Graz University of Technology

Institute for Computer Graphics and Vision

Master's Thesis

Fast Object Tracker on a Smart Camera

Gernot Loibner

Graz, Austria, Sep 2014

Thesis supervisor

Univ.-Prof. DI Dr. Horst Bischof

Instructor

DI Dr. Peter Roth

Abstract

This diploma thesis addresses the problem of tracking various objects in real time on a smart camera. As starting point an existing tracking by detection based algorithm is used which uses histogram of oriented gradient (HOG) features for object detection and Lukas Kanade Tomasi (KLT) point tracking for motion estimation to combine those detections into trajectories. As the current implementation does not include distinct object information it likely mixes up objects passing nearby when tracking, leading to wrong object trajectories. To overcome this drawback the idea of including an additional verification step into the tracking process came up where object identity information can be included. With respect to real time capability, feature descriptors are reviewed and evaluated on two datasets. The most promising approaches in terms of quality (i.e., DCT, LBP, BRIEF) are included into the current algorithm. The extended tracking algorithm is evaluated using the CLEAR MOT metric against the baseline method. In addition, we compare to results available in the literature on two datasets with different image quality. The results show a significant drop of object identity switches whereas the overall performance of the algorithm doesn't improve in a satisfying manner.

Keywords. Multi-object tracking, detection, motion, segmentation, smart camera, feature descriptor, clear mot

Kurzfassung

Diese Diplomarbeit behandelt das Problem des Tracking von mehreren Objekten innerhalb eines Videostreams auf einer intelligenten Kamera in Echtzeit. Dazu wird ein bestehender Algorithmus, der auf dem Prinzip Tracking durch Detektion basiert, analysiert. Dieser Algorithmus bedient sich zur Detektion dem Prinzip der Histogramme von orientierten Gradienten (HOG) und kombiniert Detektionen zu Bewegungsbahnen mit Hilfe von Bewegungsabschätzung auf Basis der Arbeit von Lukas, Kanade und Tomasi (KLT). Da die aktuelle Implementation auf individuelle Informationen der Objekte verzichtet, treten häufig Verwechslungen bei der Zuordnung von Objekten zu Bewegungsbahnen auf, vor allem wenn Objekte sich nah aneinander vorbei bewegen. Um dieses Problem zu lösen, wird ein zusätzlicher Verifikationsschritt in den Algorithmus integriert. Dazu werden einige Eigenschaftsbeschreibungen für Objekte, mit Hinblick auf ihre Echtzeitfähigkeit, analysiert und mit Hilfe zweier Datensätzen miteinander verglichen. Die qualitativ besten - DCT, LBP, BRIEF - werden implementiert und in den bestehende Algorithmus integriert. Die Evaluierung des Algorithmus geschieht dann unter Zuhilfenahme der CLEAR MOT Metrik. Die Ergebnisse des zugrundeliegenden Algorithmus, der drei erweiterten Algorithmen und Ergebnisse aus der Literatur werden miteinander verglichen. Augenscheinlich ist der signifikante Abfall an Zuordnungsfehlern bei der Kombination von Objekten zu Bewegungsbahnen aber kaum Verbesserung in der allgemeinen Qualität des Algorithmus.

Schlüsselwörter. Multi-Objektverfolgung, Detektion, Bewegung, Segmentierung, Intelligente Kamera, Eigenschaftsbeschreibung, clear mot

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……. ……..
date (signature)

Contents

List of Figures

List of Tables

Chapter 1

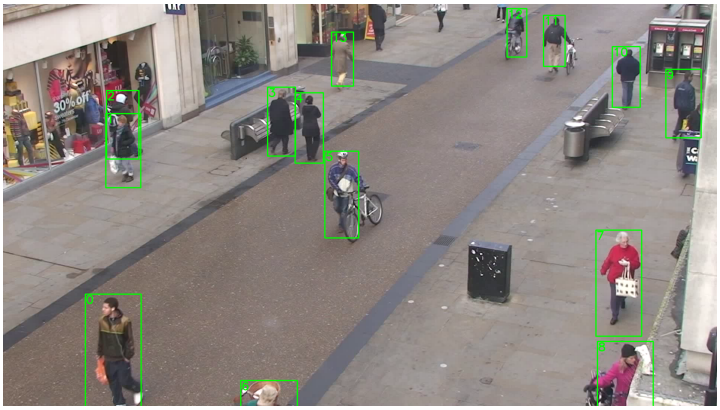
INTRODUCTION

Automatic tracking of objects like people or cars in video streams has increasingly gained more interest in past years due to the growing number of traffic cameras and surveillance cameras. The huge amount of data produced daily from such cameras makes it nearly impossible (or at least economically infeasible) to extract motion information by hand. The information gained through a complete analysis, e.g., from people walking in a specific direction, can give very useful information when planning layouts of airport terminals, arranging shops or for predicting possible dangerous situations on entries and exits of public spaces due to overcrowding. The difficulties of tracking objects in real world scenarios one has to deal with are different appearances of the objects (different car types, different clothing of humans) and changing background and illumination in the scenes.

The main goal of automatic tracking is to convert visual information automatically into trajectory information which then can be processed. Trajectory information can be given in different ways, e.g., by a list of bounding boxes of persons (each person is identified by an ID) per video frame [6]. Figure 1.1 shows example trajectory entries for a single frame containing 14 persons and its corresponding frame with overlay drawing of the body positions. Table 1.1a contains following columns from left to right: person ID, frame number, body upper left x coordinate, body upper left y coordinate, body lower right x coordinate and body lower right y coordinate.

0	0	235.925	770.142	371.546	1101.029
1	0	285.748	291.418	370.556	493.414
2	0	286.849	230.501	365.794	416.410
3	0	719.708	220.830	786.893	408.145
4	0	793.722	235.410	861.430	427.627
5	0	876.425	392.142	956.344	628.416
6	0	656.185	994.592	791.581	1387.335
7	0	1607.143	601.733	1717.096	892.799
8	0	1615.588	890.117	1748.101	1277.631
9	0	1790.455	174.285	1877.452	359.453
10	0	1645.396	113.399	1713.707	277.873
11	0	1459.905	28.546	1511.704	167.011
12	0	1361.498	10.463	1408.828	142.648
13	0	890.294	73.968	942.941	220.703

(a) Sample trajectory entries for person IDs 0 to 13 on frame 0 with head and body bounding box coordinates.



(b) Visualization of the body bounding box coordinates and the corresponding person ID.

Figure 1.1: Example of a trajectory data visualization. (Data and image taken from Benfold *et al.* [6].)

This data can be processed later on, for instance, to count people walking in a specific direction on a specific daytime. If the data would be processed in real time it can be used, e.g., to estimate the number of people walking in an airport towards a check in counter and to open an additional one on time.

As there exist various approaches for object tracking Chapter 2 will give an overview and the main motivations behind three chosen tracking approaches:

- Tracking by detection based approaches [3] [6] [9] [18] [19] [20] [33] [36] [37] [39]
- Motion based approaches [14] [38]
- Segmentation based approaches [25] [31] [45].

The main motivation of this thesis is to implement a tracking algorithm suitable for running in real time on a smart camera. The intention of using a smart camera system for object tracking is to have a compact sensor which retrieves trajectory information of objects directly rather than acquiring an image stream from a camera and then to process it on a PC. In environments where tracking information from multiple cameras needs to be extracted those cameras could be replaced by tracking sensors. The main advantage of such a system is that it is easier scalable because the computational effort is distributed among the smart cameras and the network traffic is limited to trajectory information only. Also privacy related concerns can be minimized because only anonymous trajectory data is transferred from the sensors. A disadvantage is the lack of computational power on a smart camera system which makes it difficult to run complex algorithm in real time.

Throughout the analysis of different algorithms the tracking by detection based algorithm presented in Chapter 2.4 (SLR Track) was selected for further investigation and implementation as it's detection algorithm is highly optimized to work in real time on a low computational power device. Although the detector is very fast, the algorithm's main problem lies in the combination of those detections to trajectories especially in situations with objects crossing each others way. As the current algorithm does not use object identity information, it likely mixes up objects passing each others way as presented schematic in Figure 1.2.

To overcome the identity switch problem the idea of including fast to calculate and fast to match object identity information into the tracking process came up. For this per tracked object a description must be calculated and verified on each extension of the trajectory. Therefore, Chapter 3 will investigate feature matching approaches which allow the tracker to verify if it is still tracking the correct object or if an identity switch has occurred. The investigated work covers keypoint feature approaches [1] [10] [22] [35] where interest points of objects are described separately giving several descriptors per object. Also feature descriptors describing the whole object at once [27] [40] [41] giving only a single descriptor per object are reviewed.

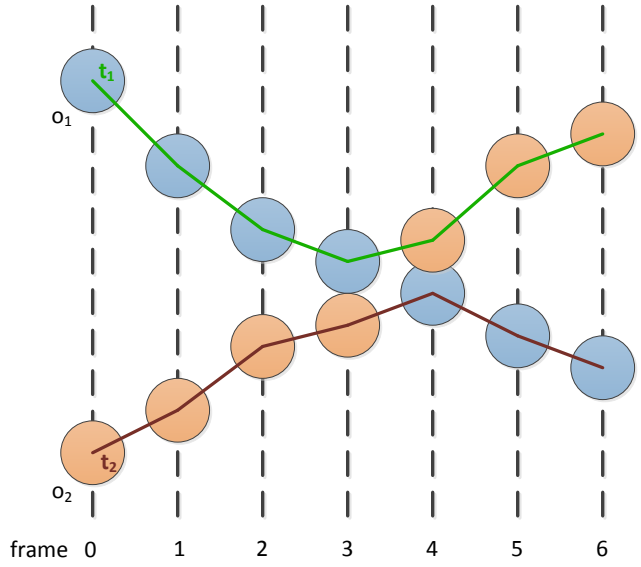


Figure 1.2: Schematic presentation of an identity switch which is likely to occur with the current tracker implementation. The circles (blue and orange) represent objects on their paths through an image sequence (frames 0 to 6). The tracker identifies those objects and tracks them (green and red line). On frame 4 the tracking algorithm mixes up the objects and follows the wrong one. (Adapted from [7].)

Chapter 4 compares different feature matching approaches using publicly available as well as self annotated datasets. The feature matching approaches are tested on their ability to determine between samples belonging to a single object and samples from distinct objects. After that the influence of the verification step, using different feature matching approaches, to the trackers overall quality is reviewed.

Chapter 2

TRACKING

Contents

Tracking of objects in a visual system deals with the problem of identifying and associating objects in subsequent video frames. There are various applications for object tracking like, e.g., tracking of multiple persons inside a shopping center for customer frequency analysis, car and person tracking on a crossroad for statistical analysis etc. Typical difficulties tracking algorithm have to handle are reflections, changing background, varying lightning conditions, moving cameras, deformable objects and full or partly occlusions.

As there are a huge variety of tracking algorithms and describing all in detail would go beyond the scope of this thesis, three most interesting approaches have been selected for detailed investigation:

- Tracking by Detection
- Motion-based Tracking
- Segmentation-based Tracking.

The following sections present different algorithms from the literature to solve the tracking problem. First Tracking by Detection approaches, which rely on combining single object detections into trajectories, are presented followed by motion-based approaches relying on moving pixels between frames and finally segmentation-based approaches working with foreground-background segmentation in consecutive frames for tracking moving objects are reviewed. In the end the current SLR Tracker is described which uses different ideas from the beforehand presented approaches.

2.1 TRACKING BY DETECTION

A broad field of research is provided by Tracking by Detection approaches. Figure 2.1 shows the basic steps done by the algorithms discussed in this section. A frame is analyzed by a detector which yields positions of found objects. Existing trajectories are then combined with the new detections giving at the end a path of the object through the field of view.

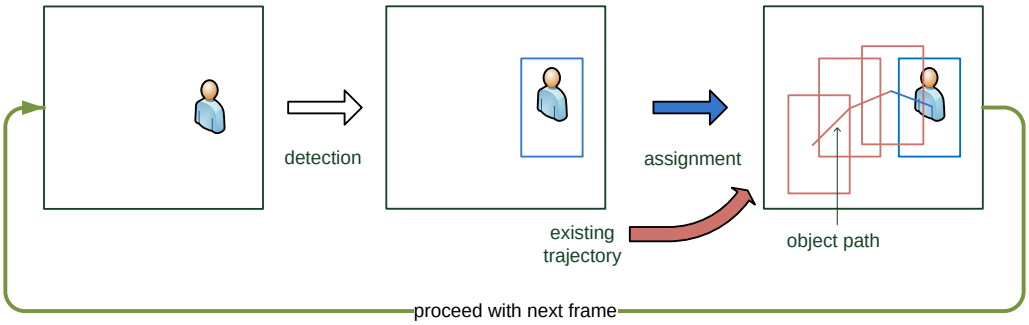
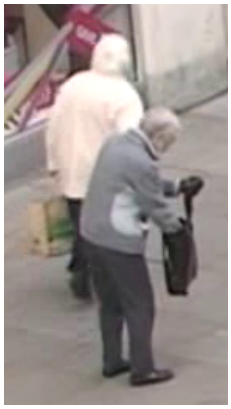


Figure 2.1: Schematic presentation of the Tracking by Detection approach showing one image frame where an object is first detected and afterwards the newly found detection is assigned to an existing trajectory which is the objects path through the image.

The main advantage of such systems is that they are modular which means that the detector or association part can be easily exchanged. A disadvantage is its dependency on the detector output. Whenever the detector yields a false positive or oversees an object in the image, the tracker is miss leaded and erroneous trajectories are generated.

Nevertheless, due to progress in object detection, (e.g., by Dollar *et al.* [15]) the approach of combining detections into trajectories has gotten more and more interesting. As mentioned before Tracking by Detection uses the output of an object detection algorithm which usually describes position, size, scale and sometimes also provides a confidence of the detected object. A challenging part of Tracking by Detection algorithms is to combine those single detections of an object into an object path (trajectory).

This tracking step is named the association problem and is solved by estimating the future state of an object based on the information currently available and then match the newly found detections to the prediction. The problem can be addressed in different ways using only parts of the detector information (like the position and size) or introducing additional information like the direction an object is moving and it's velocity or the objects appearance. Moreover the quality of a Tracking by Detection algorithm is heavily influenced by it's occlusion and false detection handling. When dealing with scenes from, e.g., a crowded town center trying to track people [6] objects are likely to occlude each other (see Figure 2.2a) or disappear behind static foreground objects like trees or something like it which in addition may produce false detections (see Figure 2.2b).



(a) Partly occlusion of two persons.

(b) False positive detection due to background clutter.

Figure 2.2: Possible difficulties arising when detecting people in real world scenarios.

The next sections will give an overview over some ideas to solve the Tracking by Detection problem including learning based approaches, algorithms using Random Forests and finally presenting the current SLR algorithm.

2.1.1 LEARNING BASED TRACKERS

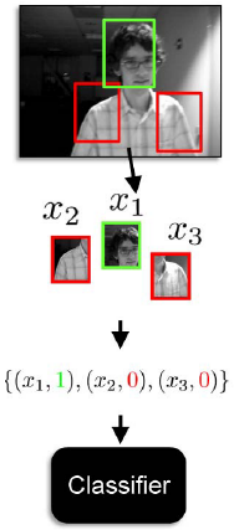
As Ross *et al.* [33] state there are many tracking algorithms performing well in a controlled environment, but often failing when the objects' appearance or the illumination changes significantly. For a really long-term tracking it seems reasonable to not hypothesize a static model describing an object but instead learn such a model over time. Learning based tracking approaches make use of this idea and this chapter is intended to give an insight into this type of algorithms and their main functioning.

The longer an object has to be tracked the more it's appearance may change over time due to varying lightning conditions, transformation of the object itself or variations in the background. This makes it hard for tracking algorithms, which rely on static appearance models, to follow such objects. The approach learning based trackers follow, is to adapt a given model over time to increase the accuracy of the objects representation. The adapted model is then able to find the object in subsequent frames more robustly than a model defined only once before the tracking process (like, e.g., modelling the overall shape of persons) started. Figure 2.3 shows an example of the difficulties when trying to define a static model.

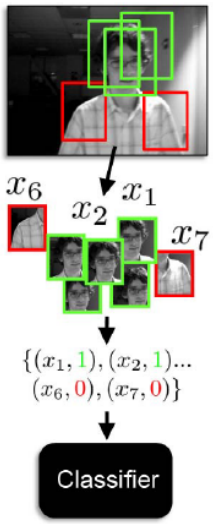
Figure 2.3: This figure shows the difficulty of tracking persons over a long time. As can be seen the shape changes significantly when the person turns.

2.1.1.1 ADAPTIVE APPEARANCE MODEL

According to Babenko *et al.* [3] to design an adaptive appearance model some decisions have to be made. One of them is whether to include the background into the model or just to model the object itself. The idea stems from the fact that some object detection algorithms have shown to perform better when the background is included in their training process. Another decision is how to choose positive and negative examples for updating the model. It is very common to take the trackers location of the object as location for extraction of positive examples and the surrounding of the yielded location to extract negative examples. If, for some reason, the tracker produces a suboptimal object location the model will be updated with inaccurate data decreasing the overall performance. Figure 2.4 shows some example extraction strategies Babenko *et al.* investigated.



(a) Extraction of examples using only a single positive example from the location reported by the tracker.



(b) Extraction of multiple examples around the location reported by the tracker.

Figure 2.4: Two different example extraction strategies to use for updating an adaptive appearance model. (Images taken from [3].)

Additionally Ross *et al.* [33] state that a robust tracking algorithm has to model following two appearance variabilities of an object:

- Intrinsic: Intrinsic appearance variability of an object includes pose variations and shape deformations which are intrinsic because they are object self influenced.

- Extrinsic: Extrinsic appearance variabilities of an object include non object influenced appearance changes. These are illumination changes, camera motion, camera viewpoint and occlusions which change the appearance of an object significantly.

2.1.1.2 UPDATE STRATEGIES

After defining the information included in the adaptive appearance model, one has to cope with the problem of updating such a model.

Kalal *et al.* [20] roughly divide the learning based tracking approaches into two groups based upon their model update strategy:

- "Every-frame-update"
- "Selective-update"

For adaptive trackers they state that the "every-frame-update" is most common. This strategy makes the assumption that the tracker always performs correct and uses every observation to update the objects model with. On the one hand this is an advantage because the learned model adapts very quickly to changes of the objects appearance. On the other hand the assumption of always correct tracking results make the tracker fail more quickly because erroneous results are also included in the objects model update.

The other group defined by Kalal *et al.* , the "selective-update" approaches, assume that the tracker is not always correct. This leads to a more complex update strategy of the model where for instance a semi-supervised framework may be used or the update is only done if the tracker is not too far away from the current model.

2.1.1.3 ALGORITHMS

After defining the basic idea behind the learning based tracking approaches now follow the descriptions of the algorithms capable of performing learning based tracking.

The algorithm from Ross *et al.* models the object to track by using a low dimensional adaptive subspace representation of the target object which leads to a compact representation of the "thing" to be tracked. This compact representation is further on used for object recognition and facilitates it compared to, e.g., models using a set of independent pixels.

The appearance model of choice is an eigenbasis representation which is normally learned off-line from a set of training images. To update the eigenbasis representation of a tracked object the simplest way is to retrain it using newly acquired images of the object. When tracking an object over a long time, this approach will lead to huge memory consumption because all the images of the object have to be stored for re-train. As solution to this problem they make use of an incremental PCA (principal component analysis) algorithm which updates the eigenbasis correctly using one or more additional training data.

To reduce the influence of previously made updates, respectively the current eigenbasis, they introduce a forgetting factor to weight the influence of previously made updates. A forgetting factor is important for learning algorithms because in tracking, recent information are more indicative to the appearance of an object than older ones. When time progresses the history of an object can be very large and without using a method to forget outdated information, the recent (and mostly more accurate information) gets lost in the volume and will finally lead to an inaccurate representation of the current state of the object.

For modeling the location of the object they use particle filters to model their six parameters of an affine transformation (x and y translation, rotation angle, scale, aspect ration and skew direction). A Markov model is then used to predict the most likely object position according to the learned model and the learned object appearance.

In comparison to the algorithm of Ross *et al.* , Kalal *et al.* present their Tracking-Modeling-Detection (TMD) framework which works as follows.

An object is selected in the first frame and a feature vector is extracted from the object. The object has now a representation in image space (the bounding box) and in feature space (the feature vector). The feature space representation serves also as appearance model for the object.Using a short term tracking algorithm the object is

tracked for a few frames giving new image space representations (bounding boxes) of the object. The set of consecutive bounding boxes represents the trajectory of the object in image space. Out of the image space trajectory a trajectory in feature space can be calculated. This feature space trajectory is then analyzed to update to the appearance model using growing events for bounding boxes which are likely to contain the object and pruning events for bounding boxes considered as wrong. This leads to an appearance model of an object as a subspace of the feature space. When considering all possible object representations as a subspace of the feature space, the aim of the TMD tracking system is to converge the current objects feature subspace to the (unknown) subspace of all possible object representations. Figure 2.5 shows the algorithms growing and pruning of the appearance model subspace.

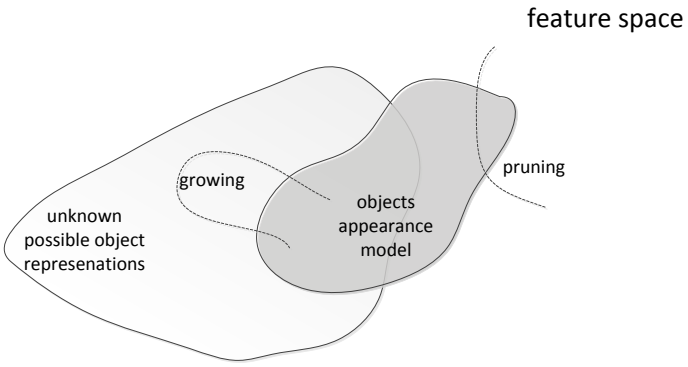


Figure 2.5: The current objects appearance (in feature space) is adapted to meet the subspace of all unknown possible object representations. Therefore growing and pruning events are performed to include correct regions and exclude wrong regions from the current objects appearance (Adapted from [20]).

2.1.2 TRACKING USING RANDOM FORESTS

Random Forests are a widely used learning technique due to their high performance during training and evaluation and are nowadays more often under research in tracking algorithms. This section is intended to give a short introduction on Random Forests

in general and additionally discusses an on-line learning approach for Random Forests, which is very useful when data arrives sequentially (like in tracking applications). Next, the Hough forest approach is explained leading finally to an on-line Hough Forest approach which allows robust tracking of highly deforming objects.

2.1.2.1 RANDOM FOREST

To understand the concept of Random Forests one has to be familiar with the main concept of trees and decision trees which are explained in the following paragraph.

Decision Tree A tree is a specialized graph with no loops and containing hierarchically structured nodes which are connected through edges [12]. A specialized form is a decision tree which defines three types of nodes: Root node, internal or split node and terminal or leaf node. The data a decision tree holds is stored in it's leaf nodes which have no further child nodes. For a given value the internal or split nodes decide which path from the the top (root node) the value takes to arrive at a terminal or leaf node. When using decision trees, normally binary decision trees are used where every node, except the terminal or leaf nodes, has exactly two sub-nodes. Figure 2.6 shows a general binary decision tree where the internal nodes split the data using binary tests.

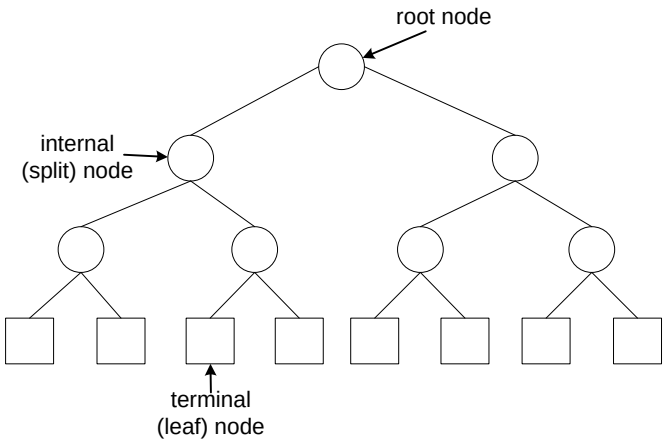


Figure 2.6: An example binary decision tree. The root node is the entry point for each value. The value is then passed through the tree according to the decisions made by the internal or split nodes ending up at a terminal or leaf node.

When using decision trees in computer vision applications the input data normally consists of d-dimensional feature vectors. Given a single data point $\mathbf{v}$ as input to a decision tree, at each split node a splitting test is executed and the input $\mathbf{v}$ is passed to the left or right child node. Finally, the input $\mathbf{v}$ reaches a leaf node where, e.g., in case of a classifier, a class label is associated to $\mathbf{v}$. Figure 2.7 illustrates the path of an input $\mathbf{v}$ through the tree ending up in a leaf node.

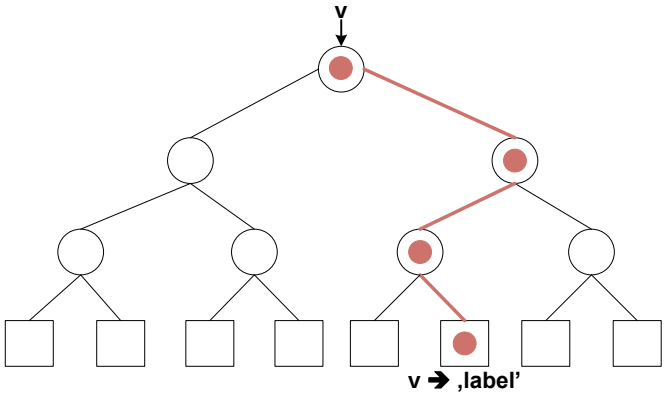


Figure 2.7: A binary decision tree showing a classification example where to an input value $\mathbf{v}$ a label *label* is assigned. The input $\mathbf{v}$ is passed through the nodes using binary tests ending up at a terminal or leaf node. The terminal or leaf node assigns the label **label** to $\mathbf{v}$.

In order to split data in a reasonable way decision trees have to learn which are the best tests at each split node. This training of a tree happens off-line using existing training data and its ground truth. The training phase optimizes the parameters used by the split functions and grows the tree. A criterion to optimize could be for instance the information gain.

Definition 1 *The information gain measures the quality of a split by comparing the entropy (amount of random information) before splitting the data and after performing the split. The lower the entropy in the resulting two subsets, the higher the information gain [12].*

The training of a single tree and using it as a classifier, as shown before, will produce good results under some constraints. To improve the robustness of decision trees, multiple trees (a forest) can be used to vote separately for an output. When they are trained randomly the whole system improves further. This thoughts lead to the idea of Random Forests.

Random Forest A Random Forest consists of randomly trained decision trees [12] and is defined by following components:

- Weak learners (family of split functions)
- Training objective function
- Leaf predictor
- Randomness model.

The following paragraphs give a short overview over the components of Random Forests.

Weak learners As every split node has to decide whether to pass arriving data to the left or right child node, for each split node a binary split function

$$h(\mathrm{v}, \theta_j) \in \{0, 1\} \tag{2.1}$$

is defined, where j indicates the split node, $\mathbf{v}$ indicates an input and θ indicates the parameters. θ consists of the three parameters ϕ (a filter function to select some features from the input $\mathbf{v}$), ψ (a geometric primitive to separate the data) and τ (the thresholds which are used in the binary testing function). Depending on the output of the split function h, the input $\mathbf{v}$ is sent, as described before, to the left or right child-node.

For a linear data separation an example weak learner, where τ_1 is the upper and τ_2 the lower threshold, would be defined as

$$h(\mathrm{v}, \theta_j) = [\tau_1 > \phi(\mathrm{v}) \cdot \psi > \tau_2]. \tag{2.2}$$

For further details on weak learners take a look at [12].

Training objective function The definition of the parameters of a split function is not feasible. Therefore, a training process is needed to automatically determine (learn) the optimal parameters. This can, e.g., be achieved by maximizing the information gain (see Def. 1) for the split.

The optimal parameters θ_j^* of the split node with index j are defined as follows:

$$\theta_j^* = \arg\max_{\theta_j} I_j, \tag{2.3}$$

where I is the information gain function defining the information gain of the j^th split node as

$$I_j = I(S_j, S_j^L, S_j^R, \theta_j). \tag{2.4}$$

The maximization operation can be achieved by simply performing an exhaustive search operation.

Leaf predictor To understand the functionality of a leaf predictor an example for a classification tree is given where a leaf stores the empirical distribution over the classes associated to the subset of training data that has reached that leaf [12]. This leads to following formula for the predictor probability model of the t^{th} tree:

$$p_t(c|\mathrm{v}), \tag{2.5}$$

where c is the label of the class assigned to the input v. So the predictor probability p_t tells us the likelihood the t^{th} tree has assigning the class c to input v.

Randomness Model The randomness of forests is achieved during the training phase by either randomly sampling the training data or randomly optimizing the split nodes. Therefore the single trees of a random forest are randomly different from each other which leads to de-correlation of the individual tree predictions. Furthermore having randomly different predictions from individual trees help to improve the generalization, lead to higher robustness and make the Random Forest less sensitive to noisy data.

Other Properties of Random Forests

- Ensemble model: For testing a sample it is traversed down all of the trees of a forest until a leaf is reached. To obtain an overall forest prediction for instance a simple averaging over all predictions can be done. Another possible ensemble model would be to multiply the outputs of each tree together. Both methods can be used, but the advantage of averaging is that the possible noise of the tree outputs can be reduced.
- Stopping criteria: defines when to stop the tree growing in the training phase. Common criteria would be to stop when reaching a given depth, when a node contains less than a certain number of training points or by imposing a minimum information gain (see Def. 1).

2.1.2.2 ON-LINE RANDOM FOREST

Because Random Forests have shown remarkable results in the field of computer vision [8], the idea of an on-line Random Forest approach came up. Saffari *et al.* [36] denote on-line as a learning method with no additional memory consumption because after learning the sample is discarded unlike in incremental learning where the samples would be stored. In contrast to the usual usage of a random forest which is trained off-line needing a huge amount of previously known training data, the on-line approach does not need any data in advance.

To lead over from an off-line to an on-line approach Saffari *et al.* define two main problems. The first is how to do bagging in on-line mode and the second is how to grow the trees on-the-fly.

For bagging they stick to the approach of Oza *et al.* [28] where each tree is updated for every sequentially arriving sample k times where k is estimated according to a Poisson distribution.

For on-line tree growing several methods exist like ETrees [29] or Hoeffding trees [16]. The solution discussed here will present the method of Saffari *et al.* in more detail because they state that their method fits better to the inherent nature of decision trees. This is achieved by adding for instance a continuous measuring of the information gain for a potential split node. Their algorithm works as follows:

1. The tree growing starts with only one root node holding a set of randomly selected tests.

2. A node only splits if

   - the minimum number of samples ended up in the node is reached or
   - the minimum information gain a split has to achieve is reached.

3. If a node is splitted, the collected statistics of the split node are propagated to the leafs, so the leaf nodes can perform classification right away without the need of observing new samples.

2.1.2.3 HOUGH FOREST

Gall *et al.* [18] combine the idea of *local appearance codebooks* and *Hough transform* for object detection. The approach can be used for Object Detection, Tracking and Action Recognition. In this section the focus lies on the usage of Hough Forests in object detection and tracking.

The Hough part of Hough Forests refers to the "Hough transform" used in this approach which is simply a detection process relying on additive aggregation of evidences, so called "Hough votes". These "Hough votes" are created from local elements of the image. The aggregation is calculated in a parametric space called "Hough space" where each point corresponds to an instance in a particular configuration.

Hough Forests combine "Hough votes" of local image elements with Random Forests where the local image elements arriving at a leaf, make a probabilistic vote in the Hough space (see Figure 2.8a and Figure 2.8b).

The training of Hough Forests can be summarized as follows:

1. For each class a set of training samples (positive and negative ones) must be available.

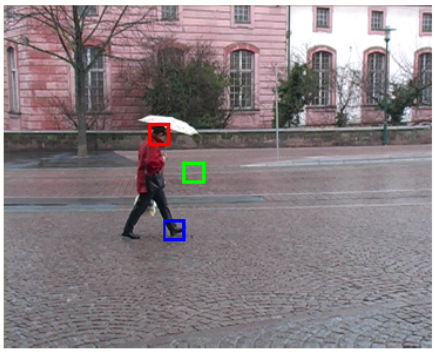
2. For positive training samples a bounding box must be provided to determine center and size of the patch.

3. All trees are constructed by randomly picking image patches (normally using the size of 16×16) from each sample. The trees' training is performed by extracting features from the image patches. The tests in the split nodes are then optimized to propagate the samples to either the left or the right child according to the split nodes criteria to optimize.

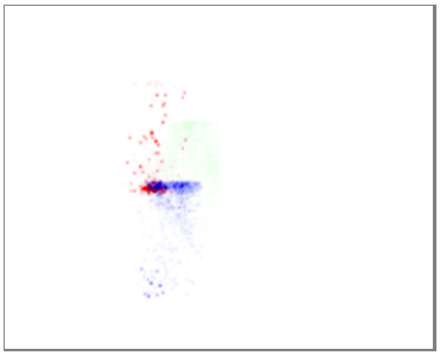
4. The leaf nodes finally store the probability of the trained image patches belonging to a specific class and a displacement vector of the patch to the objects center.

5. The training is stopped if one of the stopping criteria is met. The criteria are a maximum depth of a tree which must not be exceeded and a minimum number of image patches which have to end up in a node to split it.

The detection process passes image patches through each tree of the learned Hough forest. The image patches arrive at leaves which are then used to vote to the Hough space. The votes coming from the different leaves produce local maxima in the Hough space. These local maxima are used for object position determination. Figure 2.8 shows this process by an example.

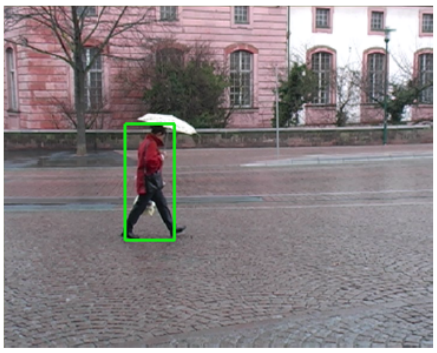


(a) Three input patches to analyze using a Hough forest.



(b) A pedestrian Hough forest votes a position for each patch.

(c) The single votes are aggregated into a single result.



(d) Finally the position of a pedestrian can be extracted.

Figure 2.8: This figure shows the process of detecting a person using Hough Votes(taken from [18]).

2.1.2.4 ON-LINE HOUGH FOREST

The idea from Schulter *et al.* [37] of using Hough Forests in combination on-line forest update leaded to On-line Hough Forests. Therefore, an on-line learning approach similar to the one explained in Sec. 2.1.2.2 is combined with a local appearance codebook Hough voting based Random Forest as explained in Sec. 2.1.2.3.

On-line growing The on-line ability of the approach is achieved by growing the tree as the image data arrives which forces the bagging and the tree growing to work on-line. As mentioned in Sec. 2.1.2.2 the bagging is also done by updating all trees k times where k is estimated using a Poisson distribution. The growing starts with a single root node. The samples arriving from the subsequent frames are then propagated through the trees leaf nodes. The leaf nodes update their statistics if less than n samples have arrived. If n is exceeded the node produces a set of random splitting functions and the split function which optimizes either the information gain (for classification nodes) or the variance of the object center vector (for regression nodes) is chosen turning this leaf node into a split node. The samples collected in this node are propagated to the left and right child node according to the found split function. The tree growing is stopped if a maximum depth is reached.

Tracking application For their tracking approach Schulter *et al.* perform a so called "one-shot" learning where they rely on the tracked object to be labelled by hand in the first frame of a sequence. This initial image is then virtually warped five times to generate in total 100 positive and 500 negative patches to update the On-line Hough Forest model. For the subsequent frame a search area twice the size of the bounding box of the labelled object from the first frame is used to find a location l with a confidence c for the object.

To cover the scale change of objects Schulter *et al.* simply calculate additional Hough images using scales relative to the scale of the object in the current frame. According to the confidence c the bounding box center of the object and its scale is updated, the Hough forest model is updated, both are updated or neither the model nor the position and scale are updated.

To solve an additional problem when tracking non-rigid, previously unknown objects, where a bounding box representation might not represent the objects shape accurately and therefore will lead to tracking problems, Godec *et al.* [19] introduce the HoughTrack approach. The disadvantage of the information current approaches use is that the background is included in the learning process and the detector is more likely to drift. A solution to this problem is to make a rough foreground-background segmentation of the object using GrabCut leading to an accurate description of the objects shape.

2.2 MOTION-BASED TRACKING

In addition to tracking approaches relying on an object detector yielding positions of objects in a frame, motion-based approaches rely only on movements in the image sequence. Analyzing motion for object tracking has many applications including video surveillance (especially for tracking of deformable objects), motion analysis and extraction for computer animation, human computer interface (HCI) and object-based video compression [38].

A simple way to track objects using motion is to track regions of differences in consecutive frames using adaptive background generation and subtraction which is efficient when tracking in low-noise environments. Having noisy background or even moving cameras this method often fails due to the erroneous background generation and subtraction.

Another approach from Shin *et al.* [38] is a blob tracking algorithm. Blob tracking uses simple geometric models like ellipses or rectangles to track the centroid of an object again assuming a stationary background for this kind of approach. Additionally, to get more robust results, they mention shape-based tracking algorithms. These algorithms use a priori shape information gained from, e.g., an active shape model or an active contour model and project them onto the closest shape in a given frame. The shape-based approaches are able to deal with partial occlusion but their disadvantages are first the a priori training of a shape model and second the iterative modelling procedure for convergence. The first problem often leads to difficulties when tracking highly deformable objects and the second leads to high challenges when implementing a real-time tracker because of the computational complexity.

A very basic and widely known motion tracking algorithm is the Kanade-Lucas-Tomasi (KLT) [42] point tracking algorithm. The next section will describe it in more detail.

2.2.1 KLT

The algorithm of Kanade-Lucas-Tomasi (KLT) [42] relies on a point tracking method presented in detail by Tomasi *et al.* They state that images taken at near time instants are strongly related to each other. This correlation is expressed through moving patterns in an image stream.

This correlation is expressed by Tomasi *et al.* for two subsequent images I from an image stream the following way:

$$I(x, y, t + \tau) = I(x - \xi, y - \eta, t), \tag{2.6}$$

where x and y are coordinates of a point inside the image, t is the image time, τ is the time difference between the subsequent images and ξ and η define the amount of motion or displacement. The displacement d of a point is therefore defined as follows:

$$d = (\xi, \eta). \tag{2.7}$$

In words, an image taken at $t + \tau$ can be expressed by moving every point of an image taken at time t by the amount of the displacement d.

The problem to solve is to find the displacement d of a point for two subsequent frames. For a single pixel this would only be possible if its brightness is very distinctive compared to its surrounding pixels. As one can imagine in natural scenes with occlusions, changing lightning conditions, objects entering and leaving the field of view it is very hard to find a single pixel in subsequent frames. To be independent from single pixels, the tracking is performed using windows of pixels with sufficient texture. Windows are tracked from frame to frame only if their appearance has not changed too much to be sure to track the correct window. A problem arising from window tracking is the displacement inaccuracy of pixel displacements inside a window when considering the windows' displacement. To overcome this problem instead of a displacement a more complex window transformation is used, e.g., an affine map. Using this, different pixels of the window can be assigned to different velocities. The drawback of this approach is

that complex transformation representations tend to over-parametrize the system and require larger windows to track. Therefore Tomasi *et al.* use only two parameters for their window transformation model, the displacement vector d (formed by the two parameters $delata_x$ and $delta_y$) for small windows. The formal redefinition is as follows:

$$J(\mathbf{x}) = I(x, y, t + \tau), I(\mathbf{x} - d) = I(x - \xi, y - \eta, t), J(\mathbf{x}) = I(\mathbf{x} - d) + n(\mathbf{x}), \quad (2.8)$$

where $\mathbf{x}$ is the point and n is the noise. Tomasi *et al.* dropped the time variable for brevity as can be seen.

To then choose the displacement vector d the residue error ϵ in the window W is minimized:

$$\epsilon = \int_W |I(\mathbf{x} - d) - J(\mathbf{x})|^2 \omega \mathrm{d}\mathbf{x}, \quad (2.9)$$

where ω is a weighting function which, in the simplest case, could be set to 1. Using a Gaussian-like weighting function for instance would emphasize the central area of the window.

2.2.2 MOTION EXTRACTION

As discussed before motion based tracking approaches deal with moving objects in an image stream. Therefore, the motion information has to be extracted which can be done in several ways. Two of this approaches are presented in the following paragraphs.

For motion extraction out of subsequent frames Shin *et al.* rely on the Lucas-Kanade optical flow algorithm especially - to achieve real-time performance - the Lucas-Kanade method based on block motion model. The resulting motion regions are then clustered into four directions - up, down, left, right. Because motion information is very noisy Shin *et al.* use morphological operations to overcome this problem and are then able to extract an object as shown in Figure 2.9.

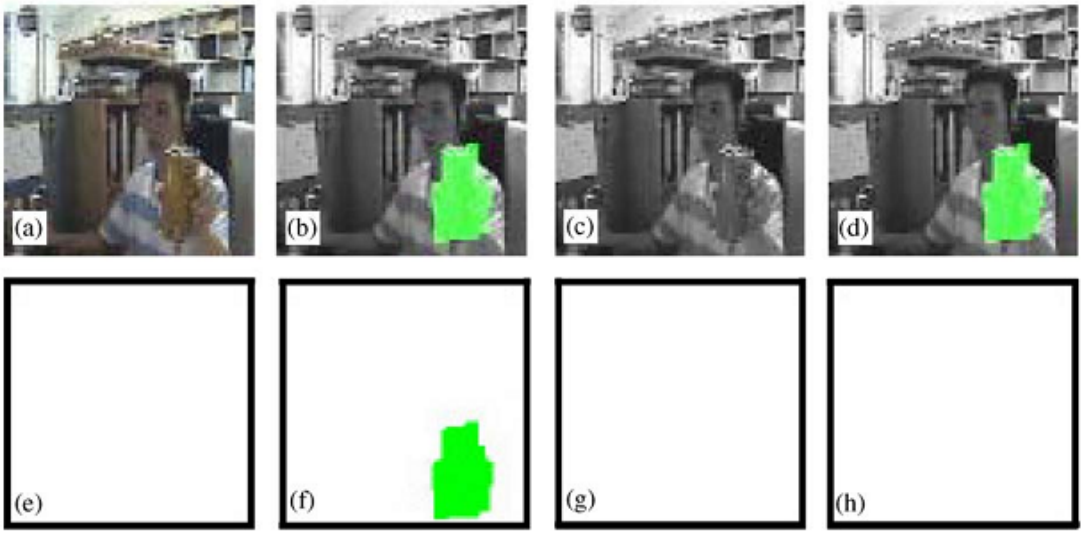


Figure 2.9: Extracted motion in upward direction (green) after morphological processing (Extracted from [38]).

Denman *et al.* [14] use a different technique to extract motion from the frames. They rely on an algorithm proposed by Butler *et al.* which is an adaptive background subtraction method. The algorithm clusters the pixels according to their luminance and chrominance giving a multi-modal distribution for each pixel. The distributions are stored in a background model and matched to a given frame. Applying a threshold on the matching result then gives the information of a change (= motion). The clusters are updated over time to adapt to changes in the background through varying lightning conditions or new background objects. When a pixel yields a motion the optical flow for that pixel is extracted by examining the surrounding. The size of the analyzed area is given by the maximum allowed acceleration – in pixels for x and y direction – the user selects. The analysis is done from the center pixel to the outside in rings giving the flow in integer precision.

2.2.3 TRACKING

After having extracted the foreground objects using motion information, the objects have to be tracked over various frames. For this two approaches have been investigated in more detail.

The approach Shin *et al.* follow in their paper is a feature-based approach. They make use of the non-prior training active feature model (NPT-AFM). Their algorithm extracts moving objects using the motion between subsequent frames. Out of these objects feature points are extracted which are then predicted using a spatio-temporal prediction algorithm. Additionally a feature correction process restores missing or failed tracking processes. In advantage to shape-based methods the feature-based method is capable of tracking objects without any constraints of camera position, object motion or complicated background because the feature points are assigned inside the object and not at the near boundary.

Denman *et al.* present a person detector and tracker in their paper which uses motion detection or optical flow to detect people. For detecting people using optical flow the velocity has to bee predicted which is only possible for persons tracked for sufficient time. Therefore, the initial detection and the tracking of a person in the first frames is performed using motion detection. Later on the detection is performed using optical flow where a person is segmented out of a frame using its expected horizontal and vertical movement. To do this reliably a persons position and average flow has to be observed over a minimum number of frames. The detection process is included in an existing tracking system which is presented by Denman*et al.* in [11].

2.3 SEGMENTATION-BASED TRACKING

Because tracking is very challenging due to variation of the objects shape, appearance and scale, several approaches for tracking objects using foreground/background segmentation techniques were developed. As motivation for segmentation-based tracking approaches, Ren *et al.* [31] point out the inaccurate representation of detected objects a lot of tracking algorithms have. They often rely on rectangular or elliptical bounding boxes, which may be sufficient for the shape of faces or cars but lack when representing the shape of a moving person, especially in sport scenes where non-rigid transformations are likely to occur.

Tracking algorithms have to maintain an appearance model to identify **which** object is tracked and a spatial model to describe **where** the object is located. This algorithms often maintain additional information like the scale of an object or a background model. According to Ren *et al.* if a trackers spatial model holds an accurate support mask, describing the shape of the tracked object in detail, it will be able to predict future positions of the object more reliably. An appearance model will also improve its behavior in representing the object using an accurate support mask because the background clutter rectangular representations may include is greatly reduced.

To give a short introduction into the field of segmentation-based tracking the algorithms of Ren *et al.* [31] and Lu *et al.* [25] are investigated in more detail in the following paragraphs.

2.3.1 "SUPERPIXEL" EXTRACTION

For their segmentation based tracking approach Ren *et al.* and Lu *et al.* both start with a preprocessing step on the frames by dividing them into "superpixels". This step is executed to reduce computational complexity and to increase robustness due to the consistency enforced inside a "superpixel".

Firstly, the approach Ren *et al.* use will be described. They use a three stage approach doing the following: First they compute a soft boundary map using local brightness, color and texture contrast. Then they apply a fast image partitioning technique to build an approximation of the boundary map. Finally, the CDT (constrained Delaunay triangulation) is calculated which produces triangle shaped "superpixels".

An example of such "superpixels" is shown in Figure 2.10 where triangular shaped "superpixels" are extracted.

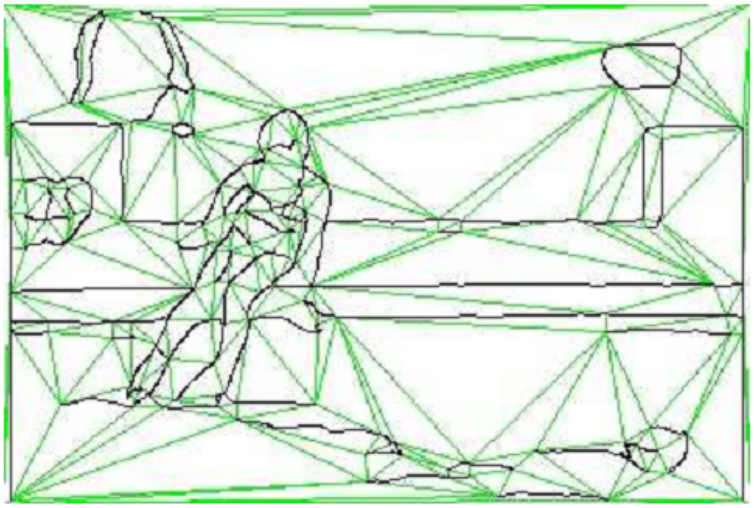


Figure 2.10: Triangular shaped "superpixel" creation (Extracted from [31]).

The approach of Lu *et al.* to generate their "superpixels" on the other hand relies on the graph-based method from [17].

2.3.2 FIGURE/GROUND SEGMENTATION

After preprocessing the image Ren *et al.* and Lu *et al.* do a figure/ground segmentation where each "superpixel" is either labeled to belong to the background or to the object (the figure).

The approach of Ren *et al.* is based on Conditional Random Fields (CRF) which is also used by Yin *et al.* in their work [45]. The following paragraph will shortly explain how the figure/ground segmentation using a CRF works:

With I_i denoting sets of image "superpixels" and s_i the corresponding labels where $s_i = 1$ if I_i belongs to the foreground and $s_i = -1$ otherwise. Giving a graph A, B where A denotes pixel node labels s and B including all links between neighboring nodes. When conditioning on the observation I the joint distribution over label s is

$$P(s|I) = \frac{1}{Z} exp \left\{ \sum_{i \in A} \sum_{k} \lambda_k \Phi_k(s_i, I) + \sum_{(i,j) \in B} \Psi(s_i, s_j, I) \right\} \propto \frac{1}{Z} exp\{-E\}, \qquad (2.10)$$

where Z is a normalization factor and E denotes the energy function to be minimized. Φ_k denote data association potentials generated by different segmentation cues which are linearly combined using the weights λ_k. Finally Ψ represents the pairwise interaction potential between neighboring nodes.

2.3.3 TRACKING

After successfully segmenting background and foreground the next step is to track the foreground object throughout a sequence of frames.

Ren *et al.* maintain during their tracking approach three models of temporal coherence: scale, appearance and spatial support, which are updated after every frame and represent the internal state of the tracker. The three models consist of following parameters:

1. Scale: size of the object in pixels, median horizontal distance to the object center and median vertical distance to the object center.
2. Appearance: brightness or color distribution as histograms in the RGB space for the object (= foreground) and the background.
3. Spatial model: correspondence between superpixels from the current frame and the previous frame leading to a linear transportation problem which has to be solved.

On the other hand Lu *et al.* follow a different approach by defining two tracking algorithms namely "Location Tracking" and "Segmentation Tracking". Both rely on first classifying image patches and second updating models using the newly classified patches. The approach described here is the "Segmentation Tracking".

Starting with one or more annotated frames at the beginning of the sequence where the foreground and the background is labeled, a model is created. The subsequent frames are then sampled into segments ("superpixels") and the obtained segments are then classified using the model obtained from the labeled data.

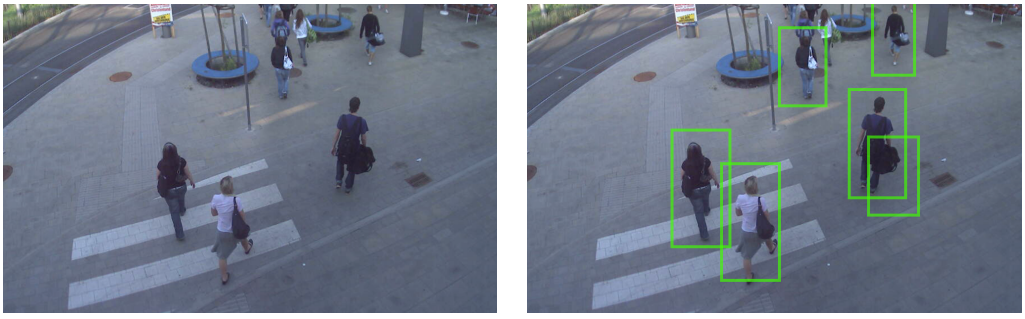
2.4 SLR TRACKER

The SLR tracking algorithm is based on a Histogram of Oriented Gradients (HOG) detector (see Sec. 2.4.2) which continuously yields detections (bounding box positions) to ensure that all objects in the field of view are recognized.

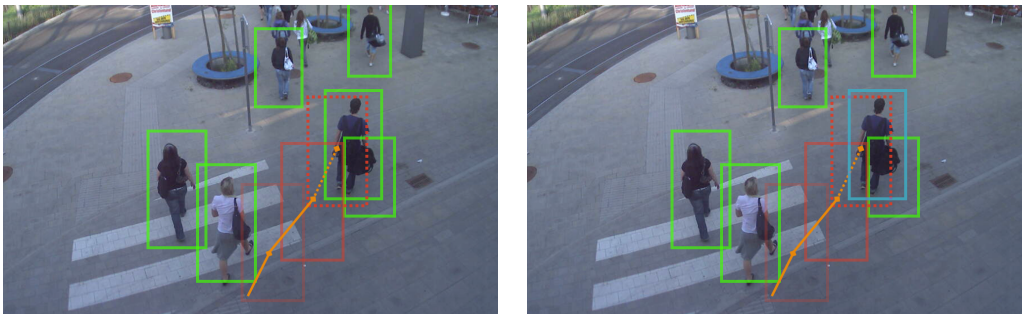
To combine the bounding box object positions yielded by the detector in every frame into trajectories, the Kanade Lucas Tomasi (KLT) point tracking (see Sec. 2.2.1) is used for estimating the objects movement direction and speed. For existing bounding boxes in combination with the KLT information, a prediction bounding box is calculated, giving the estimation of the objects location in the next frame.

Detections in the next frame are then matched against the calculated predictions using an overlap criterion of the bounding boxes. To speed up the detection process and to achieve the desired real-time capability up to a resolution of 1024x768 pixels, Sidla *et al.* implement a simple background model (see Sec. 2.4.1) which masks all uninteresting regions of the current frame. The detector is then only executed on the foreground regions yielding bounding boxes of the objects in the current frame.

For better understanding of the main part of the tracking process, Figure 2.11 shows an example of the SLR tracking algorithm where an existing trajectory is successfully matched against a newly yielded detection.



(a) (b)



(c) (d)

Figure 2.11: Tracking process using the SLR Tracker: First from the input frame (a) the HOG detections are extracted (b). For existing trajectories the predicted bounding box is shown in (c). Figure (d) shows the matching of a new HOG detection with a prediction. (Images taken from [23].)

If for an existing trajectory the prediction is not matched against a detection, the tracker is able to continue regarding the object for some frames but the object is marked as *tentative*. When a new HOG detection is successfully assigned to a *tentative* trajectory, the state is reverted to *tracked*. This mechanism lets the tracker fill gaps between HOG detections of an object which may occur due to occlusions, changing lightning conditions, etc.

The tracker counts for every trajectory the number of interpolated positions (positions with no matched HOG detections) and the number of the HOG detector positions. From that a quality measurement for each trajectory can be calculated, which is defined as the ratio of interpolated positions relative to the actual HOG detections. If this measurement falls beneath a certain value, the trajectory is regarded as not robust enough and therefore closed.

Next follows the detailed description of all parts of the SLR tracker starting with the background model, continuing with the detector and finishing with the tracker.

2.4.1 BACKGROUND MODEL

As mentioned before, the SLR Tracking approach is real-time capable. To achieve this, the HOG detector is only applied on image parts with current movement defined by an adaptive background model. Sidla *et al.* review two typical background modelling options which are standard for surveillance systems (which are the main field of operation

for the SLR Tracking approach). In opposite to the computational expensive Gaussian Mixture Models (and variations of them), they decided to implement image blending operations which update the background BG slowly with content from a frame F. The background model is defined as follows:

$$BG_T = (1 - \alpha) * BG_{T-1} + \alpha * F_T, \tag{2.11}$$

where BG_{T-1} is the current background model, F_T is the current frame, α is a weighting constant defining the update rate and BG_T is the new updated background model. The obtained background model BG_T is then used to generate a binary mask which speeds up the detection process by masking static background regions. Figure 2.12 shows some example frames and their binary masks. The white image regions are the foreground where the HOG detector is applied on whereas the black masked regions are ignored by the detector.

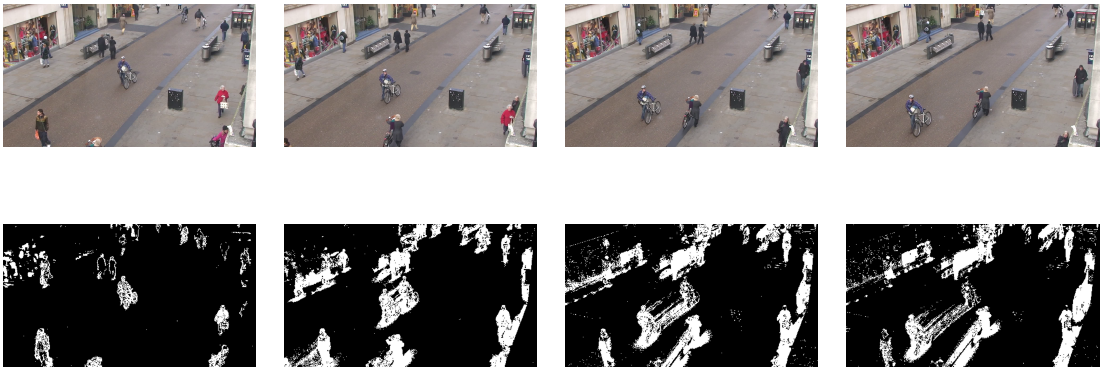


Figure 2.12: Example background models for a crowded scene. The leftmost column shows the initial frame and its corresponding background model. The columns two to four visualize the update process of the model over time.

2.4.2 DETECTOR

The applied detector builds on the pedestrian detection framework of Dalal and Triggs [13], where HOG features are extracted in a dense grid using a sliding window from an image and classified into person and non-person windows using a support vector machine (SVM). The next paragraphs describe their detection framework in detail.

Histogram of Oriented Gradients To describe objects, Dalal and Triggs make use of gradient orientations inside a rectangular bounding box. In their approach they divide an image window into *cells* where for each *cell* a 1-D histogram of gradient directions or edge orientations is built. These histograms are then combined to form the representation of the object.

The main idea is that the shape of an object can be characterized by the distribution of local intensity gradients or edge directions even without knowing the exact location of the gradient or edge. To extract a feature vector for a detection window, a dense overlapping grid of HOG descriptors are combined forming the feature descriptor. For performance improvements in case of changing illumination environments, Dalal and Triggs recommend to contrast-normalize every *block* (a larger spatial region formed by a number of *cells*). These contrast-normalized *blocks* are referred to as *Histogram of Oriented Gradient (HOG)* descriptors.

Detection process For detection Dalal and Triggs move the detection window (sliding window) over the input frame and classify each window position as object or non-object position. The detector's result is the positions of every object classified sliding window position.

For classification of the detection windows a linear Support Vector Machine (SVM) is applied. The SVM requires a number of object and non-object (background) samples for training. The trained SVM is then used in the detection process to classify the detection windows into object or non-object window.

They have also defined a default detector using the RGB color space, a gradient filter $[-1, 0, 1]$ without smoothing, histogram of gradients using angles from $0° - 180°$ divided into 9 bins (giving $20°$ per bin), 8×8 *pixels* cell size, 4 cells per block, overlapping of blocks by 1 cell giving 105 blocks forming an image window of 64×128 *pixels*. Figure 2.13 illustrates this default detector configuration. The default detectors extracted feature descriptor has a length of $9\ bins \times 4\ cells \times 105\ blocks = 3780$, which is classified using a linear SVM into person or non-person as mentioned earlier.

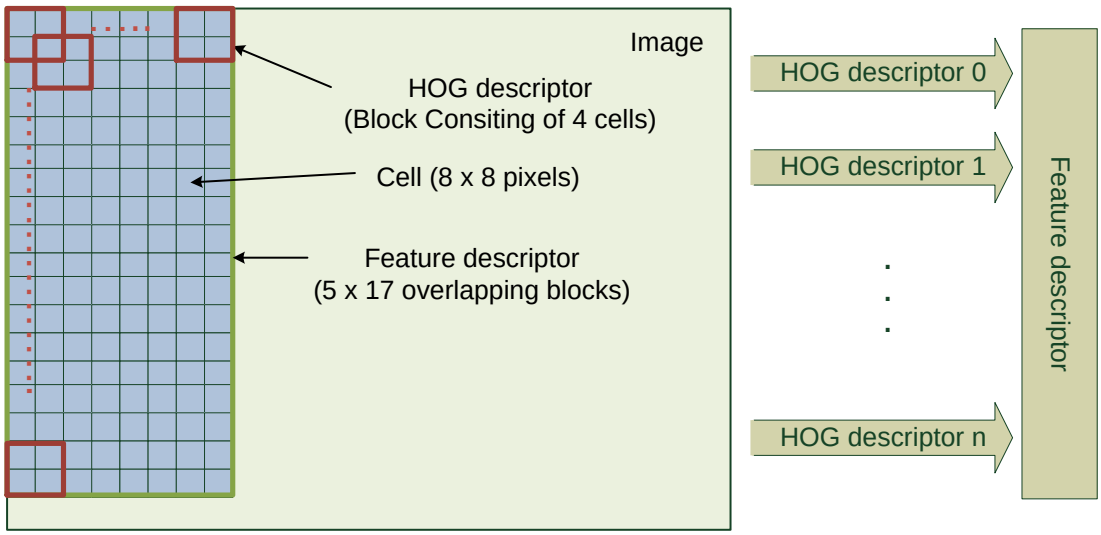


Figure 2.13: Default HOG detector dividing an image into windows of the size of 64×128 pixels using 105 blocks, 4 cells per block and 64 pixels per cell.

Multi-Scale and Multi-Stage HOG Using the default detector on a given video stream, for instance from a surveillance camera, detecting persons with only one scale, e.g., a 64×128 window, will not lead to satisfying results. Therefore, the image has to be analyzed in different scales which is usually achieved by constructing an image pyramid and applying the detection algorithm on every stage of the pyramid separately. This approach leads to a high increase of computational costs (Zhu *et al.* [47]). To overcome this drawback, we can apply a cascade of classifiers with different block sizes, which is inspired by the work of Viola *et al.* [43].

This cascaded classifier approach is adapted from Sidla [39]. The detector is split up into 8 stages using different number of blocks giving different numbers of features to classify in a SVM. Early stages use less blocks (and features) and a linear SVM to eliminate background samples as fast as possible. The last stage, using a slow and accurate polynomial SVM, is left with only a few samples. Table 2.1 shows the used HOG configuration for every SVM stage.

Stage	# Blocks	# Features	Classifier
1	1	36	linear SVM
2	2	72	linear SVM
3	6	216	linear SVM
4	4	144	linear SVM
5	15	540	linear SVM
6	26	936	linear SVM
7	15	540	linear SVM
8	17	612	polynomial SVM

Table 2.1: HOG configuration used for every stage of the SVM classifier which classifies image windows into person and non-person windows (taken from [39]).

2.4.3 TRACKER

On the first frame the HOG detector is applied and new trajectories are initialized using this newly found detections. To extend the trajectories in subsequent frames, the Kanade Lucas Tomasi (KLT) point tracking (see Sec. 2.2.1) is applied to get a motion estimation (speed and direction) of every trajectory (see Figure 2.15). For estimating the position of the trajectory in the next frame using KLT only points lying in the middle of the trajectory bounding box are selected. With the selected KLT points a new position of the trajectory is predicted. In the subsequent frame the HOG detector then yields new detections and the positions of the newly found detections are matched to the predicted ones using a position overlap measure. Figure 2.14 shows this process for a single object.

Additionally to the matching of positions for extending trajectories shown above, DCT features are computed for existing trajectories and close enough detections to verify the position match. If the position match is valid, the trajectory is extended. Figure 2.15 shows an example trajectory and KLT output produced by the SLR tracking algorithm.

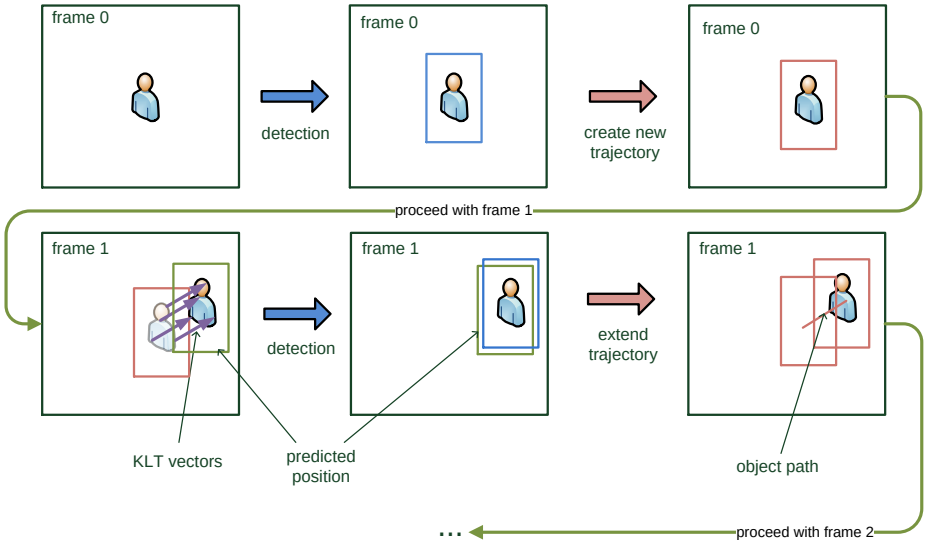


Figure 2.14: The first row shows the process of the SLR Tracker when a new object appears. With this new detection a new trajectory is initialized. In the second row, the prediction and association step of the SLR Tracker is shown. First the KLT vectors are calculated giving a predicted position of the object in the current frame. A detection obtained by the HOG detector is then matched to the prediction. When an overlap of the predicted bounding box and the detection bounding box is above a given threshold, the trajectory is extended by the new detection.

Figure 2.15: Sample SLR-Tracker output showing the KLT trajectories in blue used for bounding box position prediction and a HOG detector confidence drawn below the bounding box.

To handle missing detections and false positives a quality measurement for trajectories is introduced. This quality measurement combines the length of a trajectory and the number of missing detections for trajectory positions to determine if a trajectory

is valid or not. The quality of a trajectory is defined as follows:

$$q = n_{inter}/l_{traj}, \tag{2.12}$$

where n_{inter} is the number of positions of the trajectory where no valid detection is present and l_{traj} is the total length of the trajectory.

If the detector fails the tracker uses only the predicted position (which increase the interpolation counter n_{inter}) to update the trajectory which decreases the quality. If the quality drops below a defined threshold the trajectory is marked unreliable and is finally removed.

2.5 DISCUSSION

The discussed algorithms in this section give an overview over the different approaches to solve the tracking problem. While some approaches focus on first finding the objects and then combining those single findings with additional information into trajectories, others rely on motion information or separating objects from it's background. As the aim of this thesis is to track multiple objects in real time on a low computational device (smart camera), with the focus on creating a object tracking sensor, some of the algorithms discussed will not fit these needs.

First the algorithms where the objects of interest have to be defined by hand in the first frames are not suitable. Although these algorithms seem very interesting because of their capability of adapting dynamically to the objects appearance like presented in Chapter 2.1.1.3 and Chapter 2.1.2.4 they will not fit to be implemented as the smart cameras should work independently from user input.

Another problem arising when selecting a suitable algorithm for a smart camera is the beforehand mentioned low computational power. The selected algorithm is the SLR Track (see Chapter 2.4) because it's multi-scale and multi-stage HOG detector in combination with their simple background model implementation and their association part based upon KLT vectors runs on a 3Ghz Intel Core 2 pc with 30 frames per second at 640x480 pixels [39].

The high processing speed of the SLR Track has one drawback, it's association part. It works good in simple situations where objects do not occlude each others but fails in more difficult situations with inter-object occlusions. This is because it relies on motion estimation using the KLT algorithm only, which fails on occluded objects. To overcome this drawback, the idea of including additional information into the association part came up to describe single objects and to be able to do a occlusion detection and a tracking verification. As there exist a lot of feature descriptor algorithms for identifying objects, the next chapter presents the methods which were analysed for their ability to improve the association part.

Chapter 3

TRACKING VERIFICATION USING FEATURE MATCHING

Contents

As the output of a tracker might not yield correct results due to false detections etc., this chapter focuses on approaches for verifying the output of tracking algorithms. For this, a feature matching approach would be a valid choice to determine if the tracker still sticks on the same object as in the previous frames. Taking into account that the tracker should be able to run in real time, the following approaches will be discussed because the investigated papers for the presented approaches showed a good relation between quality and computational complexity. The following approaches for feature extraction are discussed in detail:

- Binary Robust Independent Elementary Features (BRIEF [10])
- Oriented FAST and Rotated BRIEF (ORB [35])
- Binary Robust Invariant Scalable Keypoints (BRISK [22])
- Fast Retina Keypoints (FREAK [1])

- Discrete Cosine Transform (DCT [41])
- Local Binary Patterns (LBP [26])
- Color Histogram (CHist [40])

3.1 KEYPOINT FEATURES

Keypoint features, as the name implies, rely on descriptions of regions around distinct points of an image. There exist several fast ways to extract those points like, e.g., the famous FAST keypoint detector [34]. The keypoint feature descriptors then extract a feature descriptor around every keypoint.

3.1.1 BRIEF

The **B**inary **R**obust **I**ndependent **E**lementary **F**eatures (BRIEF) as presented by Calonder *et al.* [10] are described as highly discriminative also when using a short feature vector for description. Due to the compactness of this feature descriptor it is very fast to build and to match.

The approach of Calonder *et al.* relies on a small number of pairwise intensity tests on an image patch after smoothing it. The feature vector of an image patch is defined as the result of testing a uniquely set of $n_d(x, y)$-location pairs where n_d is the number of pairs. This leads to following definition for the feature vector f_{n_d}:

$$f_{n_d}(p) := \sum_{1 \leq i \leq n_d} 2^{i-1} \tau(p; x_i, y_i), \tag{3.1}$$

where n_d is the number of binary tests and $\tau(p; x_i, y_i)$ is the intensity test function which evaluates a test on the image patch p using the locations x_i and y_i.

To decrease the noise on the intensity tests a Gaussian smoothing operation is applied as a preprocessing step. Calonder *et al.* showed experimentally that a Gaussian kernel with a size of 9×9 pixels and a variance of 2 yields the best results.

As there are many possible intensity test locations inside an image patch p, Calonder *et al.* examined various methods for determining the most promising way of selecting them. Five approaches were evaluated including uniform distribution, Gaussian distributions, randomly chosen locations and locations on a polar grid. Their experiments show that an isotropic Gaussian distribution ($Gaussian(0, \frac{1}{25}S^2)$ leads to the highest recognition rate shown in Figure 3.1.

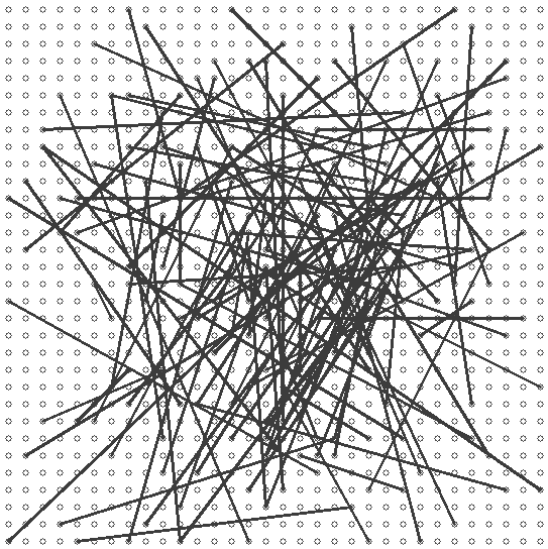


Figure 3.1: Gaussian sampling pattern of the BRIEF descriptor. The area shown is the patch where the intensity tests are extracted. The solid lines denote the pixel pairs which are tested against each other (taken from [10]).

In fact, BRIEF is very efficient and thus really suitable for the limited computation power of a SLR Smart Camera. Calonder *et al.* showed that on a 2.66 GHz PC the descriptor computation for 512 keypoints using 256 intensity tests ($n_d = 256$) takes only 8.87 milliseconds. The matching using the Hamming distance can be computed in 4.35 milliseconds.

3.1.2 ORB

The main drawback of BRIEF is its sensitivity to in-plane rotation. To deal with this drawback, Rublee *et al.* [35] introduce the **O**riented FAST and **R**otated **B**rief descriptor. They use the FAST [34] feature detector for detecting keypoints because of its high performance. In contrast to SIFT (introduced by Lowe in [24]) and SURF (introduced by Bay *et al.* in [5]), FAST does not provide a rotation operator. Therefore, Rublee *et al.* introduce their own orientation calculation for FAST keypoints using following approach:

As FAST does not produce a measure of cornerness, a Harris corner measure is introduced. To get a number of N keypoints, Rublee *et al.* use the FAST detector to produce even more than N keypoints, order them according to their Harris measure and pick the top N keypoints.

For corner orientation Rublee *et al.* use the intensity centroid approach from Rosin [32]. According to Rosin the moment of a patch is defined as

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y), \tag{3.2}$$

where m is the moment matrix, p and q are the moment matrix indices, x and y are the pixel coordinates and $I(x,y)$ is the intensity at the pixel coordinate.

Using these moments, Rublee *et al.* calculate the orientation of a patch θ by using the quadrant-aware version of the arctan function

$$\theta = atan2(m_{01}, m_{10}), \tag{3.3}$$

where m_{01} and m_{10} are entries from the patch moment defined before.

Rublee *et al.* compared their orientation approach to two gradient-based methods (BIN and MAX). They state that the intensity centroid approach outperforms the other two approaches by testing the methods on artificially rotated patches with noise added.

Based on the orientation Rublee *et al.* developed a steered variant of BRIEF. Taking a set of binary tests defined via

$$S = \begin{pmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{pmatrix}, \tag{3.4}$$

a "steered" version of this test location is constructed using the rotation matrix R_θ for the orientation θ:

$$S_\theta = R_\theta S. \tag{3.5}$$

This leads to the steered BRIEF operator where f_n is the BRIEF operator:

$$g_n(p, \theta) := f_n(p) | (x_i, y_i) \in S_\theta \tag{3.6}$$

To speed up the descriptor calculation, Rublee *et al.* created a discretized lookup table (in 12 degrees steps) of precomputed BRIEF patterns.

The drawback of the steered BRIEF descriptor is that binary tests are more uniform when applied to oriented corner keypoints. To cope with this problem, Rublee *et al.* introduce an approach for learning "Good Binary Features". They show that their approach significantly reduces the correlation between learned tests. The resulting descriptor is named **rBRIEF** and an example sampling pattern is shown in Figure 3.2.

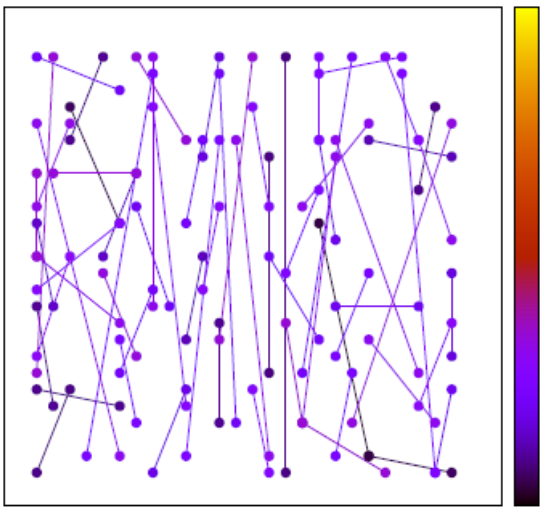


Figure 3.2: Example sampling pattern of the ORB descriptor. The lines denote the intensity tests used for descriptor creation. These test positions have to be learned (taken from [35]).

The evaluation of Rublee *et al.* showed very good results compared to SIFT, SURF and BRIEF when testing matches on synthetic in plane rotated images with a Gaussian noise added. They also compared their matching behavior against SIFT with different noise levels and came to the conclusion that rBRIEF is, in contrast to SIFT, relatively unaffected by the noise.

3.1.3 BRISK

Leutenegger *et al.* present a fast method for keypoint detection, description and matching named BRISK (**B**inary **R**obust **I**nvariant Scalable **K**eypoints) [22]. They point out that their approach is modular which allows the BRISK detector to be combined with other descriptors and vice versa.

Keypoint Detection For their keypoint detection Leutenegger *et al.* rely on the FAST detector. Their aim is to produce high-quality keypoints which are invariant to scale. To achieve this, the BRISK framework uses scale-space pyramid layers consisting of n octaves c_i and n intra-octaves d_i for $i = 0, 1, \ldots, n-1$ where typically $n = 4$. The pyramid layers are obtained by down sampling the original image c_0 the following way, where t is the layers' scale:

$$t(c_i) = 2^i \tag{3.7}$$

$$t(d_i) = 2^i \cdot 1.5. \tag{3.8}$$

As first step, the FAST detector is applied on all octaves (c_i) and intra-octaves (d_i) to produce regions of interest. Next, a non-maximum suppression is applied to the 8 neighboring FAST scores in the same layer and the corresponding square sized patches in the layer above and below. Finally, the position of the keypoint is interpolated over these consecutive scales to receive more salient keypoints.

Keypoint Description Leutenegger *et al.* use a concentric circle sampling pattern for their keypoint description. To obtain a keypoint orientation, sampling point pairings with a distance greater than a threshold δ_{min} are used (denoted as $\mathcal{L}$). Estimating the local gradients using the Gaussian smoothed intensity values $\mathbf{g}$ at the desired points in $\mathcal{L}$ leads to a global gradient determination for this keypoint.

Similar as BRIEF (see Sec. 3.1.1), BRISK relies on simple brightness comparison tests. Defining $\mathcal{S}$ as the intensities of short distance point pairings in the sampling pattern a bit b for a point pairing p_i and p_j is defined as follows:

$$b = \begin{cases} 1, & I(p_j^\alpha, \sigma_j) > I(p_i^\alpha, \sigma_i) \\ 0, & \text{otherwise} \end{cases} \quad \forall (p_i^\alpha, p_j^\alpha) \in \mathcal{S}, \tag{3.9}$$

where $\alpha = arctan2(g_y, g_x)$ and I is the intensity at the given point (with Gaussian smoothing using a standard derivation σ).

The smoothing with a Gaussian kernel is necessary to remove noise. The standard derivation of the Gaussian kernel for a point is dependent on the distance to the center. The closer to the center, the smaller becomes σ. The sampling pattern used for extracting the feature descriptor is shown in Figure 3.3.

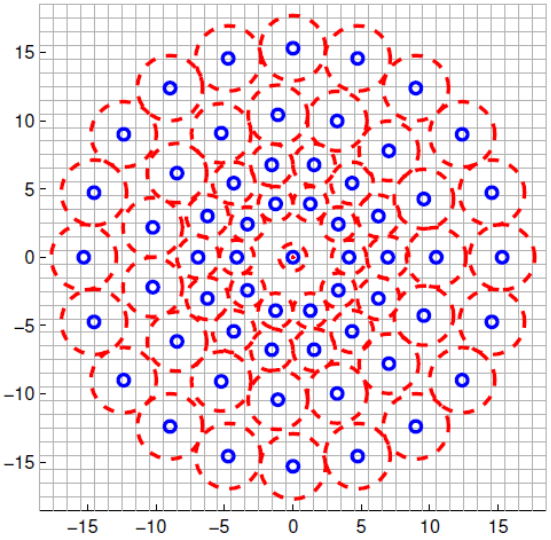


Figure 3.3: Sampling pattern of the BRISK descriptor with the varying Gaussian standard derivation as dotted red circles for smoothing. The solid blue circles denote the pixels taken for intensity tests (taken from [22]).

3.1.4 FREAK

The **F**ast **Re**tin**a** **K**eypoint (FREAK) from Alahi *et al.* [1] is a alternative to SIFT, SURF or BRISK. The intention is to compute a cascade of binary strings, extracted from intensity tests using a retinal sampling pattern. This approach was inspired by the human visual system. They show in their paper that FREAKs are faster to compute, consuming lower memory and are also more robust when compared to existing keypoint approaches.

The main question Alahi *et al.* are addressing, is how to select the ideal pairs of pixels in an image patch for intensity tests. This problem is also discussed in the sections BRIEF (see Sec. 3.1.1), ORB (see Sec. 3.1.2) and BRISK (see Sec. 3.1.1). Taking research from neuroscience into account, Alahi *et al.* develop a retinal sampling grid. Their approach is quite similar to the approach of BRISK (see Sec. 3.1.3), where concentric circles are used, with the difference that the density of points increases exponential when getting closer to the center

Alahi *et al.* also use a Gaussian kernel for smoothing their pixels before intensity testing to reduce noise. Here they also stick close to the approach in BRISK, using a varying standard derivation dependent on the distance to the center The FREAK approach sticks to this idea but expanding it by coupling the standard derivation of the Gaussian kernel to the log-polar retinal pattern [2]. The sampling pattern of the FREAK descriptor with overlapping Gaussian standard derivations is shown in Figure 3.4.

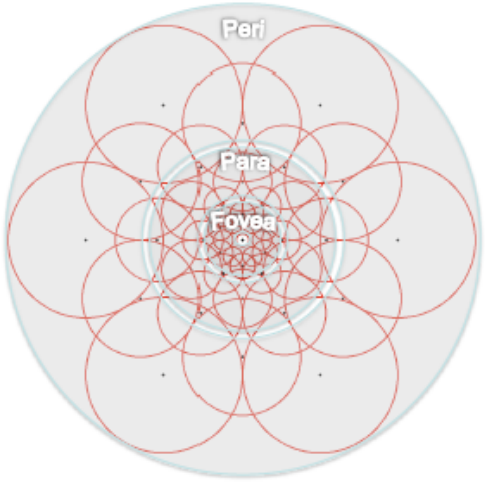


Figure 3.4: Sampling pattern of the FREAK descriptor showing the retinal pattern with overlapping Gaussian standard derivations for smoothing the pixels used for intensity tests. The closer to the center the higher is the density of pixels used for intensity testing and the smaller are the standard derivations. The labels "Fovea", "Para" and "Peri" denote different areas of the ganglion cells of the human eye. (Taken from [1])

This leads to the FREAK descriptor F, which is a binary string formed the following way:

$$F = \sum_{0 \leq a \leq N} 2^a T(P_a), \tag{3.10}$$

where N is the descriptor size and P_a is a pair of receptive fields and $T(P_a)$ is defined as follows:

$$T(P_a) = \begin{cases} 1 & if(I(P_a^{r_1}) - I(P_a^{r_2})) > 0 \\ 0 & otherwise \end{cases}, \tag{3.11}$$

with $I(P_a^{r_1})$ defined as the intensity of the first pixel of the pair P_a smoothed, using a Gaussian kernel.

Alahi *et al.* found that their selected pairs are not discriminant. To solve this, they use a learning approach similar to the one from Rublee *et al.* use for their ORB descriptor (see Sec. 3.1.2) to get more discriminant pairs.

The keypoint orientation estimation of Alahi *et al.* is very similar to BRISK (see Sec. 3.1.3). But where BRISK uses pairs with long distances, FREAK relies on a symmetric selection of pairs. In contrast to BRISK, where several hundreds of pairs are used, Alahi *et al.* use only 45 pairs which reduces the memory load significantly.

For their tests the AGAST detector (see Sec. 3.1.3 (BRISK)) is used. Alahi *et al.* propose FREAK to be the most robust descriptor on both of their testing environments. In their evaluation they show that FREAK is much faster than SIFT and SURF and also faster than BRISK for description of keypoints and matching the descriptors.

3.2 NON-KEYPOINT FEATURES

In opposite to the already examined keypoint features which rely on noticeable points in an image, there exist some completely different methods capable of representing regions of an image in bulk. Three of the most common methods are examined in more detail in this section.

3.2.1 DCT

As Tjahyadi *et al.* [41] state the Discrete Cosine Transform is a popular image compression technique which is used, e.g., for JPEG encoding and decoding. The DCT

therefore transforms images from their spatial representation into a frequency representation. For DCT calculation each image is divided into blocks of the size 8x8 and each block is then independently transformed using the 2D-DCT basis function:

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N}, \tag{3.12}$$

having x and y as coordinates in the image block, u and v as coordinates in the DCT coefficients block and C defined as

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & for\, x = 0, \\ 1 & otherwise. \end{cases} \tag{3.13}$$

This leads to an image representation of 8x8 blocks where each block is aligned in a zigzag pattern having the high energy DC coefficient in the upper left corner defined as

$$F(0,0) = \frac{1}{8} \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y), \tag{3.14}$$

because of the cosine of zero is one. The other 63 are referred to as AC coefficients. Figure 3.5 shows the zigzag pattern of a block.

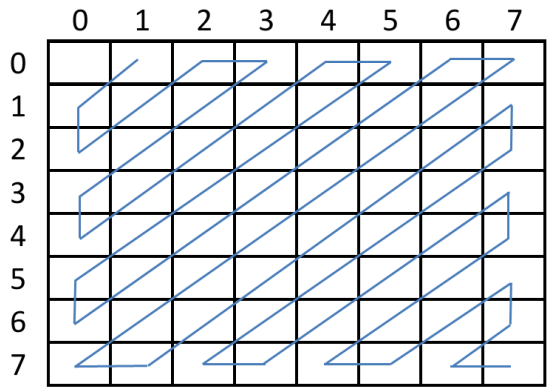


Figure 3.5: A 8x8 DCT block with the zigzag pattern.

3.2.2 LBP

The **L**ocal **B**inary **P**attern descriptor is based upon textures inside a local neighborhood around a center pixel. Ojala *et al.* [27] derive the LBP as follows.

First they define a texture in a local neighborhood of a grayscale image where the texture is defined as

$$T = t(g_c, g_0, ..., g_{P-1}), \tag{3.15}$$

where $t(x)$ is a distribution of values, g_c is the center pixels gray value in a local neighborhood and $g_p(p = 0, ..., P-1)$ are the gray values of equally spaced pixels located on a circle around the center using a given radius R and a given number of circle pixels P. The gray values are extracted at the positions calculated using the center and the radius. If a position does not match the pixel grid, it's gray value is interpolated. Figure 3.6 shows some example LBP configurations.

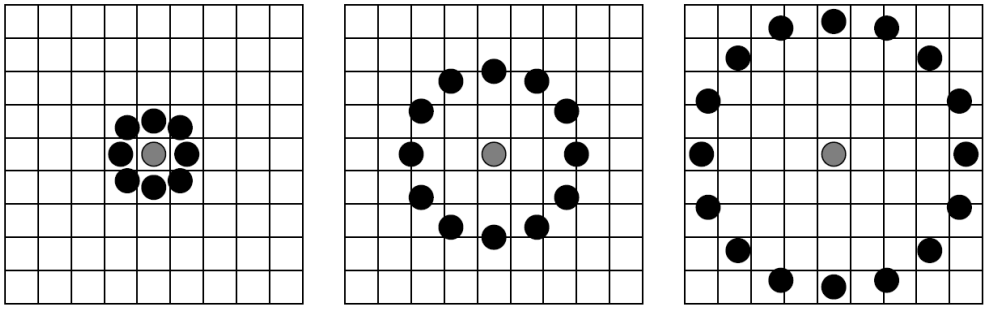


Figure 3.6: Examples of LBP configurations using a specified number of circle pixels P on a given radius R. The gray dot denotes the center pixel and the black dots the circle pixels where gray values are extracted. If the circle positions do not match the pixel grid exactly their value is interpolated.

Leading from a texture representation to a binary representation (as the name LBP tells us) requires additional work. Because the local texture can be represented without losing information by subtracting the value of the center pixel from the neighbors values T looks as follows:

$$T = t(g_c, g_0 - g_c, ..., g_{P-1} - g_c). \tag{3.16}$$

By assuming that the differences are independent from g_c and $t(g_c)$ describing the overall image luminance the texture representation becomes

$$T \approx t(g_0 - g_c, ..., g_{P-1} - g_c). \tag{3.17}$$

To achieve invariance to scaling, only the signs are considered using the sign function

$$s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{3.18}$$

the texture becomes

$$T \approx t(s(g_0 - g_c), ..., s(g_{P-1} - g_c)). \tag{3.19}$$

To transform this into a unique LBP code a binomial weight 2^p is assigned the following way:

$$LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(g_p - g_c)2^p, \tag{3.20}$$

where P is the number of circle positions, R is the radius and x_c and y_c are the center pixels coordinates. When reviewing the equation above it is obvious that the LBP in practice is interpreted as P-bit binary number.

3.2.3 COLOR HISTOGRAM

Swain *et al.* [40] denote that a color histogram is defined by a number of color axes (e.g., red, green, blue for RGB color space) for an image in a discrete color space. To obtain a color histogram for an image its colors have to be discretized and each discrete color has to be counted. This leads then to a 3D histogram for a color image shown in Figure 3.7.

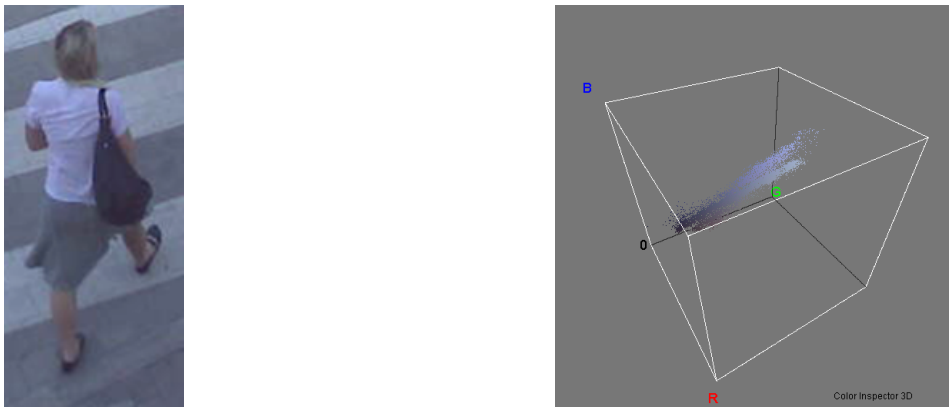


Figure 3.7: Example 3D histogram of a person. The 3D histogram was constructed using the 3D Color Inspector/Color Histogram plug-in for ImageJ (`http://rsb.info.nih.gov/ij/`).

The disadvantage of histograms for recognition is that the equivalence function they define is that two colors are the same when they fall in the same bin. This is not ideal because two colors lying close together but falling in different bins are not considered the same although they might have a distance smaller than a bin size. This leaded to ideas for defining a region around a color with respect to changing lightning conditions or sensor noise to get better results for matching colors.

Despite those problem color histograms do work because of the nature of objects surfaces. A surface of an object tends to span over a region of color and because of shading and camera noise these regions fall into more than one bin in a histogram. When matching a model histogram, obtained, e.g., from a database, to an image with the same object, the obtained value will be high although a point-by-point match would fail.

Another idea to obtain color histograms which are robust to changing lightning conditions are cumulative histograms C which are defined as follows having H as the non-cumulative histogram:

$$C(x, y, z) = \sum_{i=1}^{x} \sum_{j=1}^{y} \sum_{k=1}^{z} H(x, y, z). \tag{3.21}$$

To match two color histograms the so called *Histogram Intersection* can be used. This matching method is robust to varying background of the object, changing viewpoints, occlusions and different image resolutions concerning the database (model) and the image to be matched. For a pair of histograms where each of the histograms contains n bins the *Histogram Intersection* is defined as

$$\sum_{j=1}^{n} min(I_j, M_j). \tag{3.22}$$

The obtained result is the number of pixels from the model having the same color as pixels from the image. The following equation shows a way to normalize the obtained discrete number to a fractional result:

$$H(I, M) = \frac{\sum_{j=1}^{n} min(I_j, M_j)}{\sum_{j=1}^{n} M_j}. \tag{3.23}$$

When using histograms of equivalent size, which is possible when verifying, e.g., a person detection match, the histogram intersection problem can be reduced to the sum of absolute differences (*city-block* metric). Assuming that the number of pixels of both histograms is identical:

$$\sum_{i=1}^{n} M_i = \sum_{i=1}^{n} I_i = T, \tag{3.24}$$

gives the following intersection equation:

$$1 - H(I, M) = \frac{1}{2T} \sum_{i=1}^{n} |I_i - M_i|. \tag{3.25}$$

3.3 Discussion

The discussed feature descriptor approaches in this section give an overview over feature descriptors usable for the verification step which will be integrated into the SLR Tracking algorithm. The interesting part about the keypoint based feature descriptors is their fast matching possibility which is important to retain the real-time capability of the SLR Tracking algorithm. The disadvantage of, e.g., ORB and FREAK is their need of a learning step which can not be done in a real-time tracking application. Another advantage is that the keypoints used for feature descriptor extraction do not need to be calculated in the SLR Tracking algorithm as for the KLT motion estimation part of the tracker keypoints are already calculated (see Section 2.4.3).

For the non-keypoint features it will be interesting if the tracking algorithm retains it's real-time capability when calculating and matching DCT, LBP or color histogram feature vectors. Especially for the color histogram approach it is doubtful if it is able to describe distinct persons as the color variability of clothes is low.

Chapter 4

EVALUATION

Contents

This chapter will first present the evaluation results of the most promising feature description approaches in terms of quality. Therefore object sample databases have been build and the feature vectors have been extracted from each sample. The matching distance of all samples (where samples belong to same and distinct objects) is then used to analyse the ability of the feature description approach to separate samples belonging to a single object from samples belonging to distinct objects. The most promising approaches have been implemented into the SLR Track algorithm.

To be able to compare to the state of the art, the SLR Track output was analysed using a common metric described in Chapter 4.4. For these evaluations two datasets were used from which one is publicly available and one is an SLR own dataset. Finally the tracking quality is compared to results available in other publications to compare to the state of the art and to the implementation without the newly introduced feature verification step.

4.1 DATASETS

The following figures show example frames from the two datasets used in this evaluation chapter. Two typically scenarios where the SLR Track algorithm will be used have been selected. Both provide a surveillance camera view with different quality and background objects. One is an SLR own dataset described in Chapter 4.1.1 named SLR Pedestrian and one is the towncentre dataset (see Chapter 4.1.2).

(a) Example frames of the SLR Pedestrian sequence.

(b) Example frames of the towncentre dataset.

Figure 4.1: Example frames of the two used datasets in the evaluation.

4.1.1 SLR PEDESTRIAN

The SLR pedestrian sequence shows a town center viewed from a typical surveillance camera angle. The left part of the scene includes a building with an awning leading to variations of pedestrian appearance when walking from the shaded into the unshaded part of the street. The sequence consists of 3000 annotated frames with a resolution of 640×480. Figure 4.1a shows a frame of the dataset and Figure 4.2 shows sequences of some pedestrians extracted from the dataset.

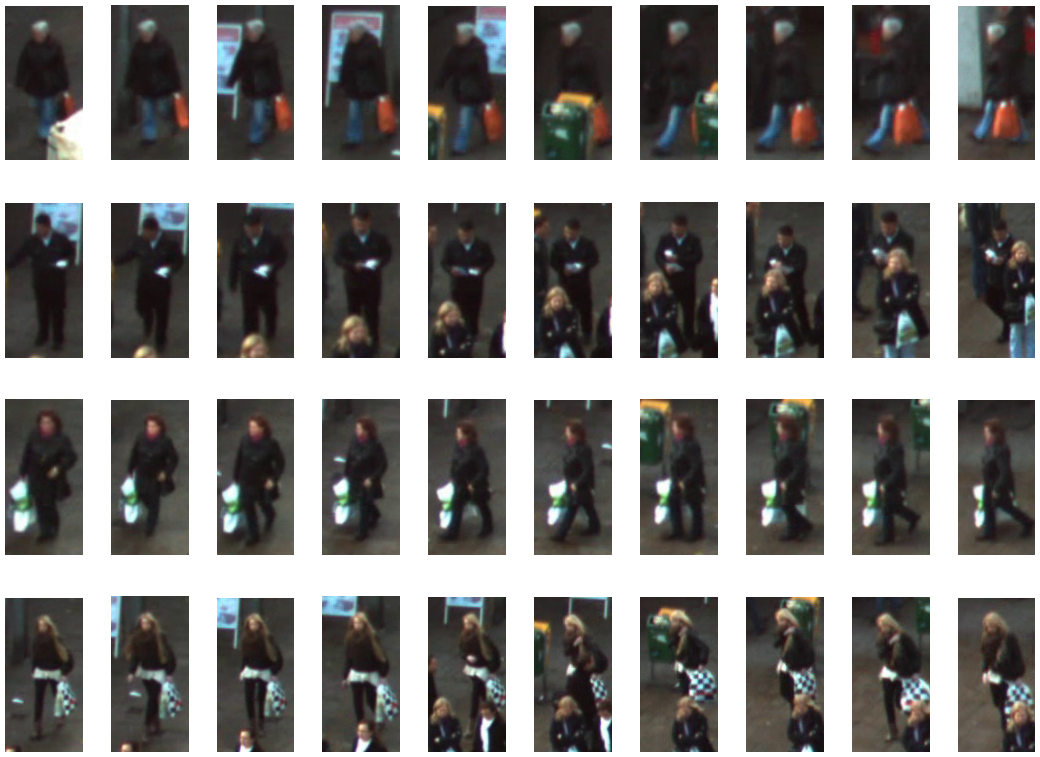


Figure 4.2: Four person sample sequences extracted from the SLR Pedestrian dataset with a difference of five frames between each image.

The samples above show four distinct persons (objects) which should be tracked by the algorithm. The difficulties shown here are occlusion of the person by background objects, occlusion of objects by other objects (possibly leading to identity switching) and high background variability. What also can be seen is that the clothes the persons wear are dark which adds additional complexity for the verification step.

4.1.2 TOWNCENTRE

The towncentre dataset presented by Benfold *et al.* in [6] includes a total number of 71500 hand labeled head locations from a high definition (1920x1080/25fps) video showing a busy towncentre. The groundtruth file also includes full body regions which are estimated using the head position and the camera calibration with approximate human dimensions. An example image of the dataset is shown in Figure 4.1b. Figure 4.3 shows four short sequences of pedestrians walking through the towncentre with partly occlusions, backpack, bag and a bicycle.

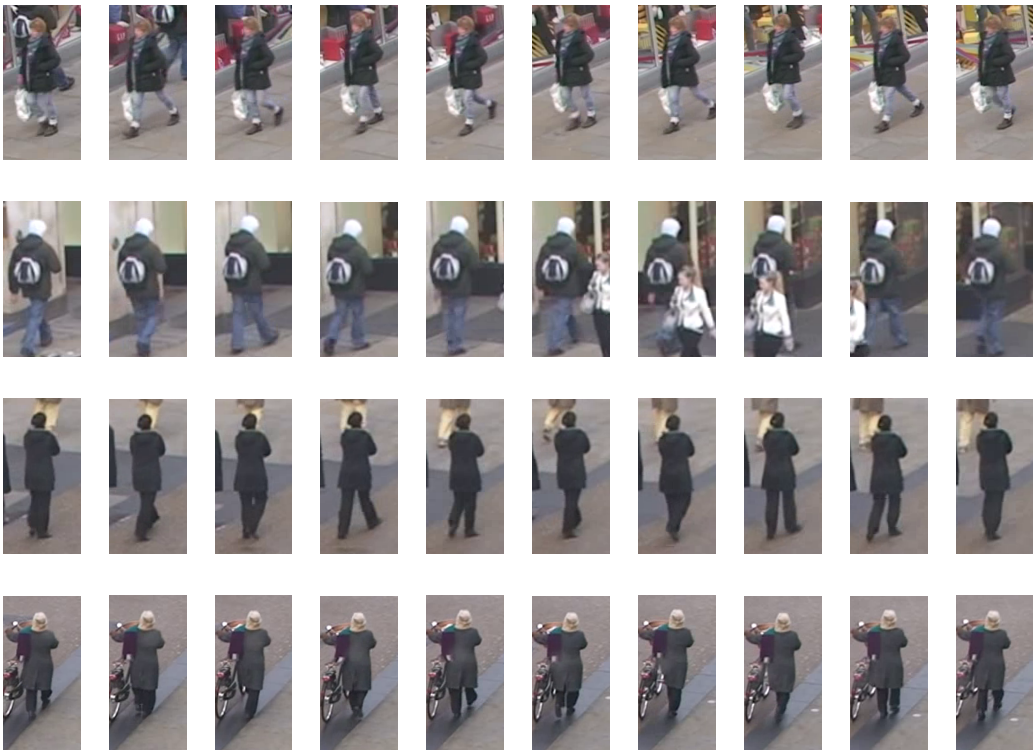


Figure 4.3: Four person sample sequences extracted from the towncentre dataset with a difference of five frames between each image.

As in the slr pedestrian sequence the clothing of the persons is also quite dark. The additional accessories of some of the persons like a bag, bagpack or bicycle may add additional information for the feature matching but may also confuse the HOG person detector.

4.2 FEATURE DESCRIPTOR EVALUATION

For descriptor evaluation the above defined datasets are used. Out of the datasets, person samples like in Figure 4.2 and in Figure 4.3 are extracted.

For all n extracted samples feature descriptors are calculated and stored. Then an $n \times n$ matrix is created where every cell holds the matching distance between two sample feature descriptors. As for every sample an object ID and a frame number is known, out of the matrix a distribution over the distances between samples with the same object ID (intra object) and a distribution over the distances between samples with distinct object ID (inter object) can be computed.

Those distributions are presented as two histograms in a single coordinate system, where the intra object distribution is presented in green and inter object distribution is presented in red (see Figure 4.4).

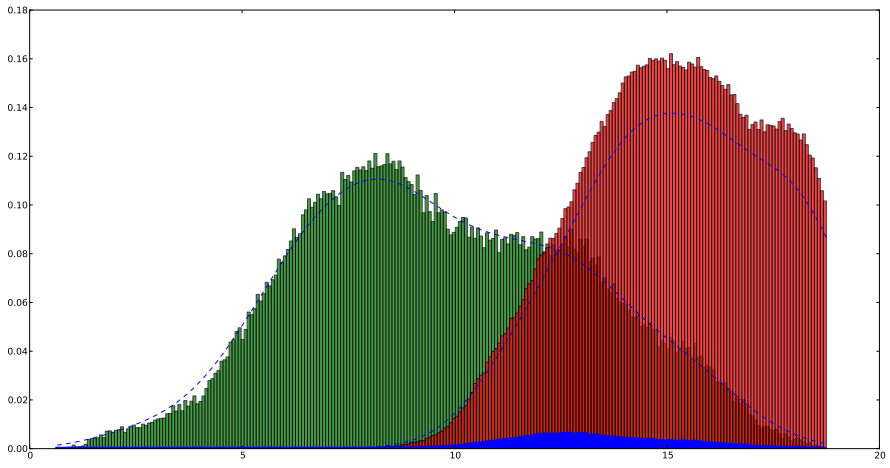


Figure 4.4: Feature descriptor distance evaluation histograms where the distribution of distance of samples of the same objects are presented in green and distances of samples from distinct objects are presented in red. The histogram overlap is colored in blue. The better a feature descriptor, the smaller the histogram overlap.

The feature descriptor matrix for the different investigated descriptors were calculated using a self developed Qt/C++ tool which takes a video sequence and an annotation file as input. In opposite to the towncentre dataset where the existing annotation file was used, for the SLR pedestrian sequence an annotation had to be created upfront by hand. The $n \times n$ distance matrix is stored as csv file for later use. For the further analysis of the distance matrix a python tool was written which produces the distribution diagrams and calculates the histogram intersection values.

To measure the quality of a feature descriptor the histogram intersect (as shown in Figure 4.4) is investigated. The histogram intersect of the histograms a and b $I(a, b)$ is defined as follows:

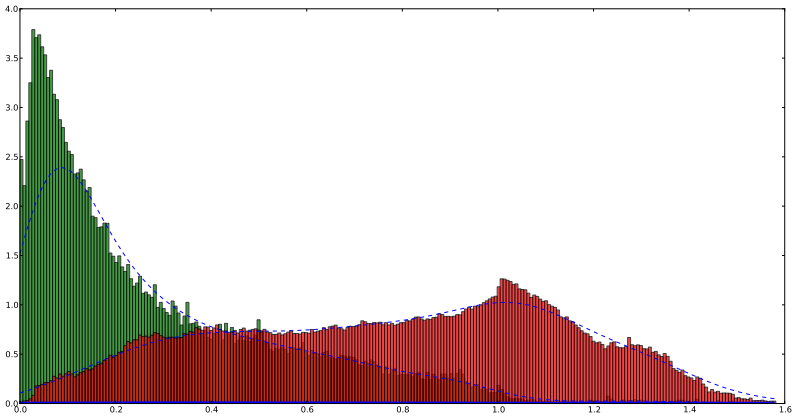
$$I(a, b) = \sum_{i=0}^{k} min(a_i, b_i), \tag{4.1}$$

where k is the number of bins. Inter object distances should be low and intra object distances should be high. Using this definition the distribution histograms of a good feature descriptor should hardly overlap and lead to a low $I(a, b)$ value.

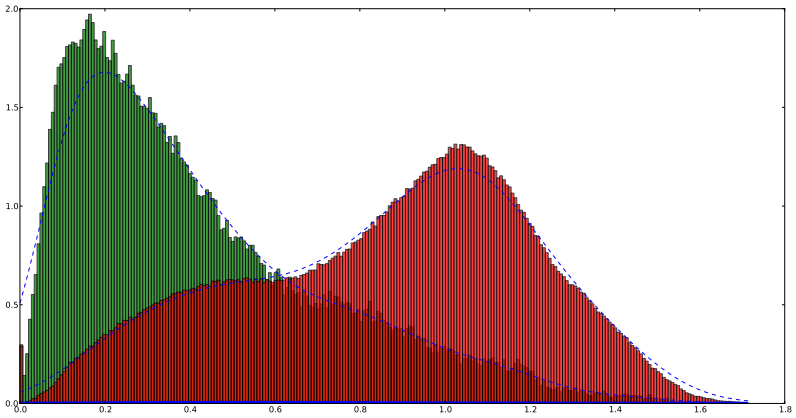
4.2.1 DISTANCE EVALUATION

The above described evaluation is performed on all features described in Chapter 2 using the presented datasets from Section 4.1 giving two distribution images for every feature. First the three non-keypoint feature descriptor approaches are investigated. Each of those descriptors yields only one feature vector per object sample and those can be directly matched to each other without the need of further processing to obtain a distance measure. In opposite to the keypoint feature descriptors which yield multiple feature vectors per object sample requiring further processing to obtain a match between two samples (see Sec. 4.2.2).

Color Histogram The color histogram feature descriptor as described in Sec. 3.2.3 is not able to separate inter and intra object samples in a satisfying manner as can be seen in Figure 4.5a and in Figure 4.5b. The figures show a medium overlap of the distance distributions having mostly intra object sample matches with low distances but also a lot of inter object sample matches with low distances leading to the conclusion that the color histogram descriptor is not optimal for separating samples of the same object and distinct objects samples.



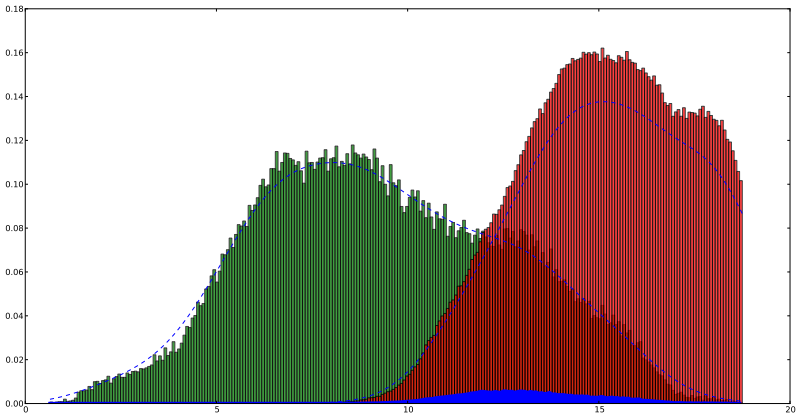
(a) Color histogram towncentre distance results.



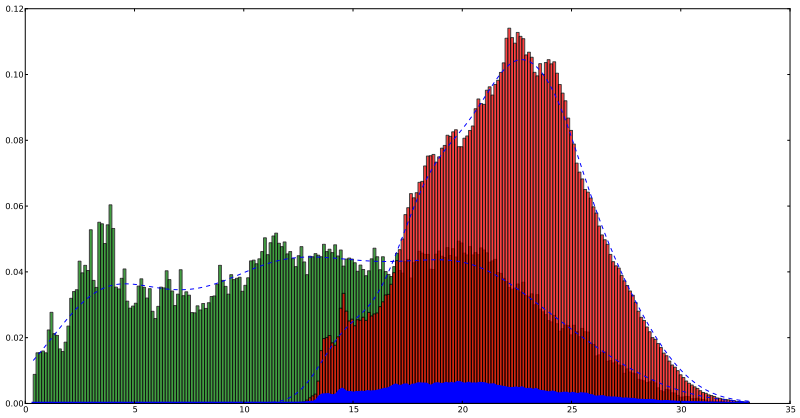
(b) Color Histogram SLR Pedestrian distance results.

Figure 4.5: Distance distribution of the Color Histogram descriptor for samples of the same object (green) and for distinct object samples (red).

DCT The DCT feature descriptor described in Sec. 3.2.1 is able to distinguish better between inter and intra object samples. In the high quality towncentre dataset (see Figure 4.6a) the result is even better than in the low quality SLR Pedestrian dataset (see Figure 4.6b), which can be determined because for inter object samples most of the matching distances are high and form a clear peak whereas, especially in the high quality data, the intra object sample distances are low.



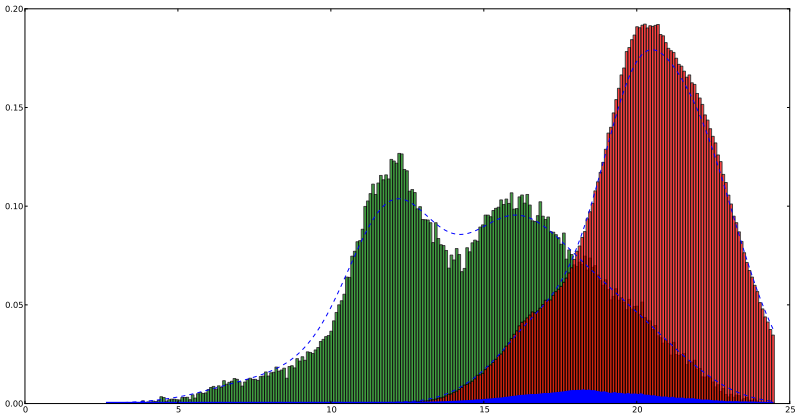
(a) DCT towncentre distance results.



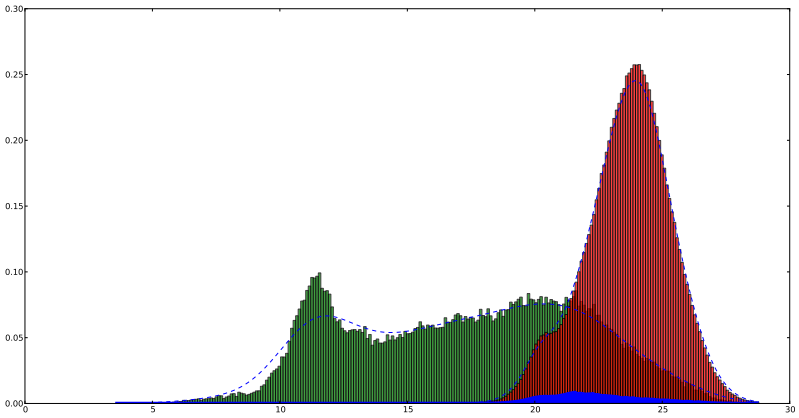
(b) DCT SLR Pedestrian distance results.

Figure 4.6: Distance distribution of the DCT descriptor for samples of the same object (green) and for distinct object samples (red).

LBP The LBP descriptor defined in Sec. 3.2.2 is also able to separate inter and intra object samples quite well. Where a higher image quality like in the towncentre dataset (see Figure 4.7a) leads to better results than a lower quality like in the SLR Pedestrian dataset (see Figure 4.7b). Especially the clear peak for inter object sample matching distances helps to distinguish if two samples belong to the same object or not.



(a) LBP towncentre distance results.



(b) LBP SLR Pedestrian distance results.

Figure 4.7: Distance distribution of the LBP descriptor for samples of the same object (green) and for distinct object samples (red).

4.2.2 KEYPOINT MATCHING

The description of an object using keypoint descriptors is slightly more complex. First the interesting points of the object have to be found. For all of those interesting points a descriptor is calculated. The matching process of two objects with several keypoint descriptors works as follows:

All keypoints descriptors of object A are matched to all descriptors of object B, giving for every keypoint k_i^A of object A a set of matches. Those matches are sorted ascending concerning the distance. To determine if a match for a keypoint is valid, the distances of the best and second best match are investigated. If the distance of the best match is significantly smaller than the distance of the second best match, the best match is marked as reliable. The distance between two objects is calculated as the average of all reliable matching distances.

As distance measure for the BRIEF, ORB and FREAK descriptor, which provide a binary string as feature vector, the Hamming distance suites best.

Definition 2 *The Hamming distance of two binary strings is a bit wise XOR operation followed by a bit count [1] and is very fast to compute on modern processors.*

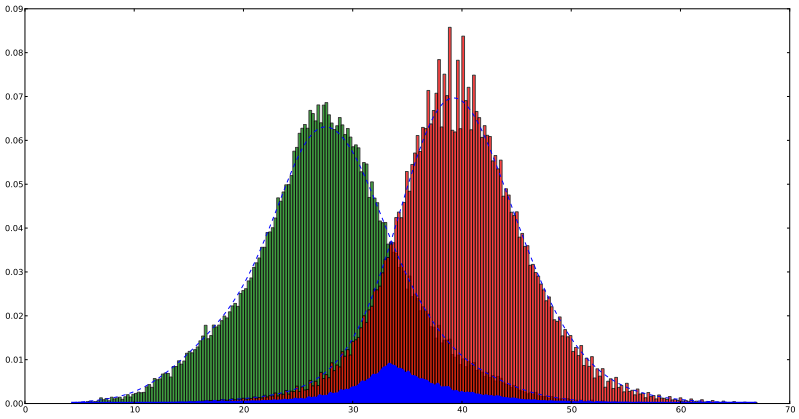
Giving, e.g., a sample A with a set of keypoints $\{k\}$ and a sample B with a set of keypoints $\{g\}$. For every keypoint k_i the distances to every keypoint g_j are calculated giving a set of Hamming distances $\{h\}$. For all keypoints in $\{k\}$ the distances h_{ij} are sorted ascending. If

$$h_i^{min}j < h_i^{min2}k * \tau, \tag{4.2}$$

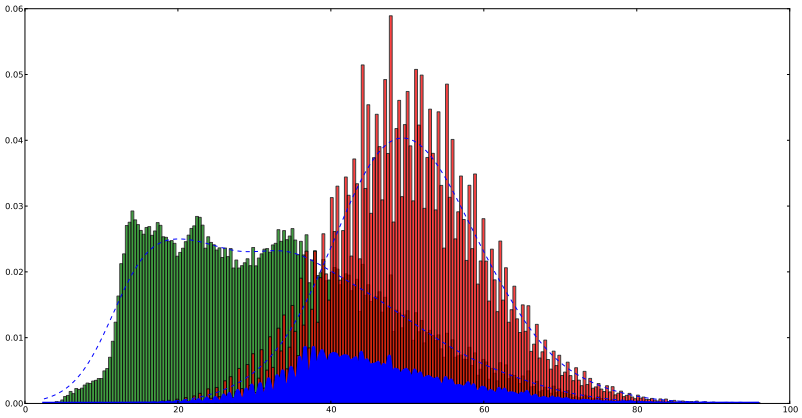
where $h_i^{min}j$ is the best match of keypoint i of sample A to the keypoint j of sample B, $h_i^{min2}k$ is the second best match of keypoint i of sample A to the keypoint j of sample B and τ is a weighting factor, equals true, the match is considered to be valid and is assigned the distance $h_i^{min}j$.

After iterating over all keypoints in $\{k\}$ and matching them to all keypoints in $\{g\}$ all valid matching distances are averaged giving the final distance for the samples A and B.

BRIEF The BRIEF feature descriptor (see Sec. 3.1.1) results in Figure 4.8a and Figure 4.8b show especially in the higher quality data of the towncentre dataset good peaks for inter and intra object distances making this descriptor a good choice for separating samples of same objects and distinct objects. In the lower quality SLR pedestrian sequence the peak for intra object distances is quite significant but the distances for inter object matches are often very high making it hard to split between objects using this descriptor in the low quality sequence.



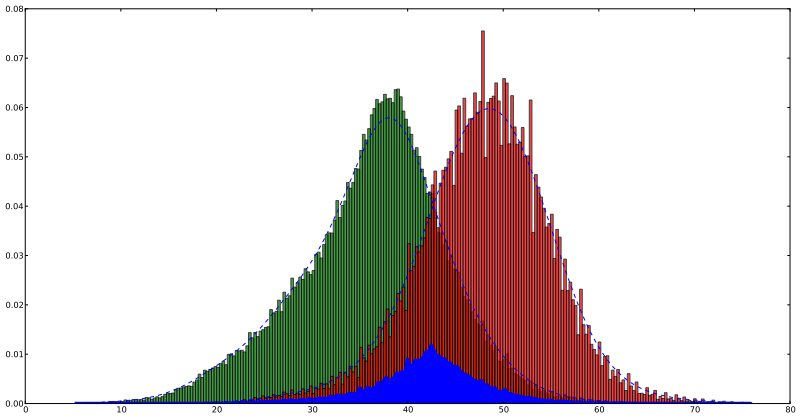
(a) BRIEF towncentre distance results.



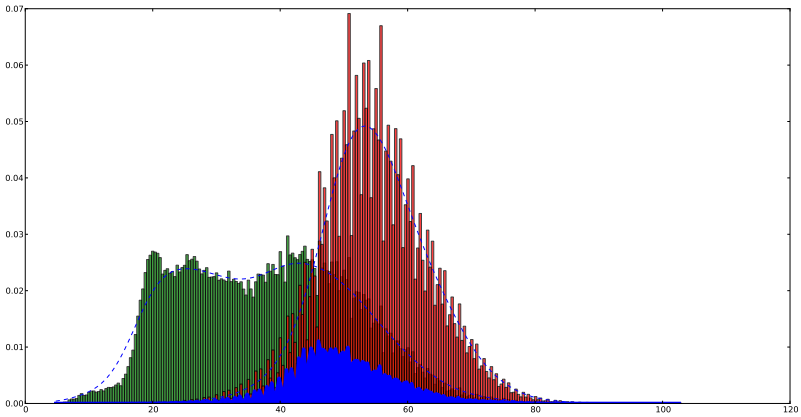
(b) BRIEF SLR Pedestrian distance results.

Figure 4.8: Distance distribution of the BRIEF descriptor for samples of the same object (green) and for distinct object samples (red).

ORB The ORB (see Sec. 3.1.2) results in Figure 4.9a and in Figure 4.9b show nearly similar results to the BRIEF descriptor but having the peaks lying closer together making it harder to determine between inter and intra objects than as it is with the BRIEF descriptor. Also for the low quality dataset the results are similar to the one presented for the BRIEF descriptor. Inter object matches with high distances make it hard to split the samples.



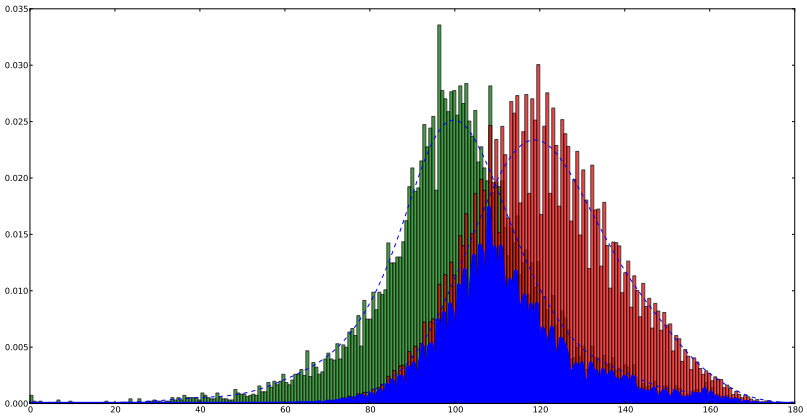
(a) ORB towncentre distance results.



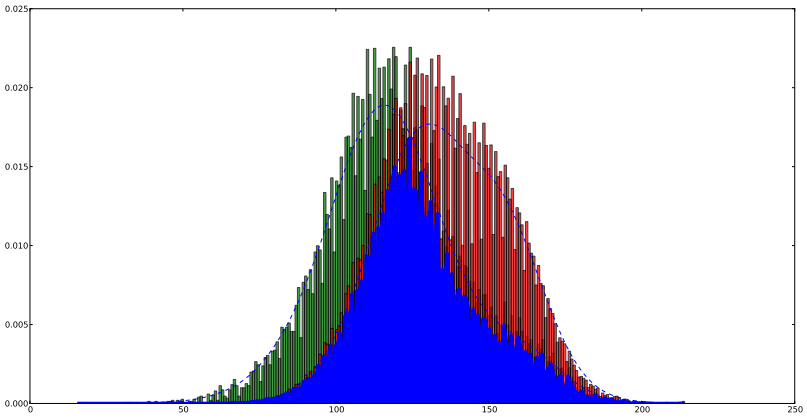
(b) ORB SLR Pedestrian distance results.

Figure 4.9: Distance distribution of the ORB descriptor for samples of the same object (green) and for distinct object samples (red).

BRISK The BRISK descriptor (see Sec. 3.1.3) tests as reported in Figure 4.10a and Figure 4.10b show peaks for inter and intra object sample matching. Given that the peaks lying close together leads to a bad performance when trying to distinguish between same object samples and distinct object samples although the results for the higher quality dataset are slightly better.



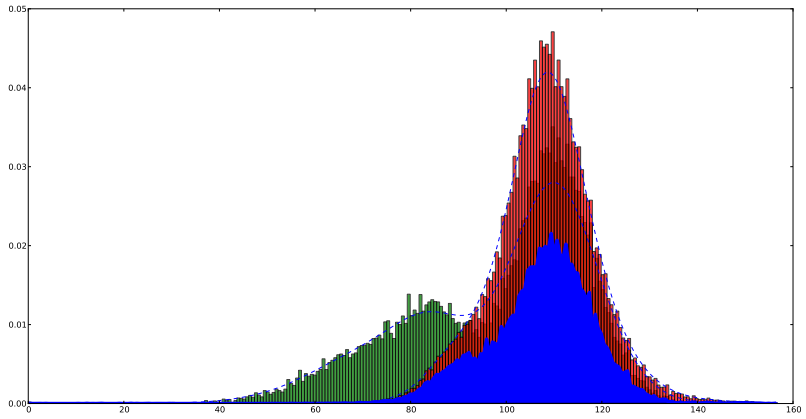
(a) BRISK towncentre distance results.



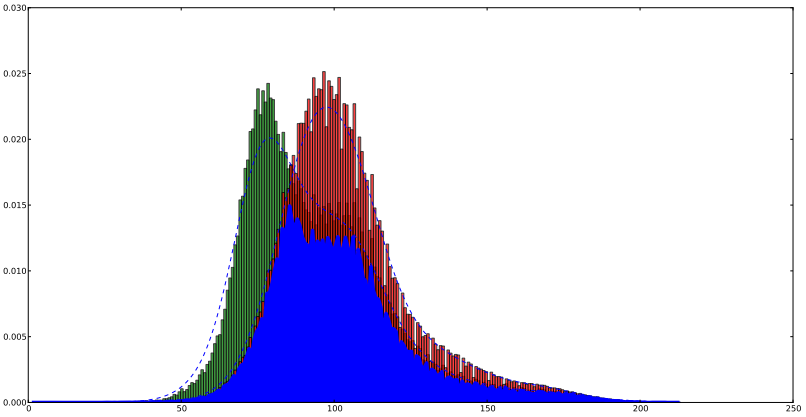
(b) BRISK SLR Pedestrian distance results.

Figure 4.10: Distance distribution of the BRISK descriptor for samples of the same object (green) and for distinct object samples (red).

FREAK The FREAK descriptor (see Sec. 3.1.4) tests as reported in Figure 4.11a and Figure 4.11b shows poor results for the data separation. Although, for the high quality dataset, the inter object sample matching distances form a peak at a high distance, the intra object samples do not form a peak at a low distance to be able to separate the samples correctly. The peaks in the low quality dataset are very close together and make it very hard to distinguish between inter and intra object samples.



(a) Freak towncentre distance results.



(b) Freak SLR Pedestrian distance results.

Figure 4.11: Distance distribution of the FREAK descriptor for samples of the same object (green) and for distinct object samples (red).

4.2.3 DISCUSSION

The performed inter- and intra-object distance evaluation showed that, out of the keypoint feature descriptors only the BRIEF descriptor was able to yield good results especially if the image resolution is high (as it is in the towncentre dataset). For low resolution data the LBP approach seems to be the best choice for determining between samples of same object and distinct object samples.

Table 4.1 additionally shows the histogram intersection values for the different descriptors and datasets. Presenting good results for DCT, BRIEF and LBP in the towncentre dataset and good results for the LBP descriptor in the SLR Pedestrian dataset.

Descriptor	Towncentre	SLR Pedestrian
BRIEF	0.3116	0.4359
DCT	0.3373	0.4638
ORB	0.3914	0.4167
LBP	0.3412	0.3269
Color Histogram	0.4620	0.4942
BRISK	0.5234	0.6711
FREAK	0.7506	0.7273

Table 4.1: Histogram intersection results.

The bad results of the Color Histogram descriptor lead to the conclusion, that color distribution on it's own is not feasible to identify distinct objects. This is obvious when tacking a look at the samples in Section 4.1.1 and in Section 4.1.2 where most of the people wear dark coats.

An explanation of the bad results the FREAK descriptor yielded is that there was no learning step performed to identify the mos discriminant pairs like described in Section 3.1.4. This step was skipped because the tracking process should work completely unsupervised on the smart camera.

Also the results of the BRISK descriptor imply that a concentric pattern around keypoints for feature descriptor extraction is not feasible because the BRIEF descriptor uses the same principle but uses a Gaussian pattern for extraction and yields much better results.

Out the found descriptor matching results, the most promising have been selected to serve as tracking verfication step in the SLR tracking algorithm as the next chapters will describe.

4.3 TRACKER IMPROVEMENTS

Using the obtained results of the evaluation section (see 4.2) the current SLR tracking algorithm is improved by implementing different verification feature descriptors and some other optimizations which are presented in the following paragraphs.

According to the results found in feature descriptor evaluation (see 4.2) the following feature descriptors were implemented for tracking verification:

- BRIEF (see Sec. 3.1.1)
- LBP (see Sec. 3.2.2)
- DCT (see Sec. 3.2.1)

To integrate the feature verification into the existing tracking algorithm following extensions have been made:

1. For all newly initialized trajectories calculate the feature vector and store it.
2. Predict trajectories according to their KLT displacements to the next frame.
3. Mark newly found detections in the next frame as candidates if the bounding box overlap between the prediction and the new detection exceeds a threshold.
4. For all trajectories where candidates have been found do the following:
   (a) Calculate the feature vector for all candidates and match the candidates' feature vector to the trajectory feature vector.
   (b) If the feature vector distance is smaller than a given threshold the trajectory is extended.
   (c) The current trajectory feature vector is replaced if the age of the feature vector exceeds a defined number of frames or HOG detector confidence of the new detection is higher than a given threshold.

5. For all trajectories where no candidate detections have been found the prediction bounding box is taken as pseudo detection. For this pseudo detection a feature vector is calculated and the feature vector is matched to the trajectories existing feature vector. If the distance of the feature vectors lies below a defined threshold the trajectory is extended. The trajectories feature vector is not replaced by the new one.

4.4 TRACKER EVALUATION

Evaluation of single object tracker is quite straightforward. The groundtruth positions of the object in the given frames only have to be matched against the trackers' output positions. If the distance between the trackers' output and the groundtruth lies below a given threshold the frame is identified as true positive.

Multi-target object tracking evaluation is more complex. The tracker data and groundtruth consist of multiple trajectories where every trajectory is identified by an ID. To evaluate multiple trajectories it is not feasible to match single trajectory positions to the groundtruth because of nearby trajectories, intersecting trajectories, identity switches, etc.

Bernardin *et al.* [7] present a bundle of performance metrics for multiple object tracking, where they define two main design criteria:

1. A metric should be able to judge the precision of the tracker by determining exact object locations.

2. A metric has to reflect the trackers ability to track an object over time producing exactly one trajectory per object.

Additional criteria are:

1. Simplicity: Have as few free parameters as possible to keep the evaluation process straightforward and make the output comparable.

2. Understandability: The metrics should be clear and easily understandable and behave according to human intuition.

3. Generality: The metrics should allow comparison of different types of trackers (2D tracker, 3D tracker, object centroid tracker, etc.).

4. Expressiveness: The metrics output should consist of few numbers but stay expressive so that they may be used for comparison of many different systems in large evaluations.

Taking into account the criteria Bernardin *et al.* formulate an evaluation procedure where for every time frame t a set of multiple tracker outputs (hypotheses) $h_1, ..., h_m$ and a set of visible objects (groundtruth) $o_1, ..., o_n$ is needed. For every time frame t the following steps are performed:

1. Find the best possible correspondence for hypotheses h_j and objects o_i.

2. Compute the position error for all found correspondences.

3. Sum all correspondence errors:

   (a) Count objects without hypothesis as misses.

   (b) Count hypotheses where no objects exist as false positives.

   (c) Count as mismatch error if for an object the tracking hypothesis changes over time. This happens if, e.g., the tracker reinitializes a former lost object track or an identity switch occurred between objects passing close to each other.

To clarify the steps of the evaluation procedure, Figure 4.12 shows examples of object and hypothesis matches for simple cases. The big circles denote object positions, the solid lines denote object tracks, the small circles denote hypothesis positions and the dotted lines denote hypothesis tracks.

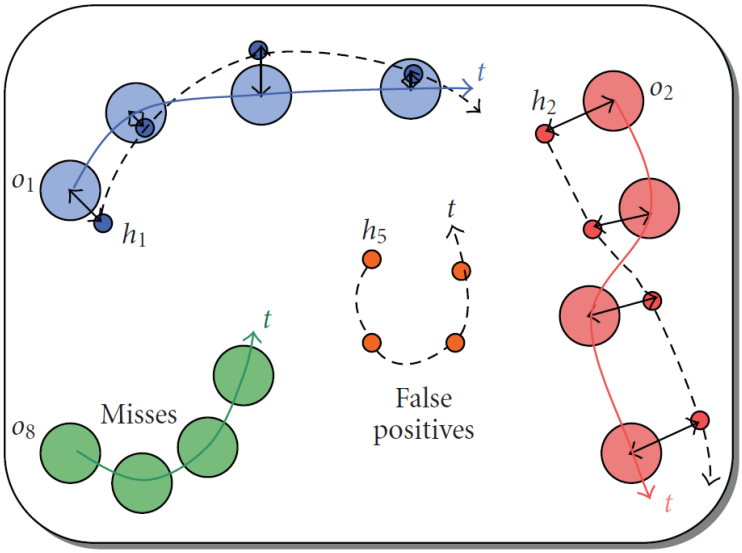


Figure 4.12: Matching tracker hypotheses to object trajectories over 4 frames. The objects o_1 and o_2 are matched against hypothesis h_1 and h_2 successfully. The object o_8 is identified as a miss because not corresponding hypotheses exists and the hypothesis h_5 is identified as false positive because no corresponding object exists (taken from [7]).

Performing the above mentioned steps a tracker's performance can be intuitively expressed by two numbers:

- "Tracking precision", which expresses the accuracy of position estimations.
- "Tracking accuracy" expressing the mistakes of the tracker in terms of misses, false positives, mismatches and failures to recover tracks etc.

4.4.1 MATCHING OBJECTS WITH TRACKER HYPOTHESIS

Before defining the two above mentioned numbers it is important to take a more detailed look at the algorithm used in Bernardin *et al.* for establishing correspondences between objects and tracker hypotheses.

The first step is to define whether a correspondence between an object o_i and a hypothesis h_j is given. To do so, a distance $dist_{i,j}$ between o_i and h_j is calculated and if a certain threshold T is exceeded no correspondence between the object and the hypothesis is established. For example a useful distance for bounding box trackers would be to consider the overlap between the object and the hypothesis or, e.g., for trackers reporting only object centroids on could use the Euclidean distance.

Next the consistency of a tracker – its ability to assign a label consistently to an object – has to be analyzed. The main difficulty here, is the identity switching. For example the best mapping for an object and a hypothesis would be (o_i, h_j). When now an identity switch occurs assign to it's hypothesis h_j another object o_k giving a new mapping (o_k, h_j). All those correspondences (o_k, h_j) would be counted as errors because the initial mapping was (o_i, h_j). In fact the tracker only made one error, the identity switch. Figure 4.13 shows an example where a hypothesis is mapped to two different objects.

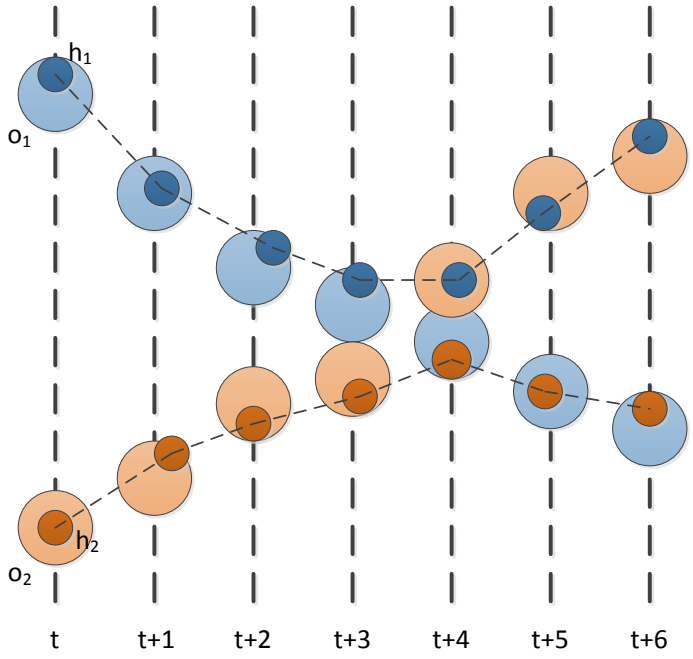


Figure 4.13: Initially the object o_1 (big light blue circle) is mapped to the hypotheses h_1 (small dark blue circle) and o_2 (big light orange circle) is mapped to h_2 (small dark orange circle). At time stamp $t + 4$ the identity switch occurs where the hypotheses are mapped to the wrong objects (adapted from [7]).

To avoid the above described multiple error counting, Bernardin *et al.* construct a set of object-hypotheses mappings, where $M_t = \{(o_i, h_j)\}$ is a set of mappings at time t and $M_0 = \{\cdot\}$. If at time $t+1$ a new mapping, e.g., (o_i, h_k) is created, the current mapping M_t is checked if for the object o_i a mapping exists and if the mapped hypothesis is still the same. If not a mismatch error is counted and the mapping (o_i, h_j) is replaced in M_{t+1} with the new mapping (o_i, h_k). The construction of a mapping set cannot only help to avoid multiple error counting but can also help to identify optimal hypothesis – object correspondences if multiple valid choices exist. Figure 4.14 shows an example where for a given object two valid hypotheses exist. The mapping from previous frames can now help to decide which on is the optimal mapping by giving priority to the existing mappings.

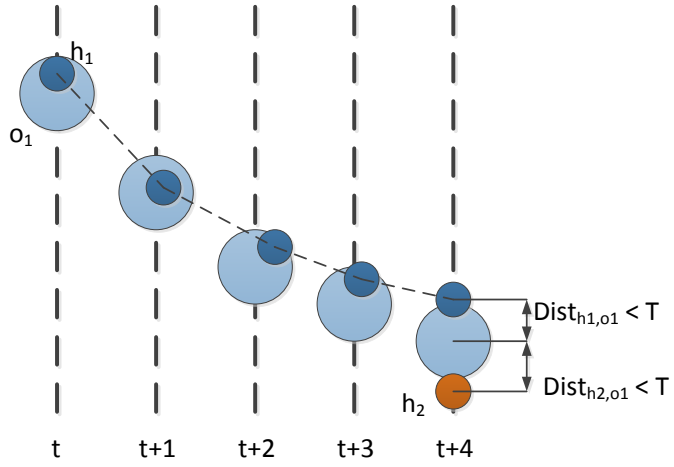


Figure 4.14: The object o_1 is mapped to the hypothesis h_1. At time $t+4$ a second hypothesis h_2 is created by the tracking algorithm nearby the object. Using the existing mapping from $t+3$ the mapping at $t+4$ is still (o_1, h_1) (adapted from [7]).

4.4.2 METRICS

Having defined the basic idea and strategy behind the metrics of Bernardin *et al.*, the metrics mentioned earlier can now be discussed in detail which follows in the next two sections.

4.4.2.1 TRACKING PRECISION

The multiple object tracking precision (MOTP) is defined as follows:

$$\text{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t}, \tag{4.3}$$

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

Thus, the MOTP metric provides a measurement over the trackers ability to estimate object positions. Further more the MOTP is independent of the trackers skill of keeping consistent trajectories, detecting identity switches etc.

4.4.2.2 TRACKING ACCURACY

The multiple object tracking accuracy (MOTA) is defined as follows:

$$\text{MOTA} = 1 - \frac{\sum_t (m_t + fp_t + ne_t)}{\sum_t g_t}, \tag{4.4}$$

where t again denotes the time stamp, m_t is the number of misses, fp_t is the number of false positives, ne_t is the number of mismatches and g_t is the number of objects present at time t.

Interpreting the above definition, MOTA is a sum over different error ratios (ratio of misses, ratio of false positives, ratio of mismatches) leading to an overall tracking accuracy.

4.4.3 IMPLEMENTATION

The CLEAR MOT metric evaluation script is written in python adapted from the existing code of Bagdanov *et al.* [4]. For the existing groundtruth data several parser classes have been written in python to deal with the different file formats. The groundtruth and the result data files are loaded into internal data structures and the metrics defined above are calculated as described in Chapter 4.4.2.

4.4.4 PROTOCOL

As mentioned before out of the results found in Section 4.2 the three most promising descriptors (DCT, LBP and BRIEF) have been implemented to perform a verification step when tracking to avoid identity switching. The implemented trackers and the tracker without the verification step were executed on both datasets presented in Section 4.1. The annotated groundtruth of the datasets and the output of the various tracking algorithms are then analyzed using the implemented python tool to calculate the CLEAR MOT metric.

Additionally to the defined CLEAR MOT metrics, the identity switches, precision and recall are calculated. Precision and recall are defined as follows:

$$precision = \frac{tp}{tp + fp}, \tag{4.5}$$

$$recall = \frac{tp}{tp + fn}, \tag{4.6}$$

where tp are the true positives, fp are the false positives and fn are the false negatives. For evaluating trackers these terms mean in detail:

- True positive: A true positive is a position of the hypothesis which matches the position of the corresponding object. In our case this means that the overlap area of the hypothesis and the objects bounding boxes is greater than 50%.

- False positive: A false positive is a position of the hypothesis which does not match the position of the corresponding object. This can occour if, e.g., the tracker drifts to the background so the hypothesis and the object position differ too much or an identity switch has occured so the hypothesis and the object are not corresponding anymore.

- False negative: A false negative is a object position for which no hypothesis position exists. It is also called a 'miss' because the objects position was missed by the tracker.

The following tables will show the results found on the two datasets for the above defined metrics. Table 4.2 shows the results on the towncentre dataset presented in Chapter 4.1.2. The SLR Track without verification step is compared to the improved SLR Track with DCT, LBP and BRIEF verification. Additionally results from the literature are also included.

Method	**MOTP**	**MOTA**	**ID switches**	**Precision**	**Recall**
SLR Track without Verification	66.6	34.9	566	72.2	57.7
SLR Track with DCT	66.4	38.0	263	78.0	53.2
SLR Track with BRIEF	68.9	32.2	465	71.1	55.1
SLR Track with LBP	67.5	24.6	322	65.3	53.1
Benfold *et al.* [6]	80.3	61.3	-	82.0	79.0
Zhang *et al.* [46]	71.5	65.7	-	71.5	66.1
Pellegrini *et al.* [30]	70.7	63.4	-	70.8	64.1
Yamaguchi *et al.* [44]	70.9	63.3	-	71.1	64.0
Leal-Taixe *et al.* [21]	71.5	67.3	-	71.6	67.6

Table 4.2: Comparison of the results of the towncentre dataset with the literature.

The results on the SLR Pedestrian dataset are presented in tab. 4.3. The non verification SLR Track algorithm is again compared to the improved using DCT, LBP and BRIEF verification steps.

Method	MOTP	MOTA	ID switches	Precision	Recall
SLR Track without Verification	75.3	50.4	341	86.0	61.4
SLR Track with DCT	74.6	58.7	88	90.6	65.8
SLR Track with BRIEF	78.6	50.8	251	86.4	61.1
SLR Track with LBP	79.2	35.9	116	87.8	42.1

Table 4.3: Comparison of the different SLR tracking implementations on the SLR Pedestrian dataset.

The results in the tables show that the SLR Track algorithms quality in terms of identity switches improves a lot when using a verification step. Especially the DCT verification step approach yields good results. Comparing the results of the towncentre dataset using the SLR Track without verification and the DCT improved SLR Track, the recall value has decreased. As the recall represents the ratio of correct object positions to correct and missed object positions, this is a hint that the DCT verification step drops correct object positions. This can also be seen for the BRIEF and LBP improved SLR Track on both of the datasets. Only on the SLR Pedestrian dataset the recall value improves using the DCT improved SLR Track algorithm.

Taking a look at the MOTP and MOTA values, the verification step implementation does not improve the trackers quality dramatically. But taking into account the decrease of identity switches the quality improvement of the DCT implementation is obvious. For MOTA and MOTP calculation identity switches are only counted as one mismatch error although all further frames of this object are wrong positions. This is why the SLR Track implementation without verification steps MOTP and MOTA values are nearly the same when compared to the DCT improved version.

Comparing the results to the state of the art presents a gap especially for the MOTA metric and the recall value which state that the, although fast, SLR Track algorithm needs further improvements in it's detection and association part. What is also noticeable is that the SLR Track algorithm was also tested on a real smart camera from the SLR company. The CPU is a 1.6GHz Intel Atom single core processor with hyperthreading where the algorithm runs with eight frames per second at a resolution of 640x480.

Chapter 5

CONCLUSION

Tracking multiple objects in video sequences is a broad field of research and the problems arising from different viewpoints, lightning conditions and occlusion of objects are very challenging. In this thesis some of the most interesting approaches and algorithms for solving the tracking problem have been reviewed keeping an eye on their ability to be implemented on a smart camera or if their findings can contribute to develop such an algorithm. Having a low computational device like a smart camera, the solution to the tracking problem becomes even more challenging. That is why a proven to be fast SLR Track algorithm was taken as starting point for the implementation.

The SLR Track combines a state of the art detection part with fast motion estimation for it's association part but lacks on object identity information. This information was then generated using different, again fast, algorithms which were compared on their ability to match person samples and to make assumptions about two samples belonging to the same object. Out of the results the most suitable algorithms were chosen to be implemented for an overall tracking analysis.

A common metric was then used for comparing the overall tracking quality of the SLR Track and its improvements to state of the art tracking algorithms from the literature on a widely used publicly available dataset. Although the improvements implemented show promising results when comparing the SLR Track to it's improved versions, the comparison to the state of the art reveals a quality gap.

The fact that the verification step improved algorithm can be executed on a low computational power device with eight frames per second at 640x480 pixels resolution can be seen as success. Especially when taking into account the decreasing of identity switches by 53% on the towncentre and 74% on the slr pedestrian dataset using the DCT improved algorithm.

Future work on the algorithm will mainly address the detector part of the algorithm. The used detector will be revised and a new training with more annotated pedestrian and car data will be made. This can help to reach higher MOTA and MOTP scores due to less false positive detections and a lower missing rate. In further works the DCT verification step will be a fixed part of the tracking algorithm but it's association part based solely on KLT position predictions will be investigated for improvements.

In conclusion it can be said that the thesis aim of implementing an object tracker on a smart camera running in real time was achieved. There is still a gap in the quality to present a tracking sensor but the identity switching errors were reduced by half due to the implementation of object identity information. Also a tool chain was built including a Qt/C++ tool to extract matching distances for different feature descriptor approaches from object samples which are graphical represented with a tool written in python and a python clear mot metric implementation for overall tracking performance comparisons. This tool chain will help to integrate further improvements into the tracker by providing an easy and standardized way of illustrating the quality of the implemented changes.

Bibliography

[1] Alahi, A., Ortic, R., and Vandergheynst, P. (2012). Freak: Fast retina keypoint. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 510–517.

[2] Araujo, H. and Dias, J. (1996). An introduction to the log-polar mapping. In *Proc. Workshop on Cybernetic Vision*, pages 139–144.

[3] Babenko, B., Yang, M.-H., and Belongie, S. (2011). Robust object tracking with online multiple instance learning. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33:1619–1632.

[4] Bagdanov, A. D., Del Bimbo, A., Dini, F., Lisanti, G., and Masi, I. (2012). Compact and efficient posterity logging of face imagery for video surveillance. *IEEE Multimedia*, pages 48–59.

[5] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. *Proc. European Conf. on Computer Vision*, pages 404–417.

[6] Benfold, B. and Reid, I. (2011). Stable multi-target tracking in real-time surveillance video. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 3457–3464.

[7] Bernardin, K. and Stiefelhagen, R. (2008). Evaluating multiple object tracking performance: the clear mot metrics. *Int. Journal on Image and Video Processing*, 2008:1:1–1:10.

[8] Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32.

[9] Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E., and Van Gool, L. (2011). Online multiperson tracking-by-detection from a single, uncalibrated camera. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33(9):1820–1833.

[10] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). Brief: Binary robust independent elementary features. In *Proc. European Conf. on Computer Vision*, volume 6314, pages 778–792.

[11] Chandran, S. D. V. and Sridharan, S. (2005). Tracking people in 3d using position, size and shape. In *Proc. Int. Symposium on Signal Processing and its Applications*, pages 611–614.

[12] Criminisi, A. and Shotton, J. (2013). *Decision Forests for Computer Vision and Medical Image Analysis*. Springer Publishing Company, Incorporated.

[13] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 886–893.

[14] Denman, S., Chandran, V., and Sridharan, S. (2007). An adaptive optical flow technique for person tracking systems. *Pattern Recognition Letters*, 28(10):1232–1239.

[15] Dollár, P., Wojek, C., Schiele, B., and Perona, P. (2012). Pedestrian detection: An evaluation of the state of the art. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 34(4):743–761.

[16] Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proc. SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 71–80.

[17] Felzenszwalb, P. and Huttenlocher, D. (2004). Efficient graph-based image segmentation. *Int. Journal of Computer Vision*, 59(2):167–181.

[18] Gall, J., Yao, A., Razavi, N., Van Gool, L., and Lempitsky, V. (2011). Hough forests for object detection, tracking, and action recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33:2188–2202.

[19] Godec, M., Roth, P. M., and Bischof, H. (2011). Hough-based tracking of non-rigid objects. In *Proc. IEEE Int. Conf. on Computer Vision*, pages 81–88.

[20] Jiri, Z. K., Matas, and Mikolajczyk, K. (2009). Online learning of robust object detectors during unstable tracking. In *Proc. IEEE Int. Conf. on Computer Vision*, pages 1417–1424.

[21] Leal-Taixé, L., Pons-Moll, G., and Rosenhahn, B. (2011). Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker. In *Proc. IEEE Int. Conf. on Computer Vision*, pages 120–127.

[22] Leutenegger, S., Chli, M., and Siegwart, R. Y. (2011). Brisk: Binary robust invariant scalable keypoints. In *Proc. IEEE Int. Conf. on Computer Vision*, pages 2548–2555.

[23] Loibner, G. and Sidla, O. (2013). Feature descriptors for object matching in real-time tracking applications. In *Proc. SPIE Conf. on Electronic Imaging*, pages 86630B–86630B.

[24] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2):91–110.

[25] Lu, L. and Hager, G. (2007). A nonparametric treatment for location/segmentation based visual tracking. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1–8.

[26] Mäenpää, T. (2003). *The Local Binary Pattern Approach to Texture Analysis: Extenxions and Applications*.

[27] Ojala, T., Pietikäinen, M., and Mäenpää, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(7):971–987.

[28] Oza, N. C. (2005). Online bagging and boosting. In *Proc. IEEE Int. Conf. on Man and Cybernetics*, volume 3, pages 2340 – 2345.

[29] Pakkanen, J., Iivarinen, J., and Oja, E. (2004). The evolving tree—a novel self-organizing network for data analysis. *Neural Processing Letters*, 20:199–211.

[30] Pellegrini, S., Ess, A., and Van Gool, L. (2010). Improving data association by joint modeling of pedestrian trajectories and groupings. In *Proc. European Conf. on Computer Vision*, pages 452–465.

[31] Ren, X. and Malik, J. (2007). Tracking as repeated figure/ground segmentation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1–8.

[32] Rosin, P. L. (1999). Measuring corner properties. *Computer Vision and Image Understanding*, 73(2):291–307.

[33] Ross, D. A., Lim, J., Lin, R.-S., and Yang, M.-H. (2008). Incremental learning for robust visual tracking. *Int. Journal of Computer Vision*, 77(1–3):125–141.

[34] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. *Proc. European Conf. on Computer Vision*, pages 430–443.

[35] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF. In *Proc. IEEE Int. Conf. on Computer Vision*, pages 2564–2571.

[36] Saffari, A., Leistner, C., Santner, J., Godec, M., and Bischof, H. (2009). On-line random forests. In *Proc. IEEE Workshop on Online Learning*, pages 1393–1400.

[37] Schulter, S., Leistner, C., Roth, P., Bischof, H., and Van Gool, L. (2011). On-line hough forests. In *Proc. British Machine Vision Conf.*, pages 128.1–128.11.

[38] Shin, J., Kim, S., Kang, S., Lee, S.-W., Paik, J., Abidi, B., and Abidi, M. (2005). Optical flow-based real-time object tracking using non-prior training active feature model. *Real-Time Imaging*, 11(3):204–218.

[39] Sidla, O. (2010). Object tracking by combining detection, motion estimation, and verification. *Proc. SPIE Conf. on Intelligent Robots and Computer Vision*, 7539:753906–753906–11.

[40] Swain, M. and Ballard, D. (1991). Color indexing. *Int. Journal of Computer Vision*, 7:11–32.

[41] Tjahyadi, R., Liu, W., and Venkatesh, S. (2004). Application of the dct energy histogram for face recognition. In *Int. Conf. on Information Technology for Application*, pages 305–310.

[42] Tomasi, C. and Kanade, T. (1991). Detection and tracking of point features. Technical Report CMU-CS-91-132, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

[43] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume 1, pages 511–518.

[44] Yamaguchi, K., Berg, A. C., Ortiz, L. E., and Berg, T. L. (2011). Who are you with and where are you going? In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1345–1352.

[45] Yin, Z. and Collins, R. (2009). Shape constrained figure-ground segmentation and tracking. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 731–738.

[46] Zhang, L., Li, Y., and Nevatia, R. (2008). Global data association for multi-object tracking using network flows. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1–8.

[47] Zhu, Q., Avidan, S., Yeh, M.-C., and Cheng, K.-T. (2006). Fast human detection using a cascade of histograms of oriented gradients. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 1491–1498.