VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VIZUÁLNÍ SLEDOVÁNÍ OBJEKTŮ V REÁLNÉM ČASE VE VIDEU

REAL-TIME OBJECT TRACKING IN VIDEO

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MARTIN ŠIMON
AUTHOR

VEDOUCÍ PRÁCE Ing. JAROSLAV ROZMAN, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Práce se zaměřuje na vizuální sledování objektu v reálném čase ve videu s důrazem na problémy vznikající při dlouhodobém sledování. Mezi tyto problémy patří především okluze, ať už částečná či úplná, a vizuální změny objektu. Dále se práce zaměřuje na objekty na hranici rozlišitelnost a trhavý pohyb kamery, jakožto problémy přítomné při sledování vzdálených objektů. Součástí práce je shrnutí současného stavu s ohledem na zmíněné problémy a návrh systému s vysokou kvalitativní stabilitou a odolností vůči zmíněným problémům, především malé velikosti objektů. Navržený systém byl implementován a z vyhodnocení vyplynulo, že je schopný tyto problémy částečně řešit.

Abstract

This thesis focuses on real-time visual object tracking with emphasis on problems caused by a long-term tracking task. Among theses problems belong primarily an occlusion problem, both the partial and the full one, and appearance changes of the object during the tracking. The work is also concerned with tracking objects of a very limited size and unsteady camera movements. These two particular problems are relatively common when tracking distant objects. A part of this work is also a summary of related work and a proposal of a system with high qualitative stability and robustness to problems mentioned. The proposed system was implemented and the evaluation demonstrated that it is capable of solving these problems partially.

Klíčová slova

Sledování objektů, Real-time sledování, Malá velikost objektů, Trhavý pohyb kamery, Částečná okluze, Úplná okluze, Násobné obousměrné sledování, Dlouhodobé sledování

Keywords

Object Tracking, Real-Time Tracking, Limited Object Size, Unsteady Camera Movement, Partial Occlusion, Full Occlusion, Multiple Bidirectional Tracking, Long-Term Tracking

Citace

Martin Šimon: Real-time object tracking in video, diplomová práce, Brno, FIT VUT v Brně, 2015

Real-time object tracking in video

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci na téma „Vizuální sledování objektů v reálném čase ve videu“ vypracoval samostatně pod vedením vedoucího diplomové práce Ing. Jaroslava Rozmana, Ph.D., a odborného konzultanta Ing. Davida Hermana a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

....
Martin Šimon
May 25, 2015

Poděkování

Poděkování patří především Ing. Davidu Hermanovi za jeho ochotu, konstruktivní nápady a velkou pomoc, a také Tereze Kučerové za její velmi rychlou jazykovou korekturu.

Contents

Chapter 1

Introduction

A visual object tracking in video has become a popular topic in recent years, but it still remains generally unsolved. There is a significant amount of very efficient object tracking systems which are sufficiently accurate and which work in real-time speed. Unfortunately, many problems, such as occlusion, no matter if full or partial, background clutter, noise or unexpected rough camera movements, are bottlenecks of many recent visual object tracking related projects and actually exclude them from daily professional usage. Among these problems also belongs a problem of appearance changes caused by a long-term tracking, when the object may vary.

In this work I would like to focus on recent solutions in the field of tracking and try to build an object tracking system whose efficiency and speed performance are at least the same as the performance of the state of the art trackers. Also, I would like to improve some critical parts which have serious impacts on accuracy or are too narrowly focused and lack generality. As the visual object tracking is a basic requirement for numerous applications in contemporary robotics, the system should be focused also on limited sized objects, which are basically all objects that are far enough, and a poor quality video input with low signal to noise ratio.

Based on the recent work and the problems mentioned, a new long-term real-time general object tracking system is proposed. The proposed system is implemented and evaluated with usage of standard visual object tracking dataset as well as a few original video sequences which were taken for this purpose. The proposed tracking system is designed in such manner that it would handle both the standard dataset as well as the new one, ideally without any significant reduction in speed or accuracy.

This work is organized as follows. In chapter 2, current progress in the visual object tracking field and the issues related are summarized. The proposed system able to handle the mentioned problems is described in chapter 3, including a newly introduced method to increase quality of tracking points. The chapter also contains goals of this work and the background on which this project is built. The implementation details are shortly summed in chapter 4. Finally, the system is widely evaluated in chapter 5. This contains several experiments focused on quality, stability, performance, and experiment focused on newly proposed components as well as the comparison with the other state of the art solutions. At the end, the thesis is summarized and the further work is outlined in chapter 6.

Chapter 2

Related Work

A significant amount of work has been written recently regarding the field of object tracking in a video. In this chapter, related solutions to all parts of long-time object tracking in a video frame-to-frame are described.

The object representation sets the apriori abilities of every tracking system. Some widely used object representations are described in section 2.1. The section also contains notes about the approaches to measuring similarity of these object representations. Afterwards, the localization approaches are described in section 2.2, which is the actual core of every tracking algorithm. The last section summarizes reasons behind writing this work - it is an outline of commonly known and widely solved, yet unsolved problems.

2.1 Object Representation and Matching

The object representation is the first and very important task before developing a successful tracking system. The object representation is the model how it is treated internally in the system. The system abilities are frequently correlated with the model type - the object representation.

Every object representation has its own way to compare these objects; the measurement system. This is also very important because it has a direct impact on the resulting system quality as well as on the speed and efficiency.

2.1.1 Visual Templates and Blobs

The most straightforward way to define the object of interest is by a visual template of the object. Such a template is obtained in the very first frame when the user chooses the object and it is stored internally in a form of matrix.

The basic idea is to use static templates [25, 40] (Figure 2.1 on the left side). The obvious problem is the inability to cover changes in the object's appearance, which may occur during tracking non-rigid objects or during tracking for longer time. However, this can be more or less solved with adaptive templates [26]. Furthermore, such systems are also able to solve partial occlusion; another very frequent and generally unsolved problem in the object tracking process.

Another approach is to represent an object as a blob (Figure 2.1). Blobs are understood as a set of points, widely represented as a rectangular matrix and a matrix of weights, also called a *mask*. The advantage of this approach is a better object representation and therefore a more accurate comparison between them [8]. Basically, the blobs are still visual

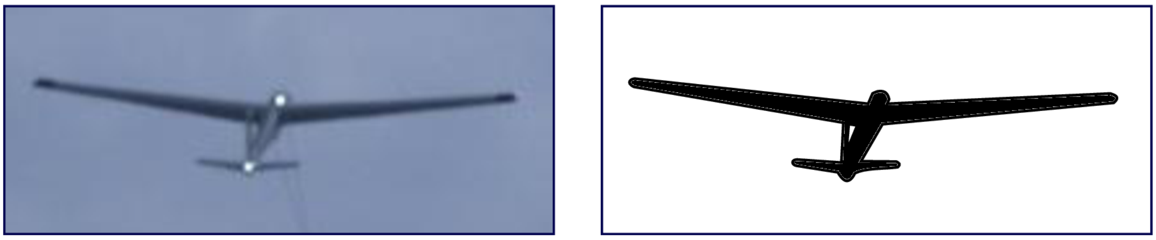
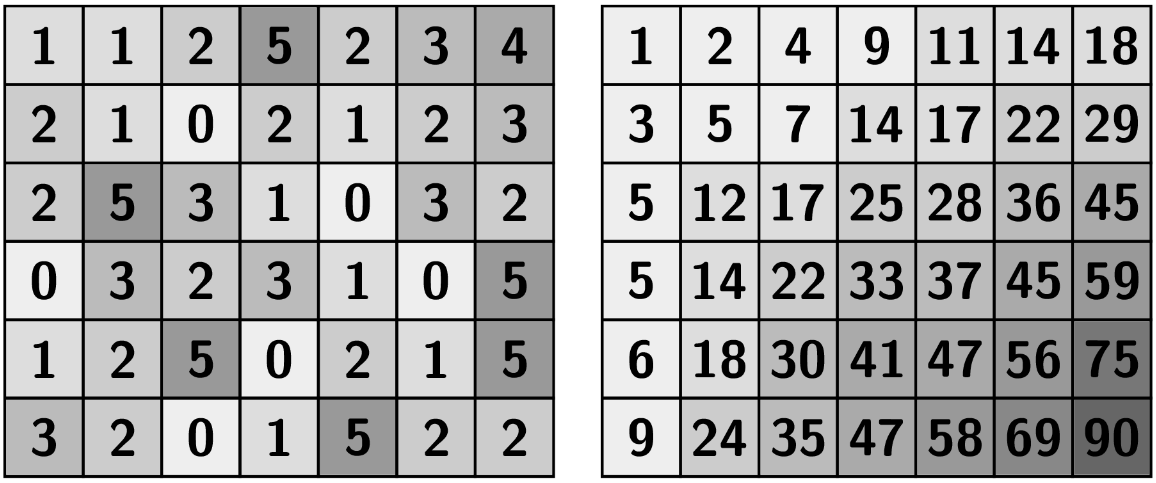


Figure 2.1: The left figure is a general template pattern. In combination with the mask on the right it is considered to be a blob.

templates and by their spatial information we are able to compare them statistically with methods like color histograms, blob proportions or similar [42].

The template similarity comparison can be done directly. The common ways are to compare variations of such templates [17, 28], brute-force matching or using technique called *Integral Image* or *Summed Area Table* [43, 38]. The *Integral Image* is able to make rapid comparison of two matrices, no matter if they are monochromatic images, variances or anything else. It is also very common to perform these methods on input templates after filtering with e.g. edge detection techniques [31].



1	1	2	5	2	3	4
2	1	0	2	1	2	3
2	5	3	1	0	3	2
0	3	2	3	1	0	5
1	2	5	0	2	1	5
3	2	0	1	5	2	2

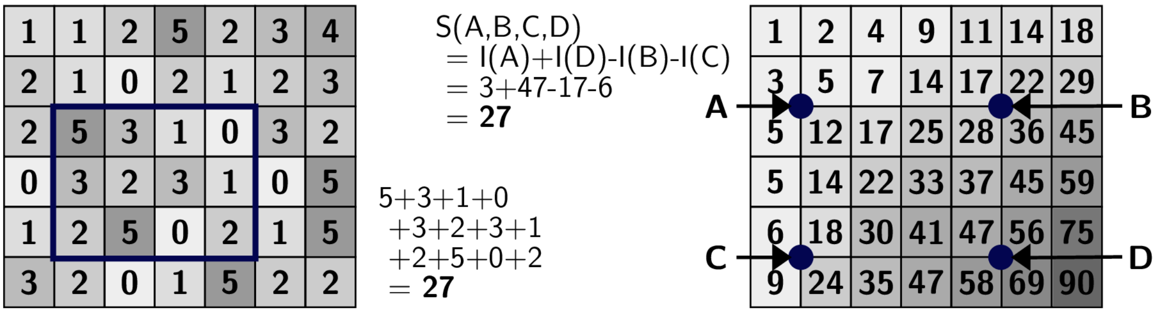
1	2	4	9	11	14	18
3	5	7	14	17	22	29
5	12	17	25	28	36	45
5	14	22	33	37	45	59
6	18	30	41	47	56	75
9	24	35	47	58	69	90

Figure 2.2: Illustration of an Integral image computation.

The main idea of the *Integral Image* method is an alternative image representation, where every value of point (x, y) is a sum of all point values above and to the left of the computed point, inclusive. That can be seen on equation 2.1, where $I(x, y)$ represents a value of the point (x, y) on the integral image and $i(x', y')$ represents a value of the point (x', y') in the original image. An illustration is in figure 2.2, where on the left is an original image (represented as a matrix of values, that can be e.g. intensities or variances) and on the right there is a corresponding integral image.

$$I(x, y) = \sum_{x' \leq x \wedge y' \leq y} i(x', y') \tag{2.1}$$

Then, the value of any rectangle of the obtained integral image can be done in constant time in exactly 4 matrix accesses [43], as can be seen in figure 2.3. In the image there is a rectangle defined by 4 vertices A, B, C and D and the corresponding area in the original



1	1	2	5	2	3	4
2	1	0	2	1	2	3
2	5	3	1	0	3	2
0	3	2	3	1	0	5
1	2	5	0	2	1	5
3	2	0	1	5	2	2

S(A,B,C,D)
= I(A)+I(D)-I(B)-I(C)
= 3+47-17-6
= **27**

5+3+1+0
+3+2+3+1
+2+5+0+2
= **27**

1	2	4	9	11	14	18
3	5	7	14	17	22	29
5	12	17	25	28	36	45
5	14	22	33	37	45	59
6	18	30	41	47	56	75
9	24	35	47	58	69	90

Figure 2.3: Example of an Integral image computation.

image. The sum of the area $S(A, B, C, D)$ is easily computed, as shown on equation 2.2.

$$S(A, B, C, D) = I(D) + I(A) - I(B) - I(C) \tag{2.2}$$

Unfortunately, the *integral image* approach ignores the local spatial information. It is a price for the speed of the method. Another method is a *sum of absolute differences (SAD)* (equation 2.3a) or alternatively a *sum of squared differences (SSD)* (equation 2.3b). The idea of both is similar; compute differences of corresponding points on both templates and sum them.

$$||T_1, T_2||_{SAD} = \sum_{i=0,j=0}^{m,n} |T_1[i,j] - T_2[i,j]| \tag{2.3a}$$

$$||T_1, T_2||_{SSD} = \sum_{i=0,j=0}^{m,n} (T_1[i,j] - T_2[i,j])^2 \tag{2.3b}$$

The equations represent going through the both 2D templates and computing the differences. As there is no normalization part, both methods are quite inclined to illumination changes. On the contrary to the *integral image*, the sum reflects the spatial information, but it also brings higher computational cost; the number of calculations depends on the pattern size and is not constant as in the *integral image* approach.

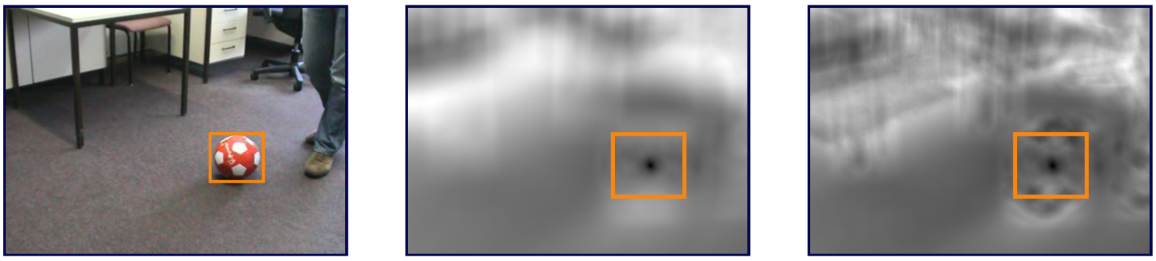


Figure 2.4: Illustration of the template matching with usage of the SAD and the SSD. On the left there is an original image and the searched pattern. In the middle there is a result matrix for the SAD and on the right side there is a result matrix of the SSD.

In a set of figures 2.4, an illustration of *SAD* and *SSD* can be seen. In the left image, there is an image to be searched with a highlighted template to be searched for. In the middle there is a 2D matrix describing the *SAD* results; the darker the point is, the lower the output $||T_1, T_2||_{SAD}$ is. In the right image there is similar 2D matrix, but reflecting the *SSD* results.

A different way to compare matrices can be *normalized cross correlation (NCC)* [20], which is due to the normalization also invariant to uniform illumination changes.

$$\gamma = \frac{\sum_{x,y}(f(x,y)-\overline{f})(t(x,y)-\overline{t})}{\sqrt{\sum_{x,y}(f(x,y)-\overline{f})^2\sum_{x,y}(t(x,y)-\overline{t})^2}} \tag{2.4}$$

The equation to compute *NCC* values can be seen in 2.4 [20]. The equation result is a value of *NCC* between two two-dimensional templates of the same size $M_x \times M_y$. The $f(x,y)$ denotes a value in the first image on position (x,y) and hence the $t(x,y)$ represents a value in the second image on the same position. The $\overline{f}$ represents mean value of the whole template f, the $\overline{t}$ is set analogically as a mean value of the template t.

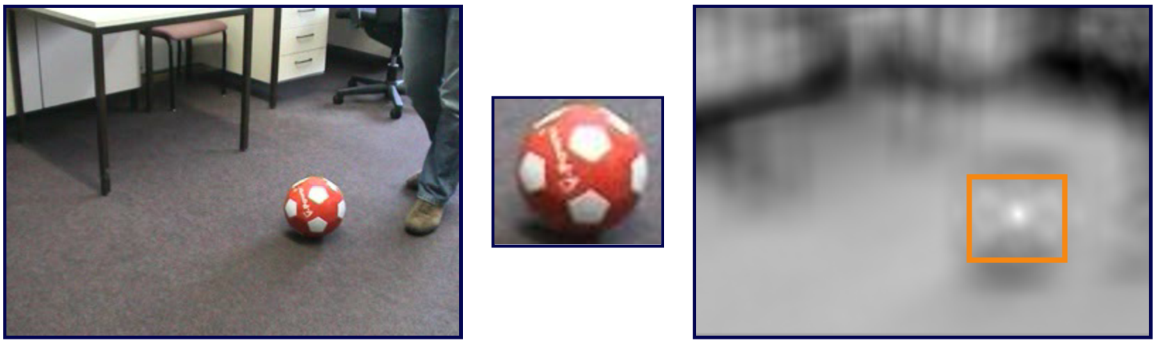


Figure 2.5: From the left: A sample image, a template, which is searched on the sample image by NCC metric and the result values of the NCC represented as a 2-dimensional matrix with a marked maximum.

A visualization on *NCC* method can be seen in 2.5. The lightest location symbolized the highest values, which means the most probable position of the template T in image I. The computation of the *NCC* for two large images can be computationally quite expensive, but the method gives generally more accurate results than the *Integral Image*.

2.1.2 Keypoints

Another option to represent the object of interest is to describe it with a limited set of good-to-track keypoints. The term keypoint should be in this work understood as a significant point like corner or peak with its neighborhood, commonly known as a *point descriptor*. The point descriptor should be invariant to rotation, scale and illumination changes. There are various widely used descriptors, such as [3, 23, 1, 37, 19]. An illustration of detection some of these keypoints can be seen in figure 2.6.

This method is more sophisticated than templates, but the quality of the keypoints is critical. The benefit of such approach is a highly robust system with the ability to handle object's appearance changes as well as the partial occlusion. This is achieved with the ability to track an object with a very limited set of original points.

Also, the methods based on keypoints are mostly quite fast with high efficiency, which was approved by [29] or [17]. A problem occurs in case of overly various and/or cluttered background, where a number of keypoints are detected. Therefore, many of true negative comparisons are performed in contrast with a very few of true positive ones. In that situation, the performance of such a method is very poor at least in the area of speed and computational cost.

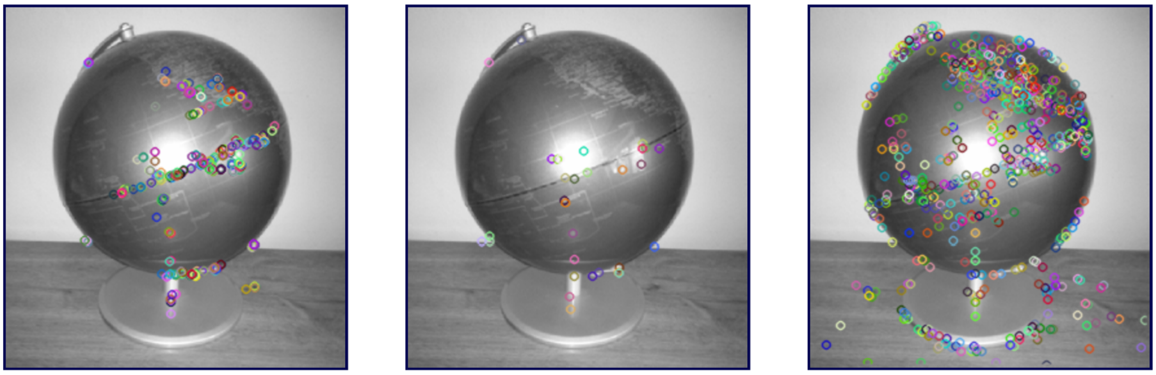


Figure 2.6: Images with highlighted positions of detected keypoints. From the left to right they are ORB [37], BRISK [19] and Good Features To Track [40].

In [40] they measure quality of the keypoints to track also in the time domain and therefore more stable keypoints can be chosen for tracking. However, even extremely various scene can be missing significant stable keypoints and therefore provide very bad tracking quality. This is an opposite problem to the abundance of such points described in the previous paragraph, but the problem is equally serious. The effect of the lack of any significant features is the lost ability to repeatedly succeed in finding the object of interest and therefore inability to track.

The point descriptor is encoded in the form of a feature vector. Then, the similarity can be measured simply with computing Euclidean or Mahalanobis distance between these vectors [3, 23, 24], or in case of e.g. ORB [37] or BRISK[19] descriptors, which are both encoded in binary vectors, the distance is computed using *Hamming distance* algorithm.

2.1.3 Contours

From time to time it is convenient to represent the object its geometry, also called a *contour* [5]. This approach is often better than others when we are able to segment the object from its background effectively. The suitable objects to represent with contour can be cars on the road, where the road - the background - is relatively solid color and quite clearly distinguished from the car - the object. Also, methods to track humans with representing head as a contour and the body as an adjacent blob occurred. However, those methods have poor results if occlusion appears.

The contours are usually based on response from edge highlighting filter. Based on the image structure it is common that a lot of not wanted edges come up. The object is the described with a set of connected edge fragments. This approach is generally more resistant to object non-rigidity as well as to the occlusion problem. In figure 2.7, an illustration of detecting contour described object can be seen.

In case of measurement of similarity of contours, it is convenient to use the limited geometric information. Therefore, the contours are usually compared by their normalization to predefined shape while measuring parameters needed to do the conversion [8].

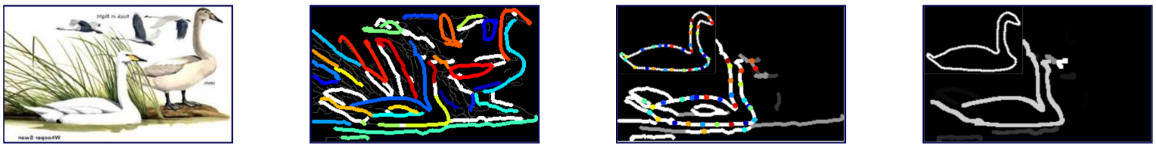


Figure 2.7: Illustration of a contour representation and matching. From left to right: an original image, detected long salient contours, correspondent contours found and marked and a final image with joint contours. The images and the example are taken from [47].

2.1.4 Complex Representation

The last representation method mentioned here is the combination of the ones above. The resulting model is then built from other models (keypoints, templates, contours) which are connected together. The connection can be set e.g. as a range of allowed angles for the given parts [46, 21].

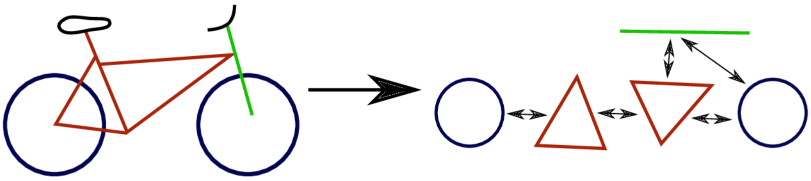


Figure 2.8: Illustration shows a simple object - bicycle - and its decomposition to simple elementary parts - triangles, a line and ellipses - which are connected by relations.

The advantage of a model built from a graph of multiple parts is its ability to detect the missing part and therefore handle partial occlusion and appearance changes. The disadvantage can be the need for a big enough object to split it into submodels, which can lead to inability to handle custom objects. An appropriate object for complex representation can be a *bicycle*, as illustrated in figure 2.8.

A similarity measurement approach can be seen in [39], where they experimented with *Euclidean* and *Cosine* distance on separate connections among the parts of the object.

2.2 Object Localization

The last missing part, if we already have the object representation and the similarity measurement tool is the object localization. There is a short overview of the current approaches to object localization in the scene in this section.

For simplicity, the section is divided into two subsections - the object detection in subsection 2.2.1, which contains methods of detection without any knowledge about the scene history, and the recursive tracking in subsection 2.2.2, which is a short summary of methods working through time domain and not merely in the present frame.

2.2.1 Object Detection

The object detection is the very first approach to object tracking. The task is to detect the object on the scene on every frame and to give its position; otherwise, to inform about the object's absence. This section is on different approaches to the search of the object in the frame.

Keypoint Detection

Despite its name, the *keypoint detection* is not relevant only to keypoints described in 2.1.2. The idea of this approach is to localize all significant points (e.g. corners, peaks, Good Features to Track in [40]), compare them with the searched one and choose the most similar. As this is the basic idea (and works very well with keypoints like SURF [3], SIFT [3] or BRISK [19]), the searched keypoint can be basically anything, from blobs, geometric shapes and so on.

The only thing which needs to be guaranteed is the detection and comparison speed. The critical point is to decide quickly whether the keypoint (or blob, shape, contour) is or is not the same type as requested. Then, the located objects of the same type are compared with the original one, but this time it is not a binary decision, but more often a probabilistic one. This comparison does not need to be as fast the binary matching in the previous phase because we have only a limited set of the keypoint of the same pattern.

Sliding Window

Similar to the previous approach is the localization method called *sliding window*. The main difference is that the binary comparison is omitted and it is substituted with the probabilistic comparison instead. This approach can bring the possibility of comparing complex objects like templates, which we are frequently unable to classify as binary in the early detection.

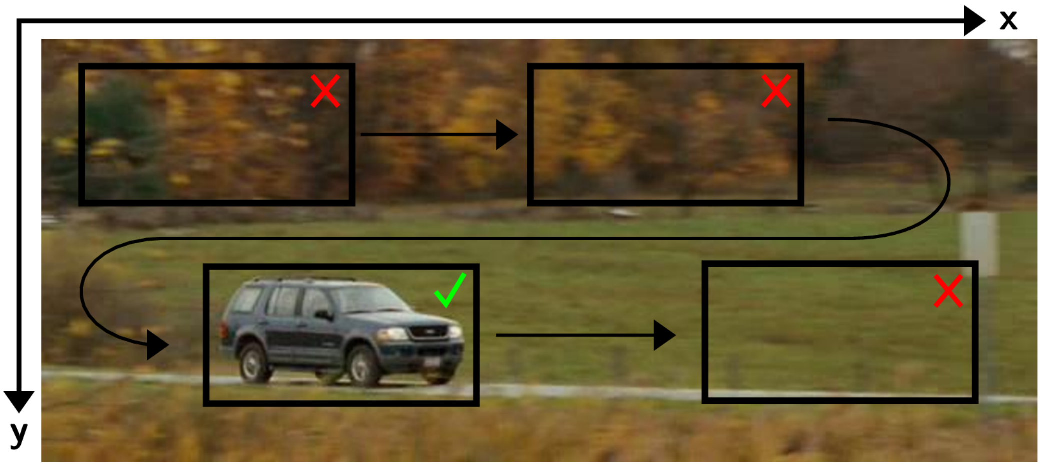


Figure 2.9: Sliding window detection illustration. The distance between every allowed position and the searched template is measured by *sliding* the template through the image.

This approach is a basic brute force method of detection. The main idea is to divide the whole frame into multiple subwindows, do comparison of every subwindow with the original template and choose the most similar subwindow. If the similarity is bigger than the given threshold, the result is marked as valid; otherwise the user is informed that the object is not found. For such window-to-window comparison, methods like previously described *Integral Image* or *NCC* are commonly used.

With equation described above (Equation 2.4) the *NCC* can be computed for all of the possible template positions of the investigated image. The result is then a set of *NCC*s for different shifts of the template in the image. Therefore, it can be easily represented as a map, as can be seen in figure 2.5.

The biggest disadvantage of the sliding window approach is the number of subwindows to compare, and therefore a very high computational cost. The reason of such a high number of subwindows is a small step (*slide*) to manage as many possible object positions as possible. Moreover, all the windows can be in different sizes due to the object scale allowed. All of these bring us a huge number of subwindows which are not easy to compute in a reasonable time [43, 18, 28, 11].

A way to reduce the computational cost of the sliding window approach can be for example fixed rectangle size and fixed sliding step (e.g. not by 1 pixel) [36] with so-called *pyramidal image* to obtain object scale.

2.2.2 Recursive Tracking

Unlike the previous section, the *recursive tracking* describes a group of methods which do the tracking itself. In this work, the term *tracking* stands for the detection with respect to the previous frames and usage of the history

It brings better handling of missing information, but a potential processing error can be cumulated over time. There is also more information in a frame-to-frame recursive tracking then in a single frame image.

Therefore, recursive tracking is an approach much related to the object tracking. The reason is clear - the object tracking is performed on top of a sequence of frames and it comes with higher density of information than a single image. If it is possible, it is a good approach to use this kind of additional information.

Optical Flow

Optical flow is a method to determine flow of points in two consecutive images. Basically, it is exactly the movement which is interesting in the object tracking.

The idea of optical flow is quite old. The differential optical flow (because they use mainly Taylor series approximations; that is, they use partial derivatives) is understand as a 3 dimensional vector $(\Delta x, \Delta y, \Delta t)$, representing the movement of distance given by vector $(\Delta x, \Delta y)$ in time Δt. This is shown on equation 2.5a. The $I(x, y, t)$ represents a value of pixel at location (x, y, t) and the Δx, Δy and Δt represent the movement between two frames in times t and $t + \Delta t$.

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{2.5a}$$

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + \dots \tag{2.5b}$$

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \tag{2.5c}$$

The equation 2.5b is based on assumption of very small movements and therefore the Taylor series can be developed and the higher-order terms can be omitted. These equations directly lead to 2.5c, where V_x and V_y are components of the optical flow of the intensity $I(x, y, t)$, and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are the derivatives of the image in the corresponding dimensions [4, 10].

Although many approaches have been developed, only two significant ones will be mentioned here. The first is an optical flow method by Farnebäck [9]. He computes dense optical flow; it means that for every point of an image, an optical flow is estimated. It can be generally more inclined to image noise and the computational cost is high due the method density. On the other hand, the density can bring higher accuracy.

The second one is the method of sparse optical flow from Lucas and Kanade. They invented the optical flow method based on the expectation that none of the points is moving in frames independently, but it is always in some blocks [25]. According to that, the Lucas-Kanade optical flow based trackers are resistant to noise and outliers, which makes them relatively stable and very fast. Also, they do not need such good keypoints to track, although they still need them [16].

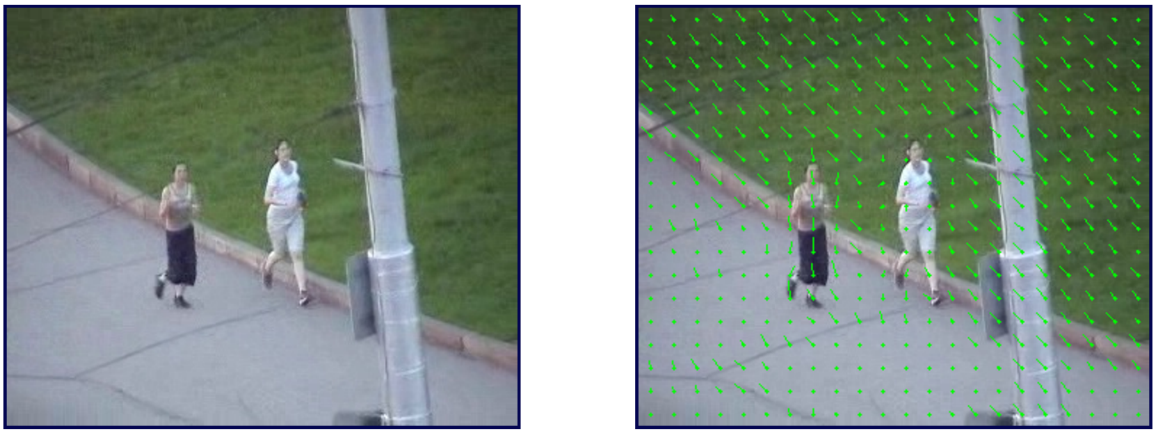


Figure 2.10: Illustration of the optical flow. On the left there is one frame (out of the pair) and on the right there is visualized optical flow (subsampled).

The size of the neighborhood is important and it leads from the Lucas-Kanade optical flow itself. As described above, the points move in blocks. This *block* is our *neighborhood* and it is basically an equivalent of the resolution ability of the tracker. A smaller block size means higher resolution of resulting tracker, but it is also limiting value for pixel displacement.

According to that, one of the requirements of Lucas-Kanade optical flow algorithm requires small object movements. That can be very limiting for many applications. Fortunately, this is solved with pyramidal implementation [6], which allows computing the flow in different levels of frame size. Therefore, we are able to get compute optical flow in different levels while keeping the neighborhood small without losing spatial information. To give an example, for the pyramid of depth $L_m = 3$ pixel displacement gain can be 15. This actually means rather large movements while keeping the neighborhood small and high quality resolution.

Another assumption of the method related to the spatial coherence is also solved with pyramidal implementation. It says that all the points in one block move in the same direction. As written above, the pyramidal implementation allows having small block size and still keeping the performance of spatial point displacement, therefore we can have small enough blocks to meet the spatial coherence without any performance reduction.

Kalman Filter

A Kalman filter is an approach for filtering of usually exceptionally noisy input values to produce the best estimation of the output values. The output values tend to be more accurate than using a single noisy input values. It is based on the recursive nature of the filter, which means that the output value estimation is based on the previous values.

The filter is very versatile. It is very good in applications such as signal filtering mentioned above, when we have a lot of noisy input values. Other application can be related to object tracking. It is basically the same - we have a lot of frames in a second and we are processing real-time. So we can look at every frame result (meaning a position of the object we are tracking) as at a noisy one, put them into Kalman filter and produce result, and if the prerequisites for Kalman filter are passed, we can expect that the Kalman's output will be more accurate.

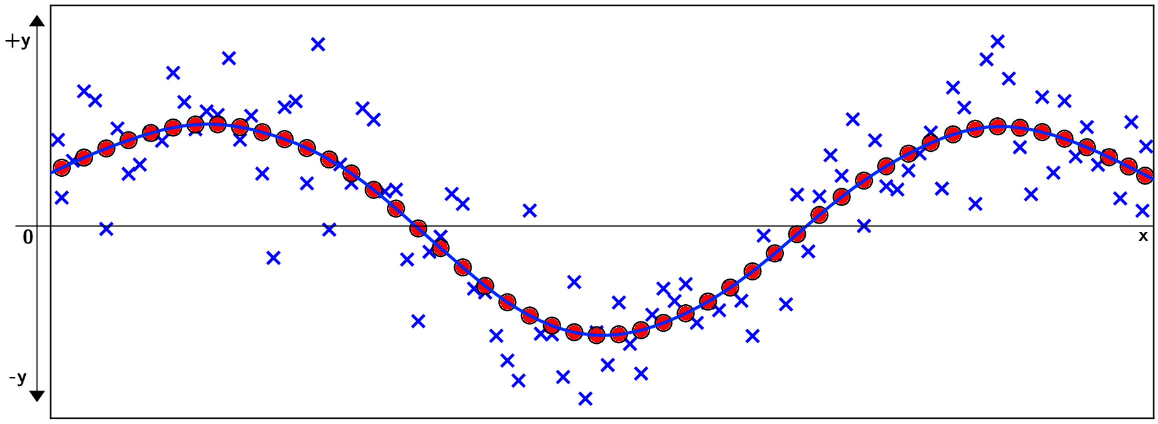


Figure 2.11: Illustration of the Kalman filter. The blue curve is the correct sine, the blue crosses are raw noisy sine data and the red dots are points filtered by Kalman filter.

An illustration can be seen in figure 2.11. It shows a raw data from noisy sensor, which are simulated as a sine values with randomly added noise. Then, you can see the original sine curve and a filtered curve estimated by Kalman filter.

We can go even further. As well as the Kalman filter can estimate a value in time t based on noisy values from times t_0 to $t-1$, we can shift the meaning and estimate a value in time $t+1$ based on values from times t_0 to t instead. This way we can get estimation of the object position in next sequence frame. It is still an estimation, but if the object's motion model keeps unchanged, it will be quite accurate.

The algorithm consists of two steps; the *prediction* and the *update*. During the prediction step, the new system state (the *object position* in our case) is a new state and the covariance is predicted from the previous ones respectively. The estimation is based on the actual state and transitional function, and the Gaussian noise added to every prediction. Consequently, during the update step, the correction and the new posteriori state are computed. In figure 2.12 an illustration of computation of Kalman filter can be seen.

The power of Kalman filter is in the correction. In focus on the object tracking, if the estimation is successful and is very close to the real position, in the very next step the algorithm will be concentrating to smaller area because it *trusts* in its predictions. Otherwise, if the estimated position is far from the real position, the next step searched area will be bigger [44, 45].

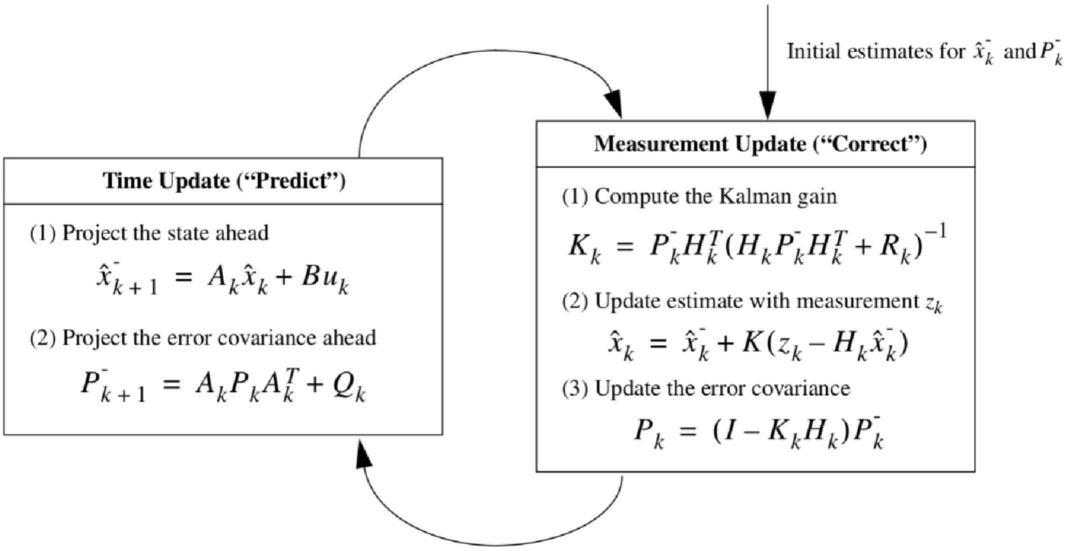

Initial estimates for $\hat{x}_k^-$ and P_k^-
Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_{k+1}^- = A_k \hat{x}_k + B u_k$
(2) Project the error covariance ahead
$P_{k+1}^- = A_k P_k A_k^T + Q_k$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K(z_k - H_k \hat{x}_k^-)$
(3) Update the error covariance
$P_k = (I - K_k H_k) P_k^-$

Figure 2.12: A complete diagram of Kalman filter computation (taken from [44]).

According to the generality mentioned above, the Kalman filter has been widely used among many object tracking algorithms [33, 12].

Particle Filter

The Particle filter is another kind of a probabilistic estimation approach. It removes the limitation to the Gaussian distribution used in the Kalman filter and can also cover non-linear system complexity. In the particle filter system, the possible states are modeled with a set of samples (the *particles*) and their weights.

A typical representative of particle filters can be a sequential Monte Carlo particle filter. The algorithm itself consists of two main steps called *prediction* and *update*.

During the prediction step, each particle which represents the next possible object's position is modified according to the previous values and its state model, and a noise in order to simulate the effect of the noise on the state [7].

Then, the weight of each sample is recomputed, based on the new data when their success is measured. In the object tracking application, this means measuring probability of every sample that it covers the real position of the object in the new frame. It is measured with model specific similarity measurement which depends on the chosen model.

The last step, which is very often considered as a part of the *update* step, is the *resampling*. This brings the *non-linearity* mentioned before. The idea is that the low-weighted particles cover most likely the space where the object is with the least probability. Instead, high-weighted particles most probably cover the real object position on the new data. Therefore, the low-weighted samples can be omitted from further computation and their sources can be re-used as replications of high-weighted samples.

The *resampling* step basically covers the most probable area with more particles of those, which are situated in the object's least probability position. Then we are able to cover the real object area with much density and do not lose computational performance on computation probably position in the area where the real object almost surely is not.

This is illustrated in figure 2.13, where the particle values (positions) are represented with blue dots and their weight is represented with their size.

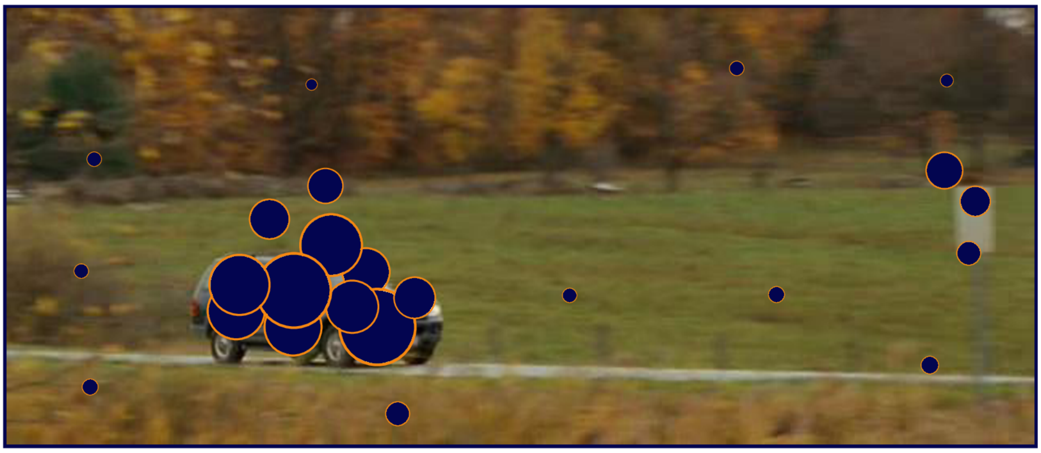


Figure 2.13: Illustration of the particle filter. The particles (blue dots) represent expected positions and their size is proportional to their weight. As can be seen, the most particles are concentrated around the most expected position, while only a few cover the rest of a space.

Another advantage over the Kalman filter is the system's non-Gaussianity. This comes from the particles' character. We are able to model probability peaks on more places at once because every particle can stand for one. This leads to possibility of multi-object detection as well as much better single object detection [14, 7].

It is obvious that the particle filter is suitable for the object tracking purpose. According to that, many object tracking issues have been solved with the particle filter as the prediction unit [30, 42].

2.3 Unsolved Problems

As was noted in the introduction (Chapter 1), some unsolved problems still remain in the field of the object tracking (and especially the long-term variant). The most problematic issues are mentioned in this section.

2.3.1 Partial Occlusion

Partial occlusion is a very common problem in the area of object tracking. In most of the cases, we are not tracking an object in ideal video without any other objects on it and with solid color background. Instead, we have various scenes where the object is moving randomly; sometimes it is partially hidden behind other fragments and sometimes it is hidden completely. Partial occlusion is a problem when the object is still partially visible, yet we want to track it, as seen in figure 2.14.

To be more precise, the partial occlusion is such situation when a part of the object is behind other object. The problem comes from the projection of 3D space of the real world into the 2D space of a video. In the 2D space an object can be easily hidden as the depth dimension is generally omitted. To go further, such an occlusion can last only few frames

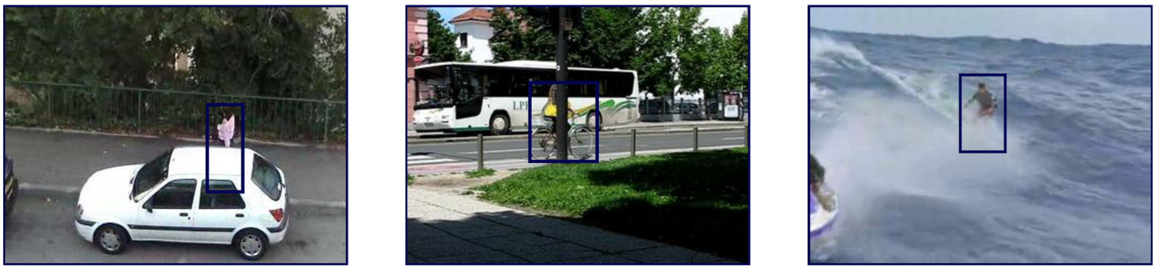


Figure 2.14: The partial occlusion. The objects are partially hidden, yet we want to track them.

as well as a half of a video sequence; both options are permitted and the system generally does not have any clue about it.

In the section about object representation (Section 2.1), the problem of occlusion was mentioned briefly. As we want to continue in tracking even if it is occluded partially, we need to choose such object representation and matching algorithm, which are immune enough to noise and missing data.

This problem can be solved with trackers and detectors which are able to handle incomplete models. It can be generally done with keypoints. We are able to say that the limited set of original keypoints, which was successfully founded (the rest was unable to found due the occlusion), represents a part of the very same object/part of it, if the relation between them remains unchanged. Of course, this can be problematic in case of non-rigid objects, but generally it works.

Another way can be a usage of image patterns. In case of searching for a whole template and a half of it is not visible, we have 50% error at the best. This is quite low, because we need to count with the template changes themselves. As a solution may seem to divide the whole pattern to a set of patches and to search them individually, which brings us to the advantage of keypoints mentioned before. An intensive research on the occlusion problem was done [12, 46].

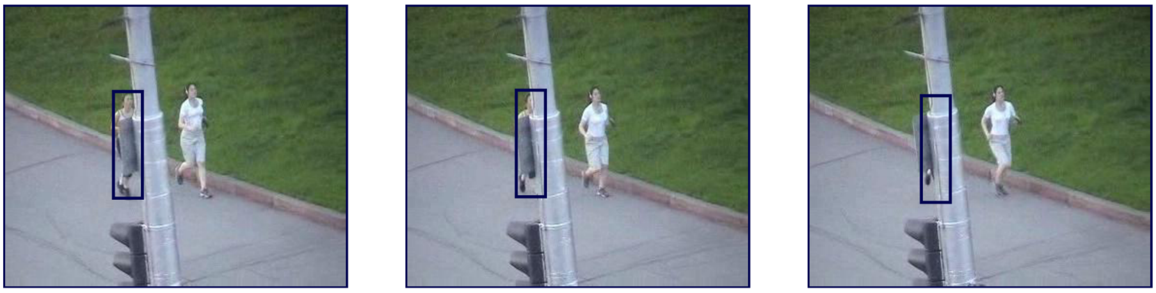


Figure 2.15: Problematic partial occlusion situation. In the illustration there is an object - the runner - which is almost completely hidden behind barrier. The question is if it is a partial occlusion or if the object is hidden already.

To determine whether the object is still partially visible or it is hidden completely is question for deeper investigation. Very often the problem is self-solved by the tracker/detector limits. The illustration of unsolvable problem can be seen in figure 2.15.

2.3.2 Full Occlusion

Despite resembling the partial occlusion, full occlusion needs to be solved differently. In general, full occlusion happens when the object disappears from the scene or it is behind another object on the scene. In these situations, the user should wait until the object reappears.

The full occlusion problem almost always follows after the partial occlusion. The moment, when the partial occlusion changes to full occlusion can be illustrated with in figure 2.15. Regarding to that, the partial occlusion very often precedes and follows the full occlusion. The only thing is to determine the moment when the partial occlusion changes to the full occlusion.

The problem of full occlusion results very often into a failure of the tracking system. An exception can be systems based on *particle filters* or other prediction approach, which can model the object movements even when it is fully hidden, with the movement history only. The movement model will be less accurate every step without the correction and a deviation between the correct and the predicted position will grow. Furthermore, unexpected object movement (in meaning of difference from the movement model) while it is behind a barrier as well as the unexpected camera movement during the full occlusion will break the prediction.

Another option is simply admit that the object is lost. Then the tracker cannot continue in its job and the situation is commonly solved by cooperation with a detector, which is started in the particular moment. The detector then searches for the object with usage of information it gained during the period when the object was visible. As it does not have any information about object position, it searches generally the whole frame. After the object is found, the full occlusion ends and it usually continues in partial occlusion again, which can be handled by the tracker again.

The duration of the full occlusion is a critical part. Not from the point of view of time, but in terms of the period at which the object may change. If the object changes significantly during full occlusion (the example can be a human which changes clothes in a changing room), then it is usually considered as a different object by the detector and the full occlusion ends with failure.

2.3.3 Long-Term Aspect

Another generally unsolved problem in tracking is the *long-term aspect*. It includes changes in behaviour and appearance of the object during time, primarily scale changes, rotation of the object and ideally other object deformations. The illustration is in the group of figures 2.16.

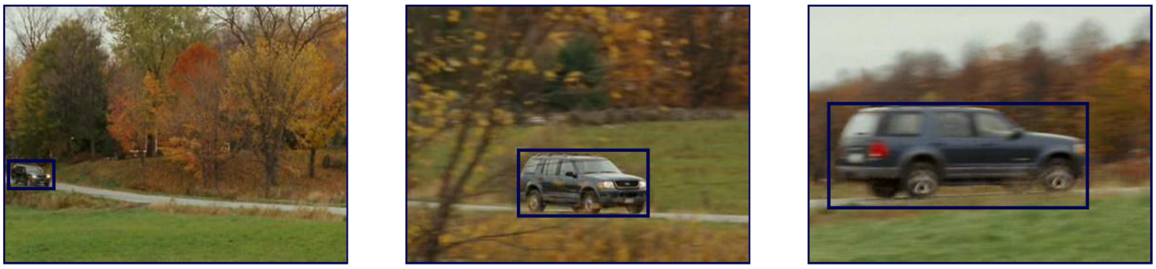


Figure 2.16: The appearance changes during time. From left to right there are frames nr. 1, 181 and 252 of the same video sequence.

The important thing is to limit how much the object can change and still be referred as the same object. The problem was described in previous subsection, and it is hard to finally decide. An example can be a sheet of paper and a question if it is still the same object when a paper plane is build out of it.

The first problem which needs to be solved is the shortest aspect of long-term tracking. We should be able to detect the object changes like scaling, rotation, uniform illumination changes or non-rigid object changes. They should be solved by the system in part of its robustness.

A different problem from the previous short aspect are the genuine long-term changes. In real long-term tracking, the object evolves and as it changes the appearance in time, the system should adapt to the changes. The ability to cover such long-term changes comes from the object description type and from the ability of the system to learn.

An example that the way of object representation can have significant influence on the ability to cover object long-term appearance changes can be [29]. In that work, the object is represented by a set of keypoints which are connected to each other. As the system is model-free, the keypoints and the connections may vary over time and it is quite easy to update them as the changes come.

Generally, to handle the appearance changes during the time is an engagement of an adaptive detector/tracker. Some ways of solving it in the tracking were described, but there are also a number of adaptive detectors [26, 15, 2, 30]. The idea is to update a detector's knowledge base with some kind of online learning, or at least updating of the base.

2.3.4 Cluttered Background

One of the last mentioned problems here is cluttered background. This problem is very frequent in the real world scenes, especially from outdoor scenes, e.g. urban intersections [33] or others [24, 14]. The illustration in figure 2.17, where the object's position is difficult to determine even for a human, can be considered as a cluttered background.

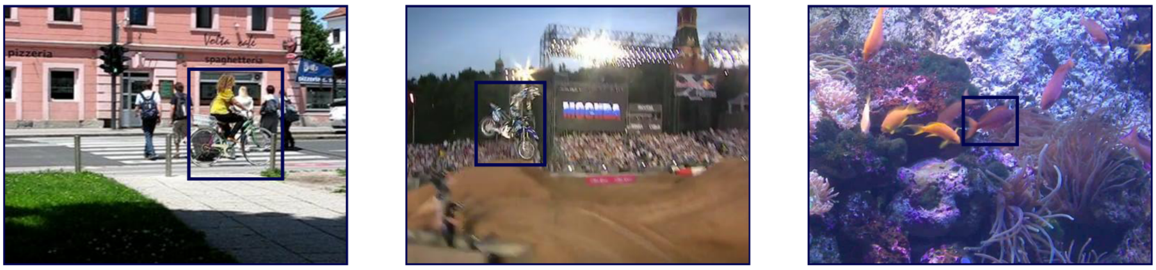


Figure 2.17: The cluttered background. Three different images from separate dataset to visualize the background clutter. In all of the images, the object is really hard to find.

For keypoints, the intensive background clutter can be critical; too many false positive points can be detected and therefore the performance of the matching can be very poor. This is explained in section 2.1.2. In general, the detection in clutter scene in computer vision is basically identical to the same situation in human vision; finally we are able to find the object, but we do a lot of false match comparisons.

2.3.5 Similar Objects

The term similar object refers to an object very similar (identical) to the object tracked. The problem occurs very often if we track an individual in a group of the same ones. In such case it is very important to keep attention and the continuity of the tracker. The problem is generally unsolvable in two frames which are not consecutive (Figure 2.18).

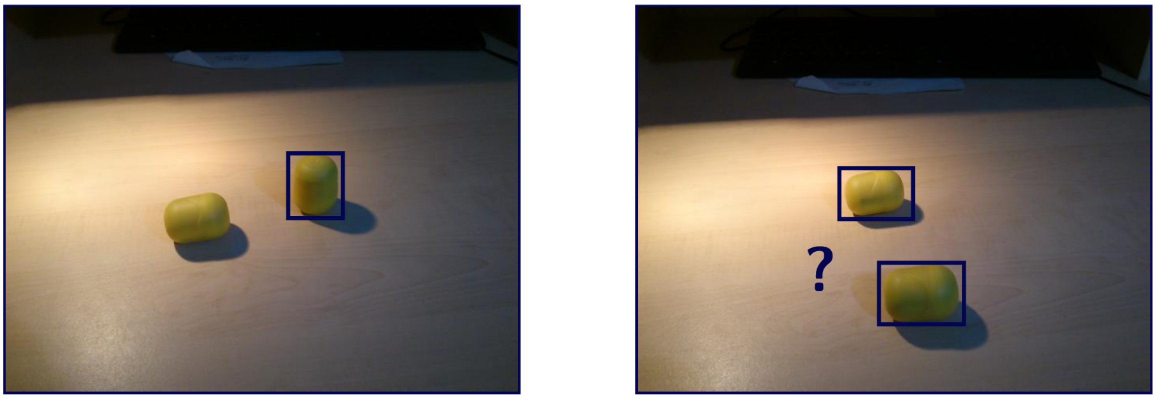


Figure 2.18: It is impossible to determine which object is which one in two frames which are not directly consecutive. On the left image there are two objects, which are well known. On the right image, both objects changed their position and it is impossible to distinguish them.

Apart from the obviously unsolvable problem, the problem of *similar objects* is a task for the tracker. The solution is possible only with the usage of objects trajectory and therefore the position estimation.

The problem can be solved also with modelling of *all* objects in a scene. Also, this method requests knowledge of history, but it could be easier as we know how many, where and how much similar objects are in the scene. The disadvantage is usually higher computational cost than in tracking a single object [27, 41].

2.4 Summary

In this section, some important work related to object tracking in a video was summarized. At first, the possibilities of representing an object were discussed, as this predetermines the whole system abilities. Then, various approaches to localization of the object in a scene were presented, which was split into two main ways; in-frame localization and frame-to-frame localization. It was shown that both ways have their advantages and disadvantages.

At the end of this section some generally unsolved problems of object tracking systems were described. Those problems occur in almost every real world tracking system and ways how to solve them are very often contradictory.

Chapter 3

Proposed Solution

This chapter describes a design of the proposed system. Observed problems with current technologies in all parts of the long-term object tracking and proposed solutions to these issues are described here. Furthermore, a way to incorporate them into the system is proposed. The system is also described from the view of expectations and requirements set before the beginning.

First of all, the expected behavior and the system requirements, as set in planning phase of this thesis (Section 3.1), are described. Afterwards, the basics and background on which this system will be built are summarized in section 3.2. Subsequently, the main part of the design itself is discussed.

Section 3.4 is focused on the tracking itself, observed issues in the new application and the solutions to these issues. The detector design used in the system is described in section 3.5. Section 3.6 is focused on the long-term task concerning the meaning of learning and updating the detector's model.

3.1 Expected Behavior

In this work, a system with the ability to track custom objects across many frames and time is going to be developed. The aim is to track either a small object without any significant points good to track, or bigger objects - the standard tracking objects, such as people, cars, balls, and so on.

The input of the new system is a video or an image sequence in a grayscale color model and a user input. The user is supposed to mark the object with a bounding rectangle in order to enable the tracking itself. The system output should be the position of the object – the centre coordinates and the bounding rectangle size. A complete pipeline can be seen in figure 3.1.

In case of the object disappearance, the system output should also notice this fact and it should be able to continue the tracking as soon as the object reappears. The new system should be able to cope with a partial occlusion as long as possible.

3.1.1 Objectives to Achieve

A goal for speed and accuracy domain was set, too. The speed requirement for the proposed system is real-time or near real-time processing, independently on images size and on a present-day standard personal computer. Regarding the accuracy, the accuracy should be

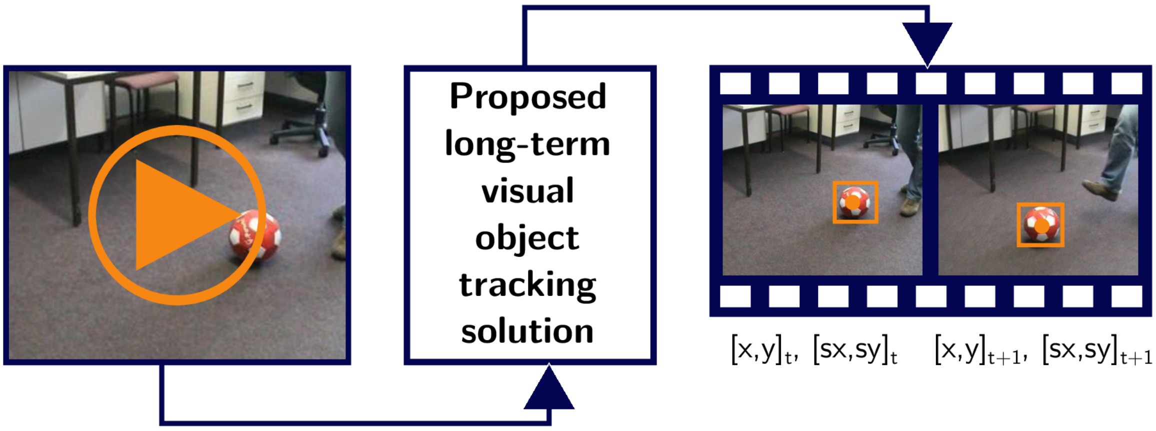

Proposed long-term visual object tracking solution
$[x,y]_t$, $[sx,sy]_t$
$[x,y]_{t+1}$, $[sx,sy]_{t+1}$

Figure 3.1: Expected behavior diagram.

at least as high as the state of the art solutions. Therefore, the goal is not to decrease the current trackers quality, but to increase it instead.

Besides reaching the state of the art accuracy, the system should be generalized enough to handle object not depending on their size or appearance correctly. The problem with smaller objects is particularly interesting. The smaller objects, which are generally all the objects distant enough, mean also less of information, either due to the low contrast quality or higher impact of an image noise.

Such objects should be also treated in monochromatic color model. The reason is again the possible object distance; the color information is getting worse with increasing distance; and system which relies on color information could be misled.

3.2 Basics and Background

The long-term tracker's task, which should be solved in the proposed system, is not easily solvable with a system based on a tracker only. Majority of trackers need to be re-initialized after their loss or after a full occlusion occurs, as was described in section 2.3.

Therefore, the system proposed in this thesis is a combination of three parts; the tracker, to track object with respect to movement history, the detector, to re-initialize tracker after object loss, and an adaptive learning system. The adaptive learning system is needed to update knowledge base for the detector during the long-term tracking.

The proposed solution is built on Georg Nebehay's master thesis system called *OpenTLD* [28], where he implemented an improved version of TLD [17] by Kalal et al. The reason to use this is that the system consists of the exactly same 3 parts as is described in previous paragraph. Another reason, but equally weighted, is that it is considered as a state of the art system for a long-term object tracking in this work.

As was mentioned, the OpenTLD consists of three parts, as well as the system proposed here. The schematic diagram of the proposed solution can be seen in figure 3.2. The tracking part from the OpenTLD is fully replaced with a new one, presented in section 3.4.

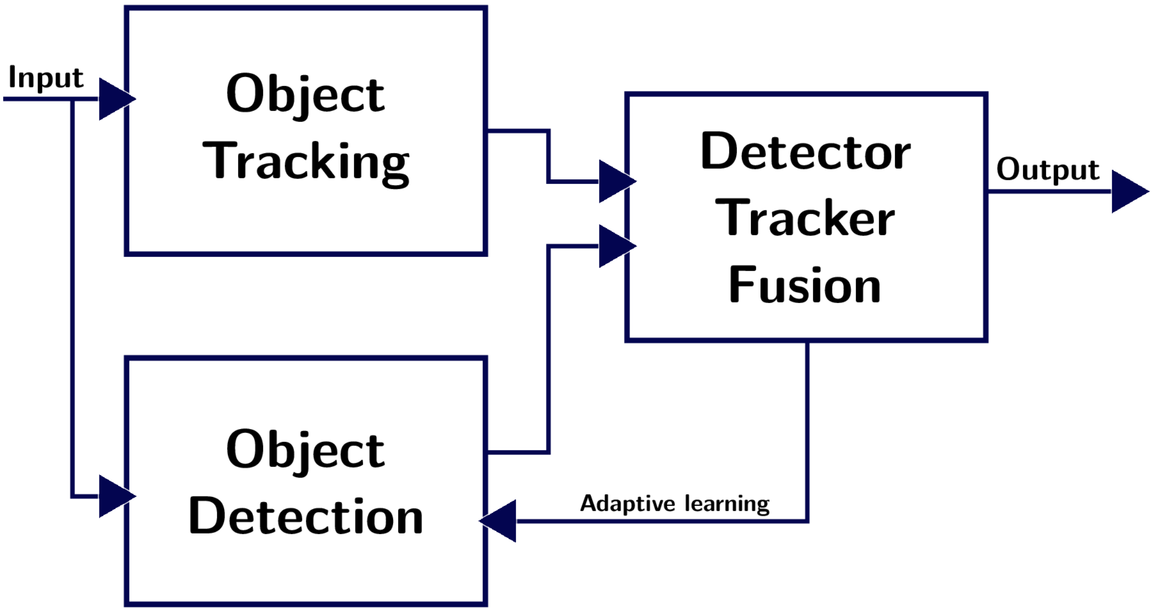


Figure 3.2: Proposed solution diagram.

The detection part is taken from the OpenTLD and remains almost unchanged. The whole solution of the detection part is described in section 3.5, including the proposed changes. The nature of the changes is rather cosmetic than functional.

As well as the detector changes nature, also the adaptive learning system is primarily taken from OpenTLD. The importance of the adaptive learning system was described in 2.3.3. As the proposed system tends to be long-term related, such a learning system should be present. The overtaken adaptive learning system is described in 3.6.

The last significant part of the proposed system is a detector-tracker fusion system. Such a part exists in the OpenTLD, but it does not do an actual fusion - it only prioritize the tracker's output before the detector's one or vice versa, depending on their successes and confidences. The proposed detector-tracker fusion is introduced in section 3.7.

3.3 Object Representation

The object representation is the way the object is treated in the system internally. Several principal methods were described in section 2.1, with main ways to compare and match them.

As the object representation affects the whole system's performance significantly, it needs to be chosen with caution. In this section an object representation is proposed. At first, the object model for tracker is described, as it is the main task what the system should perform. Then, as the system also consists of the detector, a different object model used by the detector is described in the second part of this section. Both parts contain also reason, advantages and disadvantages of these proposals.

3.3.1 Object Model for Tracker

The main task of the proposed system is an object tracking. As it was noted in 2.2.2, the tracking means a *frame-to-frame* tracking. During this task the object needs to be represented by its model, and the object model description is content of this subsection.

During the investigation phase, some critical areas where standard and state of the art trackers are failing were observed. As could be seen in many works [17, 13, 29], the standard solutions are able to track standard objects quite well. However, when they were tested to track an object on a specific dataset, which is narrowly delineated in section 5.1, they performed very poorly.

The main problem of the majority of trackers was the inability to track limited size objects due the absence of significant features to track, e.g. SURF or SIFT keypoints. In the specific dataset, the objects are typically very limited in size and good feature points are often missing completely or there are very few of them, like one or two. This is very often beyond the edge of successful tracking. To perform successful point-based tracking on such low information objects, also the points representing the objects need to be less informative.

Although the quality of such points will be low, we still can choose the best of them. The object in the tracker is represented by 16 points, which are chosen with respect to [40]. The total amount of points was set empirically, based on their success during tracking.

The points are represented in their 11 × 11 neighborhood as a raw patch. This results in 16 patches of size 11 × 11, which represent the object. An illustration of selection of the point selecting from the detection to description can be seen in 3.3.

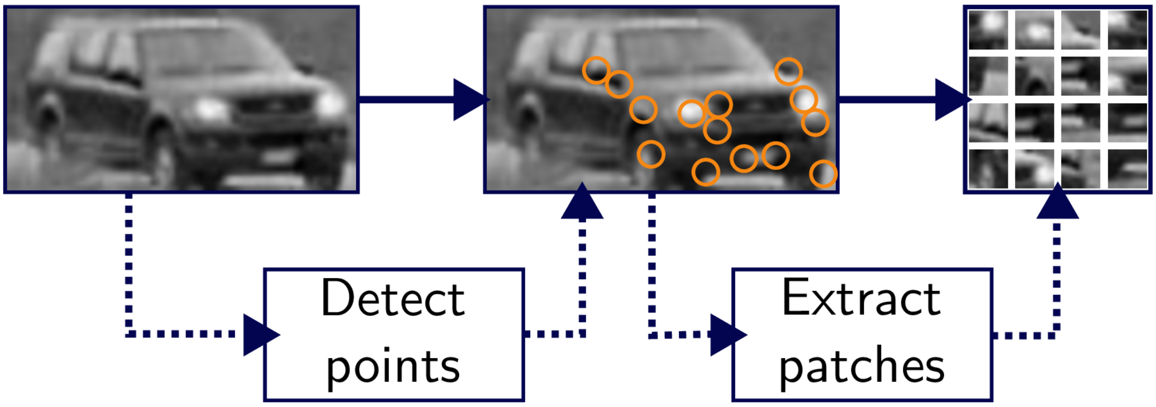


Figure 3.3: Object model for tracker.

To increase overall quality of points to track, it is important to ensure that the points are generated from the object area. As stated before, the object is determined by its rectangular bounding box. But generally, the object is not rectangle-shaped and detecting points in the whole rectangle could end with points detected on background, instead of on the object. This is obviously wrong, so a mask computation approach is presented.

The mask computation comes from the assumption that inside the object's bounding box is the object and partially a background and outside the bounding box there is only the background. The mask should reflect the difference between the object and the background to distinguish between them.

Then, the mask is 2-dimensional matrix of the same size as the original bounding box symbolizing a difference inside and outside the box. The difference is also enhanced with coefficient, which reflects a distance from the bounding box borders. The reason for that is to penalize regions in the middle of the bounding box, as the object is expected to be in the middle rather than near the borders.

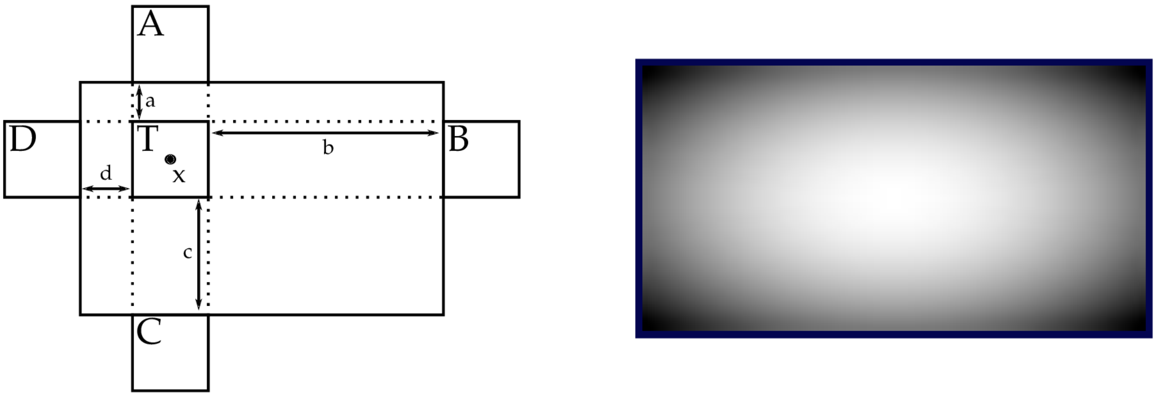


Figure 3.4: Computation of a mask for tracker point generation.

In figure 3.4 on the left, there is a diagram of mask computation. It symbolizes regions which are compared to compute their similarity and the distances for coefficient penalization. The corresponding equation is in 3.1.

$$\begin{aligned} p \quad = \quad & NCC(T,A) * (1-(a/h)^3) + \\ & NCC(T,B) * (1-(b/w)^3) + \\ & NCC(T,C) * (1-(c/h)^3) + \\ & NCC(T,D) * (1-(d/w)^3) \end{aligned} \tag{3.1}$$

On the right side of figure 3.4, there is a pure mask which represents the impact of the coefficient. The lighter area means that it is more likely the object than the background, and the level of brightness is the certainty.

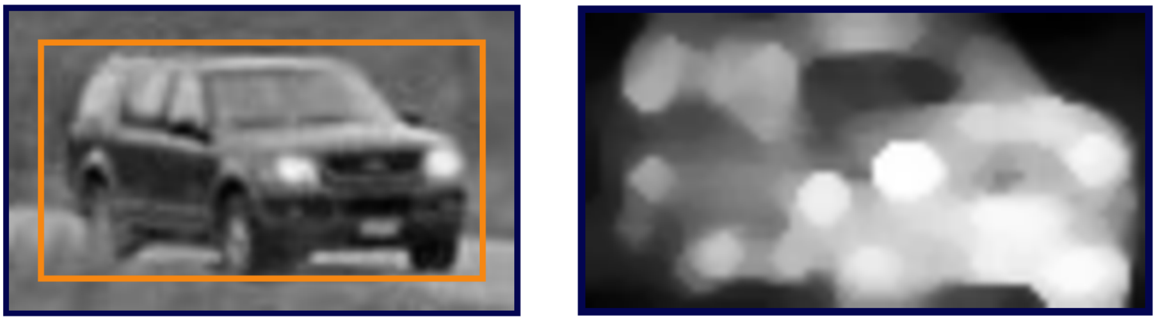


Figure 3.5: The original bounding box with surroundings and a corresponding mask.

The figure 3.5 shows the mask results themselves. On the left, there is an original bounding box with a bit of surroundings, because it is required as is written above. The original bounding box is highlighted. Then, on the right side, there is a resulting mask. It is clearly seen that the background, although it is quite cluttered, is recognized by the algorithm and the darker areas primarily shape the object.

The points to track are then generated from the bounding box and with usage of the mask. The mask is binarized and used and the chosen points to track should be mainly from the object area rather than from the cluttered background.

As mentioned in 2.1.4, the generated points could be connected to provide better overall quality. A limited connection will be proposed in section 3.4.3, but it is only a weak connection. We suppose that any type of tighter connection would bring additional computational cost or inability to handle non-rigid object. The connection proposed further should solve the points relations well.

3.3.2 Object Model for Detector

As the detector treats the object differently from the tracker, the object model will be different, too. The straightforward solution is to use template, as was presented in section 2.1.1.

The object model in the detection task is an inseparable part of the detector type as well as the adaptive learning system. For that reason and the fact that the detector development is not going to be solved in this work, the object model of the detector is also taken over.

Therefore, the model in the detector is a simple rectangle-shaped *patch* of the object area. To eliminate an influence of the background (as the object is usually not rectangle-shaped as well), the patch is normalized. The normalization is done by subtracting a patch overall mean, to reduce patch elements standard deviation. As a side effect, this normalization leads also to partial uniform illumination invariance, which is also an appreciable benefit.

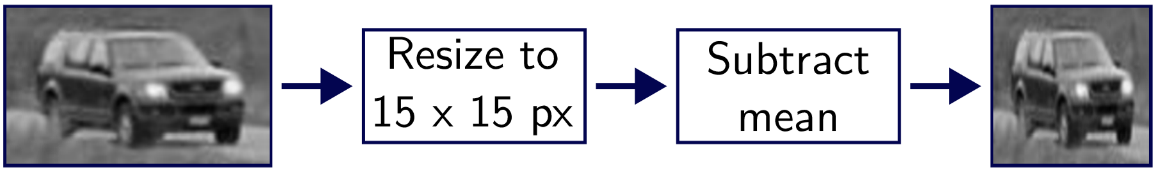


Figure 3.6: Object model for detector.

The patch is finally re-scaled into 15 × 15 dimensions. The main reason is to unify patches, which will by compared pixel-by-pixel, yet they can be in different sizes as the object may scale in time. The whole process of transformation the object in the scene into internal object model can be seen in figure 3.6. As mentioned already, the advantages are primarily in speed domain, but also a lower influence of e.g. uniform illumination or scale deformation.

3.4 Object Tracking

The main part of the proposed complex system is the tracking itself. This is going to be the core part and its success is crucial for the whole system. As was shown, the object representation intended to be used in the tracking are weak corner and peak points. Their quality needs to be supported in the tracking so the new method called *Multiple Bidirectional Tracking (MBT)* is presented in section 3.4.1.

Then, the tracking itself is proposed in section 3.4.2. This is based on sliding window approach and the proposed method *Multiple Bidirectional Tracking*. The tracked patches are weak points in their limited neighborhood and the comparison metric used is *normalized cross-correlation*.

The final part is about an estimation of a new bounding box parameters, based on the original points to be tracked and their tracked images. The method is based on *median flow* [16] and contains also a scale prediction.

3.4.1 Multiple Bidirectional Tracking

A conventional technique in the tracking field is an approach called *forward-backward* tracking. It comes from a simple idea, that tracking of a point a in a frame p_{t0} *forward* to a point b in a frame p_{t1} should work as well as the tracking of the point b in the frame p_{t1} *backward* to the point a in the frame p_{t0}. Therefore, every tracking should be followed by the same tracking, only with switched frames and points, to validate the tracking.

The conventional method was extended by Kalal et al., when they introduced a method called *Forward-Backward Error (FBE)* [16]. The difference between the traditional way and the *FBE* is in the error measurement. The distance between the original point and the backtracked point is measured simply using the Euclidean distance and it is called *FBE*. If the error overcrosses a given threshold, then it is rejected. The threshold can be also set dynamically and with respect to the target application.

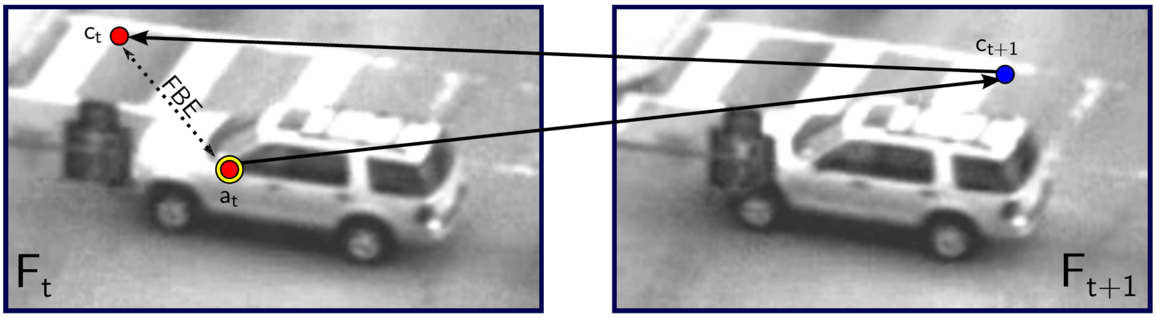


Figure 3.7: Kalal's Forward-Backward Error.

The quality of the *FBE* has been proven and it does exactly what it should do. An illustration can be seen in figure 3.7, where the point a_t from the frame F_t is tracked into the point c_{t+1} inf frame F_{t+1}, which is then tracked backward into the point c_t. The distance between the points a_t and c_t is computed and it is stated as the *FBE*.

The weakness of the *FBE* comes with usage of poor quality points to track. A poor quality point refers to a point which is hardly repeatably searched and barely unique. Such a point is therefore tracked forward, its image is tracked backward and the error is measured. As the error is very high due to the poor quality, the tracking is rejected. It should be noted that the refection is correct and expected, as the tracking most probably failed.

The extreme case of this problem can be a situation, when all or almost all points are rejected and the overall tracking fails. This work could represent such situation, because it was mentioned that such poor quality points should be used in the tracker part (section 3.3.1). It is expected that the proposed points in problematic scenes (e.g. background clutter et al.) would have high overall *FBE* and the whole tracking process would fail.

The proposed method is based on the forward-backward principle, but in contrast with the traditional *FBE*, it proposes to go deeper in tracking algorithm itself and to rather substitute the tracking criteria with the backward tracking ability. In simple words, this means that it does not only measure the error, but effectively increases the tracking quality.

The extension is based on results of continuous testing which shows that the best candidate tracking image is not always the correct tracking image, but almost every time the correct one is among three the best tracking candidates. Therefore, the proposed method suggests tracking every point multiple times in forward direction in order to choose more the best candidates than traditionally only one.

For the backward tracking, which is technically the same tracking only with swapped data, the multiple tracking is suggested as well. The method is called *Multiple Bidirectional Tracking* and it ends with n candidates for every tracked point and with m candidates of backward tracked points. Then, the *FBE* is computed for every backtracked point (we have $n \times m$ of them), and if more of them is below a threshold, the lowest is chosen.

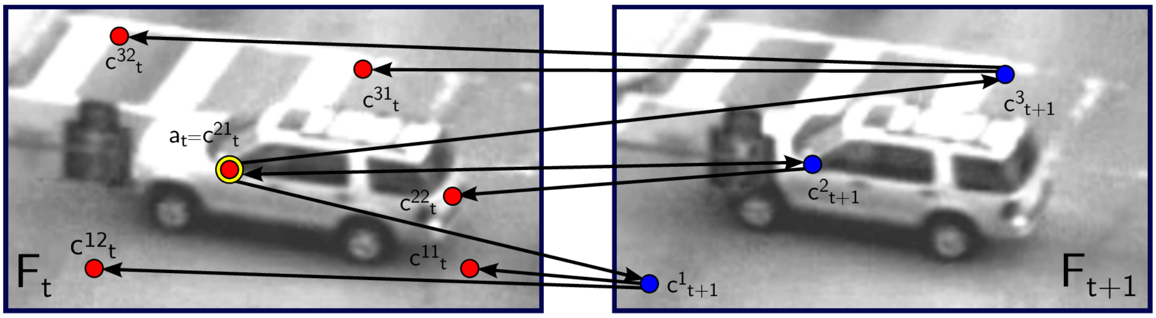


Figure 3.8: Multiple Bidirectional Tracking.

The whole proposed method is much easily understandable from an image. In figure 3.8 there are two consecutive frames of the same video sequence just like in the previous example of *FBE* (figure 3.7). In the first frame F_t a point a_t is set and its image a_{t+1} in frame F_{t+1} is to be found.

Then, 3 out of the most probable candidates of the point a_{t+1} are computed in frame F_{t+1}; they are called c^1_{t+1}, c^2_{t+1} and c^3_{t+1}. Each of the candidates is then backtracked twice, which ends in 6 candidates of the backward test. As can be seen in the figure, the backtracked point c^{21}_t is similar to original point a_t and therefore its *FBE* is equal to zero. As the searched point a_{t+1} is then claimed the forwardly tracked point c^2_{t+1}, because the point c^{21}_t is among its backtracking candidates.

The exact amount of forward and backward points is 5 and 3 respectively and it was chosen experimentally. Higher number of candidates in forward tracking has serious decreasing impact on computational performance, and both the forward and the backward number of candidates bring beside the more correctly tracked points also a number of wrongly tracked points, yet undetected. An experimental evaluation of different values of both numbers is evaluated in section 5.5.

As stated already, this improvement should cover the correct correspondence and the tracker should get more quality tracked points. Therefore, the absence of significant points to track in the tracked object due to its small size should be compensated slightly, as the system should be able to track also with low quality points.

3.4.2 Patch Tracking

The object is tracked in the form of a set of points with their neighborhoods, as was described in section 3.3.1. The point in its neighborhood is in scope of this work called a *patch*.

Regarding the described new approach in tracking called *Multiple Bidirectional Tracking*, every single tracking has to end in a list of probable future positions to enable sorting and choosing more of the probable positions. This was described in the previous section.

As a sequel to that, the Lucas-Kanade optical flow [25] cannot be used. It is designed to follow a ridge or a valley in a matrix of partial derivations. Therefore, it basically leads to only one local maximum and no multiplication is possible. It was described in section 2.2.2.

More of the local maximums need to be localized. Thus, a *sliding window* approach is used. This was described before as well in section 2.2.1. The resulting 2-dimensional matrix (figure 2.5) makes it possible to find out the local maximums and to measure their values. This is a fundamental requirement for the *Multiple Bidirectional Tracking*.

The difference from simple object detection (as the sliding window approach is one out of primary object detection methods) is a limitation of sliding window search space. This is also one of the parameters of this approach; to set where the patch can be searched for, which equals to a maximal allowed object movement.

The parameter is internally called *maximal allowed object movement (maom)* and it represents exactly what it stands for. Every single patch is searched in a window with a center in the original patch's center position and with both the width and the height of two times the *maom* size.

The searching metric is *NCC* as well as the metric in mask computation (section 3.3.1). The local maximums are then easily obtained and regarding the *MBT*, a very similar tracking is performed with these points, only with opposite direction.

3.4.3 Next State Estimation

This section is dedicated to a problem of next state estimation from the tracked points. The input of this functional block is a set of tracked points, both the original ones and the tracked images, and the original bounding box. On the output there is a new bounding box, estimated for the new state.

The first thing which is needed to investigate, is whether the object moved, and if so, to where. This is done separately for x-axis direction and y-axis direction. The movement is applied to the original bounding box, so the next state is recursively estimated on the basis of the previous state.

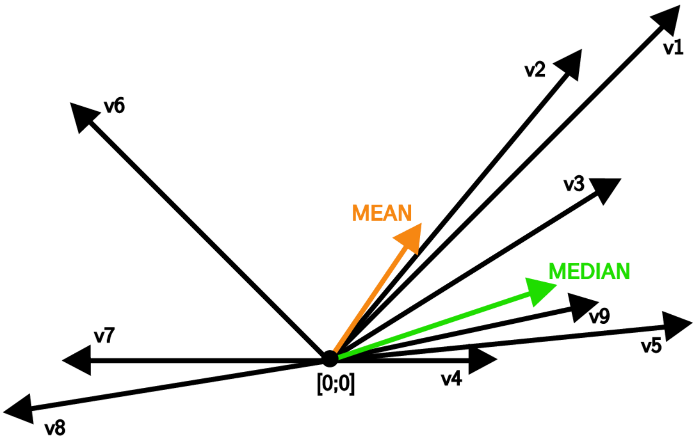


Figure 3.9: Median Flow illustration. In the figure there exists 9 vectors **v1**...**v9** and their median vector and a mean vector. The vectors **v6**, **v7** and **v8** can be considered as outliers and it can be seen that the median vector is more robust to them from its definition and it follows the majority flow. Instead, the mean is affected by the outliers.

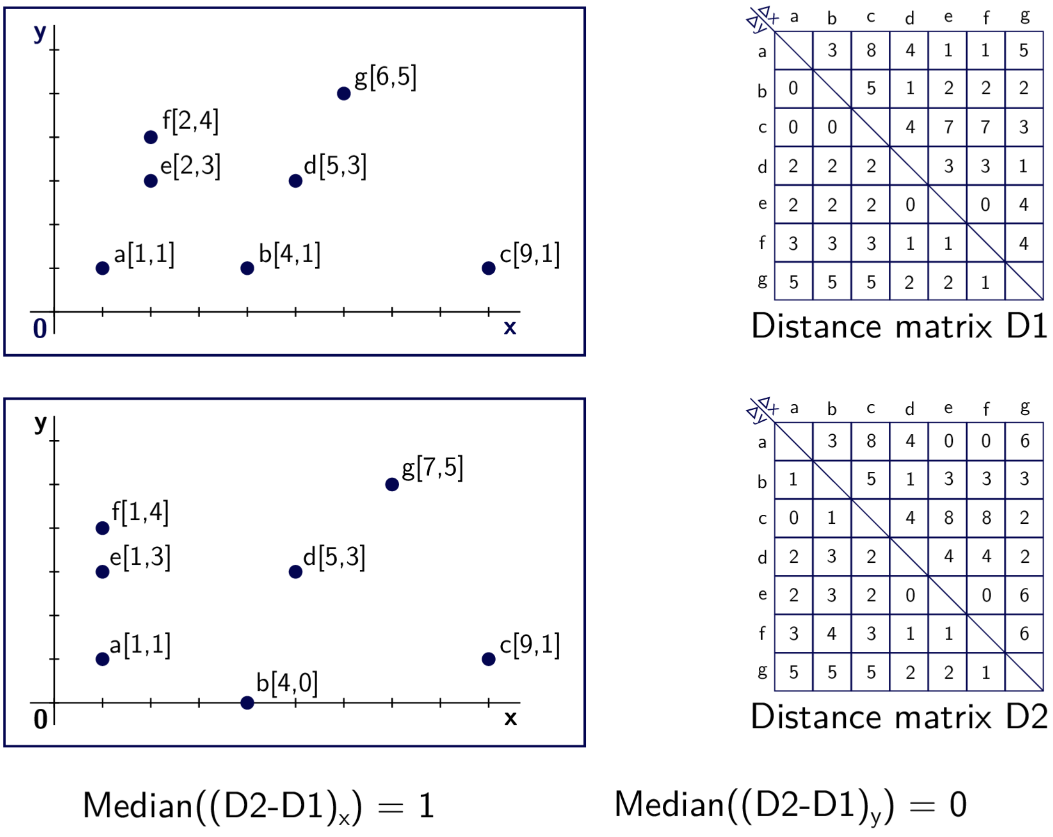
The method of finding the movement is called *Median Flow* [16]. The method builds on computing movement vector for every single successfully tracked point and choosing a median of these vectors as the resulting movement vector. As was said, this is done independently for x-axis and y-axis. An illustration of the median flow of a set of points' movement vectors is in figure 3.9

The *median flow* has a significant advantage in high robustness of error which can be caused by outliers. Therefore, we can easily do the *median flow* without taking them into account. A different situation is in scale estimation.

The scale estimation is the second step in the next state estimation process. The scale is not necessary an integral part of the new state estimation, but it is worthy. A reason can be extending the object bounding box and thus extending a space for better point to track generation. And of course the opposite reason, the reducing of the object bounding box when its reducing can improve the *object-to-background* ratio.

The outliers are filtered out by their deviation from the mean. Thus, a standard deviation σ of the movement in both the x-axis and the y-axis direction is computed. It is known from tye definition that the distance shorter or equal to $2 \times \sigma$ from the mean includes 95% of all values for normally distributed samples. According to that, the tracked points which are too distant from the mean by more than $2 \times \sigma$, are discarded.

This outliers removing requirement also comes from the use of *Multiple Bidirectional Tracking*, which on one side increases a number of successfully tracked points, but on the other side increase the number of false positive trackings as well.



	a	b	c	d	e	f	g
a		3	8	4	1	1	5
b	0		5	1	2	2	2
c	0	0		4	7	7	3
d	2	2	2		3	3	1
e	2	2	2	0		0	4
f	3	3	3	1	1		4
g	5	5	5	2	2	1	

Distance matrix D1

	a	b	c	d	e	f	g
a		3	8	4	0	0	6
b	1		5	1	3	3	3
c	0	1		4	8	8	2
d	2	3	2		4	4	2
e	2	3	2	0		0	6
f	3	4	3	1	1		6
g	5	5	5	2	2	1	

Distance matrix D2

$\text{Median}((\text{D2-D1})_x) = 1$ $\text{Median}((\text{D2-D1})_y) = 0$

Figure 3.10: Scale computation.

Since the outliers are filtered out, the scale can be computed. This is performed by comparing distances among original tracking points and their tracked images. Above the original points an each-to-each absolute distance is measured and a triangular matrix of distances is stored. This is illustrated in figure 3.10, with the original points topology and the corresponding each-to-each distance triangular matrices, where the x-axis distances are in the upper triangular matrix and the y-axis direction distances are in the lower triangular matrix. Both the triangular matrices are without diagonal values, as that measurement is meaningless.

On the bottom line, there are the tracked points in a new, x-axis scaled topology. The y-axis position of these points stays unchanged. Then, there is also the matrix with triangular matrices, firstly for the x-axis direction and then for the y-axis one.

To compare those matrices, a difference between the original topology matrix and the new topology matrix in a by-element meaning is computed. Finally, the median value of the difference is considered as a majority scale. Again, this is done separately for the x-axis and y-axis directions.

This approach provides an incremental change of the bounding box, with respect to the recursive tracking itself. The only disadvantage can be the possibility of cumulative error, but this is common for the whole recursive tracking process. The rotation or other object deformations are not considered in this work.

3.5 Object Detection

When the tracking is not working for any reason, a detector is necessary. This can happen if the object disappears from the frame completely, if it is covered with another object (no matter whether entirely or only partially) or when we are not able to track because we lost our tracking points, e.g. due to frame-cuts or because of fast camera movements.

The detector used in the proposed solution is taken from OpenTLD is left almost unchanged and hence is not an original work here. But since the detector is a part of proposed system, it is described in this section, including the updated parts.

The core idea of the detector is described in the subsection 3.5.1. There are also some problematic parts mentioned, when the detector could fail instantly, as well as the proposed changes against the original OpenTLD's detector.

3.5.1 Cascade Detector

The detector used in OpenTLD uses the sliding window approach described in 2.2.1. The detector respects cascade (or pyramidal) principles and therefore consists of four stages. The stages are the following:

1. Foreground Detection
2. Variance Filter
3. Ensemble Classifier
4. Template Matching

Obviously, in the intended general use, any background image will not exists, hence the *Foreground Detector* is meaningless. The next 3 stages are taken from OpenTLD only with minor changes. An illustration of the suggested cascade is in figure 3.11.

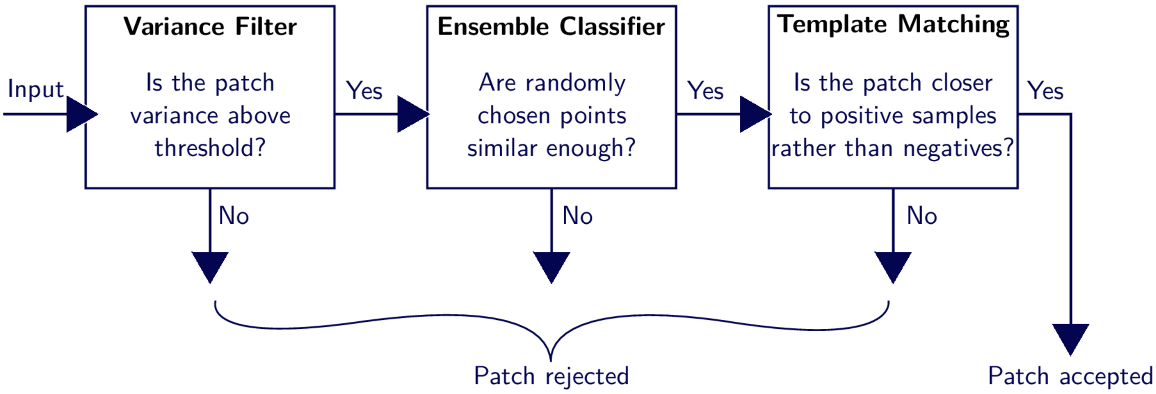

Input
Variance Filter
Is the patch variance above threshold?
Yes
Ensemble Classifier
Are randomly chosen points similar enough?
Yes
Template Matching
Is the patch closer to positive samples rather than negatives?
Yes
No
No
No
Patch rejected
Patch accepted

Figure 3.11: Proposed cascade detector. The detector consists of 3 stages - the *Variance Filter*, the *Ensemble Classifier* and the *Template Matching*.

The first stage in the cascade detector is then the *Variance Filter*. As it is the first stage, the main task is to reject as many false positives as possible. The measure metric is a variance σ_min and the purpose is to reject patches with lower variance than the patch with tracked object. The threshold is set as a σ^2_min and it is taken from OpenTLD unchanged.

The variance filter generally denies all the solid or near solid texture subwindows, where the object most probably is not, and it does it rapidly. As the variance of a subwindow increases, it becomes more interesting and the probability that it contains searched object also grows.

The next cascade stage is the *Ensemble Classifier*. This classifier uses a method known as *random fern classification* [32]. The classifier decision is based on comparing several randomly chosen pixels instead of the whole frame, and therefore it is a good trade off between the fast variance filter and the next stage, the template matching.

The last stage of the detector is the already mentioned *template matching*. To be more accurate, the approach is build on a method called *nearest neighbor* and the distance metric is also described *NCC*. To be able to perform any *nearest neighbor* method, samples from all classes must exist. Therefore, the detector with cooperation of adaptive learning system (described in section 3.6) stores a set of both positive and negative templates.

The difference between the last stage from the classical *nearest neighbor* approach is in the output. The traditional *nearest neighbor* method outputs the nearest class from the sample data to the new input sample. In the cascade classifier, the output has to be either *yes* or *no*. So the evaluation searches for the nearest neighbor from the positive class as well from the negative class, inverts them and then computes a new value called *confidence*.

$$d_N = 1 - min(NCC(T, n_i)) \tag{3.2a}$$

$$d_P = 1 - min(NCC(T, p_i)) \tag{3.2b}$$

$$conf = \frac{d_N}{d_N + d_P} \tag{3.2c}$$

The whole computation can be seen in equations 3.2a, 3.2b and 3.2c. The function $min(NCC(T, n_i))$ stands for the nearest neighbor of template T from a set of negative samples n when the distance metric is set as *normalized cross-correlation (NCC)*.

$$res = (conf < \Theta^P)?True : False \tag{3.3}$$

The thresholding part needed by the cascade classifier is in equation 3.3. The value of the Θ^P is set as 0.65 as in [28]. In the res there is a final cascade detector decision whether to accept the patch or not.

3.5.2 Detector's Confidence

It was shown that the last part of the cascade detector also contains a confidence measurement. This is important for further fusion, because the detector's and the tracker's outputs are not fused uniformly, but the fusion is weighted.

The confidence is used also for the tracker. It is done easily by performing the very same computation with identical knowledge base, only with the tracker's output. The fusion is described in section 3.7.

3.6 Model Learning

As mentioned before (long-term factor is described in section 2.3.3), a long-term tracking needs to cope with the object appearance changes. It is necessary to cover these changes in the proposed system to successfully perform the long-term tracking. The obvious and at the same time the easiest thing is to try generate some variations with added rotation or scale in detector itself. Unfortunately, that can be quite computationally expensive and results are at least unsure.

A different approach is to develop an adaptive learning model to update and extend the positive samples database when object is certainly found. This solution should cover more object appearance changes, it even enables a complete object evolution. On the other hand, it expects more sophisticated solution. Such a solution can be computationally expensive as well as quite cheap and therefore very fast. The same diversity can supervene from the point of view of memory consumption.

The proposed solution is an adaptive learning system, which is actually a rather a naive approach. The main idea is to manage two databases of positive and negative sample objects. Those samples are normalized and stored, so no actual learning is done and the knowledge base is simply extended.

The solution is overtaken from OpenTLD [28]. All the changes made on it are described in this section. Primarily, the limits to allow the learning in every single frame were made stricter in order to prevent a false positive learning. The adaptive learning system can be divided into an initial learning part and an in-progress learning part. This is the contents of this section with all the subsections.

3.6.1 Initial Learning

The first point where the model learning system is called is immediately after the user chooses the object rectangle. By drawing rectangle around an object of interest on the first frame, the user indicates that he wants to track that object. The rectangle given by the user is stored as the first positive sample.

Then, more positive and negative samples are generated from the initial frame. The whole frame is covered by many different rectangles preserving the original rectangle aspect ratio, with a limited scale and with fixed moving step. It ends with the whole frame covered with rectangles, as can be seen in [28].

The positive samples are generated from the set of the rectangles according to their overlap with the original one. Up to 10 samples sorted by their overlap are chosen. The lowest overlap which can be added to positive samples set is 60%. The similar approach is done for the negative samples. The only difference is that up to 100 rectangles with overlap lower than 20% are added.

It is noticeable that the quality of the rectangle set by the user is critical. The object should be fully inside the rectangle and at the same time the rectangle should be as small as possible. However, the user-caused error is a limitation of every manual or semi-automated system, where the user has its active role.

3.6.2 In-progress Learning

It is highly recommended to update the detection model in time in order to improve results in the long-term tracking. Fortunately, the system based on a set of positive and negative samples is very easy to update. If the system confidence is high enough, it just adds a positive sample to the positive knowledge base and negative samples to the negative sample knowledge base.

The important thing is to determine whether to learn and whether rather not to learn. As any other system, also the proposed system is not perfect and occasional error can occur. It does not need to be a problem, if the system can detect the mistake and does not perform online learning. However, this could end in cumulative error which is described in detail in the next section (3.6.3).

If we expect the system is correct in its output, we can learn. As was mentioned, the learning is simply a new sample addition. The new sample is normalized as any other sample, which means that the mean is subtracted and the sample is resized to 15×15 size. Also the negative samples (those which do not overlap with the positive one) are added to the negative sample set with the same normalization.

The sample problem, which can be reached with this adaptive learning model, can be linear growth on memory needed to store all the positive and the negative samples. It is a question of an experiment whether that problem becomes a real or stays only in the hypothetical area. Nevertheless, this issue is not dealt with in this thesis.

With the system runtime, also the quality of the detector increases, which comes from the last stage. The critical part is right after the initial learning, when the object model in not described well. It also requires a low factor of false positive learning, but it generally becomes better in time.

3.6.3 False Positive Detection

The already mentioned critical point of the adaptive learning system is the involvement of false positive detection result among the positive sample set. From that point, the detector results should be considered untrustworthy due to the presence of a negative sample inside the positive sample set. The problem becomes more crucial if we realize that the last part of the detector, the *template matching*, uses the nearest neighbor approach.

Therefore, the learning is enabled only in some specific cases, when it is safe to say that the confidence of the system is high enough. One of the cases is a successful fusion. Then it is really safe to note that the system is sure enough and the learning can be performed. The second case is after successful and high enough confident tracking. This case was actually a part of original OpenTLD, only the limits to involve new sample into positive database set were increased.

Another solution of this problem can be extending the nearest neighbor algorithm, to use e.g. 3 of the nearest samples from both the positive and the negative sample set. Hence, the minority of false positive samples in the positive samples database should not have any big influence, as the voting is extended. This solution will be incorporated only if the problem becomes more serious.

That should be enough to handle the most critical part. After all, a solving of the learning system is not a part of this thesis and is mentioned here only because it is an inseparable part of long-term tracking.

3.7 Detector-Tracker Fusion

The last part of the proposed tracking system is called a *Detector-Tracker Fusion*. It is an original work and the purpose is to improve the quality and the stability of the overall output result by fusion of both the tracker's and the detector's outputs, if both of them are valid.

It was mentioned before that the last stage of the cascade detector contains also a confidence estimation. It was also mentioned that the very same approach can be used to estimate a confidence for the tracker, which is comparable to the detector's one. Therefore, if we have both results valid, the fusion can be considered.

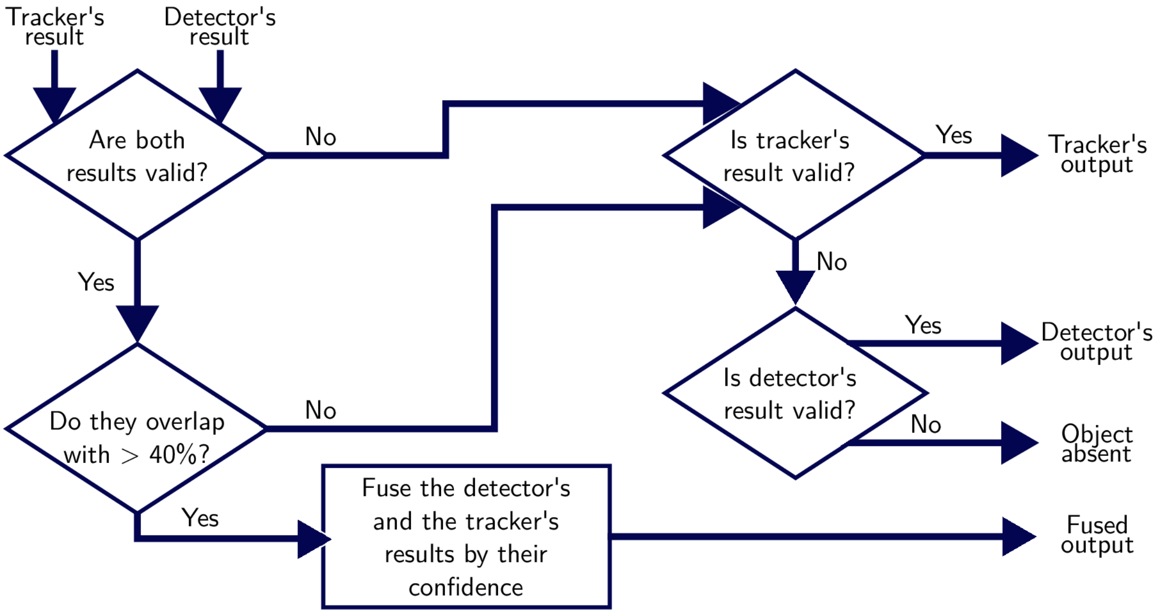

Tracker's result
Detector's result
Are both results valid?
No
Yes
Do they overlap with > 40%?
No
Yes
Fuse the detector's and the tracker's results by their confidence
Is tracker's result valid?
Yes
No
Is detector's result valid?
Yes
No
Tracker's output
Detector's output
Object absent
Fused output

Figure 3.12: Detector-Tracker Fusion workflow diagram. If both the detector and the tracker have their results valid, it is considerable to fuse them to achieve higher stability and quality of the proposed system.

The whole decision workflow diagram can be seen in figure 3.12. It is shown that another condition for successful fusion is at least 40% overlap of the detector's and the tracker's results. The reason is simple - if they do not overlap, one of them can be expected to be wrong and a potential fusion of *possibly correct* and *probably wrong* can end only in *wrong* result.

Therefore, they need to overlap at least a little. If they do not overlap or confidence of any of them is too low, no fusion is performed. Then, the tracker is prioritized and if it is at least a bit confident (more than 0.2, which basically means that it is not totally lost), its result is used.

Otherwise, the detector's confidence is tested by the very same threshold value. Analogically, if it passes, the detector's result is used. Finally, if none of these conditions is satisfied, the object is marked as missing.

The fusion is computed as a weighted mean of both results and the weights are already discussed confidences. The appropriate equations are shown in 3.4a, 3.4a, 3.4c and 3.4d.

$$x = \frac{conf_t * x_t + conf_d * x_d}{conf_t + conf_d} \tag{3.4a}$$

$$y = \frac{conf_t * y_t + conf_d * y_d}{conf_t + conf_d} \tag{3.4b}$$

$$w = \frac{conf_t * w_t + conf_d * w_d}{conf_t + conf_d} \tag{3.4c}$$

$$h = \frac{conf_t * h_t + conf_d * h_d}{conf_t + conf_d} \tag{3.4d}$$

$$\tag{3.4e}$$

The $[x_t, y_t]$, resp. $[x_d, y_d]$ are coordinates of a center of the tracker's result, resp. the detector's result. As a result, new coordinates $[x, y]$ are computed. The fusion is done also for the bounding box size, as it is shown in the last two equations.

The last part is a resulting confidence. The confidence is also fused, but this time it is only averaged arithmetically. The only thing is that the higher confidence is counted twice, as is expected the fused result if more confident than two single results. The equation can be stated as in 3.5.

$$conf = \frac{2 * max(conf_t, conf_d) + min(conf_t, conf_d)}{3} \tag{3.5}$$

As it was already mentioned, the proposed fusion system should bring better system stability and higher quality. A specific problem, which can be improved or partially reduced, can be tracker's *drift*, when both the object and the camera does not move but due to the noise and the internal tracker's work the resulting bounding box is drifting. After incorporating the proposed *Detector-Tracker Fusion*, the tracker's drifting result can be corrected by the detector from time to time.

3.8 Summary

This chapter described the proposed solution including the parts, which are not original. The proposed solution is a complex tracking system with all parts needed to perform the long-term visual object tracking.

In the first part, some expected behavior targets, especially system's inputs, outputs and problems which should the system be able to handle are summarized. As the system background was chosen the OpenTLD from Nebehay [28], as it contains all the parts required for long-term tracking task. Also some limits which should be met are set in the first sections, like real-time or near real-time performance.

Then the object representation was proposed. It was done separately for the tracker and for the detector in appropriate section. The tracker's object representation was proposed as a set of 16 points in their 11 × 11 neighborhood, searched by *Good Features To Track* [40] concept. The detector's object representation is a patch reduced to size 15 × 15 with subtracted mean of the patch.

The object representation in the tracker was used and the points quality was improved by a newly proposed method called *Multiple Bidirectional Tracking*. This method builds on the idea of tracking every point bidirectionally to measure its quality, but adds the points tracking multiplication to improve the quality instead of simple measurement.

The points are then tracked by *sliding window* approach and the new bounding box is estimated recursively. To determine majority tracking vector, a *median flow* is used. The scale is estimated, too, and outliers are removed.

In the next section, the cascade detector is described. The cascade from OpenTLD is overtaken with only minor changes, which are described. Generally, the cascade is built from 3 stages - the variance filter, the ensemble classifier and the template matching.

Then the adaptive learning system is described. Again, the learning system is mainly overtaken from OpenTLD, but for the completeness it is described here. Such a system is generally necessary in a long-term tracking.

Finally, when both the tracker and the detector have their results, they can be fused. The new *Detector-Tracker Fusion* system was proposed in the last section and it is expected that it should bring both higher stability and confidence to the whole system.

Chapter 4

Implementation

This chapter is intended to describe briefly the implementation background. The implementation part of this thesis is attached to the printed version of this thesis and this chapter should describe only the interesting and non standard ways, or algorithms of the newly proposed methods.

Therefore, in the section 4.1 there is described the background, the original OpenTLD [28], with a limited set of changes which were implemented in the original code. The section is followed with the related section about the whole system structure (4.2).

The next three sections are dedicated to the new original methods proposed in the previous chapter. These are namely the *Points To Track Generation* in section 4.3, where the mask and points to track are generated, then the *Multiple Bidirectional Tracking* describing implementation details in section 4.4 of the originally proposed method for increasing the quality during tracking, and finally the section *Detector-Tracker Fusion* where the short description of the implementation of the proposed fusion is presented.

4.1 Technical Background

As it was numerously mentioned, this project builds on basics of OpenTLD by Nebehay [28]. Beside the structure advantages (the tracking-detection-learning), the implementation itself has also several advantages. Thus, the source code, which is publicly accessible on the Internet[1], it was decided to use the code directly.

The source code is written in C++ and it is divided into a few logical blocks. The one not related to tracking, but surely required is a system of reading input images or video or input stream and providing the single frames one-by-one to the tracking system itself. Also the output format is ready to be parsed automatically, hence the automatic evaluation scripts were easy to implement.

Because the source code is already 4 years old, a structure update was needed. First of all the component update was done. The OpenTLD uses OpenCV framework[2], but due to the long time, the version was obsolete. The update to the latest version was done, which means that every use of a data type or a call of a function from OpenCV was replaced with an alternative in the latest version.

Also, several third party dependencies were removed, if the function was replaceable, which was often the case, as for example the new OpenCV contains much more functionality.

[1] The OpenTLD repository is on `https://github.com/gnebehay/opentld.git`

[2] OpenCV homepage is at `http://opencv.org/`

Therefore, the dependency list was reduced rapidly. The code itself was updated from the same reason. From time to time some approaches are more easily and effectively done with standard C++ functions or again by the new OpenCV.

The initial code size was reduced by almost 20% only with removing redundant code, unused (and probably not even intended to be used) functions and part of the code.

4.2 System Structure

The system structure is mainly overtaken. The basic diagram can be seen in figure 4.1. The structure is divided into a library part and a utility part. The utility part contains the input/output interface including configuration, mouse events handling and results printing system.

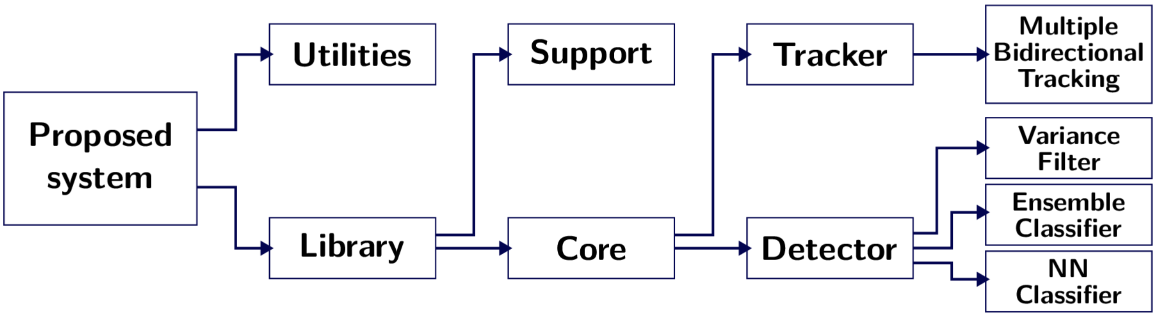


Figure 4.1: Basic system structure.

In the utility part, the result application is named *rtgot*, which stands for *Real-Time General Object Tracker*. That differs from the original *opentld* and the application also differs internally.

The library part contains the tracking system itself. In conformity with the object oriented approach the logical blocks are in separate files. The blocks can be summed as a *Core* part, which is divided into two equal parts a *ObjectTracker* and a *ObjectDetector*. The tracker part then contains primarily a *MultipleBidirectionalTracking* logical block, as this is a core part of the tracker. The detector part contains all three stages as was described in previous chapter. They are namely a *VarianceFilter*, a *EnsembleClassifier* and a *NNClassifier*, which stands for the template matching part.

The *Detector-Tracker Fusion* is an internal part of the *Core* logical block. Also the learning part is initialized from the *Core* part and the learning itself is performed by the *ObjectDetector*, because the detector is learned.

Next to these two main parts, there is also a non-homogeneous block with support functions. There are mainly conversion function, but also a bounding box definitions, normalized patch structures and so on. Majority of these support functions was only overtaken, the rest was extended and cleaned.

4.3 Points To Track Generation

The points used to track are generally quite weak, as was presented before, including the reasons for that. Within the implementation process two different approaches were used.

Beside the proposed solution, also a pseudo random point generator of a Monte Carlo type was used. The idea was to involve not only the strongest points, but also the weaker ones, and lower computational cost, as no detection is needed. The method works as following: generate pseudo random point coordinates $[x_i, y_i]$ from the space of bounding box. If the value of the mask in coordinates $[x_i, y_i]$ is above 0.75 (on a scale from 0 to 1, where 1 equals to 100% probability that it covers the object), then it is added to a list of points. The pseudo code can be seen in 4.1.

Algorithm 4.1: Monte Carlo Points To Track Generation pseudo code.

```
input: bbWidt, bbHeight, nPoints, threshold
points = []

while (points.size < nPoints)
do
    x = rand() % bb.width
    y = rand() % bb.height

    if (mask[x,y] > threshold)
    then
        points.add(x,y)
    fi
done
```

Some of the generated points can be seen in figure 4.2. It can be seen that some of the points are very hard to track and also some of them are multiplied. Another serious problem which occurred was the computation cost. If the ratio between the object area to the whole bounding box area is very low, the used time grows a lot, as there were many missed tries. So, the computation cost, which was expected to be low due the non requirement of the points detection, was not confirmed.

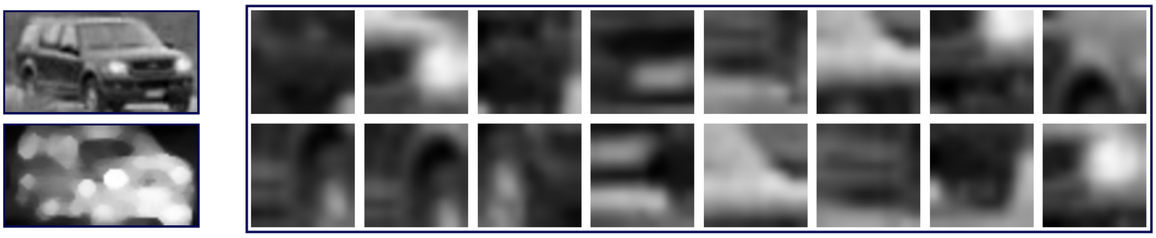


Figure 4.2: Results of Monte Carlo Points To Track Generation. On the left side there is the original bounding box with its mask. On the right there is a set of enlarged points in their neighborhood as they are supposed to be used in tracking. It can be seen that some of them are duplicated as well as barely unique. The computational time was disproportionately long due the number of missed generations.

The second approach is the one described in section 3.3.1. The chosen points should be generally weaker than similar point detected by SURF or SIFT like methods, but it should be the best which can be found. Therefore, the points are detected by the method proposed in [40].

The limit of the quality to accept points is set to a 1‰ of the quality of the best point detected. Also it is set that the minimal distance between such detected points must be at least 3 pixels. To get the points detected, the OpenCV function `GoodFeaturesToTrack` is called with the mentioned parameters.

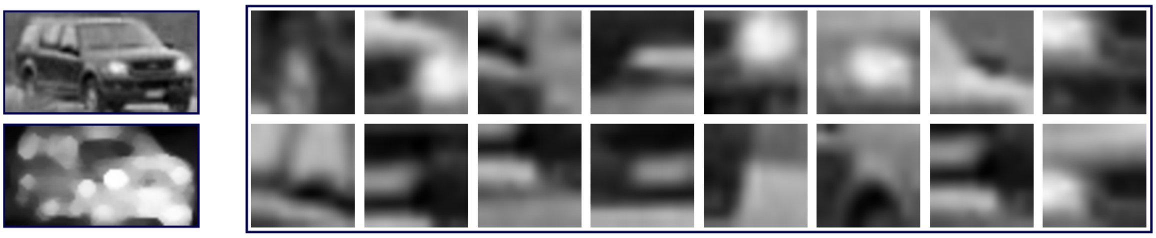


Figure 4.3: Results of GFTT Points To Track Generation. On the left side there is the original bounding box with its mask again and on the right there are enlarged points in their neighborhood generated by this method. It can be seen that the quality increased.

The results are shown in figure 4.3. The quality increased notably and the performance as well. Although the points are detected in the first step, it means a finite time consumption instead of random generation above. This approach is used in a final implementation, for the reasons mentioned.

4.4 Multiple Bidirectional Tracking

The originally proposed method called *Multiple Bidirectional Tracking* was described in section 3.4.1. The method was deeply described and in this section some implementation details are noted. The pseudo code can be seen in 4.2.

In the pseudo code, it can be seen that every point is tracked in its neighborhood separately. Beside other implications, it means that the used structures can be reused and a lot of initializations can be omitted. First of all, for every tracked point a neighborhood is obtained from the first frame `imgI`. Then the neighborhood is searched in the second frame `imgJ`, which ends with a map of distances for every position, called `fDistanceMap`. A visualization of such a map can be seen in figure 2.4.

In the distance map, the highest peaks are searched, exactly 5 of them for the forward direction and 3 of them in the backward direction. Every of them is then searched in the very same manner, but in the opposite direction. This is a content of the inner loop. At the end of the inner loop, every backtracked point is appended into a list of all candidates.

At the end of tracking of each point in forward direction, the backtracked candidates are evaluated. The candidate which was backtracked to the closest position from the original point is chosen and the distance is stored. A *sigma* is set, which is the maximal allowed distance of the backtracked point. This is the same as *FBE*. So, if the stored distance is smaller than the sigma, the candidate is stored as a correct tracking into appropriate list.

The list of the successful trackings is then used for the estimation of the next state, which is described in 3.4.3. It ends with a new bounding box, which is in the one side printed as a result and at the second side it is used as a bounding box for tracking between the next frames pair.

Algorithm 4.2: Multiple Bidirectional Tracking.

```
input: imgI, imgJ, radius = 32
output: successfullyTrackedPoints = []

foreach pt from points
do
    ptNeighborhood = getPtNeighbordhood(pt, imgI)
    fDistanceMap = computeDistance(ptNeighborhood, imgJ, radius)
    fCandidates = findPeaks(fDistanceMap, 5)
    trackedCandidates = []

    // Track every candidate backward and choose 3 of best candidates
    foreach cand from fCandidates
    do
        candNeighborhood = getPtNeighbordhood(cand)
        bDistantMap = computeDistance(candNeighborhood, imgI, radius)
        bCandidates = findPeaks(bDistantMap, 3)
        // Copy the candidated to candidates array
        trackedCandidates.add(bCandidates)
    done

    // Get distance of the closest backtracked point with the point
    //  which led to that tracking
    distance, btPt = getClosestCandidate(pt, trackedCandidates)

    // Add trackings that are closer than sigma
    if (distance < sigma)
    then
        successfullyTrackedPoints.add(btPt)
    fi
done
```

4.5 Detector-Tracker Fusion

The Detector-Tracker Fusion takes a small place in a code. It is designed as a decision tree and so it is implemented. In the pseudo code 4.3 can be seen the pipeline which precedes the very fusion.

The fusion is connected with permission to learning and with general validation sign. This is signalized with the `learnValid` and `valid` variables. The `learnValid` variable signalize that the learning can be performed, if it is not disabled at all (from configuration, mainly for testing purposes). The `valid` variable is a general sign that the tracking was valid at all. It means, that at least one part has a valid result and the object position can be determined.

The confidence limits used in the pseudo code are default values, which are stored in configuration file. Therefore, they can be changed, but this is the default settings which is used in evaluation presented further. Generally, the logic says that the confidence can be lower if the result was valid in previous frame and if not, in needs to be a bit higher.

4.6 GPU Acceleration

From the pseudo codes above is easily notable that many of the parts can be done in parallel. Moreover, many of the loops operate on the same data, like two consecutive frames for all

Algorithm 4.3: Detector-Tracker Fusion.

```
input: trackerBB, trackerConfidence, detectorBB, detectorConfidence
       wasValid
output: newBB, newConfidence, valid, learnValid

if (detectorConfidence > 0.5 &&
    trackerConfidence > 0.5 &&
    computeOverlap(trackerBB, detectorBB) > 0.4)
then
    newBB = fuseBBs(detectorBB, detectorConfidence,
            trackerBB, trackerConfidence)
    newConfidence = (2 * MAX(detectorConfidence, trackerConfidence) +
                     MIN(detectorConfidence, trackerConfidence)) / 3.0
    valid = true
    if (newConfidence > 0.65)
    then
        learnValid = true
    elif (wasValid && newConfidence > 0.5)
    then
        learnValid = true
    fi
elif (trackerConfidence > 0.2)
then
    newBB = trackerBB
    newConfidence = trackerConfidence
    valid = true
    if (newConfidence > 0.65)
    then
        learnValid = true
    elif (wasValid && newConfidence > 0.5)
    then
        learnValid = true
    fi
elif (detectorConfidence > 0.2)
then
    newBB = detectorBB
    newConfidence = detectorConfidence
    valid = true
fi
```

the tracked points. Thus, a gpu acceleration of some of the parts is involved as well. For that purpose the NVIDIA CUDA framework[3] was chosen.

By the profiling of the proposed implementation was determined, that the bottleneck of the proposed solution is in the multiple tracking in both direction for several points. This is the part where the same image is compared several times with almost identical data, with performing the same operation. That part takes by estimation almost 80% of the total computational time.

As the acceleration is not the main part of this thesis, it stays in a form of a draft. The acceleration was done mainly on the OpenCV side. The OpenCV contains modules that are accelerated in GPU by CUDA framework. Therefore, only minor changes of data structures, used functions and code structure were needed to achieve significant speed gain. Better refactoring with a highlight on reducing a data transfers (as this is the most expensive

[3] NVIDIA CUDA framework homepage is at https://developer.nvidia.com/cuda-zone

operation in GPU acceleration field) could bring even higher performance gains, probably multiplied.

4.7 Summary

The content of this chapter was a description of the implementation. As the implementation is in separate media attached to the printed version of this thesis, this chapter contains only a few important parts of it.

Firstly, the technical background was mentioned. It covered also a project structure divided in some logical blocks. The blocks were primarily the *ObjectTracker*, the *ObjectDetector* and a support functions. It is based on C++ implementation of OpenTLD [28], which uses an OpenCV. The original code was massively updated and simplified on the level of logical blocks, on the level of code itself or about the dependency list, which was pruned a lot.

Then the three main parts were described from the pseudo code point of view. It was a section about generation of points to track, which also contains one unused approach from the early implementation stages. Then, it is a section about Multiple Bidirectional Tracking itself. As the main idea was exhaustively drawn in previous chapter, the implementation details with the pipeline pseudo code were mentioned here. The last code part was about the proposed Detector-Tracker Fusion. As it is designed as a decision tree and so it is implemented. The pseudo code contains also the parts which permit the learning.

At the end the GPU acceleration is proposed. Many of the logical blocks of the proposed system use a computation above matrices and majority of these parts can be done in parallel. Some parts where the acceleration makes the biggest sense are emphasized and a further enhancement is proposed.

Chapter 5

Evaluation

The evaluation of the proposed solution is as important as the proposal itself. In this chapter, the evaluation is summarized. The chapter consists of several chapters. With exception of the first section, which is focused on an evaluation dataset, the sections are dedicated to evaluation of the proposed solution from several different points of view.

Every section is divided into two parts; a description and results. The reason is to precisely define what is tested in the particular section, how the the accurate procedure, what is the reason for testing it and what are expected results. In the results part there are the results evaluated exactly by the procedure noted before and the results are presented.

It was noted that the first section is different from the others. The section is called *Evaluation dataset* (section 5.1) and the used dataset is described in it. The dataset is a basis for every experiment proposed further.

The following sections are dedicated to experiment themselves. The first experiment is about the proposed system's quality (section 5.2), the second is about the achieved performance (section 5.3), then the section about stability evaluation in 5.4 comes.

The last two experiments are about evaluation of original parts of the proposed system (section 5.4) and about the comparison with other state of the art solutions. This is also the second most important experiment, besides the quality focused one.

The last section summarizes results from separate experiments and interprets them. All the interpretation of results is done there and not in the sections themselves. It is so because of consistency; the result is not the same as the result interpretation.

5.1 Evaluation Dataset

This section describes a dataset, which is used to evaluate all experiments, including the evaluation of the separate components as well as the comparison evaluation. The dataset itself consists of two main parts, which are combined.

The first part consists of the standard dataset for visual object tracking. The used set is a subset of dataset used in VOT'14 [22], which collects the dataset in a complete set used to evaluate visual object trackers worldwide. The reason to use this set of evaluation sequences is to evaluate the proposed system on similar data as alternative solutions.

The used sequences from VOT'14 are all in format of consecutive images, which comes from the video decomposition. The images are in resolution from QVGA to VGA and in color. However, the color information is not used in the proposed solution, so it is not important. All the images are labeled manually by a committee of VOT in the meaning of bounding rectangle and its orientation, which is done severally and the result is done by a mean. The groundtruth then consists of coordinates of four vertices of a bounding rectangle $[x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4]$. This format comes from the evaluation used in VOT, where they compare the overlap of the groundtruth bounding box and the tracker's bounding box, which should be able to estimate also the rotation.

In this thesis the evaluation is done by computing a distance from the object center. The reason is noted in the following paragraphs and comes from the labeling of the new dataset and the fact that the proposed system should not be designed to estimate rotation, only scale. It means that the VOT groundtruth has to be recomputed. The equation is shown in 5.1 and it comes from the groundtruth representation mentioned in the previous paragraph.

$$\begin{aligned} x &= \frac{x_1 + x_2 + x_3 + x_4}{4} \\ y &= \frac{y_1 + y_2 + y_3 + y_4}{4} \end{aligned} \tag{5.1}$$

The second part is an original newly proposed dataset for visual object tracking purposes. The sequences have been taken from airport Medlanky[1] and the tracked subjects are aircrafts. The difference from the standard dataset noted in the previous paragraphs is a limited size of the objects and their distance. The aircrafts are generally very far, which means the already mentioned limited size and quite high *SNR (signal-to-noise ratio)*. To summarize that, the objects are low contrast, blurred and very small.

On the other hand, the cluttered background problem mentioned in 2.3.4 or the similar objects problem noted in 2.3.5 almost never appears. But, the camera is very unsteady, so the full occlusion problem described in section 2.3.2 appears quite often.

The new dataset was annotated manually by me. The groundtruth is set as coordinates of the object center $[x, y]$ in particular frame. The labelling was repeated few times and the results have been averaged by a geometric mean. It was done for every proposed sequence of the new dataset.

As there are several generally unsolved problems mentioned in section 2.3, the dataset tries to cover them. The proposed system is designed to be general, hence all the mentioned problems should be tested. The problems and their occurrences in single testing sequences are summed in table 5.1.

The values noted in the table are either in percentage form, absolute form or binary form. The percentage value means a ratio between frames where the problematic parts occur and all frames. The binary form consequently signalize if the problem occurs at all. It is primarily by problems which are not exactly measurable, like scale or appearance changes.

A simple visual dataset overview can be seen in figure 5.1. There is a representative from every dataset sequence with a noted object. This is only to see how the object and the overall scenes look like.

5.2 Quality Focused Experiment

The experiment focused on quality can be considered as the most important experiment at all. The quality is the value which is the most interesting and which is very often the only

[1] Public civil domestic airport Medlanky, Brno, Czech Republic

	Name	FrCnt	FrRes	PO	FO	CB	CS	IC	S	SO	AC
standard	Ball	602	320x240	0%	0%	No	Yes	No	32%	No	No
	Bicycle	271	320x240	1%	0%	Yes	Yes	No	37%	No	No
	Car	252	620x272	10%	0%	No	Yes	No	66%	No	Yes
	Drunk	1210	508x336	0%	0%	No	Yes	No	6%	Yes	Yes
	Fish1	436	460x259	0%	0%	Yes	Yes	No	67%	Yes	Yes
	Gymnastics	207	320x180	0%	0%	Yes	Yes	No	22%	No	Yes
	Jogging	307	352x288	7%	0%	No	Yes	No	40%	No	No
	Polarbear	371	640x360	0%	0%	No	Yes	No	13%	Yes	Yes
	Skating	400	640x360	11%	0%	Yes	Yes	Yes	16%	Yes	Yes
	Surfing	282	320x240	11%	0%	No	Yes	No	0%	Yes	No
	Woman	597	352x288	57%	0%	No	Yes	No	7%	No	No
orig	Helicopter	440	640x480	4%	3%	No	No	No	34%	No	No
	Plane51	923	640x480	2%	7%	No	No	No	63%	No	Yes

Table 5.1: Dataset description. The abbreviations are following: *FrCnt* stands for *total frames count*, *FrRes* for *frames resolution*, *PO* for *Partial Occlusion*, *FO* for *Full Occlusion*, *CB* for *Cluttered Background*, *CS* for *Camera Stability*, *IC* for *Illumination Changes*, *S* for *Scale*, *SO* for *Similar Objects* and *AC* for *Appearance Changes*.

evaluation field for many solutions. Regarding that, the informative value of the quality targeted experiment was the highest.

Going further, the quality is also an entry point for next testing. Saying this in other words, it does not make any sense to evaluate performance of stability of the system which does not work as expected. This can be determined from a low value of quality.

In this case, it is not reasonable or even possible to expect any overall quality. The quality strongly depends on used dataset, besides the system itself. It can be expected that the quality for majority of the evaluation dataset will be about the higher boundaries. According to general purpose of the tracker, it is not expected to fail at all in any of the evaluation datasets.

5.2.1 Description

The quality is evaluated using all the datasets. The wider the set of datasets is, the better. The quality is meant to be an average success rate of a success rate in every frame of a particular sequence. All the rates of one sequence are averaged in a one value for every dataset.

If the object center predicted by the tracker lies within a circle with 30 pixels in diameter with center in the groundtruth, the results are considered as a 100% successful. If it lies out of the circle with a diameter of 70 pixels and with center in the very same groundtruth, the results are set to 0% instead. The success rate between these two limitation boundaries is computed linearly. The reason for the hard limits is that the evaluation script does not know the size of the object. The equation is shown in 5.2.

$$rate = 1 - \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} - \sigma}{\gamma - \sigma} \tag{5.2}$$

The σ is the lower boundary and it equals to 30 pixels. The γ is the opposite limit and

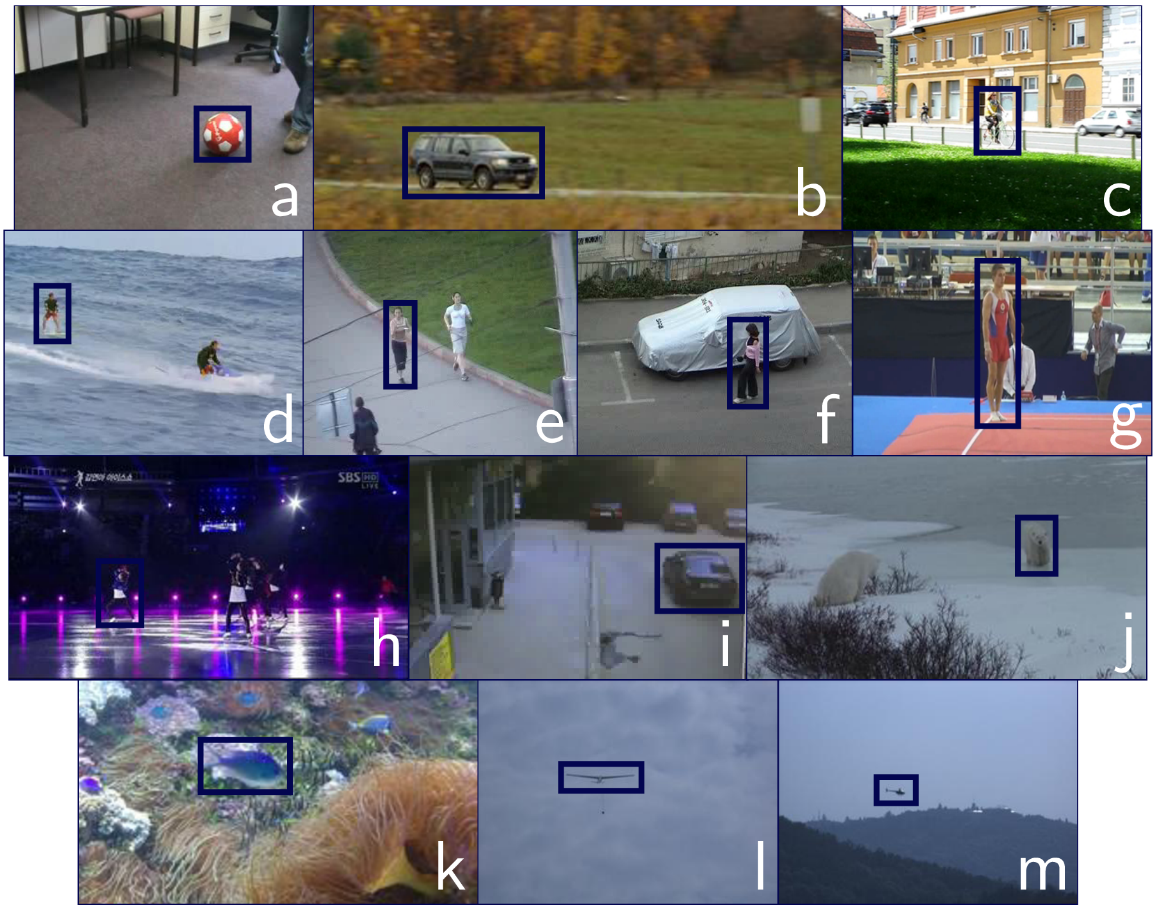


Figure 5.1: Visual description of evaluation dataset. The sequences are following: a) ball, b) car, c) bicycle, d)surfing, e) jogging, f) woman, g) gymnastics, h) skating, i) drunk, j) polarbear, k) fish1, l) plane51 and m) helicopter.

its value is already mentioned 70 pixels. The $[x_1, y_1]$ are coordinates of the groundtruth object center and the $[x_2, y_2]$ are the position of the center predicated by a tracker.

For every dataset, the evaluation is done in 10 runs, which are then averaged and additional values like standard deviation, minimum and maximum are obtained.

5.2.2 Results

On the introduced datasets the evaluation was done. The results are summarized in a table 5.2, where specific numbers can be found, and in the graph in figure 5.2, for a better perspective.

The graph aggregates all overall success rates, including the standard deviations, minimums and maximums as was described in description. The legend is attached. Both the graph and the table contain the same values, only the table contains the accurate values and the graph is well-arranged visualization.

Dataset	Overall	StdDev	Minimum	Maximum
Ball	67.8	8.3	55.3	82.4
Bicycle	79.9	10.8	65.9	99.5
Car	74.7	6.9	63.1	81.8
Drunk	57.4	29.0	14.3	87.9
Fish1	11.1	1.3	10.0	13.3
Gymnastics	39.9	3.1	36.5	47.3
Jogging	89.5	2.0	86.6	92.7
Polarbear	89.2	2.2	86.1	91.8
Skating	39.3	11.5	20.5	55.2
Surfing	100.0	0.0	100.0	100.0
Woman	90.2	8.3	69.8	98.6
Helicopter	15.5	12.0	7.0	32.5
Plane51	88.0	1.1	87.0	89.1

Table 5.2: Quality experiment - results.

5.3 Performance Focused Experiment

The proposed system performance comes from its title. It is also mentioned in section 3.1.1, where it is said that the expected performance is about real-time or near real-time.

This section describes the performance experiment and the acquired results. The system performance can be critical for some applications, where the tracker is not the primary function or where the solution computation power is limited. Such applications are typical for a field of robotics, where the tracking is only one of many and the solution hardware is typically low-end and relatively weak, as the power consumption of such devices is often more important.

5.3.1 Description

The measured value unit is a number of *frames per second (fps)*. The *fps* value is obtained as inverted value of a time consumed to proceed one frame. The relation between time consumed and the fps value is on equation 5.3.

$$fps = \frac{1}{\Delta t} \tag{5.3}$$

The overall value is a geometric mean of the *fps* value for all frames, where the tracking is performed. The „tracking is performed“ statement references to a state, when the object position is estimated in the particular frame, no matter if successfully or not. The point what matters is that the tracking algorithm does its work.

The reason is simple. When the object is lost (in meaning that the tracking is persuaded that it is lost), the tracking algorithm does not need to be executed and the speed is not relevant. Thus, the speed measurement of non working solution is a bit meaningless.

The evaluation is done on all dataset sequences. For every sequence the speed for every valid frame is measured and averaged using geometric mean, and the minimal and maximal speed for single frame is noted. The standard deviation is computed for every sequence using the values for single frames.

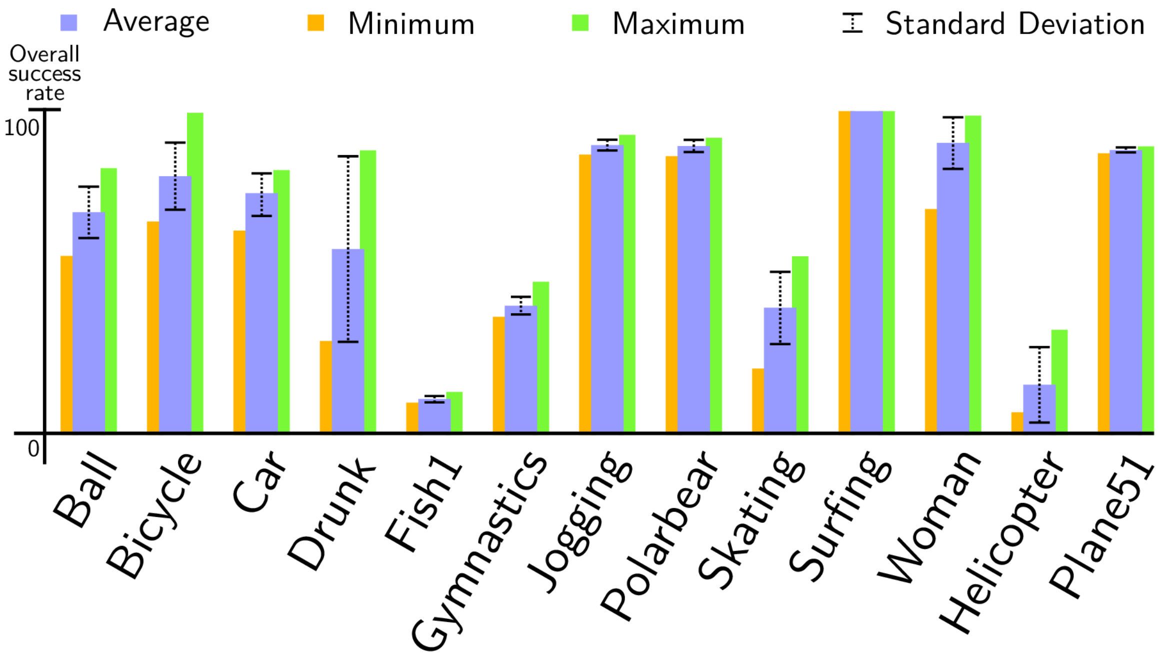


Figure 5.2: Visual results of quality focused experiment.

The proposed *MBT* multiplies the number of trackings done for one frame pair. Therefore, the performance evaluation is done for a system with usage of the *MBT* as well as for the system without it.

Every sequence is evaluated 10 times for both the *MBT* presences, to get more statistically informed values. The minimal and maximal values are absolute values over all the runs, whether the overall speed and the standard deviation are obtained by averaging the results in single runs with geometric mean.

5.3.2 Results

The experiment was executed on a machine with a Intel Core i7-3520M (the IvyBridge Mobile) CPU at frequency 2.90GHz. The speed is not meaningful in itself, but it can be observed at least in general and also as a comparison for the different parameters. The results are again in both the table 5.3 and in the figure 5.3.

The visualization shows overall average *fps* for the system with and without the *MBT*. From the results can be seen that the system's performance does not depend on the used dataset, which basically means the object size and its movement model does not affect the computational part. The occasional high or low number of *fps* is caused by higher number of failures during single tracking step.

5.4 Stability Focused Experiment

The stability of a system is often not one of the most observed factors. However, the proposed system is evaluated also in this area. The stability in scope of this thesis means a rate how much the results are stable, on a level of every frame and on a level of whole evaluation sequence.

	MBT enabled				**MBT disabled**			
Dataset	**FPS**	**StdDev**	**Min**	**Max**	**FPS**	**StdDev**	**Min**	**Max**
Ball	24.3	0.7	11.2	75.2	45.8	2.6	16.0	93.9
Bicycle	23.1	0.2	14.3	35.1	40.6	0.6	25.0	62.3
Car	20.7	0.0	12.2	23.0	34.1	1.0	19.4	41.1
Drunk	17.2	1.3	6.5	22.3	28.5	1.6	10.2	35.4
Fish1	14.4	0.1	7.2	15.6	32.9	0.1	19.4	37.9
Gymnastics	22.5	0.5	12.8	28.8	38.7	1.0	31.9	54.0
Jogging	22.7	0.2	14.3	25.4	37.4	3.7	21.4	48.3
Polarbear	16.3	0.3	9.7	17.9	24.4	0.2	13.7	30.1
Skating	16.3	0.7	7.1	34.3	26.6	2.4	7.3	42.4
Surfing	24.6	0.6	14.6	27.4	49.3	0.3	34.8	56.8
Woman	22.1	0.4	9.0	37.3	37.6	2.2	17.4	69.4
Helicopter	15.3	1.2	5.1	17.1	21.4	3.1	7.7	26.9
Plane51	16.0	0.5	4.4	16.8	22.9	1.7	10.4	24.6

Table 5.3: Performance experiment - results.

The system is not stabilized with any extra stabilization method, like Kalman filtering. The only part which could provide more stability is a fusion detected in section 3.7. As the frames are samples from continuous stream and there are no frame-cuts or any other moments which can have serious impact on the continuity, the evaluation of the stability makes good sense.

5.4.1 Description

The stability is measured as a side effect of the quality and the performance experiments. The standard deviation of the quality among the whole runs was already displayed as a part of quality experiment in section 5.2. The standard deviation of performance results was also collected, again on the level of whole runs.

This experiment is based on standard deviation among results obtained during several testings. As the system contains some parts, where the random number generator occurs as one of the inputs, the stability can be statistically measured.

Therefore, 10 runs focused on performance are executed and the standard deviation of the result is computed. The standard deviation is computed also above all overall performance results, collected from the whole run. The evaluation is done only above a subset of the whole dataset pool, when the most important results from the performance and the quality point of view were measured. Also an overall value is added, which summarizes a mean over all particular values of all datasets with a standard deviation above those results.

Similar situation is in the stability evaluation of the quality. The 10 runs are executed and overall success rate is computed. The standard deviation is the obtained from these overall results. This is already mentioned as the evaluation of the system quality earlier.

5.4.2 Results

The stability experiment was evaluated as a part of the other experiments, primarily the performance and the quality focused ones. Therefore, the table 5.4 only summarizes the

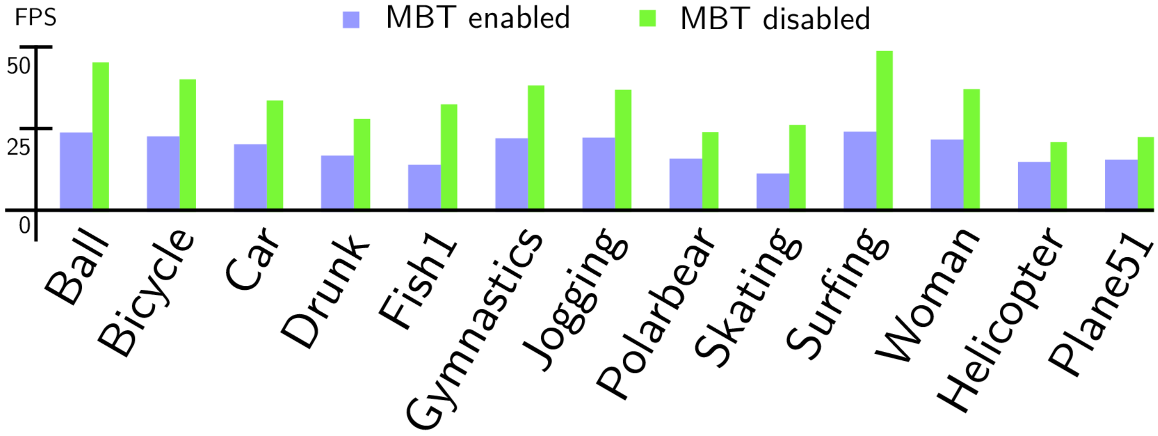


Figure 5.3: Visual representation of the performance focused experiment.

results, which are the standard deviations in this context. To understand the results, the lower value means the system is more stable, the minimum is then 0, which is equivalent of perfectly steady results of the system.

The maximum and in the same time the worst value of stability of the system's quality is 50, which basically means that the system either totally succeeds or totally fails. For the performance experiment it depends on actual *fps* value, but the general „the lower the better“ is valid, too.

	Dataset				
Experiment	**Ball**	**Polarbear**	**Woman**	**Plane51**	**Total**
Quality	8.3	2.2	8.3	0.7	7.4/7.5
Performance	0.7	0.3	1.1	0.5	0.6/0.4

Table 5.4: Stability focused experiment. The results represent standard deviation from appropriate experiments, which is here interpreted as a system stability.

It can be seen that the stability for single datasets can quite vary, which can point to some problematic parts and implicate further work.

5.5 Component Focused Evaluation

During the chapter designing the solution, some of the new methods were introduced. There are namely the *Points To Track Generation*, which consists of mask computation primarily in section 3.3.1 and the *Multiple Bidirectional Tracking*, which is the core contribution.

The both contributions will be evaluated in this section. The expected results are clear; to prove that the use of the proposed parts has an impact on better quality performance of the whole system and that the system without usage of these parts is not as good as with them.

5.5.1 Description

The component experiment was performed on subset of dataset chosen by random. The reason for this limited evaluation is a number of combinations for which the evaluation needs to be executed. The components are evaluated from the point of the quality.

It was stated that the two tested components are the *MBT* and the *mask generation*. The *MBT* has few parameters which need to be evaluated. From the section 3.4.1 it is a number of points tracked in forward direction m and a number of points tracked backward for every forwardly tracked points n. The situation when the $m = 1$ and the $n = 1$ is the same as the conventional *Forward-Backward Error* [16]. The specific numbers which will be used in for the parameters are $m \in 1, 3, 5$ and $n \in 1, 2, 3$

For the mask there are no parameters which need to be set. So the only flag about the mask computation is whether the mask is used or not. All the possibilities are evaluated in a subset of all combinations, as it does not make any sense to evaluate an acquisition for a mask for all of the combinations of *MBT*.

On the proposed dataset to evaluate component tests, the particular evaluations are executed in agreement with the performance and the quality experiment introduced above. So it means that all the results are obtained from several runs.

5.5.2 Results

The proposed solution was evaluated from the point of view of involving newly proposed component parts, namely the mask computation (stated as *mask* in the graph and the table below) and the *Multiple Bidirectional Tracking* (the *MBT* abbreviation bellow). For both the components a number of parameters were set, and all of them resulted in a table 5.5 and a simplified view can be seen in figure 5.4.

	Parameters					
Dataset	**1/1**	**1/1/M**	**3/2**	**3/2/M**	**5/3**	**5/3/M**
Ball	59.8/09.4	56.9/14.8	64.2/13.4	64.5/15.4	**67.9**/08.7	67.8/**08.3**
Bicycle	82.6/09.9	72.9/07.3	**86.5**/08.6	83.3/11.6	82.8/**06.6**	79.9/10.8
Car	74.4/08.1	74.6/08.4	**82.1**/03.0	74.2/06.7	81.0/**02.1**	74.7/06.9
Jogging	88.8/04.4	75.0/26.9	**91.2**/**01.4**	90.1/01.6	88.8/02.4	89.6/02.0
Polarbear	87.7/04.2	87.3/03.5	89.1/**01.6**	88.5/04.1	**89.5**/02.3	89.2/02.2
Woman	67.4/21.4	66.2/23.3	76.3/27.9	79.1/19.4	76.3/14.5	**90.2**/**08.3**
Helicopter	17.1/12.6	31.2/27.5	45.3/28.8	32.7/**00.8**	**63.6**/21.7	15.5/12.0
Plane51	87.1/01.2	85.5/**00.2**	85.2/00.7	85.2/00.7	83.9/**00.2**	**88.0**/01.1

Table 5.5: Component focused evaluation - results for all input parameters.

In the table 5.5 the parameters noted are in format m/n[/M], where the `m` stands for the number of forwardly tracked points, the `n` is for the backwardly tracked points and the `/M` part signalize whether the mask is used (the `/M` presence) or not. The same notation is used also in the graph 5.4.

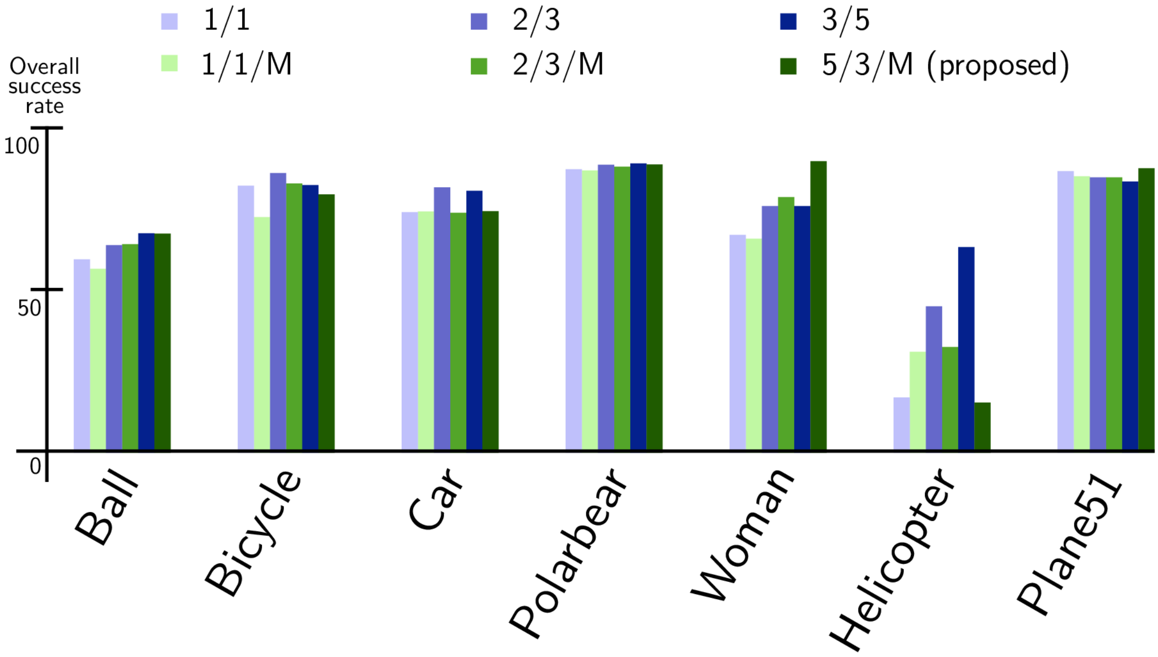


Figure 5.4: Component focused evaluation, the simplified visual representation.

5.6 Comparison to State of the Art Solutions

The final evaluation is based on comparison with other state of the art solutions. As the state of the art were chosen the original OpenTLD citenebehaythesis and the CMT [29]. The first one uses Lucas-Kanade optical flow [25] as the tracking core and the points to track are generated by uniformly distributed grid.

The second one, CMT, uses a BRISK [19] descriptors for points detected by FAST [34, 35] detector. As the tracking core, it uses a Lucas-Kanade optical flow as well.

The results are expected to be equal with the state of the art solutions on the standard datasets and to exceed them in the newly proposed datasets. This should be a general sign that the proposed solution is successful.

5.6.1 Description

All needed results of the proposed system are already gathered in the previous experiments. So we need to evaluate also the state of the art solutions, to do the comparison.

The used dataset to comparison is set as a subset of the whole dataset, chosen by overall success rate of the proposed solution, to equally cover an axis of success from the lowest success rate achieved to the highest success rate achieved.

The comparison is done on the quality level as well as on the performance one. The other trackers are compiled locally and evaluated with the same manners as the proposed tracker. It means that the runs are repeated several times and the results are combined together, to get statistically relevant results. The other reason is to evaluate all solutions in the same way.

5.6.2 Results

The evaluation was done for all the trackers in the same way. As was stated, both the performance and the quality focused experiments were performed. The results can be seen in graph in figure 5.5, or in the table 5.6 for exact values, too.

	Quality experiment			Performance experiment		
Dataset	**Our**	**OpenTLD**	**CMT**	**Our**	**OpenTLD**	**CMT**
Ball	**67.8**/08.3	**98.6**/00.0	97.8/00.7	**24.3**/00.7	119.6/03.6	**152.8**/02.6
Bicycle	**79.9**/10.8	**07.0**/00.0	65.6/02.9	**23.1**/00.2	**144.7**/04.3	35.4/00.7
Car	**74.7**/06.9	63.9/00.0	**62.4**/00.0	**20.7**/00.0	**108.9**/03.2	77.0/00.7
Jogging	**89.5**/02.0	**25.4**/00.0	81.8/00.0	**22.7**/00.2	**196.8**/07.1	132.4/01.5
Helicopter	15.5/12.0	**04.3**/00.0	**45.5**/00.0	**15.3**/01.2	56.9/09.5	**61.6**/00.9
Plane51	**88.0**/01.1	07.9/00.0	**03.5**/09.8	**16.0**/00.5	**107.2**/01.9	40.1/01.7

Table 5.6: Comparison to state of the art solutions - exact results for both the performance and the quality focused experiments. The most significant numbers (both the best and the worst) are highlighted.

In the table there are the exact achieved values for all solutions. Beside the average value (both the overall success rate and the fps), also the standard deviation is presented. The standard deviation again stands for the deviation between single runs of the same experiment which were used to get overall result.

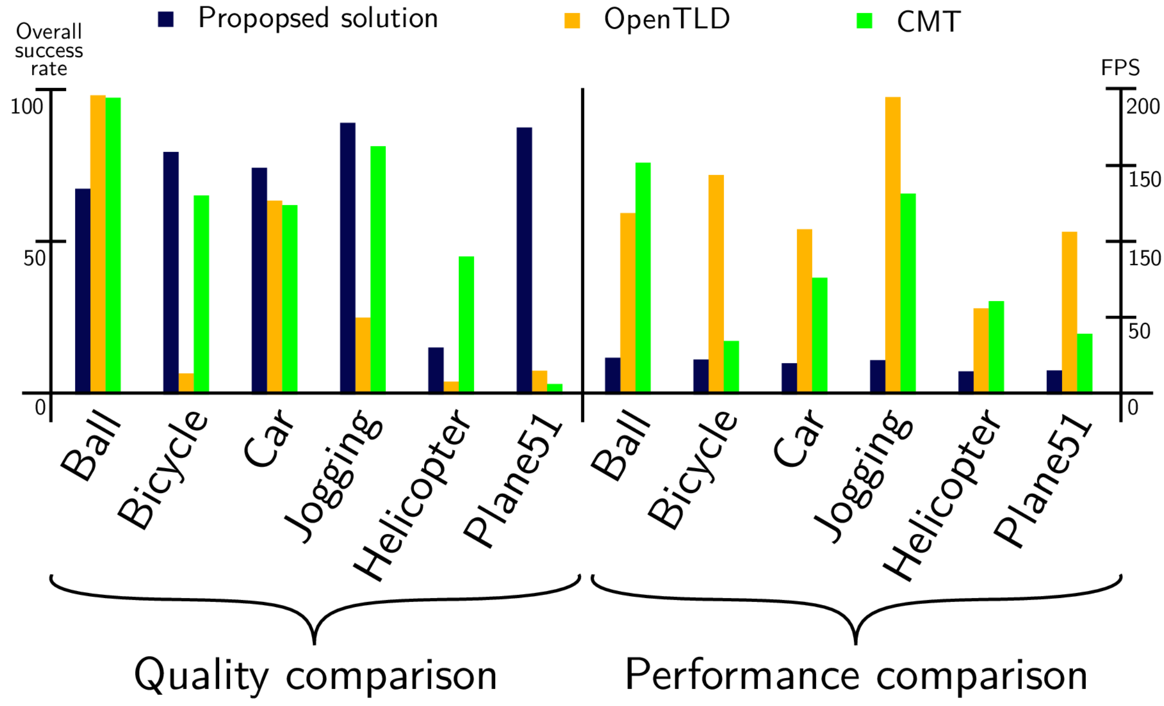


Figure 5.5: Comparison to state of the art solutions, the visual representation.

The graph quite clearly demonstrates that the proposed solution achieves better results in the quality focused experiment. The performance of the proposed solution was lower, but it still met the requirements and it is outweighed with better performance.

5.7 Summary

In this chapter, an evaluation was designed and executed. At first, the evaluation dataset was introduced in detail, including the problems which occur in the particular datasets. The next part is dedicated to the experiments themselves. Every experiment was described deeply, to permit its repetition. It contains also the expected results.

The first experiment was focused on quality. This is apparently the most important evaluation and very often also the only one, which is executed. It was shown that the proposed system reaches a satisfactory level of quality across a various set of dataset, which was proposed before. There is also no dataset, which would fail totally, which was the target result in this experiment. The mean overall success rate is 64.81 %, while the minimal is 11.1 % and the maximal 100.0 %.

The next evaluation was aimed to the performance domain. The measured value is *fps* and the expected result is real-time of near real-time. The achieved value was on average 19.7 fps for the complete proposed solution, the highest value was then 24.6 fps and the lowest was 14.4 fps. It is useful to mention that even the highest average value achieved (24.6 fps) is for the system with limited object movement to 32 pixels in all directions, which is more than the majority of the state of the art trackers. Therefore, the real-time expectation was fulfilled.

The experiment in the next section focuses on the system's stability. The stability is deduced from the standard deviation of the quality and the performance experiments. No specific result was expected, but the achieved stability can be considered as satisfactory.

The newly proposed components of the system, the mask computation and the *Multiple Bidirectional Tracking* are evaluated in the next experiment. For the majority of the tested parameters, it was proven that the *MTB* brought an increase in quality. On the other hand, the mask was far not always better and the system without the mask sometimes behaved better. In overall, with both components connected in the system, the improvement was reached. Furthermore, it can be seen from the table that the majority of best results achieved solution with parameters set as 5 forward trackings and 3 backward trackings. That seems as the best parameters observed so far.

The last part was about the comparison with other state of the art solutions. It was shown that in the standard dataset the proposed solution performed comparable to the other solutions, and in the newly proposed part of the dataset the proposed solution also overperformed them. Both results are considered as very valuable.

The proposed system was evaluated from several points of view. The evaluation demonstrated that the proposed solution including all the parts performs better than the system without the improvements. Moreover, the proposed system also overperformed the state of the art solutions in the quality focused experiment in several evaluation datasets. It was also the aim of this thesis.

Chapter 6

Conclusion

This work is aimed to the visual object tracking in video with focus on long-term task. The tracking object is supposed to be general object and the proposed system has no previous knowledge about it. The system should work in real-time speed and should learn the model adaptive.

The proposed solution was designed on top of knowledge of recent related work in a field of object trackers and object detectors. The detector is an integral part of the long-term task, which means also of the proposed solution. Some previous approaches to perform visual object tracking in context of long-term behavior and primarily used methods were summarized in chapter 2. In the chapter, some problems of object tracking, which are either generally unsolved or at least still problematic, are also described.

The system was designed as a complex system built from three main parts; the tracker, which is the core part, the detector, which allows to re-initialize the tracking process after its lose, and the adaptive model learning, which is also a required part to successfully handle long-term caused appearance changes. This design of the proposed tracking system is in chapter 3. The system is designed to be able to successfully solve tasks with the problems described in the first chapter and still be general as much as possible.

The completely new tracking core was proposed, based on the proposed method called *Multiple Bidirectional Tracking*. The method comes from the *Forward-Backward Error* from [16], but instead of simple measurement of a quality of tracking points it adds a solution how to increase the quality by the tracking itself.

The object is tracked frame-to-frame and is represented by its bounding rectangle. As the system is based on tracking weak quality corner points by approach called *GoodFeaturesToTrack* [40], it is necessary to pick out these corner points from the object area, instead of the background regions. For that purpose a mask computation method, to estimate the object shape from its bounding rectangle was also introduced.

The stability of the system was improved by another proposed part called *Detector-Tracker Fusion*. As the name suggests, it is the component called when both the tracker and the detector have valid results. It helps to reduce cumulative tracker's error as well as the tracker's *drift*.

The system is implemented on the background of the OpenTLD tracking system by Nebehay [28]. The implementation uses the latest OpenCV framework and it is written in C++ with respect to object oriented principles.

The system was evaluated from several points of view, which are all described in chapter 5. The first experiment focuses on the most expected quality, which stands in the scope of this project as an overall success rate averaged for all single frames of an image sequence.

Then, the experiments focused on the computational performance, the stability of the system results and the particular improvements follow. The last experiment compares quality and performance results of the proposed solution with other state of the art solutions.

The evaluation revealed that the proposed system behaves better in a quality domain than the other solutions, in a majority of tested datasets. Moreover, in the original part of evaluation dataset, the differences were much more significant. In the performance domain, the proposed system reaches a speed between 10 and 25 fps, which satisfies the real-time requirement.

Besides the evaluated results, a project which comes from this thesis was presented at Excel@FIT 2015, the student conference and Faculty of Information, Brno University of Technology. The paper called *Real-Time Long-Term Visual Object Tracking* was accepted for oral presentation as well as for the poster presentation, and in the associated competition won the first prize in a category *Innovation Potential* and the fifth prize in a category *Technological Level*. The accepted paper is attached in appendix A.

Regarding the proposed solution and its evaluation, including the evaluation performed before accepting the paper for the conference, the thesis tasks were accomplished.

The proposed solution can be further improved in many aspects. The first one can be a better online learning system, which performs the actual learning. This could help the detector in better behavior and therefore the overall quality should increase significantly. The tracker acceleration, which was drawn in implementation chapter, could be analyzed deeper to achieve higher computational performance. A shorter time needed to proceed frame could mean more time to improve results in a quality field. Furthermore, the system is designed to be modular and even the main parts like the detector or the tracker can be easily replaced with different methods.

Bibliography

[1] Alahi, A., Ortiz, R., and Vandergheynst, P. FREAK: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on* (June 2012), pp. 510–517.

[2] Babenko, B., Yang, M.-H., and Belongie, S. Robust object tracking with online multiple instance learning. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 33*, 8 (2011), 1619–1632.

[3] Bay, H., Ess, A., Tuytelaars, T., and Gool, L. V. Speeded-up robust features (SURF). *Comput. Vis. Image Underst. 110*, 3 (June 2008), 346–359.

[4] Beauchemin, S. S., and Barron, J. L. The computation of optical flow. *ACM Computing Surveys (CSUR) 27*, 3 (1995), 433–466.

[5] Bibby, C., and Reid, I. Real-time tracking of multiple occluding objects using level sets. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (2010), IEEE, pp. 1307–1314.

[6] Bouguet, J.-Y. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation 5* (2001).

[7] Brasnett, P. A., Mihaylova, L., Canagarajah, N., and Bull, D. Particle filtering with multiple cues for object tracking in video sequences. In *Electronic Imaging 2005* (2005), International Society for Optics and Photonics, pp. 430–441.

[8] Comaniciu, D., Ramesh, V., and Meer, P. Kernel-based object tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 25*, 5 (Apr. 2003), 564–577.

[9] Farnebäck, G. Two-frame motion estimation based on polynomial expansion. In *Image Analysis*. Springer, 2003, pp. 363–370.

[10] Fleet, D., and Weiss, Y. Optical flow estimation. In *Handbook of Mathematical Models in Computer Vision*. Springer, 2006, pp. 237–257.

[11] Forsyth, D., and Ponce, J. *a modern approach*, 2nd ed. ed. Pearson, Boston, 2012.

[12] Gabriel, P. F., Verly, J. G., Piater, J. H., and Genon, A. The state of the art in multiple object tracking under occlusion in video sequences. In *Advanced Concepts for Intelligent Vision Systems* (2003), pp. 166–173.

[13] Hare, S., Saffari, A., and Torr, P. H. S. Struck: Structured output tracking with kernels. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (Nov. 2011), pp. 263–270.

[14] Isard, M., and Blake, A. Condensation—conditional density propagation for visual tracking. *International journal of computer vision 29*, 1 (1998), 5–28.

[15] Kalal, Z., Matas, J., and Mikolajczyk, K. Online learning of robust object detectors during unstable tracking. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on* (2009), IEEE, pp. 1417–1424.

[16] Kalal, Z., Mikolajczyk, K., and Matas, J. Forward-backward error: Automatic detection of tracking failures. In *Pattern Recognition (ICPR), 2010 20th International Conference on* (Aug. 2010), pp. 2756–2759.

[17] Kalal, Z., Mikolajczyk, K., and Matas, J. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 7 (2012), 1409–1422.

[18] Lampert, C. H., Blaschko, M. B., and Hofmann, T. Beyond sliding windows: Object localization by efficient subwindow search. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* (2008), IEEE, pp. 1–8.

[19] Leutenegger, S., Chli, M., and Siegwart, R. Y. BRISK: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (Nov. 2011), pp. 2548–2555.

[20] Lewis, J. Fast normalized cross-correlation. In *Vision interface* (1995), vol. 10, pp. 120–123.

[21] Lin, L., Wu, T., Porway, J., and Xu, Z. A stochastic graph grammar for compositional object representation and recognition. *Pattern Recognition 42*, 7 (2009), 1297–1307.

[22] LIRIS, F. The visual object tracking vot2014 challenge results.

[23] Lowe, D. G. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on* (Sept. 1999), vol. 2, pp. 1150–1157.

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

[25] Lucas, B. D., and Kanade, T. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2* (San Francisco, CA, USA, Apr. 1981), IJCAI'81, Morgan Kaufmann Publishers Inc., pp. 674–679.

[26] Matthews, I., Ishikawa, T., and Baker, S. The template update problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 26*, 6 (June 2004), 810–815.

[27] Medrano, C., Herrero, J., Martínez, J., and Orrite, C. Mean field approach for tracking similar objects. *Computer Vision and Image Understanding 113*, 8 (2009), 907–920.

[28] Nebehay, G. *Robust object tracking based on tracking-learning-detection.* Wien, 2012.

[29] Nebehay, G., and Pflugfelder, R. Consensus-based matching and tracking of keypoints for object tracking. In *Winter Conference on Applications of Computer Vision* (Mar. 2014), IEEE.

[30] Nummiaro, K., Koller-Meier, E., and Van Gool, L. Object tracking with an adaptive color-based particle filter. In *Pattern Recognition.* Springer, 2002, pp. 353–360.

[31] Olson, C. F. Maximum-likelihood template matching. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on* (2000), vol. 2, IEEE, pp. 52–57.

[32] Ozuysal, M., Fua, P., and Lepetit, V. Fast keypoint recognition in ten lines of code. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on* (2007), Ieee, pp. 1–8.

[33] Rabiu, H. Vehicle detection and classification for cluttered urban intersection. *International Journal of Computer Science, Engineering and Applications 3*, 1 (2013).

[34] Rosten, E., and Drummond, T. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision* (Oct. 2005), vol. 2, pp. 1508–1511.

[35] Rosten, E., and Drummond, T. Machine learning for high-speed corner detection. In *European Conference on Computer Vision* (May 2006), vol. 1, pp. 430–443.

[36] Rowley, H. A., Baluja, S., and Kanade, T. *Human face detection in visual scenes.* School of Computer Science, Carnegie Mellon University Pittsburgh, PA, Nov. 1995.

[37] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (Nov. 2011), pp. 2564–2571.

[38] Schweitzer, H., Bell, J. W., and Wu, F. Very fast template matching. In *Computer Vision — ECCV 2002*, A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, Eds., vol. 2353 of *Lecture Notes in Computer Science.* Springer Berlin Heidelberg, 2002, pp. 358–372.

[39] Shang, J., Chen, C., Liang, H., Tang, H., and Sarem, M. A novel fragments-based similarity measurement algorithm for visual tracking. *Journal of Computers 9*, 9 (Sept. 2014), 2167–2172.

[40] Shi, J., and Tomasi, C. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on* (June 1994), pp. 593–600.

[41] Stalder, S., Grabner, H., and Van Gool, L. Beyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on* (2009), IEEE, pp. 1409–1416.

[42] Sun, X., Yao, H., and Zhang, S. Contour tracking via on-line discriminative appearance modeling based level sets. In *Image Processing (ICIP), 2011 18th IEEE International Conference on* (2011), IEEE, pp. 2317–2320.

[43] Viola, P., and Jones, M. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* (2001), vol. 1, IEEE, pp. 501–511.

[44] Welch, G., and Bishop, G. An introduction to the kalman filter, 1995.

[45] Yilmaz, A., Javed, O., and Shah, M. Object tracking: A survey. *Acm computing surveys (CSUR) 38*, 4 (2006), 13.

[46] Yilmaz, A., Li, X., and Shah, M. Contour-based object tracking with occlusion handling in video acquired using mobile cameras. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 26*, 11 (Nov. 2004), 1531–1536.

[47] Zhu, Q., Wang, L., Wu, Y., and Shi, J. Contour context selection for object detection: A set-to-set contour matching approach. In *Computer Vision–ECCV 2008*. Springer, 2008, pp. 774–787.

Appendix A

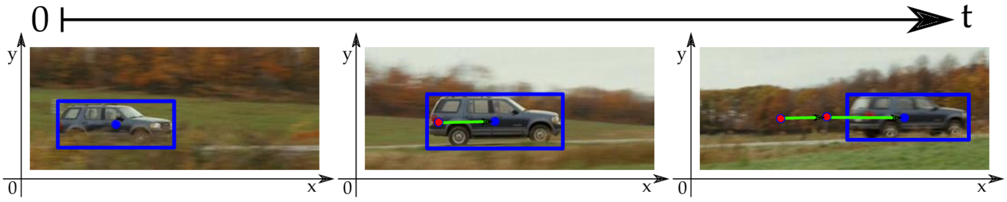
Paper at conference Excel@FIT 2015

A paper based on a work on this thesis was proposed and accepted for the student conference Excel@FIT 2015. The paper is called *Real-Time Long-Term Visual Object Tracking* was published in the conference proceedings and is attached from the next page.

Excel@FIT 2015
http://excel.fit.vutbr.cz

Real-Time Long-Term Visual Object Tracking

Martin Simon*



Abstract
Visual object tracking with focus on occlusion, background clutter, image noise and unsteady camera movements, those all in a long-term domain, remain unsolved despite the popularity it experiences in recent years. This paper summarizes a related work which has been done in trackers field and proposes an object tracking system focused on solving mentioned problems, especially the occlusion, rough camera movements and the long-term task. Therefore, a system combined from three parts is proposed here; the tracker, which is the core part, the detector, to re-initialize tracker after a failure or an occlusion, and a system of adaptive learning to handle long-term task. The tracker uses newly proposed approach of bidirectional tracking of points, which are generally weaker then commonly used keypoints. Outputs of both the tracker and the detector are fused together and the result is also used for the learning part. The proposed solution can handle mentioned problems well and in some areas is even better then the state-of-the-art solutions.

Keywords: object tracking — bidirectional tracking — partial occlusion — long-term tracking — full occlusion — rough camera movement

Supplementary Material: Demonstration Video

*xsimon14@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Despite the fact that a visual object tracking in video has become a popular topic in recent days, it still remains generally unsolved. There is a significant amount of very efficient object tracking systems which are sufficiently accurate and which work in real-time. Unfortunately, many problems, such as occlusion, no matter if full or partial, background clutter, noise or unexpected rough camera movements, are problematic parts of many recent visual object tracking related projects and actually exclude them from daily professional usage.

In this work I would like to focus on recent solutions in the field of tracking and try to build an object tracking system whose efficiency and performance are at least the same as the performance of the state-of-the-art trackers. Also, I would like to improve some critical parts which have serious impacts on accuracy or are too narrowly focused and lack generality.

Further aim of this work is to bring a new challenging evaluation dataset with some highlighted problems, which the standard trackers handle incorrectly. These problems include primarily very small objects of interest or very similar objects in comparison with the background. The new tracking system should be designed in the way to handle both the standard visual object tracking evaluation dataset and the new challenging dataset, ideally without any significant speed or accuracy reduction.

This paper is organized as follows. In section 2 related work is summarized. The section is further divided into a part describing current situation in objects representations (Subsection 2.1), tracking approaches are described in subsection 2.2 and the last subsection provides a short overview of complex tracking solutions presented in recent days.

The core contribution of the paper is in section 3, where the main parts of the proposed solution are described. The experiment is described in section 4, which contains the evaluation as well. In the last section (5) the results are interpreted and a conclusion is given.

2. Related Work

A significant amount of work has been written recently regarding the field of the object tracking in a video. In this section, related parts of tracking based systems are described.

2.1 Object Description

An object representation sets the apriori abilities of every tracking system. The object representation is a way how to treat the object model internally in the system. The system abilities are highly correlated with the type of the object representation.

Possibly the most straightforward way how to represent an object is to use a visual image template [1, 2]. The weakness of this representation can be an inability to cover object appearance changes, which may occur in case of tracking non-rigid objects or during tracking for longer periods.

Another option to represent an object could be a description with a set of good-to-track keypoints. The term keypoint is commonly understood as a significant point like corner or peak with its neighborhood. This method is generally much better than templates, but the quality of the keypoints is critical. There are various widely used descriptors, such as [3, 4, 5, 6].

There are also some other object description methods, like contours [7], which is basically an object geometry, or complex object representations [8], which combine other models together.

2.2 Localization Methods

Localization can be presented as the core of every object tracking system. The used approaches can be divided into two types; frame-to-frame (recursive) tracking and in-frame detection. The main difference is in usage of history of object position.

In the frame-to-frame tracking the history is used. It brings better handling of the missing information, but the processing error can be cumulated over time. On the other side, the in-frame detection does not use the history and searches for the object in every frame separately. The error cannot be cumulated, but it cannot handle the missing information neither. There is also less information in a single frame then in a sequence of multiple frames.

The object detection is performed with a method called *sliding window*, and it is widely used in many trackers and detectors [9, 10, 11]. This is generally a very slow approach, but the performance can be improved by developing some kind of *pyramid* to discard as many true negatives in early stages as possible [10, 12, 13, 14].

The recursive tracking is an approach much related to the object tracking. The reason is clear - the tracking is performed on top of a sequence of frames and it comes with higher density of information than single image. If it is possible, it is a good approach to use this kind of added information.

One way how to use this information is to model object with its movement, very often in form of position and velocity. Then, the future object position can be estimated and therefore the amount of possible object positions is significantly reduced. This is used in systems based on *particle filters* [15, 16] or *optical flow* [1, 17], among others.

2.3 Complex Solutions

The tracking system parts described in previous section are building blocks of complex tracking systems. Such a complex long-term tracking system requires a tracker for the basic tracking function, a detector for handling situations when the object gets lost or the tracker has failed, and some kind of learning system to handle object appearance changes. This does not necessarily require a presence of an object model [18], but the adaptation needs to be present in some way.

A huge number of visual object trackers exists. One often highlighted system is TLD [12, 19]. It is a tracking system with sufficient tracking performance, subtitled *tracking-learning-detection*. A tracking system based on strong keypoints and their voting can be CMT [20], which is considered a good representative of key-point based trackers. Another tracker which is good to mention is Struck [21], which is a tracker solution with adaptive visual templates and online SVM classifier. All of these trackers are considered state-of-the-art.

3. Proposed Solution

The long-term tracking system proposed here is based on OpenTLD [10], an improved C++ implementation of TLD [12]. The main reason is that the TLD consists of two main parts, the tracker and the detector, which is improved with adaptive learning system. In OpenTLD, there is more improvements, like breaking away from the necessity of strong keypoints.

Therefore, the proposed solution is the OpenTLD with a completely new tracker and a new system of fusion of outputs of both the tracker and the detector.

The particular tracking problem to solve is defined in 3.1. It directly leads into the object representation, which is described in 3.2. The core contribution of this paper, the bidirectional tracking proposal, is described in 3.3. Finally, the results of the detector and the tracker are fused in accordance with 3.4.

3.1 Problem Definition

The problem of visual object tracking in video sequence can be described as follows. The input of the system is a sequence of monochromatic video frames. The output can be generally a four dimensional vector $(x,y,w,h)_t$ symbolizing the object position and the size in every frame. The result can be also a zero vector, if the object is not present in a particular frame.

The algorithm is commonly initialized by the user, after marking the object with bounding rectangle. The object is generally unknown to the system and this user input is the very first information about the object.

So far the tracking is not different from the detection of an object. The difference comes with the usage of history of results to improve result in following frames. Unfortunately, an error introduced in early states leads to bigger error at the end, or eventually, a failure during the process.

3.2 Object Representation

The tracking object can be generally *anything* which exists in the real world. Therefore, every good object tracking system needs to find a proper object representation with respect to other system parts.

The most promising seems to be keypoints model described with usage of keypoints like [3, 4, 5, 6]. The advantages are that these keypoints are easily and quickly found and even if some of them are lost (e.g. due the *partial occlusion*), the rest can still represent the object well. The disadvantage comes with a situation when we are not able to find such strong keypoints on the object; that can happen e.g. due to a limited object size or poor object resolution or contrast.

In the proposed solution the object is internally treated in two different processes. According to that, there are also two different object representations. After the user sets the bounding box of the object, the image *pattern* is stored also as a very first true positive sample for the detector. The pattern is resized and normalized and stored in internal memory besides the true negative, which are taken randomly from the area in the same frame. The frames are marked as true negatives in this initial learning phase if they are disjunctive to the true positive one.

Much more interesting is the object representation used during tracking phase. We propose to use 16 points in their 11×11 neighborhood to describe the object, and to choose these points according to their quality to track [2]. As it was mentioned before, there is a very limited set of such points in our dataset (small object, poor resolution/contrast). Therefore, we select those 16 best points which are possible to find. It is highly expectable that the quality of the majority of these points will be very poor.

One of the problems of visual object tracking is called *background clutter*. It means that the background contains more information than the object itself and, hence, the best points will be chosen from the background rather than from the object, what is obviously a failure. To compensate this problem, a mask of the object is computed before the points are actually searched.

The mask represents generally a difference between the internal bounding box area and the bounding box neighborhood. The mask value is calculated for every single bounding box point. As it was outlined, the value is a sum of weighted differences of the nearest out-box regions. Each region size is set to 5×5 pixels. This is illustrated in figure 1.

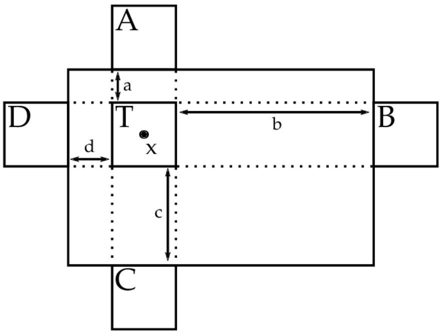


Figure 1. Mask computation illustration. The resulting value is given by a sum of weighted inverted NCC between the point neighborhood and all four nearest out-box regions

The figure 1 is described by equation 1. The symbols are the same as in the figure 1. Symbols A, B, C and D stand for the nearest out-box regions, the T is a neighborhood of the point x, which is the point of the mask. The function $NCC(M,N)$ stands for the *normalized cross correlation* and it is used to compute the similarity/difference between the templates M and N. The symbols a, b, c and d are distances of the computed region T and corresponding out-box regions. These distances are used to compute weights. The symbols w and h represents the bounding box width, resp. height.

$$\begin{aligned} p \quad = \quad & NCC(T,A)*(1-(a/h)^3)+ \\ & NCC(T,B)*(1-(b/w)^3)+ \\ & NCC(T,C)*(1-(c/h)^3)+ \\ & NCC(T,D)*(1-(d/w)^3) \end{aligned} \tag{1}$$

The mask can be seen in illustration 2. On the left there is the original bounding box used to compute the mask. On the right there is the mask. The white color means the most different regions and therefore *the object*.

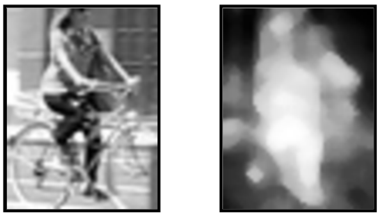


Figure 2. Original *bounding box* and its mask. The white color represents the most different areas

This mask is used to select points to track from the object regions rather than from the background. The points placed on the background can lead to serious tracking error.

3.3 Multiple Bidirectional Tracking

Regarding the expected poor quality of points used to track, it has to be improved in tracking process itself. In [19] an approach called *Forward-Backward Error* (FBE) is presented. This basically means that every point is tracked in forward direction, the result is tracked back and the distance between the original point and the returned one is measured and called the *FBE*. Points with too high FBE are then excluded from the tracking process.

This idea brings very good results as it was proved in [19]. Unfortunately, this only excludes wrongly tracked points from the process, but does not improve their individual quality.

In this paper is proposed an improved solution based on multiple tracking in both directions. Every point selected to tracking is tracked in forward direction, which results in 2D histogram (a map) of expected point positions in the next frame, which are measured as the *NCC* between the original point area and the tracking point area.

Instead of taking only the most valuable point to track back, like it is in FBE, we track back 3 most expected positions. As it was observed from experiments, the correct tracking position is not always the most valuable position in the map, but sometimes also the second one or even the third one.

After tracking them back we take into account 2 best candidates of every backward tracking, so for one tracking point we get 3 image candidates and 6 possible backward tracking images. For each of these backtracked images a distance to the original position is measured and the closest point is noted. The starting point for the noted one is then marked as a result.

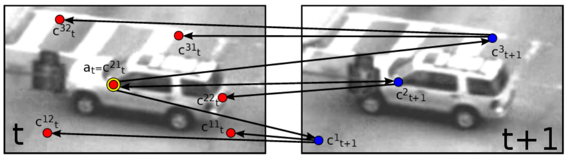


Figure 3. Multiple Bidirectional Tracking (MBT). Taking into account more candidates and tracking them back brings higher chance to find successful tracking path

The situation is better described with an image. In figure 3 can be seen point a_t on the left part (frame in time t) and three of its possible trackings to the frame in time $t+1$. Also, two possible trackings back for every candidate to the original frame can be seen, with some failing results and a successful one.

In the end, the successfully tracked points are those, which end in the same point they came from, with α as a spatial tolerance. Other points are excluded as wrongly tracked, as is in the FBE.

3.4 Detector-Tracker Fusion

The detector is an integral part of a long-term tracking system. It is the only way how to re-initialize tracking process after full occlusion or tracker failure.

The detector used in the proposed solution is taken from [10] unchanged. The detector is very fast, because it is developed with respect to cascade (or pyramidal) principles. The detector cascade consists of 3 stages. A *Variance filter*, which is the first stage with straightforward task to reject as many false positives as possible. The measurement metric is a simple variance. The next stage is an *Ensemble classifier*. This classifier uses method called *random fern classification* [22, 23]. The last stage and the slowest one is a *template matching* with usage of *NCC* as the similarity measurement technique.

With usage of both the detector and the tracker, following rules can be set. If only the detector or the tracker has result, its result is used. When neither the tracker nor the detector has result, then the object is considered as invisible.

In case both the tracker and the detector have their results, then the detector's output is used as an addition to the tracker's one in a fusion originally presented in this paper. All the combinations are for better overview in figure 4.

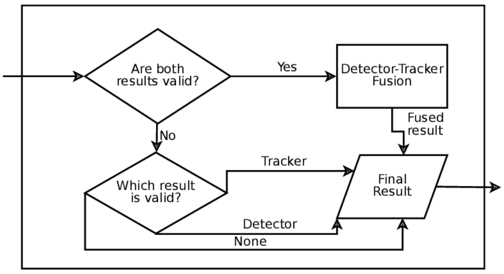


Figure 4. If only one (the detector or the tracker) is successful, its output is used. Otherwise, their outputs are fused

The tracker's and the detector's output is a position and a size of the object. Then, if the results overlap with at least 40%, they can be fused. The fusion result is computed as weighted mean and the weight is considered to be a certainty of the tracker and the detector.

The detector's certainty is computed in its last stage as in equation 2. Then $n \in negativesamples$, and $p \in positivesamples$.

$$\begin{aligned} d_N &= 1 - max(NCC(T, n_i)) \\ d_P &= 1 - max(NCC(T, p_i)) \\ a &= d_N/(d_P + d_N) \end{aligned} \tag{2}$$

The certainty of the tracker is set in accordance with the detector's one. It can be easily done with a small hack; the tracker's result put as an input for the last stage of the detector to get the very same certainty, only for the tracker this time.

The fused result position is illustrated in equations 3 and 4.

$$x = \frac{a_t * x_t + a_d * x_d}{a_t + a_d} \tag{3}$$

$$y = \frac{a_t * y_t + a_d * y_d}{a_t + a_d} \tag{4}$$

In the equations 3 and 4 the x and y are the fused coordinates of the object center, the a_t and a_d are the certainties of the tracker and the detector. The x_d, y_d, x_t and y_t are original outputs from the detector and from the tracker.

This kind of fusion should bring better movement correction in case, e.g. the tracker drifts. On the other hand, an additional error can be entered from the detector, in case of false positives.

4. Experiment and Evaluation

This section consists of two main parts. In the first part (Subsection 4.1), the experiment and evaluation prerequisites are set, including the evaluation dataset. The next part (Subsection 4.2) summarizes results of the evaluation according to the experiment presented in the first part.

4.1 Experiment Description

The dataset used for evaluation of the proposed tracking system consists of several standard video sequences and a few new ones. The standard dataset is represented with sequences known as *ball*, *bicycle*, *car* and *jogging* and it is taken from *VOT2014* [24]. The subset covers problems like object rotation (the ball), background clutter (the bicycle), object scale (the car) and occlusion (jogging).

The originally obtained sequences have work titles *helicopter* and *plane51*. This dataset focuses on small objects of interest (both), low contrast and poor quality resolution (the helicopter), object appearance changes (the plane51) and rough camera movements (both). The plane51 consists from 923 frames and the helicopter from 440 frames. The groundtruth was annotated manually by marking objects' centers.

In figure 5 can be seen representatives from all the datasets. In the top line can be seen the representative images from the standard dataset, in the bottom line are representatives from the original one.

For every frame sequence the proposed tracking solution is evaluated several times to get statistically relevant result. The only result which is measured is a position of the object center on a particular frame. The success rate for every frame is measured according to equation 5.

$$rate = 1 - \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} - \sigma}{\gamma - \sigma} \tag{5}$$

The equation 5 says that the success rate for the particular frame is equal to complement to 1 of normalized Euclidean distance between the correct center position and the position the tracker gives. The equation also contains boundaries due the manual groundtruth marking process. It says that the distances shorter than σ (30 pixels) are considered as 100% correct, and longer that γ (70 pixels) are considered as 100% incorrect.

Other long-term tracking solutions have been already mentioned in section 2.3. Only to summarize it, the compared trackers are OpenTLD [10] and CMT [20]. The reason is mainly that they are considered the state-of-the-art. Another reason is that these trackers have their reference implementation publicly available.

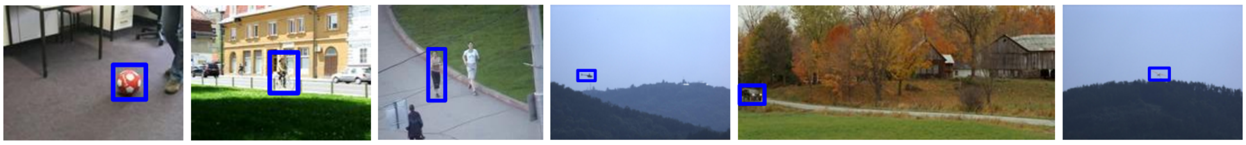


Figure 5. Dataset samples. From the left to the right they are: *ball*, *bicycle*, *jogging*, *helicopter*, *car* and *plane51*

4.2 Results

The proposed solution have been tested in quality domain as was mentioned in the previous subsection. In table 1 are summarized evaluation results for system without the mask computation nor the *MBT*, for both improvements separately and a complete system. Every results is combined from a mean of overall success rate of all sequence frames together in several runs and a standard deviation which symbolizes how unstable the system is (how different the single overall results were).

Table 1. Improvements results. Overall success rate for particular improvement usage and standard deviation. All values are percentages

	None	Mask	MBT	Complete
Ball	66 / 16	66 / 10	73 / 09	71 / 06
Bicycle	70 / 05	72 / 10	71 / 05	79 / 07
Car	73 / 09	79 / 11	78 / 10	79 / 14
Jogging	81 / 10	78 / 13	90 / 03	84 / 13
Helicopter	75 / 07	09 / 01	74 / 04	70 / 32
Plane51	87 / 02	85 / 01	87 / 02	85 / 02

The table shows how the proposed improvements mostly bring an increase of performance in comparison with a system without these improvements. The complete system is almost always better than the basic one.

Table 2 sums success rates for every evaluation dataset and the mentioned trackers to compare, and also the results of system with all proposed improvements. The overall success rate is again a mean from several runs. The most significant numbers (both the best and the worst) are highlighted.

Table 2. Evaluation results. Overall success rate and standard deviation. All values are percentages

	Our	OpenTLD	CMT
Ball	**71** / 35	**99** / 11	98 / 11
Bicycle	**79** / 38	**01** / 09	66 / 46
Car	**79** / 36	64 / 48	**62** / 48
Jogging	**84** / 29	**25** / 43	82 / 37
Helicopter	**70** / 43	**04** / 20	46 / 21
Plane51	**85** / 33	08 / 27	**04** / 17

According to the results, the proposed solution performs better in our dataset. That was expected, as the dataset contains some problematic parts that the other solutions do not solve well. This was also the main reason for the proposed solution.

In the standard dataset the quality performance of the proposed solution is comparable to other solutions. This is considered as a very good result; maybe even better than overperforming the original dataset. It also means that after the generalization to solve our dataset the common performance was not disrupted.

Table 3. Speed evaluation. The M means that MBT is applied, otherwise it is not. All values are means across all datasets.

	64	64/M	256	256/M/C	OpenTLD	CMT
fps	23.2	14.8	2.4	8.8	107.2	45.5

The computational speed is highly affected by an allowed maximal object movement and a number of backtracked points (usage of MBT). In table 3 are shown average values for few configuration with added average values of other trackers, to avoid dependency on machine performance. The values stand for frames-per-second (fps) and the proposed solutions is evaluated with 64*px* and 256*px* as a maximal object movement in all axis directions and by usage of MBT. It is good to mention that the movement by 64*px* is generally much bigger than is needed and much bigger than other solutions offer. The *C* symbolizes a partial CUDA acceleration.

5. Conclusions

In this paper a long-term tracker was introduced. The proposed solution is focused on some particular problems, like small object size and poor image contrast quality. Beside these specific problems, the tracker performance on standard dataset should not be decreased and the tracker should not lose its generality.

As the evaluation showed, the proposed solution overperformed state-of-the-art trackers in specific dataset. On the standard dataset, our tracker performed comparably to others.

The main contribution is the tracking quality measurement called *multiple bidirectional tracking*. The main idea of the method is to move tracking point quality measurement into the tracking process itself.

The system was built on *OpenTLD* background and therefore it took its structure; the tracker and the detector cooperation. The cooperation was tied closely by newly proposed *Detector-Tracker Fusion*.

The proposed solution is a general object tracker prototype generalized to solve the new problems and able to run in real-time, although it is slower then other solutions. Hence, it can be improved in any conceivable domain. Some of the first steps should be more sophisticated online learning system.

References

[1] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 2, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[2] Jianbo Shi and Carlo Tomasi. Good features to track. In *CVPR*, pages 593–600, 1994.

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[4] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. FREAK: Fast retina keypoint. In *CVPR*, pages 510–517. IEEE, 2012.

[5] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *ICCV*, pages 2564–2571. IEEE, 2011.

[6] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *ICCV*, pages 2548–2555. IEEE, 2011.

[7] Charles Bibby and Ian Reid. Real-time tracking of multiple occluding objects using level sets. In *CVPR*, pages 1307–1314. IEEE, 2010.

[8] Alper Yilmaz, Xin Li, and Mubarak Shah. Contour-based object tracking with occlusion handling in video acquired using mobile cameras. *TPAMI*, 26(11):1531–1536, 2004.

[9] Christoph H. Lampert, Matthew B. Blaschko, and Thomas Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*, pages 1–8. IEEE, 2008.

[10] Georg Nebehay. *Robust object tracking based on tracking-learning-detection*. Wien, 2012.

[11] David Forsyth and Jean Ponce. *a modern approach*. Pearson, Boston, 2 edition, 2012.

[12] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *TPAMI*, 34(7):1409–1422, 2012.

[13] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Representing shape with a spatial pyramid kernel. In *CIVR*, pages 401–408. ACM, 2007.

[14] Ondrej Chum and Andrew Zisserman. An exemplar model for learning object classes. In *CVPR*, pages 1–8. IEEE, 2007.

[15] Paul A. Brasnett, Lyudmila Mihaylova, Nishan Canagarajah, and David Bull. Particle filtering with multiple cues for object tracking in video sequences. In *Electronic Imaging*, pages 430–441. SPIE, 2005.

[16] Xin Sun, Hongxun Yao, and Shengping Zhang. Contour tracking via on-line discriminative appearance modeling based level sets. In *ICIP*, pages 2317–2320. IEEE, 2011.

[17] Jean-Yves Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5, 2001.

[18] Lu Zhang and LJP van der Maaten. Preserving structure in model-free tracking. *TPAMI*, 36(4):756–769, 2014.

[19] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *ICPR*, pages 2756–2759, 2010.

[20] Georg Nebehay and Roman Pflugfelder. Consensus-based matching and tracking of keypoints for object tracking. In *Winter Conference on Applications of Computer Vision*. IEEE, IEEE, 2014.

[21] Sam Hare, Amir Saffari, and Philip H. S. Torr. Struck: Structured output tracking with kernels. In *ICCV*, pages 263–270. IEEE, IEEE, 2011.

[22] Mustafa Ozuysal, Pascal Fua, and Vincent Lepetit. Fast keypoint recognition in ten lines of code. In *CVPR*, pages 1–8. IEEE, 2007.

[23] Mustafa Ozuysal, Michael Calonder, Vincent Lepetit, and Pascal Fua. Fast keypoint recognition using random ferns. *TPAMI*, 32(3):448–461, 2010.

[24] VOT2014. [online]. [cit. 2015-04-14].