

Tracking multiple targets on aerial videos



Miklós Homolya

Students' Scientific Conference

Supervisors:

Tamás Szirányi D.Sc., Csaba Benedek Ph.D.

Pázmány Péter Catholic University

Faculty of Information Technology

Budapest, 2011

Összefoglaló

Munkám során a MTA Számítástechnikai és Automatizálási Kutatóintézetével együttműködve egy olyan megoldás fejlesztésén dolgoztam, amely képes városi környezetben készült légi videofelvételeken a mozgó alakzatok (járművek) észlelésére, és a mozgás nyomvonalának meghatározására. A programnak célszerűen az Universität der Bundeswehr München pilóta nélküli repülőgépein kell üzemelnie.

A légi távfelügyelet régóta alkalmazott erdők, növénypopulációk és egyéb erőforrások megfigyelésére az iparban és a mezőgazdaságban, városokban a forgalom figyelésére és a rendezvények felügyeletére, valamint katonai alkalmazásokban. Sokáig a légi távfelügyelet főleg állóképek emberi vizsgálatára korlátozódott, manapság viszont a videofelvételek kerülnek túlsúlyba. A videofelvételek viszont olyan dinamikus eseményeket is rögzítenek, amelyek állóképen nem megfigyelhetők. A videók sokkal több adatot tartalmaznak, mint az állóképek, nehezebb bennük a keresés, valamint koncentrálni kell arra, hogy a fizikai eszköz a megfigyelés szempontjából érdekes eseményeket rögzítsen. E problémák megoldása intelligens támogatórendszerek kutatását és fejlesztését teszi szükségessé. A munka során elkészített rendszer alkalmazható például forgalommérésre és városi események távfelügyeletére. Fontos kihívás, hogy a videofelvételek nem rögzített kameraállásból, hanem mozgó platformról készülnek, így előre kiszámított statikus háttérkép nem lehet a segítségünkre. További fontos probléma, hogy egyszerre több célpontot is követni kell, valamint mérnöki szempontból jelentős, hogy az egész folyamatnak valós időben futnia kell a pilóta nélküli repülőgépen.

Az elkészült megoldást valódi és szintetikus felvételeken teszteltem, melyeken hatékony eredményeket produkált. Nagy hangsúlyt fektettem a számítási teljesítmény optimalizálására, aminek köszönhetően VGA felbontású videók 12-14 fps sebességgel feldolgozhatók. Ez a sebesség elegendő a nyomvonalak valós idejű rekonstrukciójához. A további tervek között szerepel a jelenlegi verzió helyes működését korlátozó tényezők csökkentése, valamint a pilóta nélküli repülőgépre való tényleges integráció.

Abstract

In cooperation with the Bundeswehr University Munich, at the Computer and Automation Research Institute of the Hungarian Academy of Sciences I have been working on the development of a solution, which is able to detect moving vehicles in aerial videos of urban scenes, and is able to reconstruct the track of these vehicles. The program is expediently required to operate on the unmanned aerial vehicles of the Bundeswehr University Munich.

Aerial surveillance has a long history for monitoring resources such as forests and crops, for monitoring traffic and crowd events in urban environment, and also in military applications. Aerial videos capture dynamic events which are not observable on still images. It is more difficult to search in videos, and more attention should be paid to identify interested objects and phenomena. To solve these problems, research and development efforts of intelligent systems are necessary.

My proposed solution may be used for traffic monitoring and for the surveillance of urban events. It is an important challenge, that the video is not captured from a static point of view, but from a moving platform, therefore we cannot use precalculated background images. In addition, an important engineering challenge is that the whole process should operate on the unmanned aerial vehicle in real time.

The proposed solution was tested using real and synthetic videos, and it produced promising results. Due to my excessive performance optimizations, the application can process videos of VGA resolution at 12-14 frame per seconds. This performance is adequate to reconstruct tracks in real time. Further plans include reducing the number of the factors which restrict the conditions of correct operation, and to actually integrate the program into the onboard system of unmanned aerial vehicles.

Contents

1	Introduction	1
1.1	Introduction to aerial surveillance	1
1.2	Brief description of the task	2
1.3	Major challenges	2
1.4	Antecedents	3
1.5	Requirements	3
1.6	Instruments	4
2	Proposed solution	5
2.1	Overview	5
2.2	Image registration	6
2.2.1	Known approaches	6
2.2.2	Implementation details	7
2.2.3	Accuracy boost	8
2.3	Foreground segmentation	9
2.3.1	Background model	10
2.3.2	Foreground extraction	12
2.3.3	Note on color space conversion	13
2.3.4	Implemenation details	14
	Warping frames	15
	Continuous matrices	16
	Integer arithmetic and low-throughput operations	16
2.4	Object detection	16
2.5	Multi-target tracking	19
2.5.1	Kalman filter	20
2.5.2	Assignment	21

3 Conclusion	24
3.1 Results	24
3.2 Further work	26
Acknowledgements	29
Bibliography	30

Chapter 1

Introduction

1.1 Introduction to aerial surveillance

Aerial surveillance has a long history in commercial world for monitoring resources such as forests and crops, for identifying illegal dumping, and in the military for observing enemy activities. Aerial images was also used for topographic mapping.

For a long time, aerial surveillance was restricted to analyzing high-resolution aerial still images. In the last decade, there is a growing interest in using video cameras for these and other task, because video captures dynamic events which cannot be observed on still images. However, this imposes several problems. First, video captures has much lower resolution. In order to see enough details to be able to identify the objects and phenomena of interest, often telephoto lens are used, with a narrow field of view. In this case, objects of interest move rapidly in and out of the field of view, frames lack larger visual context, therefore the observer has to pay constant attention. Second, video streams carry much more data to store, transmit, or process. These problems require intelligent systems which can automatize some or all of these task. A general framework for aerial video systems is described in [1].

Nowadays, unmanned aerial vehicles (UAVs) are becoming more and more important in both civil and military applications. They can be used to measure traffic density, for surveillance of crowd events, for pursuing suspicious targets, and for reconnaissance. Since there is no need to carry a pilot, UAVs are usually much smaller and lighter than pilot-driven airborne vehicles, thus their operation is more cost effective, and often requires less human attention.

One of these applications is tracking targets such as vehicles or people. Tracking may have several purposes, for example, to measure traffic density, to pursue a specific target,

or reconnaissance in the battlefield. Frameworks for this problem was created by [2] and [3]. The latter is a direct predecessor of [4], which my work is mainly based upon (see section 1.4 for details).

A similar task but with different technical requirements is targeted in [5]. Here the cameras capture high-resolution, low frame rate images of very large area. Thus in this case, the vehicles are extremely small in term of pixels, and the method must handle temporal sampling rate as low as 2 Hz.

An approach for handling parallax errors and extracting semantic layers from low altitude aerial videos is described in [6].

1.2 Brief description of the task

The task is to detect and track moving objects on video streams captured by airborne vehicles. Requirements include reconstruction of the track of moving objects to the extent possible from the given video stream.

The task is part of the MEDUSA (Multi Sensor Fusion Grid for Urban Situational Awareness) project supported by the European Defence Agency. The project goal is the integration of multi-sensor information for urban surveillance and improving crime prevention. My contributions are made possible through the Distributed Events Analysis Research Group of the Computer and Automation Research Institute of the Hungarian Academy of Sciences.

Given the above goals, the solution will operate on a UAV, which captures urban scene, and the goal is to detect and track moving vehicles in the traffic. Bundeswehr University Munich – another participant of the MEDUSA project – possesses several UAVs and other aerial vehicles, and also provides real and synthetic video streams for testing purposes.

1.3 Major challenges

Major challenges of this task stem from several sources:

- *Video stream is captured from a moving platform*, not from a fixed viewpoint. Therefore, the background image, which does not contain any moving object, cannot be precalculated. The background image must be estimated simultaneously with the foreground segmentation. Additionally, when the viewpoint is moving fast, all this has to be done from quite few samples.

- *Not just a single, but multiple targets* – every visible moving vehicle – are tracked. Thus, it is important part of the task to associate the detections of individual frames between the frames, to be able to reconstruct the track of each vehicle.
- *Objective for real-time operation.* The software has to operate real-time on the board computer of the UAV. Targeted UAVs have neither GPU nor other high-performance, parallel processing unit. Only a CPU is available.

1.4 Antecedents

For a similar multi-target tracking task, Mátyus [4] has worked out a solution earlier in the Distributed Events Analysis Research Group. However, his work was intended to process video stream taken by a camera attached on a ballon. Therefore, that solution is able to handle the tremor of camera, but is unsuitable for the case, when the camera goes far from its position. The implementation's performance was also deficient in some cases.

The series of steps, which outline the structure of the solution, were based on this previous work. Implementation was made suitable for any camera movement, implementation's performance was significantly improved, and there was also improvements on the algorithm to yield somewhat better results. Differences and improvements are detailed at the appropriate places later in this document.

1.5 Requirements

First, the solution is required to process aerial videos of urban scenes. The camera will provide approximately 640×480 resolution frames. Due to hardware capabilities and restrictions of the software framework, no more than 10 frames will be given to my application per one second.

The area covered by a single frame will be around around 3800 m^2 , therefore an average car will occupy around 40×20 pixels, but the solution should properly handle vehicles which are about twice as long and also wider.

The unmanned aerial vehicle can fly as fast as 90 km/h , while vehicles are not expected to go faster than 70 km/h , because the speed limit in residential areas is 50 km/h in Europe. The unmanned aerial vehicle also flies at high altitude, but its camera has narrow viewing angle, therefore although parallax effects occur, they are not severe.

1.6 Instruments

During the programming work I have been heavily relied on OpenCV (Open Source Computer Vision) library. OpenCV provides data structures for points, pixels, matrices and images, and a broad range of highly-optimized algorithms for real-time computer vision. It has C++, C and Python interfaces. The solution was implemented in C++ programming language, however, occasionally I have chosen to call the C version of a function due to performance reasons. I have been using OpenCV version 2.3.0.

OpenCV was built with support for Intel® Integrated Performance Primitives (Intel IPP) and Intel® Threading Building Blocks (Intel TBB). Without Intel TBB, OpenCV cannot exploit multiple cores. Intel IPP was also called directly for color space conversion.

Since real-time operation is one of our design goals, Intel® VTune™ Amplifier XE was used for profiling. It shows where execution time is spent, and provides information to identify the potential causes of poor performance.

Most measurements and testing were run on Intel® Core™ i5-520M processor. Therefore, measurement quantities related to performance (e.g. per frame execution time) are given for this processor unless explicitly specified otherwise.

Chapter 2

Proposed solution

2.1 Overview

First, we outline the major steps of the proposed solution. The flowchart on Figure 2.1 represents the general data flow among these steps. Brief description of these steps are:

Image registration

Image registration is the process of transforming two or more images into the same coordinate system. In our case, it is used to estimate the ego-motion of the camera.

Foreground segmentation

Foreground segmentation is responsible for differentiating between pixels belonging to moving objects or stationary background. This step provides a foreground mask or distance image which is used for object detection.

Blob detection

The goal of object detection is to identify and locate moving objects on a frame,

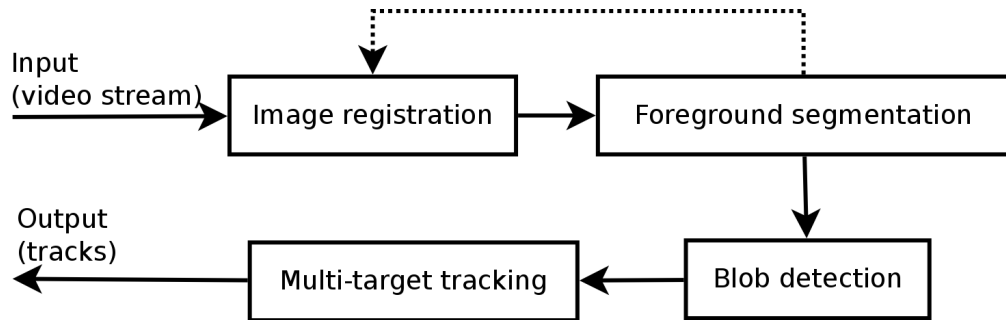


FIGURE 2.1 General outline and data flow diagram of the proposed solution. Full line represents the main flow of data, while dashed line marks feedback loops.

given the result of foreground segmentation. However, at this stage there is no link between the detections of different frames.

Multi-target tracking

This is the final step where tracks are determined. In order to reconstruct the track of moving objects, one has to assign detections from previous step across the frames.

Each of these steps are discussed in more detail in the following sections.

2.2 Image registration

Image registration is the process of transforming two or more images into the same coordinate system. In our case, it is used to estimate the ego-motion of the camera.

In our model, we assume, that the camera is observing a planar surface from above. This is a good approximation if the height of ground objects is significantly less than the altitude of the aerial vehicle carrying the camera. If this condition is not satisfied, we may expect parallax errors in the registration procedure.

In this model, the ego-motion of the camera may be described by the perspective transformation between the two images seen by the camera from different positions. The perspective transformation can be described by the homography matrix H . One point p_0 is transformed into p_1 by the following equation:

$$\begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \quad (2.1)$$

where p_0 and p_1 are given above in homogeneous coordinates. The pixel coordinates are:

$$p_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (2.2)$$

$$p_1 = \begin{bmatrix} x_1/w_1 \\ y_1/w_1 \end{bmatrix} \quad (2.3)$$

Formally, the task of image registration is to determine the homography matrix H , given two consecutive image frames from the video sequence.

2.2.1 Known approaches

Image registration algorithms are classified into two major categories: intensity-based, and feature-based algorithms.

Intensity-based approaches define an error metric based on the pixel intensity differences in the aligned images. Then the transformation which warps one image into the coordinate system of the other is searched. This method needs a computationally suitable searching technique, e.g. in the Fourier domain.

This approach can handle blurred images, where feature point detection fails or gives inadequate results. On the other hand, this method is mostly suitable for specific transformations (e.g. similarity [7], affine [8]), but it can hardly handle general homography transformations.

Feature-based techniques find feature points on the images first, then they try to find the homography transformation which aligns the feature points of the two images with each other. Popular feature point detectors are, for example, the Harris corner detector [9], the SIFT detector [10], and the SURF detector [11].

We are using a feature-based technique due to the requirement to be able handle general perspective transformations, and because of performance reasons.

2.2.2 Implementation details

Feature point detection is done using OpenCV's `cv::goodFeaturesToTrack` function with Harris corner detector. According to [4], "the Harris corner detector is more suitable in man-made environments where corners are abundant". Harris corners are also found suitable in aerial images [12]. An example of Harris corner detection is shown on Figure 2.2.

When the camera movement is slow compared to the temporal sampling rate, the displacement of the feature points between two consecutive frames are often in the range of the pixel size. To achieve accurate registration, `cv::cornerSubPix` function is used to locate corners with subpixel accuracy.

The displacement of feature points between the frames are calculated by sparse optical flow using iterative Lucas-Kanade method with pyramids. For this purpose, we use the `cvCalcOpticalFlowPyrLK` function, which is part of OpenCV's C language interface. My measurements showed that it was 4.9 times faster than its C++ counterpart on a hyper-threading capable dual-core CPU, because of the ability to exploit multiple cores and being better optimized overall.

Finally, the homography matrix is calculated by `cv::findHomography`, which implements RANSAC method [13] for fitting the transformation to the feature points. Outlier-classified optical flow vectors of feature points are shown on Figure 2.3.



FIGURE 2.2 Feature points of a grayscale image found by Harris corner detector.

2.2.3 Accuracy boost

Homography matrix estimation is, of course, not perfect. This is partly due to the inaccuracies in feature localization and calculation errors, and partly because the territory is not planar, thus parallax errors occur. The estimation error between consecutive frames of the video sequence can be sufficiently low. However, when estimation is done on consecutive pairs of frames, then the estimation errors of homography matrices are independent. This causes estimation error to accumulate over time, which results in blurry background image as demonstrated on Figure 2.4.

To reduce the accumulation of registration errors, I have chosen to do the registration between the current frame and the constructed background image, rather than between the current and previous frames. Such an example is presented on Figure 2.5. In this case, the individual errors of the homography matrices tend to cancel each other out, especially if the camera movement is stationary. This quality improvement is reduced, if new frames refresh the background image too quickly.

`cvCalcOpticalFlowPyrLK` requires its input images to have equal size. Due to this implementation restriction, the part of the background image, which overlaps with the previous frame, is extracted and used during image registration in place of the actual previous frame. Other aspects of the registration process are intact.

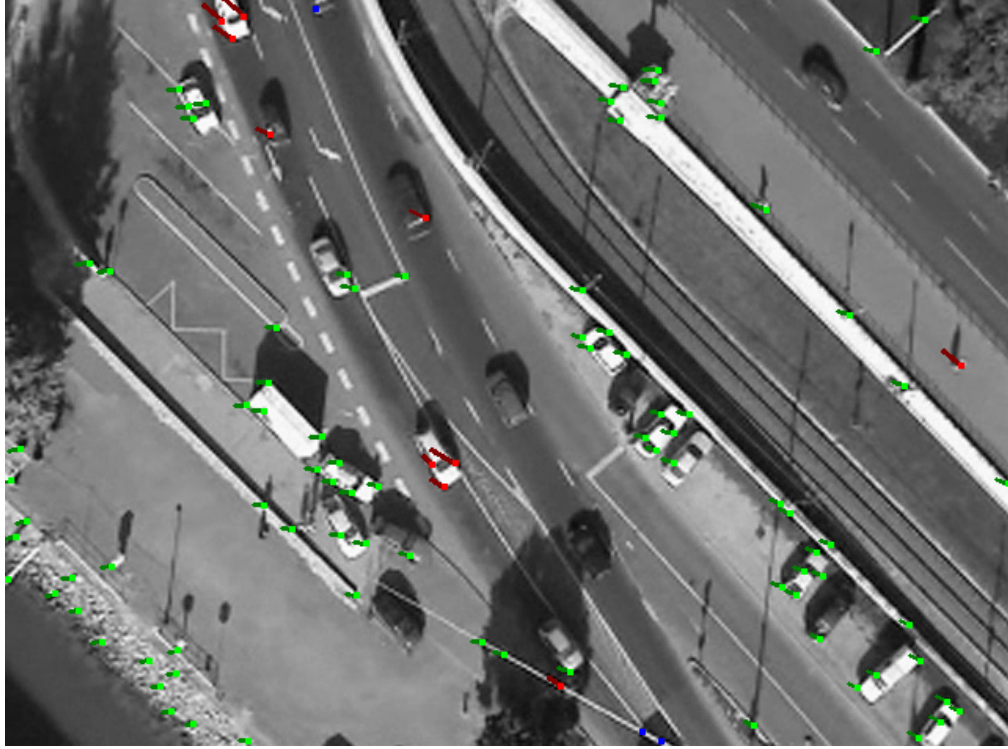


FIGURE 2.3 Optical flow vectors of the feature points. Legend: *blue* – optical flow calculation fails; *red* – considered outlier by RANSAC; *green* – takes part in homography matrix estimation.

This approach has additional benefits. The highest quality feature points are often those which belong to moving vehicles. These points are desired to be ruled out from homography matrix estimation, because they most likely will increase the uncertainty of the estimation. In the background image, moving vehicles are either not present or massively blurred, which causes optical flow calculation to fail on the feature points of moving objects. This makes RANSAC algorithm’s job easier to choose appropriate feature points to find an accurate estimate of the homography matrix.

This improvement allows to use less feature points, but still archieving adequate registration accuracy, which is beneficial to performance, because subpixel corner localization and optical flow calculation forms significant part of the execution time.

2.3 Foreground segmentation

Foreground segmentation is responsible for differentiating between pixels belonging to moving objects or stationary background.

In our case, camera often acquires images with such a high frequency, that the displacement of moving objects between consecutive frames is too low to reliably detect



FIGURE 2.4 Average of 100 frames when image registration uses only pairs of consecutive frames. Compare with Figure 2.5.

moving objects by using just two frames. For this reason, we construct a background image from the video sequence, and then foreground segmentation is done by examining the difference between the current frame and the background image.

2.3.1 Background model

Nowadays, the mixture of Gaussian approach [14] is widely used for background image synthesis. This method provides very high quality background images, but it cannot be applied in our case, since due to the fast camera motion we often cannot observe enough samples to be able to describe the distribution by mixture of Gaussian.

[4] uses exponential running average, where the mean value $\bar{x}_n$, and the variance σ_n^2 are calculated for every pixel on-line:

$$\bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n \quad (2.4)$$

$$\sigma_n^2 = (1 - \alpha) \cdot \sigma_{n-1}^2 + \alpha \cdot (x_n - \bar{x}_n)(x_n - \bar{x}_{n-1}) \quad (2.5)$$

where α is an arbitrary constant that gives the refresh rate.

However, I have ceased to use this model; reasons and consequences are detailed later



FIGURE 2.5 Average of 100 frames when frames are registered against the background image. Compare with Figure 2.4.

in this section. I use cumulative running average where

$$\bar{x}_n = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.6)$$

but I use on-line calculation as follows:

$$\bar{x}_n = \bar{x}_{n-1} + \frac{1}{n} \cdot (x_n - \bar{x}_{n-1}) \quad (2.7)$$

This model requires to store sample count for all background pixels. In my implementation, sample count n does not grow over an arbitrary chosen N constant parameter. Thus, when n reaches N , (2.7) becomes equivalent to (2.4) with $\alpha = 1/N$.

With exponential running average, it is difficult to choose a good α parameter, due to the following reasons:

- *Small α values* give too much weight to the first sample. It takes considerably long time to wash out the moving objects present on the first frame. Figure 2.6 demonstrates the difference.
- *Large α values* give much weight to the current frame. This causes moving objects to distort the background image even if there are lots of samples available. Figure 2.7 demonstrates this effect. Additionally, high refresh rates tend to neutralize my improvement on the accuracy of image registration, detailed in section 2.2.3.

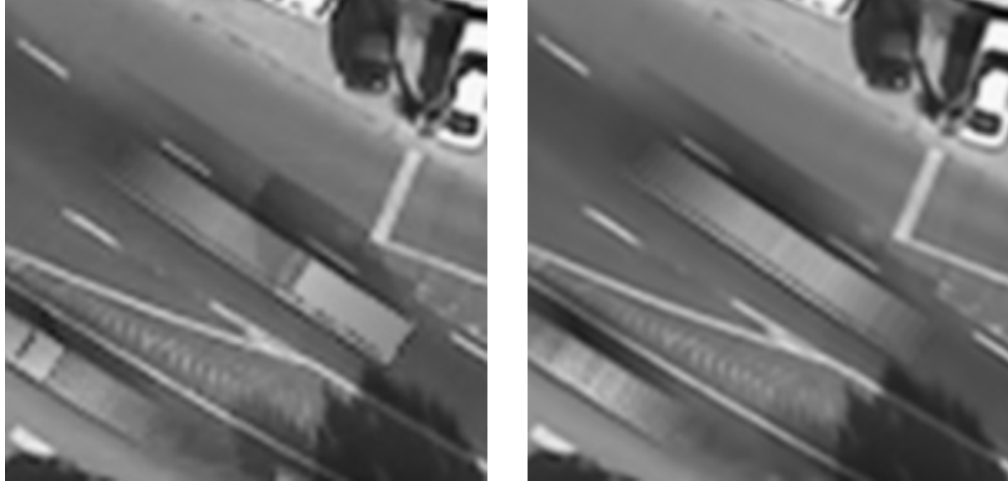


FIGURE 2.6 Magnified portion of the background image constructed from the first 20 frames of a video sequence. Left side: exponential running average with $\alpha = 0.05$; right side: cumulative average. Notice that on the left the vehicle is still clearly seen on spot where it was on the first frame.

Cumulative running average gives equal weight to all samples until sample count saturates, which allows moving objects to be washed out fast from the background image. When saturation occurred, further samples contribute to the background with a low refresh rate, thus eliminating the above problems.

2.3.2 Foreground extraction

[4] uses Mahalanobis distance to get a distance image between the current frame and the background. I simply use Euclidean distance to yield a distance image (see Figure 2.8). The essential difference is that Mahalanobis distance uses variance to weight the difference. With Euclidean distance, no variance is used, and its calculation is unnecessary. The former approach has the following disadvantages:

- *Variance tends to become very high* on roads with heavy traffic. High variance virtually prevents foreground detection; in some cases, only the front part of objects is classified as foreground. One may try not to refresh the background where pixels are classified as foreground, but in practice, implementing such a feedback would make the algorithm very fragile.
- *Variance cannot be precisely estimated* from low sample count. Fluctuation in estimated variance also causes fluctuation in the distance, making the distance image virtually useless until there are more than 10 samples available in that area. When

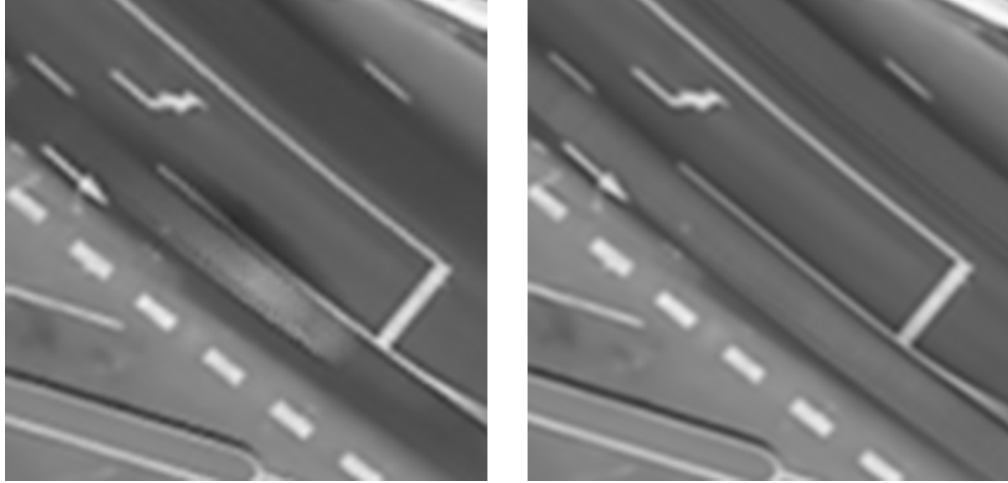


FIGURE 2.7 Another magnified portion of the background image constructed from the first 200 frames of the same video sequence. Left side: exponential running average also with $\alpha = 0.05$; right side: cumulative average. Notice that human spectator can identify the most recent position of the vehicle on the left image. On the right image, only the tracks where vehicles usually run, are identifiable.

applying Euclidean distance, distance image may be usable for object detection as early as after 3-5 frames.

- *Additional computational cost* is associated to variance calculation. Foreground segmentation may run faster if less things need to be done. Every saving is appreciated when real-time operation is a design goal.

As one may see on Figure 2.8, edges are often visible on the distance image due to imperfect registration. Since they are thin and long lines, while moving vehicles are large blobs, the object detector may distinguish them later.

In [4] the result of foreground segmentation was a binary mask obtained by thresholding the Gaussian blurred distance image. I, however, give the distance image itself to the object detector, leaving any preproccession up to the next stage, thus eliminating unnecessary threshold parameters.

2.3.3 Note on color space conversion

[4] used to synthesize the background image, and produce foreground mask in the CIE $L^*u^*v^*$ color space. This color space resembles human vision, and aims to be perceptually uniform. However, according to my measurements, the conversion itself takes about 20ms on 640x480 frames, which makes the CIE $L^*u^*v^*$ color space clearly

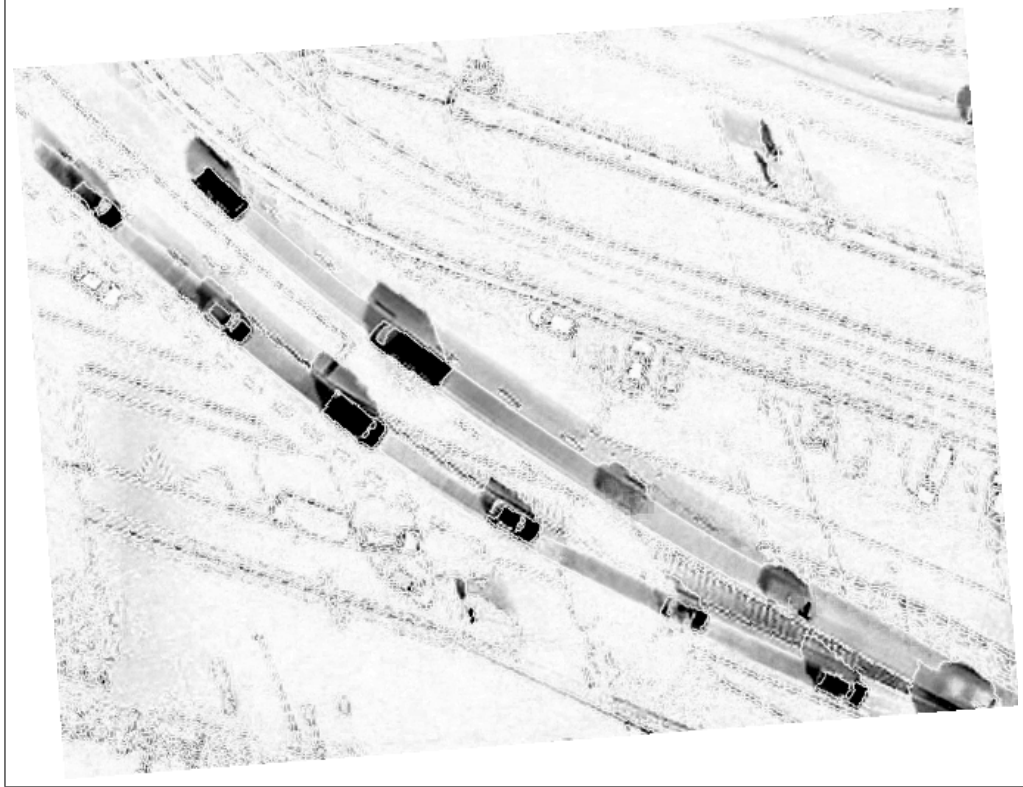


FIGURE 2.8 Euclidean distance image between the current frame and the on-line constructed background image. Darkness indicates the extent of distance. Notice, that due to registration, the image appears in a rotated rectangle.

unsuitable given our performance requirements. Conversion from RGB to CIE $L^*u^*v^*$ involves slow operations like cube root calculation, which is the most probable cause of its performance deficiency.

Therefore, solely because of performance reasons, I have chosen to use the YC_oC_g color space instead of CIE $L^*u^*v^*$. Its only advantage, besides separating luminance and chrominance coordinates, is that conversion from RGB is blazingly fast. I use the following equations:

$$Y = R/4 + G/2 + B/4$$

$$C_o = R/2 - B/2$$

$$C_g = -R/4 + G/2 - B/4$$

2.3.4 Implementation details

Most things about foreground segmentation were implemented by writing the code myself, external library functions were rarely used, because specifically written code might have more fine-grained optimizations. In the following, I detail those important consid-

erations which make my implementation run fast.

Warping frames

After the perspective transformation is determined during the stage of image registration, every frame is warped into a common coordinate system to use it as new samples of those pixels of the background image which are covered by the warped frame.

Image warping is done using OpenCV's `cv::warpPerspective` function. Despite being wisely optimized, this function is alone responsible for significant part of the whole execution time. This comes from the computationally complex nature of image warping, which mostly stems from two sources:

- *Image warping is not cache-friendly* with the usual memory layout of rectangular images. Pixels of a row in the destination image almost always span many rows in the source image, which may cause a high number of cache-misses. OpenCV handles this problem by partitioning the destination image into small quadratic blocks, and run on block after block. This makes the algorithm run moderately slow regardless of the actual perspective transformation.
- *Slow operations* are run for each pixel. Due to the perspective nature of the transformation, a division has to be executed for every pixel of the destination image. (It is not the case for affine transformations.) Division has not only long-latency, but modern CPUs usually cannot execute more division operations at once (except when using vector instructions), thus it also has low throughput.

Because of the above detailed reasons, the most beneficial optimization is to reduce the number of `cv::warpPerspective` calls to the minimum necessary. I have managed to use only two of them for most frames. One is obviously necessary, the other call does the inverse transformation from the background to be used next time at image registration as “previous” frame.

A naive implementation may use a third call to warp a totally white image, and use the warped one as a mask where the new pixel samples are. In my work, this was eliminated by telling `cv::warpPerspective` that the plane outside the source image is filled with a value which is not actually valid in YC_oC_g . This allows to reconstruct the mask from the result.

Continuous matrices

Implementation synthesizes a background image several times larger the frame size of the video sequence itself. This serves a purpose to prevent the background image being warped too often, which could lead to degradation in quality like blur.

The running time of most operations during foreground segmentation is proportional only to the size of the image which we actually operate on. In order to keep the running time moderate, region of interest for these operations is set to the bounding rectangle of the warped current frame. However this makes the data, which are subject to operation, non-continuous, which makes loop vectorization much difficult.

To ease automatic vectorization, the interesting region of the background image is copied to a continuous place in memory, all operations are executed, and the updated background region is copied back to the appropriate region of the of the background image. Applying the appropriate compiler option, the compiler justifies that it was able to vectorize the loops.

Integer arithmetic and low-throughput operations

For background synthesis, (2.7) is calculated using integer arithmetic capabilities of the CPU. Using floating-point arithmetic would be less circumstantial, but floating-point numbers are at least 32 bits wide, while 16-bit intensity level resolution is more than satisfactory.

Operand width becomes crucial when using vector instructions, like MMX, SSE or AVX instructions. For example, SSE (Streaming SIMD Extensions) instructions can operate on 128 bits at once. This can mean, for example, 4 single precision floating-point numbers, or 8 16-bit integers. Therefore, using integer arithmetic in this case allows to approximately double the procession speed.

(2.7) requires to divide by n . Division, as stated above, is a low-throughput operation. Because of the saturation of sample count, the possible values of n are very limited. Therefore, we can store the precomputed reciprocal of possible values of n , and use a lookup table and multiplication to avoid division.

2.4 Object detection

The goal of object detection is to identify and locate moving objects on a frame, given the result of foreground segmentation. As already stated about foreground extraction, I use

the distance image for object detection. Examples in this section use the distance image shown on Figure 2.9. In contrast to object detection in [4], two functional differences apply in my version:

- *Binary mask* was given to previous version as input.
- *Size detection* was a limited capability of previous version. Since the variation in the size of vehicles is pretty limited, detecting merely the position is satisfactory if large or long vehicles do not cause multiple detections.

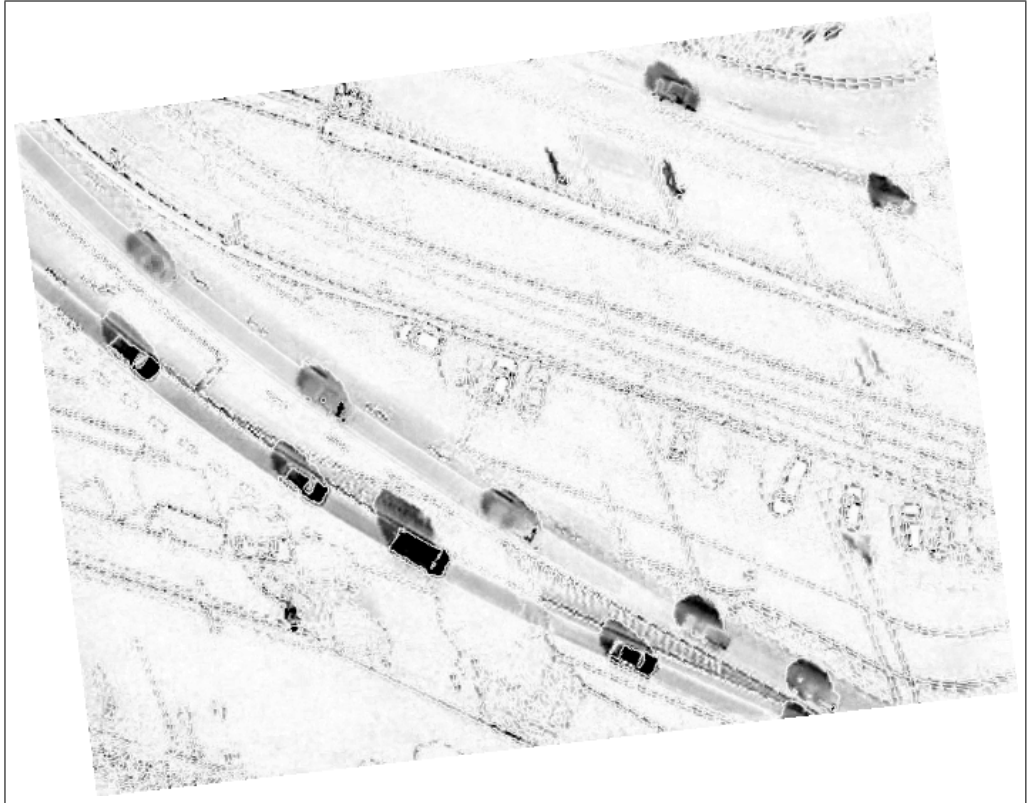


FIGURE 2.9 Euclidean distance image, which gives the input of object detection. Darkness indicates the extent of distance. Notice, that due to registration, the image appears in a rotated rectangle.

Given the above functional differences, other differences to the previous detection method are pure performance improvements. A brief summary of the old method follows.

The foreground mask was parcelled into disjunctive rectangles of the same size. The size of the rectangles was chosen to be approximately the size of the vehicles. Then coverage ratio was calculated for all rectangles, and rectangles whose coverage ratio was below a certain threshold would be discarded. Then these two kinds of operation were repeated iteratively until convergence was achieved: *a) shift* – shifts the center of

rectangles to the mass center of its content; *b) merge* – merges rectangles where overlap ratio is above a given threshold.

There are several issues to note in order to improve performance:

- *‘Shift’ operation actually implements hill climbing*, which converges into a local maximum. Therefore, ‘shift’ operation can be replaced with any method for local maxima search.
- *Recounting foreground pixels in rectangles* after every series of operations is a great luxury. Coverage ratio for *any* rectangle on the foreground mask can be answered in constant time, once we have the integral image which can be calculated in time proportional to the size of the image. This fact makes meaningless to start from disjunctive rectangles of a given size, we may run local maxima search starting from *all* rectangles of that given size.
- *‘Merge’ operation serves for eliminating local maxima found multiple times*, or local maxima *too close* to each other – using the terms from the above discussion.

Considering the above remarks, I propose the following method for object detection:

1. Calculate *average intensity* (corresponds to *coverage ratio*) for all rectangles of a given size. Calculation is done fast using integral image. The average intensities are represented as a $(W - w + 1) \times (H - h + 1)$ size image as shown on Figure 2.10, where the distance image has size $W \times H$ and the rectangle size is $w \times h$.
2. Run *grayscale dilation* on that image either once with a large structuring element, or many times with a smaller one. Example is shown on Figure 2.10. This serves for “local” maxima search. The effective structuring element defines the neighborhood which is considered “local”.
3. *Compare* average intensity image with its dilated version. Equality occurs at those spots, where the average intensity of neighboring spots is less than or equal to the spot considered. These spots include “local” maxima and plateaus. However, due to measurement noise, equality is extremely rare in real video frames, thus my implementation simply ignores the fact, that plateaus may appear. “Local” maxima are shown on Figure 2.11.
4. *Thresholding*. “Local” maxima locate the moving objects well, but they are also found on vast territories without any moving object, due to the apparently present

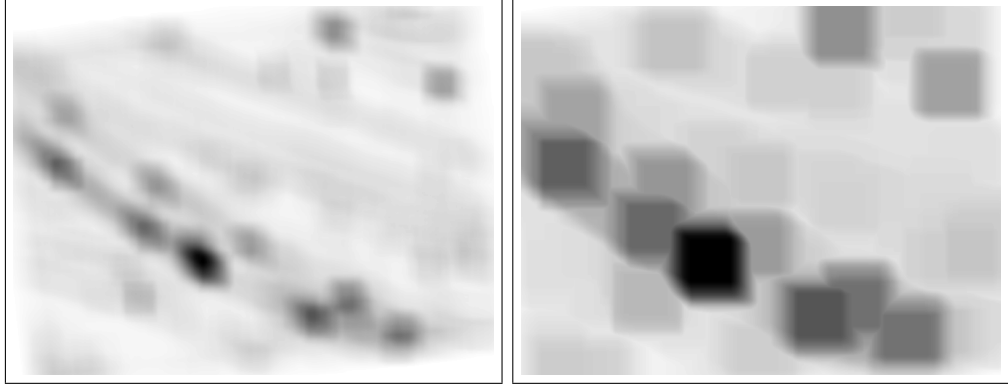


FIGURE 2.10 On the left: average intensity in rectangular areas of the image shown on Figure 2.9. On the right: grayscale dilated version of the left image.

noise. The average intensity of “local” maxima is a good basis for discrimination of real and fake detections. Thresholded result is shown by colors on Figure 2.11.

This method provides fast moving object detection, while the results are very similar to those provided by [4].

2.5 Multi-target tracking

The object detection step processes each frame independently. In order to reconstruct the track of moving objects, one has to assign detections across the frames. The difficulty is here that the number of objects on consecutive frames can vary even in case of perfect detection. Objects may enter or leave the scene, others may be temporarily occluded by occluded by buildings or overpasses. To handle disappearing and later reappearing objects, Kalman filtering is used.

Basic outline of the proposed tracking solution is illustrated on Figure 2.12. We call the detections of a given frame *measurements*; they constitute the input of the tracking algorithm. First, these measurements are assigned to tracked *objects*. The number of measurements and objects may be different: some objects can be undetected or invisible on a frame, and some measurements might be false detections or objects entering to the scene at the moment. Thus the assignment must also handle the creation of new objects, and removal of those which has not been seen for a while.

The motion state of the objects are maintained using Kalman filters. After assignment, the motion states are corrected by the measurements. Kalman filter can predict future position of objects, which is useful when assigning future measurements with objects.

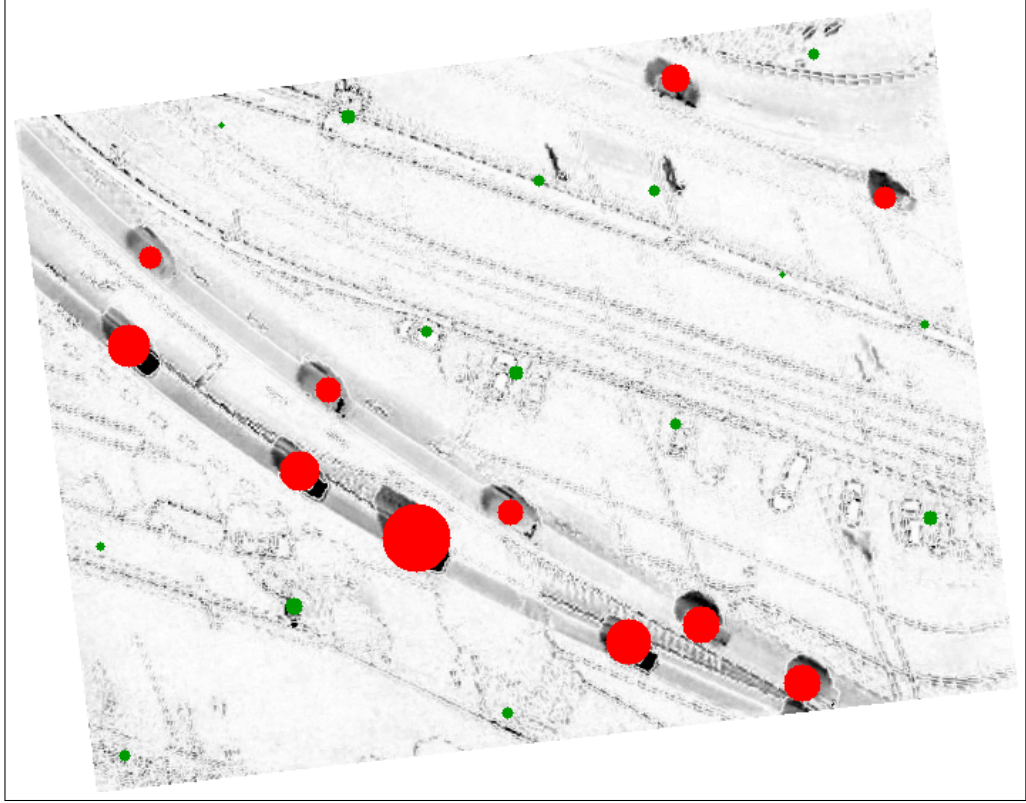


FIGURE 2.11 Result of moving object detection superimposed on the Euclidean distance image as shown on Figure 2.9. Spheres represent “local” maxima of average intensity; their radius represents the value of that average intensity. Green spheres are probably erroneous detections discarded by thresholding. Red spheres constitute the actual output.

2.5.1 Kalman filter

As stated earlier, Kalman filter is used to maintain the motion state of a moving objects. The state consists of the objects’ position and velocity, while acceleration is modelled by state noise. The equations describing the time evolution of the state and the measurement model are, in respective order:

$$x_k = \begin{bmatrix} \vec{r}_k \\ \vec{v}_k \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{r}_{k-1} \\ \vec{v}_{k-1} \end{bmatrix} + w_{k-1} \quad (2.8)$$

$$z_k = \begin{bmatrix} 1 & 0 \end{bmatrix} x_k + v_k \quad (2.9)$$

where $\vec{r}_k$ is the position, and $\vec{v}_k$ is the velocity of the object at time k . The state noise, w_{k-1} handles the case of acceleration. z_k is also a vector quantity as $\vec{r}_k$, and represents the measured position of the object. v_k is the measurement noise.

By applying Kalman filter, predictions help us to do better job at assignment, and additionally predictions allow to handle cases where the vehicle is occluded or detection

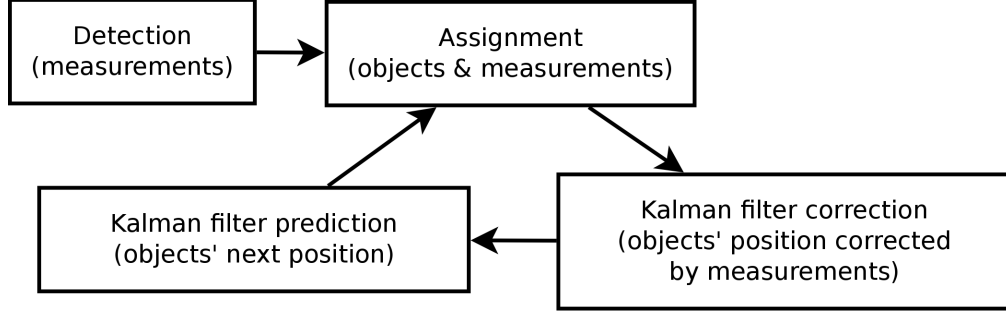


FIGURE 2.12 Block diagram representing the stages of the tracking method.

simply fails.

2.5.2 Assignment

Assignment is the key point of tracking. This is the operation where new detections are added to tracks. For each pair of tracked object O_i and detection D_j a score S_{ij} is calculated based on the Euclidean distance of the predicted and detected positions, as follows:

$$S_{ij} = \frac{1}{d(O_i, D_j)} \quad (2.10)$$

Score function in [4] also took into consideration the distance between the histogram of the object and the detection. However, since my implementation runs significantly faster than [4], less time elapses between consecutively preprocessed frames, thus the per frame displacement of objects is smaller, which results in that histogram comparison provides no real benefit.

Given all S_{ij} values, the assignment with maximal score can be determined by Hungarian method [15]. However, this method requires $O(n^3)$ time, thus we have chosen to use a greedy method which is not optimal in general, but gives good results for our assignment problem, and requires no more than $O(n^2)$ time.

Before we go into the details of assignment, first we have to list the informations stored about every object and their state. The informations stored about objects are: a) the position of first detection; b) last time it was seen. Objects may have three states: a) *active* if the tracker module verified that it is actually moving; b) *stale* if it has not been seen for a while; c) *inactive* until it turns active (or stale). Objects are immediately removed from the set of tracked objects when they become stale. The concept of activation was introduced to rule out a portion of incorrect detections, like occasionally appearing false detections. Furthermore, areas of background which are rich in edges are likely to cause false detections due to registration errors. These false

detections, although they may frequently occur, look like standing objects, thus the tracker may rule them out.

The greedy nature of the algorithm means that whenever we have chosen a detection or an object to assign, we simply choose its “best pair” which is not yet taken. We chose the locally optimal choice, and previous choices are never revised. The order in which we find pairs for objects and detections:

1. Activated objects, which were visible last time, always come first.
2. Remaining objects *or* remaining detections are matched.
 - Remaining objects if there are less objects than detections.
 - Remaining detections if there are less detections than objects.

New objects are created for unmatched detections. Unmatched objects, however, are kept until they become stale.

Tracking is demonstrated on Figure 2.13 and on Figure 2.14.

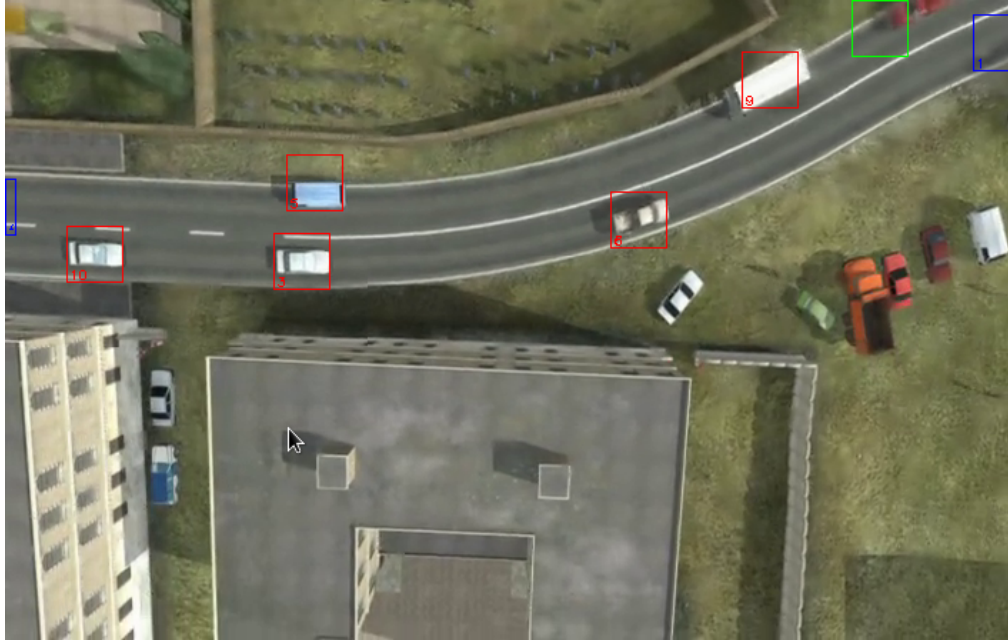


FIGURE 2.13 Tracking result with object states illustrated by colors. Green square represents a “not yet active” object which has just entered to the scene. Red and blue squares represent active objects. Red ones are visible at the moment, while blue ones not (they have just left the scene). The objects’ unique identification number is in the bottom left corner of squares. This frame is from a synthetic video sequence.

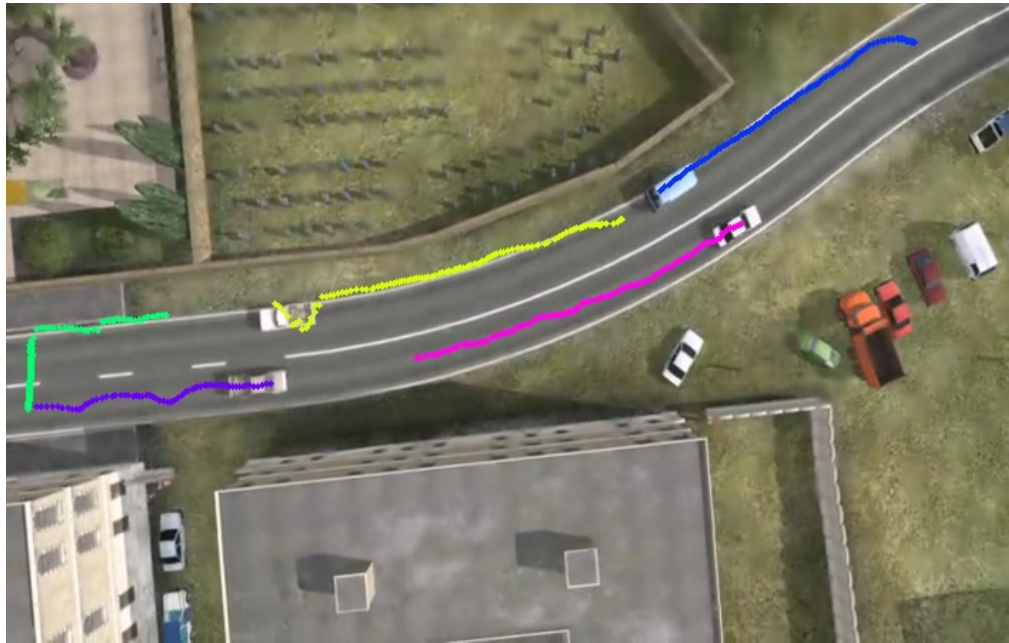


FIGURE 2.14 Detected tracks of vehicles. Different tracks are assigned different hue. This frame is from a synthetic video sequence.

Chapter 3

Conclusion

3.1 Results

I received three real aerial video streams and four synthetic ones for testing. The synthetic video streams were produced by the Bundeswehr University Munich using Virtual Battlespace Simulator 2 [16, 17], a professional tool. The length of these video streams ranges from 12 second to 3 minutes, and they contain 8 to 30 moving vehicles. Two video streams were not urban scene videos. Sometimes long vehicles, like trucks, also appear on these videos.

Synthetic video sequences are invaluable for verification purposes, because the tracks set up to drive the cars in the simulation are the ground truth solutions of our task to reconstruct tracks for videos. This allows to precisely measure the accuracy of our proposed solution. Testing on real urban scenes verifies that there are no noticeable difference between real and synthetic scenes regards to accuracy and performance.

The following measures were used to assess the quality of the results:

- **PDR** – Ratio of number of detections *vs* all the possible locations during the ground truth presence of an object (i.e. if A is present on N frames and detected for M times, M/N).
- **P** – Ratio of good detections *vs.* all detections (false positives included).
- **R** – Ratio of good detections *vs.* all possible detections (false negatives included).

Results of measurements are found in the following table:

	PDR	P	R
Real	84%	77%	84%
Synthetic	90%	93%	85%

Faults mostly belong to the following categories:

- occasional false detection in areas where no moving object present (see Figure 3.1);
- different vehicles got a shared track as if they were the same, creating in a confused track (see Figure 3.1);
- a moving vehicle gets lost and found, interrupted track (see Figure 3.2);
- failure to detect a given vehicle (see Figure 3.3);
- in rare cases, a vehicle may be detected twice (see Figure 3.4).



FIGURE 3.1 *Detected tracks with different hue.* Erroneous detection in cyan is seen at the bottom left corner. Confused track is visible in light green at the bottom right corner. This example is from a real urban scene.

Speed requirements of the task are well satisfied. Due to my performance improvements running time is reduced to 70–80 *ms* per frame at resolution 640×480 . This means about 12–14 FPS, which is confidently more than the 10 frames maximally provided in one second. In contrast, implementation in [4] runs 5 FPS at the same resolution.

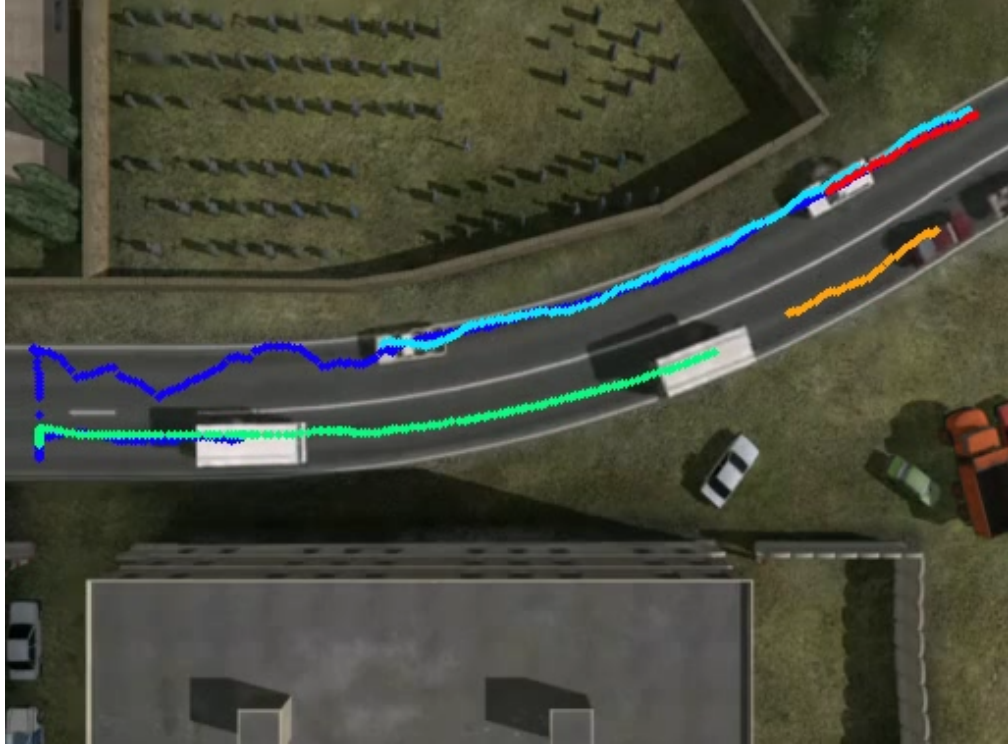


FIGURE 3.2 *Detected tracks with different hue.* The blue track is the falsely linked track of two different cars. The red car on the right side has a broken track in orange. This example is from a synthetic scene.

3.2 Further work

I have made significant effort to improve the performance of the application in order to achieve real-time operation. Although the performance of the current implementation is satisfactory for now, there is still room for improvement if larger spatial or temporal resolution will be provided in the future. Hints for further performance improvements:

- *Eliminate optical flow.* Optical flow calculation is largest time consumer of the whole process. To decrease running time, it is worth to try matching feature points of consecutive frames by feature descriptors.
- *Vectorization of `cv::warpPerspective`.* This function is another major time consumer. It may be worth investigating whether performance can be improved by using vector instructions, namely `divps` (packed single precision floating-point division).
- *Optimize integral image calculation.* Although the speed of object detection is significantly improved since [4] by applying algorithms with less time complexity, current implementation does not exploit the instruction-level parallelism offered by

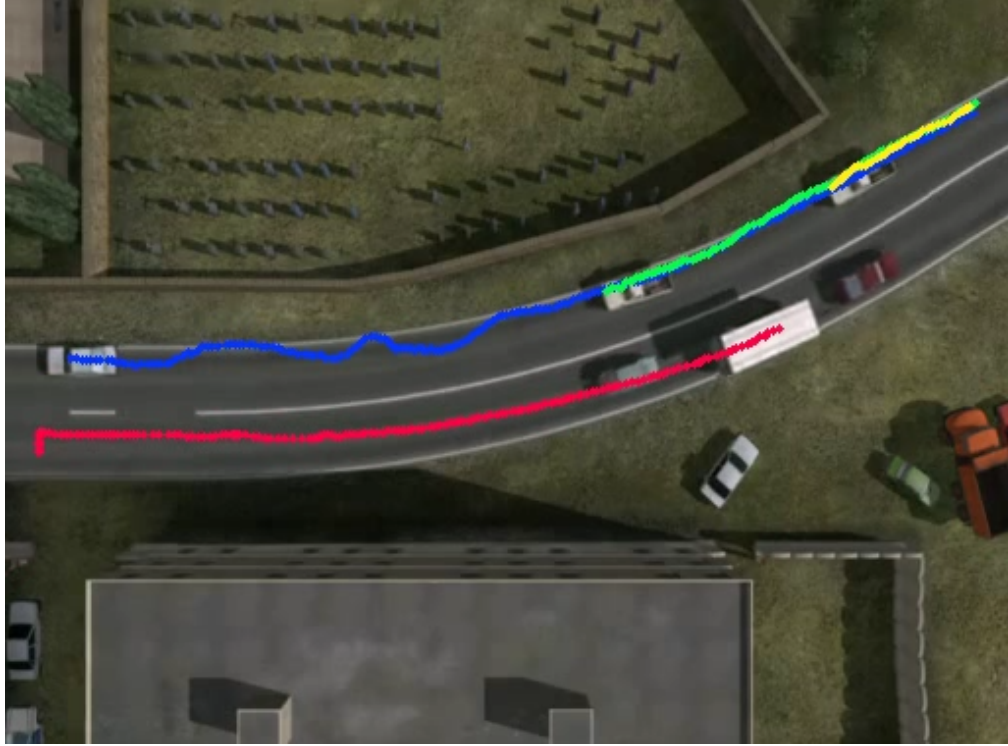


FIGURE 3.3 *Detected tracks with different hue.* Red and gray cars in front of, and behind the long white truck are undetected. This example is from a synthetic scene.

modern CPUs due to data dependencies. For ideal performance, capabilities of the actual hardware platform need to be taken into consideration.

- *Exploit thread-level parallelism* during `cv::warpPerspective`, background maintenance and object detection. Currently the only part of the application which can exploit multiple processing cores is the optical flow calculation. The listed steps would be also able to benefit from multiple running threads.
- *Task-level parallelism.* This would improve the performance of those steps, where thread-level parallelization is difficult, and which are waiting for I/O. For example, acquisition of the next frame could be initiated before the processing of the previous frame is finished.

I have several ideas to improve the quality of the results. First, *more complex modeling of vehicle motion* would allow to identify unusual tracks which are probably defective. Second, *robust image difference* between current frame and background image would probably prevent false object detections due to registration errors in areas which are rich in edges. Third, it is very hard to detect vehicles whose color is a shade of gray very similar to the road's.

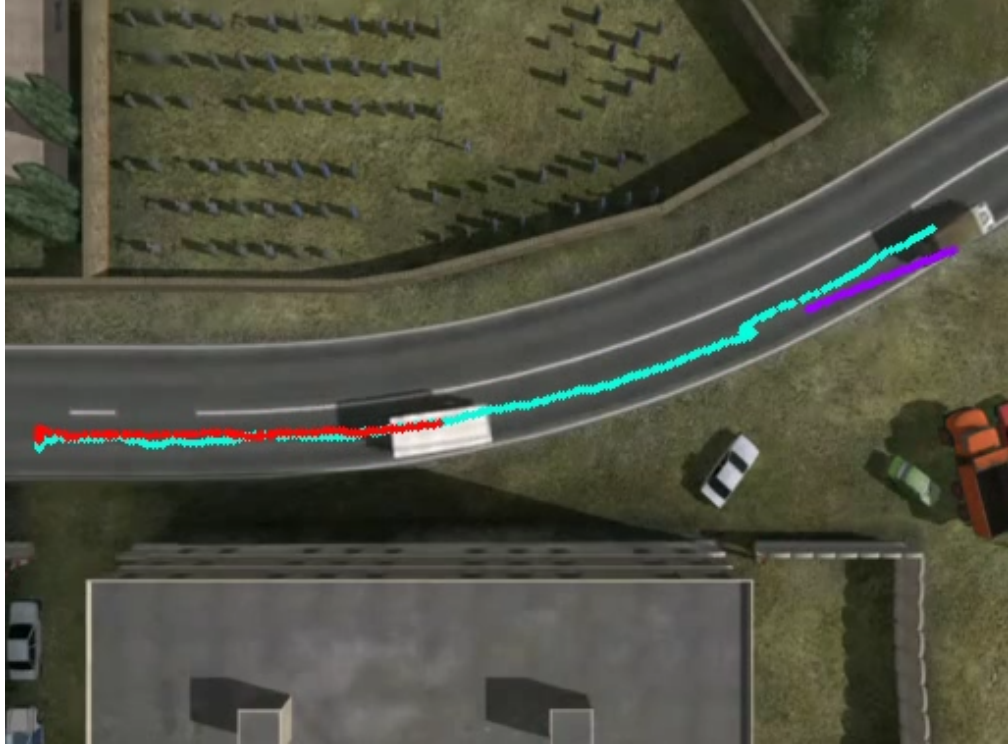


FIGURE 3.4 *Detected tracks with different hue.* The car on the right has recently been detected twice, thus it has an additional purple track beside the correct cyan track. This example is from a synthetic scene.

Long term registration and eventually tracking can be ultimately improved by the using other sensual information as well, like information from GPS, gravitational sensor and gyroscope.

Acknowledgements

First, I would like to thank my supervisor Professor Tamás Szirányi for his consistent help and the opportunity to take part in this project.

I also thank the Distributed Events Analysis Research Group for the task, and the support during my work. I am grateful to Csaba Benedek Ph.D. for his support in the theoretical background and for his help in improving my thesis. I would like to thank Levente Kovács Ph.D. whom I often used to consult about the exact requirements.

Bibliography

- [1] R. Kumar, H. Sawhney, S. Samarasekera, S. Hsu, H. Tao, Y. Guo, K. Hanna, A. Pope, R. Wildes, D. Hirvonen, *et al.*, “Aerial video surveillance and exploitation,” *Proceedings of the IEEE*, vol. 89, no. 10, pp. 1518–1539, 2001.
- [2] S. Ali and M. Shah, “COCOA: tracking in aerial imagery,” *Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications III*, vol. 6209, pp. 62090D–6, 2006.
- [3] W. Yu, X. Yu, P. Zhang, and J. Zhou, “A New Framework of Moving Target Detection and Tracking for UAV Video Application,” in *ISPRS Congress, Proceedings of Commission III*, (Beijing, China), 2008.
- [4] G. Mátyus, C. Benedek, and T. Szirányi, “Multi target tracking on aerial videos,” in *ISPRS Workshop on Modeling of Optical Airborne and Space Borne Sensors*, (Istanbul, Turkey), 2010.
- [5] V. Reilly, H. Idrees, and M. Shah, “Detection and tracking of large number of targets in wide area surveillance,” *Computer Vision–ECCV 2010*, pp. 186–199, 2010.
- [6] J. Xiao, H. Cheng, F. Han, and H. S. Sawhney, “Geo-spatial aerial video processing for scene understanding and object tracking,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.
- [7] B. Reddy and B. Chatterji, “An FFT-based technique for translation, rotation and scale-invariant image registration,” *IEEE Transactions on Image Processing*, vol. 5, no. 8, pp. 1266–1271, 1996.
- [8] L. Lucchese, “Estimating Affine Transformations in the Frequency Domain,” in *Proc. Int. Conf. On Image Processing*, vol. II, (Thessaloniki, Greece), pp. 909–912, September 2001.

- [9] C. Harris and M. Stephens, “A combined corner and edge detector,” in *Alvey Vision Conference*, vol. 15, p. 50, Manchester, UK, 1988.
- [10] D. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [11] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded up robust features,” *Computer Vision–ECCV*, pp. 404–417, 2006.
- [12] A. Kovacs and T. Sziranyi, “New saliency point detection and evaluation methods for finding structural differences in remote sensing images of long time-span samples,” in *12th International Conference, Advanced Concepts for Intelligent Vision Systems*, pp. 273–283, 2010.
- [13] M. Fischler and R. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [14] C. Stauffer and W. Grimson, “Adaptive background mixture models for real-time tracking,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, IEEE, 1999.
- [15] H. W. Kuhn, “The Hungarian Method for the assignment problem,” *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.
- [16] Bohemia Interactive Australia, *VBS2 VTK User Reference*.
<http://www.bisimulations.com/support/manuals/>.
- [17] Bohemia Interactive Australia, *VBS2 VTK Developer’s Reference*.
<http://www.bisimulations.com/support/onlinehelp/>.