image. The value θ is the allowed occlusion threshold.

3.3.2 Distance of histograms

Kullback – Leibler (KL) divergence of two probability distributions P, Q of a discrete random variable is defined as:

$$D_{KL}(P||Q) = \sum_i P(i) ln \frac{P(i)}{Q(i)} \tag{29}$$

The KL divergence is only defined if both P and Q sum to 1 and if $Q(i) > 0$ for any i that $P(i) > 0$. The quantity $0ln0$ is interpreted as zero.

In the case of normalized histograms the first condition (sum to 1) is always met. However, the second condition proves to be problematic. In many cases on our experiments only one of the two quantities $P(i), Q(i)$ was zero.

For that reason, we chose to use the Jensen – Shannon (JS) Divergence as the distance between two color histograms. The JS Divergence is defined as:

$$D_{JS}(P||Q) = \frac{1}{2}(D_{KL}(P||M) + D_{KL}(Q||M)) \tag{30}$$

where $M = \frac{1}{2}(P + Q)$.

The JS divergence can always be calculated and is a finite value (unlike KL divergence). If for some i, $P(i) > 0$ and $Q(i) = 0$, then $M(i) > 0$, so $D_{KL}(P||M)$ and $D_{KL}(Q||M)$ can always be computed. Also JS divergence has the nice property that:

$$0 \leq D_{JS}(P||Q) \leq 1 \tag{31}$$

if base 2 logarithm is used for the computation. This property is extremely useful because there is no need for a normalization of the result.

3.3.3 Updated measurement – object distance

The distance function used in the linking algorithm described in 3.2.3, is altered to also take into account the appearance model distance. The new distance between a measurement and object is defined as:

$$dist(o,m) = w * D_{euclN}\,(p_o, p_m) + (1 - w) * D_{JS}(a_o, a_m) \qquad (32)$$

where w is a weight factor, $p_o, p_m$are the 2D points associated with the object and the measurement, a_o, a_m the appearance models associated with the object and the measurement and $0 \leq D_{euclN}(x,y) = \frac{dist(x,y)}{MAX\_DIST} \leq 1$ is the normalized Euclidean distance between two points x, y. The MAX_DIST is the maximum distance on the grid, corresponding to the diagonal.

4. SOFTWARE

In this chapter, we present the software implementation aspects of the project. The project was implemented in C++ and uses the Qt library for its Graphical User Interface. It is now integrated with the existing real-time detection framework of the CVLab (see section 1.3). A standalone version was also built which takes as input the Probability Occupancy Map generated by the publicly available POM software[1]. A side-project was to extend the current software so as to provide a recording tool, which saved the videos either directly from the cameras or after the detection / tracking. Also, we added support to the framework for more than 4 cameras. Those two extensions are discussed in sections 4.4 and 4.5 respectively.

4.1 Tracking framework pipeline

The complete pipeline of the demo system is show in the diagram below (Figure 10). To avoid unnecessary delays in the communication, the various subsystems share circular buffers where they read/write. For example, such a circular buffer is used between the *CameraController* and the *Tracker* to write and read (respectively) the frame captured from a camera. All the components run on different threads.

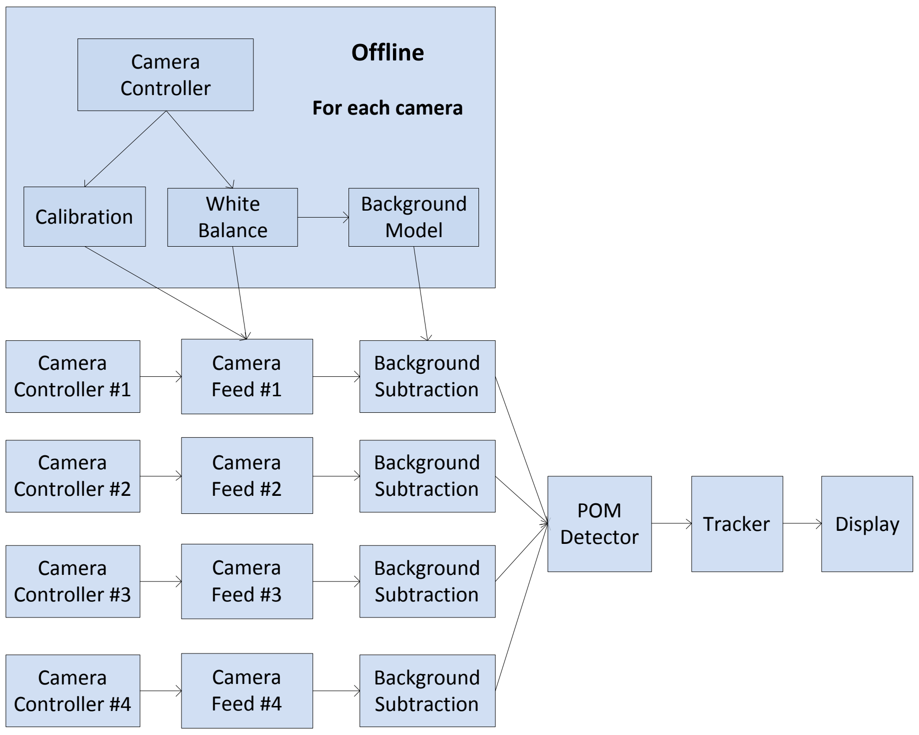


Figure 10. Tracking framework pipeline

[1] http://cvlab.epfl.ch/software/pom/

4.2 Architecture of the tracker

The tracker was designed in such a way that the Kalman Filter can be easily substituted by some other frame-by-frame filter, such as a Particle Filter or an Extended Kalman Filter. Those filters should just provide a similar interface to the tracker software.

The tracker module, as can be seen in the pipeline diagram (section 4.1), has been placed between the Detector and Display subsystem and runs on a separate thread. The Detector was altered to emit a signal each time it gets update. The Tracker catches that signal and updates the trajectories on each time step. The Tracker in turn, emits a signal that it has been updated, which is caught by the display subsystem that presents the output on the window.

The main classes of the system are:

- *KalmanFilter*. OpenCV's implementation of Kalman Filter is used. There are two main methods being used: *Predict()* and *Correct()* which perform the steps described in the corresponding section. To implement a Kalman Filter from scratch one needs to write the code to perform a few matrix operations as described in chapter 0. OpenCV's implementation was chosen in order to avoid rewriting code for matrix operations.
- *Object*, which models a person being tracked. Each object contains a KalmanFilter that keeps track of it.
- *AppearanceModel.* This models encapsulates a histogram as described in section 3.3. It also provides the methods to compute the histogram from the input image, to normalize it and also compute the distance (JS or KL) between two Appearance Models.
- *MultipleObjectsTracker.* This is the main class of the program and provides methods for linking measurements to objects (Section 3.2.3), for handling disabled objects or those entering / exiting the scene (Section 0), for computing the occlusion maps and appearance models on each measurement position (Section 0) and various other tasks. The method $TrackFrame()$ is responsible of invoking all the internal methods, to update the state of every object for that frame.
- *Measurement.* This class represents a measurement. It contains the position of the measurement, as well as the associated Appearance Model.

More information and details about the implementation can be found inside the source code files.

4.3 Standalone version

A standalone version was also implemented, so that we could evaluate our algorithm on various datasets. The algorithm and the classes are essentially the same as described before. The main difference is the input files. The main input files are the probabilities of occupancy of each grid position for each frame, in the open POM format. To visualize the detections, the bounding boxes for each grid position and each camera should be given in the same format as in the POM input files. If the appearance model is to be used, a video file of each camera should be given as input, together with the background subtractions for each frame. Those background subtractions can be either a video file for each camera or separate .png images, as is the input for POM. The paths for all the input and some parameters are set in a configuration file (similar to the one POM uses). Sample configuration files are provided for three datasets (Basketball, Soccer, Lab 6P).

This version can either run completely verbose in the console and produce an output similar to the one that KSP produces or present the tracking results on the input video and on a grid. Also it has the ability to save a video with the tracking result; both bounding boxes on each person and the tracking grid. The output is an MPEG-4 encoded .avi file in the same size and framerate as the original. All those options can be set in the configuration file.

To execute the standalone version use the command:

"./kalman_pom <CONFIG_FILE>"

4.4 Recording

We modified the configuration tool to allow the recording of raw video from every camera connected to the system.

The output video of each camera has the original camera size (1032 x 778) saved at 30 fps and encoded in MPEG-4 .avi format with a bitrate of 51385 kb/s.

To allow concurrent smooth video presentation on the screen and recording, different threads have to be started that handle the saving. For this purpose, the class *videoRecorder* was built. It's basically a wrapper around the OpenCV's *VideoWriter*[1] class. For each camera, one instance of *videoRecorder* is created and assigned to a new thread. This class reads from the corresponding camera buffer, passes the frame to the encoder and saves it to the specified output video. The tool was tested with 6 cameras and could handle simultaneous video presentation and writing on the original size and frame rate.

4.5 Support for more cameras

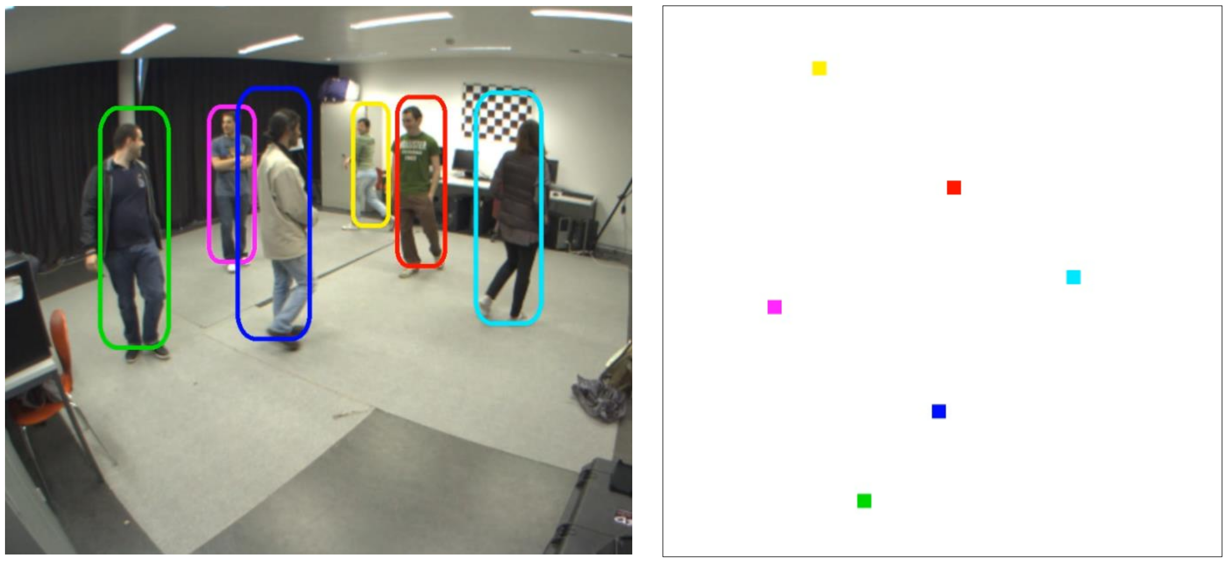
The configuration tool of the framework was extended to support an arbitrary number of Ethernet cameras. The user has now the ability to click on any of the four views and then press the number of the camera that she wishes to switch to. So each view of the four views can be connected to any of the cameras plugged into the system. The recording feature, presented above, automatically captures from all the cameras, not only those shown.

When a new camera is connected to an Ethernet adapter of the computer, the Reverse Path Filtering of the kernel for that adapter should be turned off, for the driver to be able to communicate with the camera. To do this, run the command "*sysctl -w net.ipv4.conf.ethX.rp_filter=0*" where X is the adapter number where the camera is connected. This change is temporary until the system is rebooted. To make the change permanent edit the file */etc/sysctl.conf* and set the same variable as in the command.

[1] http://opencv.willowgarage.com/wiki/documentation/cpp/highgui/VideoWriter

5. EXPERIMENTS

The complete real-time tracking application is operating on a demo room of the CVLab at EPFL. There are 4 cameras, one in each corner of the room. A screenshot of one of the cameras, together with the grid can be seen below.



(a) **(b)**

Figure 11. (a) Screenshot from one of the cameras of the CVLab demo room. (b) Corresponding detection grid

The real-time demo in the CVLab room can handle up to 5 or 6 people. This limitation is due to the size of the room and the positions of the cameras. People tend to occlude big parts of the image which results in pretty bad detections from the POM. As is shown next, our method can easily handle 20 or more persons in a scene.

Apart from the real-time experiments conducted in the demo room, we evaluated the algorithm on three publicly available datasets, using the standalone version of the software (Section 4.3). For each dataset, we calculated the GMOTA metric as defined in [**9**]. The GMOTA metric is an extension of the well known Multiple Object Tracking Accuracy (MOTA) metric [**10**], where the mme_t term is substituted by $gmme_t$. It is defined as:

$$GMOTA = 1 - \frac{\sum_t \left(c_m(m_t) + c_f(fp_t) + c_s(gmme_t)\right)}{\sum_t g_t} \qquad (33)$$

where m_t is the number of miss-detections, fp_t is the number of false positives and g_t is the number of ground truth detections. The term $gmme_t$ measures the proportion of identity switches in a global manner (unlike mme_t, in MOTA metric, which is the number of instantaneous identity switches). More information about the definition of $gmme_t$ can be found in [9]. The c_m, c_f and c_s are weighting functions set to $c_m(x) = cs(x) = 1$ and $cs(x) = \log_{10} x$, as in [9].

We tested our algorithm on three different datasets. The first one is the publicly available ISSIA dataset [11], which is a 2 minutes video footage shot at 25 fps by 6 cameras. It features 25 people, 11 of each team and 3 referees. The second dataset is a 4000 frames sequence from a basketball match, shot by 6 cameras on 25 fps. There are 14 people being tracked, five people and one coach on each team and two referees. Even though 6 cameras were used for the detection by POM, only 4 of them were used for computing the appearance models for the tracker. The last dataset is the publicly available 6 person laboratory video sequence shot in our demo room[1]. It consists of a 3000 frames shot at 25 fps by 4 cameras.

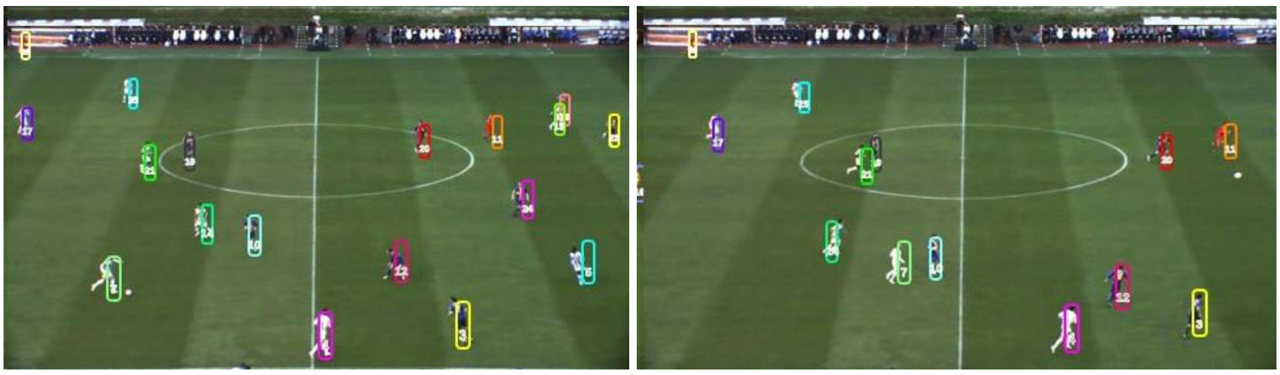


Figure 12. Two screenshots from the ISSIA dataset

We compare this algorithm to a number of different algorithms: The K-Shortest Paths method [12] (KSP), a modified version of KSP which takes into account appearance information from frame to frame (C-KSP), the state-of-the-art Linear Programming method that exploits the appearance model [9] and two tracklet versions of the LP method (Multi Commodity Network Flow), referred to as LLP and TLP. The GMOTA scores for the soccer dataset are presented below (Figure 13).

In the soccer dataset, our method achieves similar scores with the LLP method while it outperforms all the others. This happens because of the reliable POM detections, the fact that there aren’t many occlusions and the large (in general) distance between the

[1] http://cvlab.epfl.ch/data/pom/ - Laboratory Sequence – 6 People

players. When two players are close to each other, they are usually from different teams and the appearance model helps differentiating between the two.

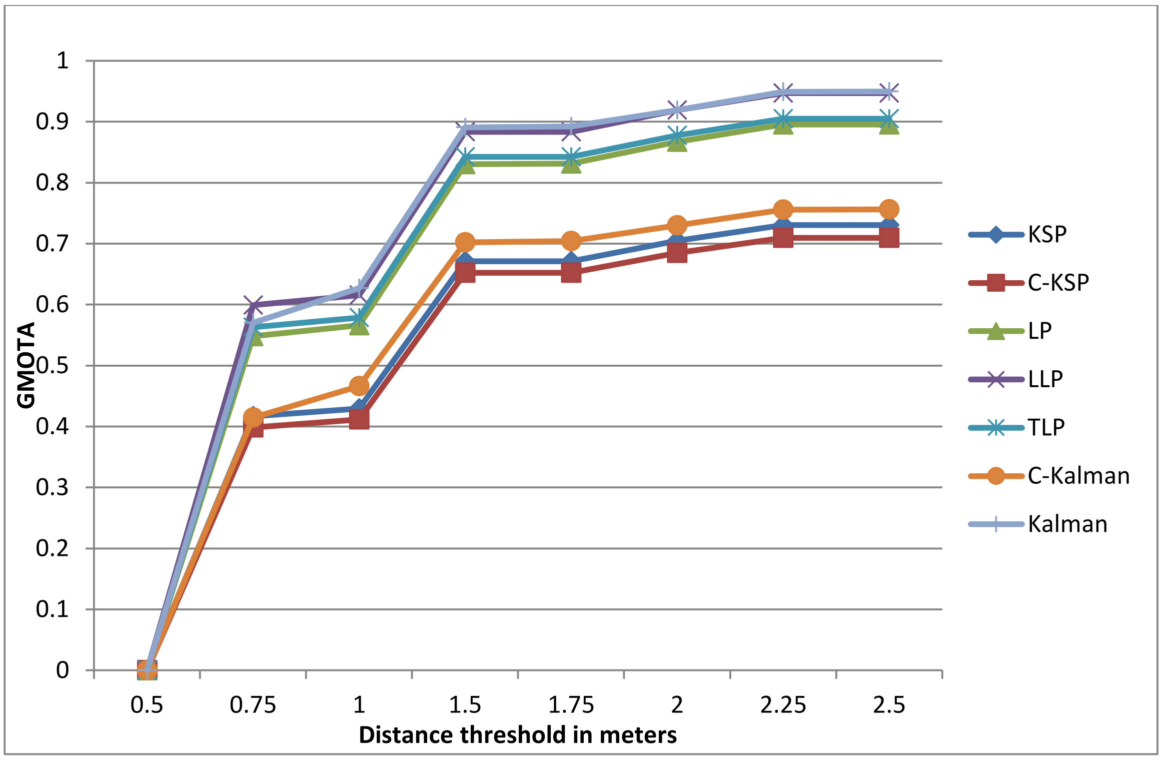


Figure 13. Soccer dataset GMOTA scores

The computational aspects of our algorithm are summarized on Table 2. The algorithm runs on a single core of an Intel i7 3.2GHz processor. The method proved to be very efficient which was a primary goal of this project, since we plan to add new features in the real-time tracking pipeline (see section 6.1).

Sequence	# Cameras	Resolution	# People	FPS
Lab	4	360 x 288	4	**173**
Soccer	6	480 x 270	25	**170**
Basketball	4	1294 x 964	14	**25**

Table 2. Frames per second, for each different dataset

As can be observed in the table, high resolution videos can slow down the algorithm; in those cases, the appearance model computation becomes a bottleneck because the number of pixels to be processed is quite large. In practice, we can subsample the original image before we compute the appearance models.

6. CONCLUSION

We have presented a real-time people tracker built on top of the POM detector. It is based on the Kalman Filter and can keep track of many people simultaneously. An unknown number of people, entering or exiting the scene at any moment, can be tracked. The algorithm exploits appearance cues to prevent identity switches. Instead of computing the appearance difference in a frame-by-frame manner, an appearance model is initially built when an individual enters the scene and is afterwards matched against the detected people. The algorithm was integrated to the existing demo application of the CVLab. Also a standalone version was built and used to evaluate our method. The frame-by-frame spatial tracking of the Kalman Filter makes the algorithm computationally efficient and the appearance model matching increases the robustness. In certain datasets it outperforms the state-of-the-art method while it's one to two orders of magnitude faster. As a side-project, we added support for more than 4 cameras in the existing demo and implemented a recording tool that captures each camera's feed.

6.1 Future Work

Even though the current tracking framework avoids some identity switches, by exploiting the image appearance, it doesn't prevent them at all. On situations where individuals wear clothes of similar colour or part of the clothes of a person match the background colour, the appearance model is essentially meaningless.

We plan to extend the framework so as to exploit the facial characteristics of each individual and build face models. The purpose of building a face model is twofold: a) the facial information, when available, could be combined with the current motion and appearance models, to provide more accurate tracking across frames. b) The face model will be used to identify the person by matching it against those in a face database. Such an algorithm could be used in applications, where specific individuals need to be tracked and their motion to be modeled.

ACKNOWLEDGEMENTS

I would like to thank my teaching assistant Horesh Ben Shitrit for his valuable feedback, the help and the advice he provided during my semester project.

REFERENCES

[1] J Black, T Ellis, and P Rosin, "Multi view image surveillance and tracking," in *Workshop on Motion and Video Computing, 2002. Proceedings*, dec 2002, pp. 169 - 174.

[2] Anurag Mittal and Larry S Davis, "M2Tracker: A Multi-View Approach to Segmenting and Tracking People in a Cluttered Scene," *International Journal of Computer Vision*, vol. 51, no. 3, pp. 189--03, 2003.

[3] J Kang, I Cohen, and G Medioni, "Tracking people in crowded scenes across multiple cameras," in *Asian conference on computer vision*, vol. 7, 2004.

[4] F Fleuret, J Berclaz, R Lengagne, and P Fua, "Multicamera People Tracking with a Probabilistic Occupancy Map," *IEEE Transactions onPattern Analysis and Machine Intelligence* , vol. 30, no. 2, pp. 267 - 282, Feb 2008.

[5] R.E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, pp. 35 - 45, 1960.

[6] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, 2006.

[7] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*.: Society for Industrial and Applied Mathematics.

[8] H.W Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, 1955.

[9] H Ben Shitrit, J Berclaz, F Fleuret, and P Fua, "Tracking multiple people under global appearance constraints," in *2011 IEEE International Conference on Computer Vision (ICCV)*, nov 2011, pp. 137 -144.

[10] K. Bernardin and R. Stiefelhagen, "Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, no. 1, p. 246309, May 2008.

[11] T. D'Orazio, M. Leo, N. Mosca, P. Spagnolo, and P.L. Mazzeo, "A Semi-automatic System for Ground Truth Generation of Soccer Video Sequences," in *6th IEEE International Conference on Advanced Video and Signal Based Surveillance AVSS '09.* , sep 2009, pp. 559 - 564.

[12] J Berclaz, F Fleuret, E Turetken, and P Fua, "Multiple Object Tracking Using K-Shortest Paths Optimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 9, pp. 1806 -1819, Sep 2011.