## 4. Analysis and Theoretical Foundation

As presented in earlier chapters, the main contributions and final goal of this thesis concerns vehicle traking which and the possibility of it being deployed from a broad range of locations under minimal placement constraints. This chapter will clarify and reson about the details of the presented methods from chapter three and thier scope within the propoposed architecture architecture. We will logically deduct the steps performed for the implementation of the system.

The figure 4.1 describes the general conceptual architecture of vehicle tracking system in which we show the flow of data between the composing modules, dependencies and services provided by the components to other components.



## 

You will find out that the architecture that we propose for our solution closely resembles the trends also approached in the works of [1?] [2?] [3?] [4?]. Our change in the architecture was purely motivated by performance reasons, and consists of the tracker module simultaneously consuming the outputs from the Detector and Background Subtractor module. We will detail later in chapter 5 where we discuss the dependencies of deploying these component on various hardware and software platforms.

Next we give a brief description of the basic building blocks that compose the architecture and the tasks that they play in achieving the final goal.

* The video camera is the primary input for the system, responsible for interfacing the system with the real world by capturing video frames of the surveillance scene.
* The Training Set Acquisition module is responsible for building the Image Training Set that is later used in extracting vehicle specific patterns with which the Classifier is trained.
* The Image Training Set is just a container for a collection of positive training samples, images that depict single vehicles in their area of concern, and negative training samples, images that are suppose not to contain any vehicles.
* The Classifier has the capacity to discriminate between patterns of vehicles and non-vehicles on an entire image, by learning to distinguish them based on the derivated features as a result from processing the training set.
* The Detector is also fed with incoming frames from the acquisition device and has the responsibility of communicating with the classifier and search the video frames for the location and size of vehicles. We call *a detection* a bounding box on the input frame which centrally encapsulates a vehicle.
* The Background Subtractor module is responsible with performing motion segmentation, i.e. identifying the moving regions, in the frames whether they are pedestrians or vehicles. The Background Subtractor module will provide the detector a foreground mask in which the value of each pixel will indicate if the corresponding pixel from the frame contains the projection of a moving object, or any kind of entity.
* The Tracker consumes detections provided by the Detector and the foreground mask provided by the Background Subtractor and performs additional processing using mechanisms to construct the tracks of moving vehicles over time and also validates the consistency of being generated by real vehicles.

Now that we have brief description of what each module is responsible for performing we will dive into the algorithmically details of the architecture and decompose each module as they are scheduled to perform their work in the general process of the system.

## 4.1 Training set acquisition

In general, all machine learning algorithms take as input thousands of data vectors made up of many features, where the number of features might also reach thousands.

The first problem encountered is how to collect the images that make up the dataset. Do you search for online datasets or do you collect it from a security camera and what kind of information do you capture: static or movement. We must end up with thousands of images.

But our job is to recognize a certain type of object, vehicles showing in specific image regions. The second problem encountered is how to label training data that falls into positive (there is a vehicle in the scene) and negative (no vehicles) cases. We must end up with thousands of images, how do we label that?

You will soon realize that vehicles appear at different scales, sizes, orientations, design, their image may consist of just a few pixels, or may cover whole screen. Vehicles will often be occluded: a vehicle passing behind a tree, a man passing in front of the car, or vehicles occluding each other.

An algorithm that recognizes vehicles from ground level might fail when applied on top views of vehicles. We need to capture the slight variations in the data: different views of vehicles, different lightings, weather conditions, shadows, different car producer designs.

The training set acquisition module has the objective of addressing the problem of capturing and labeling the whole training dataset. The proposed method for acquiring the training set has the advantage of gathering images offering meaningful patterns for vehicles that are relative to the scene and locality of our acquisition device and also variations in the data, mentioned above.

For example, if we were to deploy this system in two different locations, one filming vehicles close and frontally, and another filming vehicles from far away and sideways, we would expect that this module to collect front facing pictures of cars, in the first case, and sideways pictures of cars, in the second. Moreover the vehicles in one database will be of similar scale and capture only specific variations of orientation that the scene permits: cars having normal orientation in the direction the road, cars engaged in overtaking.

Figure 4… is the expansion of the Training Set Acquisition module from the Conceptual architecture figure 4…., depicting the internal processing modules and stages performed on the input frame in order to obtain the cropped out vehicles (positive samples) and also negative samples which are saved in the Image Training Set, as the last operation. As you can see, the architecture follows a simple pipeline model, each module taking as input the processed output of the previous module. Let us analyze what are the objectives of each module and what operations and algorithms engaged in fulfilling their purpose.



*Video Camera*

The video camera acts is the acquisition device which captures frames from the observed scene. A discussion is here necessary based on one constraint that we have imposed on our system, namely we have imposed that the mounting point of the camera should be elevated from the ground providing top-views of the passing vehicles.

Because at this stage of processing we are interested in acquiring quality training samples, i.e. pictures in which vehicles are centered under small variations of orientation and scale this constraint helps us with processing because:

* We minimize the occlusion regions between vehicles at projection of the scene onto the image plane
* It increases our efficiency in extracting individual vehicles, acquiring thousands of samples



Figure 4… sample frame

***Frame cropping***

**Figure 4.1 also demonstrates the perspective effect of projecting the scene onto the image plane, vehicles closer to the camera will appear larger than the ones farther away, another effect is the and also introduces significant occlusions. In [8] the region closer to the camera is called *entrance region*, and in this region occlusions between vehicles are minimized.**

**Also in [8] they automate the process of finding this region by running Canny Edge Detection algorithm to and then a Hough line transform to identify the orientation of the highway.**

**Our approach is different in that we demand the user to select to select with the mouse a region of interest in which he expects to find minimal occlusion between vehicles and in which vehicles appear to have a median scale within their passing through the video. Once the region of interest is selected further processing will be carried on only this region within the frames.**

**The constraints for selecting this region are as follows:**

* **It should minimize the occlusion between vehicles**
* **The vehicles should appear to have a median scale**
* **The height of the region of interest should be slightly bigger than the height of vehicles. This information will be later used to filter the detected blobs.**



**Figure 4…, Selecting the ROI(region of interest)**

***Background Subtractor***

**The Background Subtractor module has the responsibility of learning the background model of the region of interest and output the foreground mask, which is a binary image segmenting moving parts or regions of region of interest from the static regions, the background.**

**In order to learn the background model several frames in the beginning of the video are reserved for the construction of the background model, after which the foreground mask is obtained by differentiating the background model with each incoming frame, additionally the model has to be continuously updated in order for it integrate slow varying changes, like the variation of light.**

**The quality of the foreground mask will affect not only the blob detection later in processing, but also the accuracy of the classifier and because that at this stage of training we are not concerned with real-time processing we choose a high performing algorithm for this task, the MOG (Mixture of Gaussian).**

**In the approach of the Mixture of Gaussian, each pixel history is modeled by a mixture or linear combination of *K* Gaussian distributions, where a pixel history is the sequence of pixel values over the last *t* frames. The probability of observing a pixel value as a background value is given by evaluating the expression:**

**Notations:**

**When updating the pixel history with a new pixel value its *K* Gaussian mixtures are evaluated. If it does not have a high enough value against any Gaussian then the distribution with the lowest cost will be replaced with a new one and also distribution means and covariance matrices are updated.**

**The background model for each pixel is chosen to be composed of *B* most reliable and least variance distributions. When evaluating a new pixel value *X* which gives a high probability within these B distributions then the pixel value is considered to be a background pixel.**

**In this manner the MOG is continually adapting to small changes of lighting and weather. We also observed that the foreground masks resulting from this procedure are of high quality and are robust to noise and very small camera movements which we have experienced. The next figure shows the constructed background model, and extracted foreground mask for two random frames**







**Figure 4… The first and second columns represent separately the region of interest, background model, and foreground mask from two random frames.**

***Blob Detector***

**Having the foreground mask the Blob Detector’s job is to quickly find the blobs, the connected components on the foreground masks. It does this by applying a contour tracking algorithm on the foreground mask following the border white object pixels.**

**Additionally this component also performs the following tasks:**

* **It discards blobs which have the width, height and area less than predefined thresholds. In our application the thresholds for width and height are 5 and for the area the threshold is 25. This filtering step will most likely small blobs which are not of interest resulting from noise of small object movements, example: leaf movements in trees, birds flying.**
* **It also discards blobs that are on or very close to the region of interest’s border. The reason for applying this filtering step is that later compute a bounding box around the blob and we want this bounding box to be contained in the region of interest. Another reason for this operation is that we do not want for a vehicle to be saved too many times in the Training Set and thus we ignore it when entering and exiting the region of interest.**

***Blob Filtering***

**The Blob Filtering module performs additional filtering functions and lastly saves positive samples containing vehicles and negative samples which are preferred not to contain vehicles in the Training Set. This module’s goal is to make an educated and simple decision about the incoming blobs from the Bob Detector module to decide which of them are masks generated by moving vehicles.**

**The decision to classify the blobs as vehicles is done based on a feature of shape, namely the *circularity* of the *convex hull*. In the first iterations of development this module was missing and we were collecting elongated objects mainly because we were experiencing slight camera movements. Figure 4… shows that street markings were saved as the background subtraction was generating blobs under camera movement.**

E:\Academic\Licence\lic_doc\pics\chapter 4\bad_blob3.png E:\Academic\Licence\lic_doc\pics\chapter 4\bad_blob1.png 

Figure 4… Bad images saved in the absence

of the Blob Filtering Module.

**It was a necessity to use a discriminative feature that would correctly classify cars at this stage, unless we were to manually go over all the generated samples again and hand-pick them.**

**The idea of using the eccentricity of the convex hull is also mentioned in [7]. They argued that a vehicle is roughly a convex object, and the silhouette is very close to its convex hull. When two vehicles tend to occlude themselves the blob containing them generally does not have a convex shape. With this assumption in mind we can further filter passing blobs and discard joint vehicles appearing as one blob.**

**At this stage of processing we are able the robustly filter and discard the blobs which have a high probability of not belonging to the vehicle class. The last function that this module performs is to save the detected blob as a positive sample in the Training Set. This step is performed by centering a cropping rectangle over the center of gravity of the detected blob. The cropping rectangle’s size is computed relative to the size of the region of interest, namely the height of the cropping rectangle in our case is the largest multiple of 8 which is also smaller than the height of the region of interest, and we compute the width based on an empirically selected aspect ratio which in our case is 1. For our tested video a good cropping size for the saved patches was 64x64.**

Figure 4… Detected blobs

Once we have saved the cropped images as positive samples in the Training set, the next question is from where do we obtain the images serving as negative samples. In [8] they propose the idea of saving the negative samples relative to the location of the blob detections. Specifically, they select rectangular patches centered around the corners and edge midpoints.

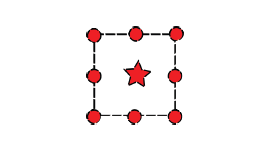


Figure 4…, Source: [8], Relative position of positive and negative training samples.

The bullets are the centers of the negative patches placed around the corners and edge midpoints of the detection.

**Another strategy of choosing the negative samples would be to select them randomly over the frame but there is the risk that one of them would end up containing a vehicle in the center which will have a negative impact on the training of every type of classifier.**

**The authors of [8] point out that this approach offers a great advantage for the training of the classifier: it is not just trained with plain and random clutter, but with hard examples generally containing margins and portions of vehicles. Thus the classifier is forced to learn to rule out zones in which it might think that it has detected parts of vehicles, and even filter the regions where vehicles are very close to each other, which is beneficial for accurate localization of vehicles. We also point out that as opposed collecting random negative training samples from different sources or internet databases this method collects samples specific to the scene we are observing thus it further reduce the risks of having false detections from the classifier. Another idea that we propose it to use the background, model obtained in previous stages, by segmenting it in random selected patches which we will also use as negative training samples. The classifier will learn scene specific features that it will easily distinguish as not containing vehicles, and reduce the number of false detections.**

**We show in figure 4… and figure 4… the positive and respectively the negative samples that we end up saving in the Training Set. Experiments have demonstrated that 100% of the captured positive samples are indeed containing a single centered vehicle. We also have to mention that we recorded having centered vehicles in negative samples also, mainly due to the fact that separate vehicle blobs were captured moving close and in the process of saving one of them the other was saved as a negative sample. Nevertheless we did not hand-remove these samples and left the Training Set exactly as it was constructed in the described process of acquisition.**



**Figure 4… Extract of the positive samples training set**



**Figure 4… Extract of negative samples training set**

## 4.2 Feature Extraction and Classifier

**4.2.1. Feature Extraction**

**One of the main challenges in developing a machine learning application concerns the features you use in training the classifier. This decision was taken only after we carried out experiments with different feature extraction methods and studying the results and improvements proposed in other papers.**

It is unavoidable that moving vehicles in the scene will suffer different scale and pose transformations depending perspective projection, their motion direction, elevation of the camera: their image may be as small as just a few pixels, or in close proximity they may be covering a large portion of the screen. The interactions and events that take place in an outside-door environment are of such high complexity that they cannot be properly understood by the computers in order to convey the semantic meaning.

Often vehicles will often be occluded: a vehicle passing behind a tree, a man passing in front of the car, or vehicles occluding each other. In the domain of tracking occlusion is a very complex problem which has been approached in many paper subjects. The reason for this is that is very hard to identify if an object is occluded by the background and other objects or it is occluding himself with new untracked parts.

As every classification problem the goal is to maximize inter-class variance or minimize intra-class variance. We want to be able to correctly recognize all type of cars, regardless of size, position, scale specific for our scene and also discard any kind of objects falling in other classes like pedestrians.

Our goal is to capture the variations in the data: different views of vehicles, different lightings, weather conditions, shadows, and so on.

Detecting cars is a considerably more difficult problem than detecting textured objects which have simple, semi-rigid structure, where the localization of components does not vary much between samples. Cars have a semi-rigid structure as well, but that structure will vary more between samples, because their shapes and configurations have been designed with product differentiation in mind. Besides the intra-class variations due to color, shape, and ornamentation, which similarly plague face and pedestrian detection systems, there are other issues that complicate car detection. Compared to the other object classes, vehicles also lack texture and have a highly reflective surface. Experimentally we observe, from the background subtraction technique, that the majority of vehicles tend to resemble the color of the pavement in some regions.

In the yearly days of our project we have analyzed the possibility of using the Bag of Words method for extracting a data vector that describes the image sample. In short this method extracts SIFT keypoints/features from the collection of training samples and clusters them to obtain cluster centroids which denote the most relevant features from the whole set. Next an SVM is trained using as feature vectors the frequencies of the most representative keypoints in each of the training samples. The disadvantage of this method is that it lacks locality, the position of the extracted keypoints is discarded, and also we were not obtaining the results that we needed.

At this stage of the process we were inspired by the works of [6] with the formulization of HOG (Histogram of Oriented Gradient) in the context of human detection and [8] with the application in the context of tracking vehicles on a surveillanced highway.

The idea of Histogram of Oriented Gradient descriptors is to represent an image patch by the distribution of gradients directions. A good approach in describing a whole image is to divide in it a grid manner where each segmented region is called a cell and then computing for each a histogram of gradient orientations. The combination of these histograms in a single data vector will represent the feature vector or descriptor of an image. To increase accuracy and account for local illumination, shadowing, foreground-background contrasts an extra step can be performed, the histograms within a cell are contrast-normalized by computing a measure of intensity across a neighborhood of cells called a block, and this value is used to normalize cells within a block. The original description of this method along with schemes of how to customize the block normalization procedure can be found in [6].

After obtaining the training set the HOG descriptors are extracted off each training sample and saved as positive or negative descriptors which are later used in the SVM training phase. In our solution we divide each training sample into cells of size 8x8 pixels and the blocks are of size 16x16 pixels.

**4.2.2. Classifier**

The next step after obtaining the descriptors of the positive and negative training samples is choosing a classifier. For our work we decided to use SVM linear because they offer fast evaluation and fast training with the expense of choosing good training features

## 4.3 Detector

The detector has the responsibility of applying the classifier over the frame in a sliding window approach providing robust detection region for the vehicles. As you have seen in figure 4… the vehicles that appeared in the positive training set have very close scales, due to the fact that they have been saved within roughly the same camera distance. The truth is that vehicles will appear at scales which are different from the one deducted from the training image set, therefore the detector has search at different scales of the image. It does this by iteratively decreasing the image size and keeping the size of the search window over the candidate image trying to identify vehicles that appear at larger scales. In our case we iteratively decrease the image by a factor of 1.05. Once the image has been scaled down several times the HOG histograms are computed for each of them. When sliding the searching window over the image it is important to have a stride in its displacement because moving it pixel by pixel is not efficient and does not even increase accuracy. In our solution we apply a stride equal to the cell size. At each position of the sliding window the histogram of the blocks is under the sliding window is extracted and feed it to the classifier. The classifier lastly compares the distance of the input histogram to the separating hyperplane learnt after termination of the learning phase and the detector records this as a response for the current position of the sliding window and continues to search new locations of vehicles by iteratively displacing the window top-down from right to left.

After the process of searching for vehicles in all scaled images we obtain a list of possible car locations each bearing a score received by the classifier. At this stage it is likely that at the real location of a vehicle we could have obtained multiple positive responses from the classifier. These positive responses are generated at small deviations of the sliding window around the true position and scale of the vehicle. The next task is to filter all these satellite detections and obtain the best one for tracking. We make this decision based on the classifier output that characterizes each location. This challenge reduces to the problem of finding disjoint sets. Specifically we apply to solution of disjoint sets in the goal of eliminating near close detections located under a specified threshold. After obtaining the disjoint sets, a voting procedure is carried out for each set to construct the position and size of best detection for that set, a representative, and finally if there are overlapped, detection representatives, above a selected threshold, then only one of them is saved depending on the size of the disjoint set that it has constructed it. Figure 4…. Show the strong responses that the classifier gave for the scene and figure 4… depicts the results of filtering the detections for a vehicle.

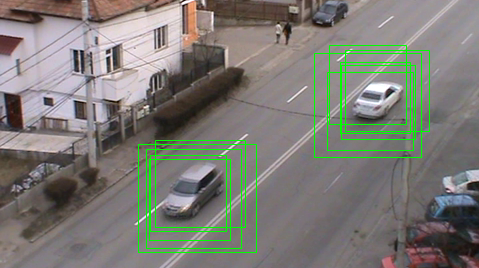


Figure 4… Multiple detections for one vehicle

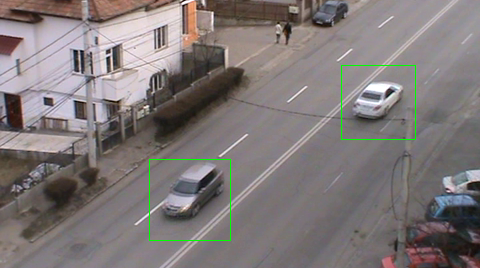


Figure 4… Filtered Detections

## 4.3 Background Subtractor

We have already discussed the Background Subtraction technique and used it in the process or forming our Training Set. The technique that we have used is called MOG (Mixture of Gaussian) and the principal strength of this approach in constructing the background model is the tolerance to noise but has the weakness that it cannot achieve the real-time standards that we are aiming to achieve. We have also mentioned that it is not a concern in the case of the Training Set Acquisition process.

This Background Subtractor module has the responsibility of providing the tracker with pixel level information of how is the scene segmented into static and moving objects, i.e. the background and potential moving cars. The tracker will further use this information to extract patterns from the surface of vehicles which can be submitted for tracking.

The main concern that we have to face here is the real-time performance. While this is not an issue in the case of the training set acquisition process, we have determined experimentally that that the quality of the foreground mask need not be high and thus we allowed ourselves to use a faster and simpler method for this stage in order to achieve a good throughput of foreground masks for the tracker.

The method that we have used here is based on maintaining a *running average* as the background model for each pixel, thus a pixel is represented by a single value, as opposed to a mixture of Gaussian distributions in the MOG. In [9] they have experimented with different subtraction methods and have concluded that the Running Average is certainly the fastest, in which the mask is just a thresholded difference between the background model and the frame, while the model adapts one or two parameters.

This method is first mentioned in [10] and they propose a background model independently at each pixel location. The model is based on ideally fitting a Gaussian probability density function on the last *n* pixel’s values. In order to avoid fitting the probability density function from scratch at each new frame time *t*, a running average is computed cumulative on line instead as:

where , is the pixel’s current value and , , the previous average, *a* is an empirical weight often chosen as a trade-off between stability and quick update, you can think of it as the learning rate. In addition to speed, the advantage of the running average is given by the low memory requirement for each pixel. At each frame time a pixel's value can then be classified as a foreground pixel if the inequality:

where *k* is a threshold above which the deviation of a pixel from the mean value leads to it being marked as an object pixel.

The following figure is an illustration of the Running Average method on our video.







Figure 4… Current frame in the video, background model, foreground mask,

*a=0.01, k=20*

As you can see from figure 4… the foreground mask is not of high quality and is subject to noise. We have several options of improving the results:

* Increase the threshold *k*, this will increase the value with which pixels can vary around their running average thus reducing noise, but as a consequence cars with colors that closely resemble the pavement will not be detected. This effect is illustrated in figure 4…
* Increate the learning rate *a*, this will lead to stabilizing the color of a pixel in the background model more quickly, it will converge to the current color faster, but as a consequence pixels covering large moving objects of constant color will be detected as background because the mean color of the pixel has stabilized too quickly when the object was in the phase of entering the pixel; this effect can be seen in the center vehicle of figure 4…, the bonnet and windshield are detected as background pixels.
* Apply morphological transforms, erosion and dilation, the filter the noisy regions, small connected regions surrounded by background. Depending on the structuring element with erosion we can filter a large portion of the noisy pixels in the foreground mask in figure 4… and with dilation we can close the holes in the blobs containing vehicles. Experimentally we have obtained satisfactory results by applying erosion followed by a closing operation. The structuring element that we have used is a cross of size 7x7 with the origin in the center of the cross. Figure 4… exemplifies the results.



Figure 4… Foreground mask obtained with an increased *k,*

*a=0.01, k = 50*



Figure 4… Foreground mask by applying erosion followed by a closing operation

## 4.3 Tracker

The tracking module is the most complex component of our solution and the subject of contribution approached and proposed by this paper. The aim of an object tracker is to generate the trajectory of an object over time by locating its position in every frame of the video. Object tracker may also provide the complete region in the image that is occupied by the object at every time instant.

Essentially our tracking module is responsible for managing the tracks of detected vehicles by:

* Initiating new tracks when new vehicles are detected
* Merging detections to tracks when detections are available
* Producing track estimations when detections are not available
* Scoring the tracks
* Identifying false positive tracks by the scoring method
* Counting true tracks

We have discussed that in order to perform these goals the Tracker receives as input detections from the Detector under the form of bounding rectangles and also the foreground mask from the Background Subtractor. The Tracker combines these two sources of information by overlapping the detections over the foreground mask and thus obtaining the pixels from the foreground mask that are thought to belong to a single vehicle. This information is useful because we later use a point tracker.

The method proposed by this paper in the subject of tracking is to use two different techniques, the Kalman filter and Lucas-Kanade. Each technique has its strong points and strengths. The main idea is to realize when one technique is not suitable to apply because in specific situations exploit its weak points and use the other’s technique strong points to compensate. In what follows we will review the main ideas and assumptions that the Lucas-Kanade point tracker and the Kalman filter assume, what is the best case and worst case when applying these methods, what are their advantages and disadvantages, the technique we propose to balance the each other’s weaknesses and what are the possibilities of modeling a track.

The next figure is a sketch of flow of the tracking algorithm performed for managing each track it is aware of.



Figure 4… Flow of the tracking algorithm

*Kalman-Filter Estimation*

At each frame we make a determination and find ourselves with an estimate of the location of a vehicle, which was detected by the Detector. This estimation is not likely to be extremely accurate. The reasons for this are many. They may include inaccuracies in the sensor, approximations in earlier processing stages, issues arising from occlusion

or shadows, or the apparent changing of shape. Whatever the source, we expect that these measurements will vary, perhaps somewhat randomly, about the “actual” values. We can think of all these inaccuracies, taken together, as simply adding noise to our tracking process. We would like to have the capability of estimating the motion of vehicles in a way that makes maximal use of the measurements we have made, specifically the detections. Thus, the cumulative effect of our many measurements could allow us to detect the part of the vehicles’s observed trajectory that does not arise from noise, inaccuracies or deviations of the detection around the true location of the vehicle.

The key additional ingredient is a *model* for vehicles motion. In our case we model the vehicles motion under the following assumption: *A vehicle enters the frame at one side and passes through the frame with approximate constant velocity*.

Given this model, we can determine where the vehicle is. This task is divided into two phases. In the first phase, typically called the *prediction phase*, we use information learned in the past to further refine our model for what the next location of the vehicle will be. In the second phase, the *correction phase*, we make a measurement and then reconcile that measurement with the predictions based on our previous measurements, in our case the measurement is the detection.

The Kalman filter builds a model for the state of the system that maximizes the a posteriori probability of those previous measurements. It means that the new model constructed after making a measurement is the model that has the highest probability of being correct, considering the previous model with its uncertainty and the new measurement with its uncertainty. In addition it is not required to keep a long history of measurements, i.e. detections, but iteratively update our model of a system’s state and keep only that model for the next iteration.

There are three important assumptions required in the theoretical construction of the Kalman filter:

* the system being modeled is linear
* the noise that measurements are subject to is “white”
* this noise is also Gaussian in nature

The first assumption means (in effect) that the state of the system at time *k* can be modeled as some matrix multiplied by the state at time *k*–1. The additional assumptions that the noise is both white and Gaussian means that the noise is not correlated in time and that its amplitude can be accurately modeled using only an average and a covariance.

We start with what we know, we obtain new information, and then we decide to change what we know based on how certain we are about the old and new information using a weighted combination of the old and the new.

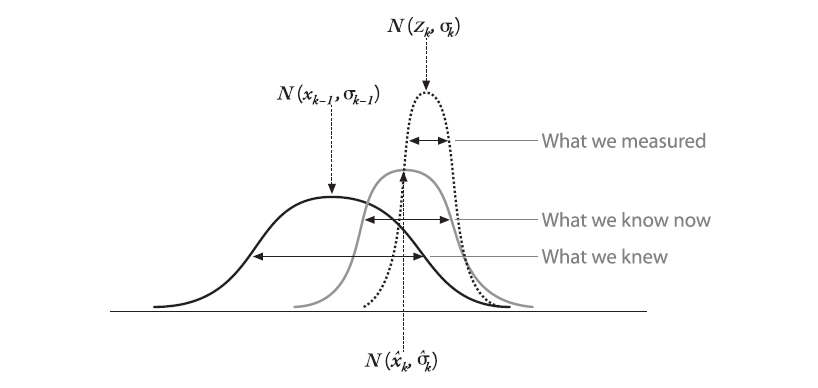


Figure 4…, Source [12], Combining our prior knowledge with our measurement observation; the result is our new estimate

It is beyond of the scope of this paper to deduce the equations the of the Kalman filter but they are presented in the next figure.

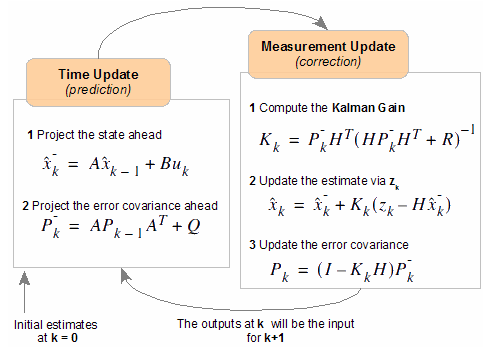


Figure 4…, Source: [11], Kalman filter phase equations

where *A* is the transfer matrix, used to express linearly as a function of ; is the prior estimate before update correction; is the estimate after the measurement update ; is the estimate after the measurement of the previous phase; is denoting the external controls on the system, is used to relate these external controls, is the measurement we are incorporating in our model, in our case it is the position of the detections; is a matrix relating the state to the measurement; is the prior error covariance; is the Kalman gain which tells us how to weight new information against what we think we already know.

As mentioned we used the Kalman filter to estimate the position of the vehicles under the assumption of constant velocity, the matrices of the filter that we use are:

, , ,

The measurement is the coordinates of the center of the bounding rectangle detection.

*Lucas Kanade Point Tracker*

A good introduction on the Lucas-Kanade algorithm can be found in [12]

The optical flow is a velocity field in the image which transforms one image into the next image in a sequence

The LK algorithm can be applied in a sparse context because it relies only on local information that is derived from some small window surrounding each of the points of interest. The disadvantage of using small local windows in Lucas-Kanade

is that large motions can move points outside of the local window and thus become impossible for the algorithm to find.

The basic idea of LK algorithm rests on three assumptions, as highlighted in [12]:

* *Brightness constancy*. Image measurements, brightness, in a small patch remain the same although the location may change. This means that we can assume that pixel brightness does not change from frame to frame as we are tracking the point.
* *Temporal persistence or small movements*. The image motion of a surface patch changes slowly in time. In practice, this means the temporal increments are fast enough relative to the scale of motion in the image that the object does not move much from frame to frame.
* *Spatial coherence*. Neighboring points in a scene typically belong to the same surface and by projection onto the image plane we would expect the same special coherence in the image as well.

We will not go into details of deriving the equations but based on these assumptions we must highlight some worst case scenarios that our scene can exhibit to induce errors in this method of tracking:

* As we have already mentioned some parts of vehicles appear as high reflective surfaces thus have a high probability of changing their surface brightness as they pass through the environment.
* Other parts of vehicles appear as patches of constant color thus the movement of neighboring pixels cannot be detected.
* Another phenomenon that takes place to break the third constraint is occlusion in which case points placed the margin of overlapping vehicles exhibit neighbors that have different flows. Such cases appear when vehicles produce occlusion among them, or when they are passing behind a pole

This method work best in controlled environments, but in outdoor environments it is difficult to rely on tracking successfully points through a large number of frames. Due to the assumptions stated above the track generally suffers drifts for the actual path or can even be tracked in a completely wrong region.

To overcome the disadvantages of tracking the same points over a long number of frames, our approach is to only track newly selected points from frame to frame. The advantage over tracking the same points is that consecutive frames do not vary dramatically and so we can rely on the assumptions stated, while points visible in the first frames will often get occluded as they are carried by the car surface into the frames in which the car is exiting.

The method of how we use the Lucas-Kanade tracker to estimate the position and scale of a vehicle in the next frame is the one proposed in the works of [13]. They propose a method of self-evaluating the tracking reliability and estimating the displacement from frame to frame of a bounding box. The proposed method is based on the principle of forward-backward consistency of correctly determined tracks, more precisely, given a sequence of frames the track of an object should be independent of the direction of the time-flow. Algorithmically the method can be implemented as follows.

* First we track a point *forward*, over several time frames, obtaining the forward trajectory.
* Secondly we track the point from the time-frame we stopped to the time-frame we started; this is equivalent to reversing the flow of time as we pass through frames, thus producing the *backward* trajectory.
* Thirdly, we can compare these two trajectories and verify how consistent they are with one another. If the difference is significantly, then this means that it is not reliable to follow the forward trajectory. A simple scheme to evaluate the difference in trajectories which we also use is to compute the Euclidean distance between the initial location of the point and the location of the backward tracked point.

.

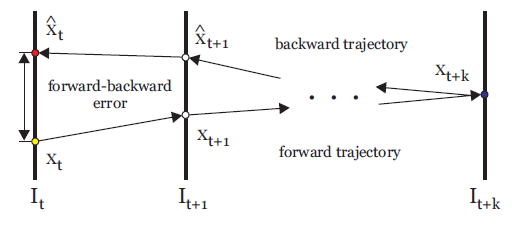


Figure 4…, Source: [13], Forward-Backward trajectories and error evaluation

The authors of [13] also propose an object tracker called the Median Flow Tracker. They formulize the problem of tracking an abject delimited by a bounding box. Their method is based on the observations that points are generally not independent, but are part of bigger objects or parts that move together.

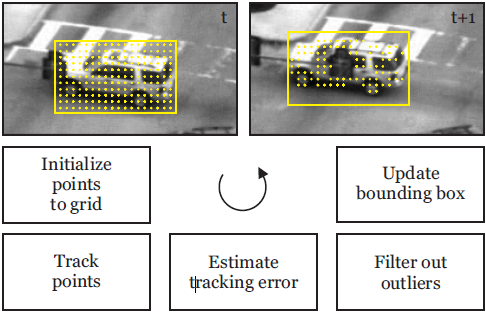


Figure 4…, Source: [13], Median Flow Tracker phases

The Median Flow filter accepts as input a bounding box around an object that we wish to track, within a frame , and will output the updated bounding box in the next frame with the estimated location and scale of the object. The algorithm of the Median Flow is shown in figure 4… and has the following steps:

* Initialize a grid of points within the bounding box with a selected stride
* Perform forward LK tracking with these points into the next frame
* Perform backward tracking of the resulted points back into the original frame
* Fix a threshold above which the distance between which the Euclidean distance between the initial points and the backward tracked points are considered outliers and can be discarded for the following computations because their tracking is not reliable. In our case this threshold is 2 pixels.
* Next estimate the displacement between frame and of the bounding box by computing the median of distances between the initial points and the forward tracked inlier points.
* For determining the scale we determine the ratio of distance between a pair of inliers in frame and the same pair in frame. The scale of the bounding box is estimated to be the mean of the computed ratios.

This method works best with rigid objects. Parts of the objects that do not satisfy this constraint because they exhibit a flexible nature and also object boundaries are discarded in the process of evaluating the inliers and are discarded. There is a possibility of keeping static inliers, which do not move from a frame to another, which are not part of the vehicle object. The displacement of these inliers will be zero and will affect the computation of the median displacement and also the median scale. To avoid this effect we have modified the algorithm to only select the points within the detections which are on the foreground mask, the blobs representing the vehicles.

*4.4. Merging Track Predictions, Merging detections to tracks*

At this stage of the algorithm we should decide how do the tracks model the identity of the objects (vehicles); we should ask which object representation is suitable for tracking vehicles? Which are the relevant features that uniquely identity an object: aspect, motion, shape? After answering this question we need to decide how should these features are abstracted in data structures and how do we extract them from the object. This decision relies on analyzing the environment and the goal for which tracking information is sought. Choosing features is critical in tracking, and in general it is preferable that feature representations of multiple objects these are easily separable in feature space. For example, color and contour representations usually used as features. In general, many tracking algorithms use a combination of these features.

After we have obtained two predictions upon the location of the vehicle for the current frame, from the Median Flow Tracker and the Kalman filter, it is necessary that we decide to assign one of them as the true location of the vehicle at this time step to the track history. Thus we want to determine what is the best estimation that the trackers provided. In order to evaluate this we need to compare the features stored in the track for the vehicle it represents with the features that the estimations provided. The detection which provides the best similarity measure with respect to the features of the track is considered to be the best, and the location and size of the track is update to correspond with this prediction. Thus the *Merging Track Predictions* stage of the algorithm is determining the best prediction for which the distance between the features of the track and the prediction is minimized.

At each frame we are provided with detections on behalf of the Detector. These might be detections for vehicles which we are tracking, or detections for vehicles entering the scene. It is also very probable that for a number of frames we will not receive detections of cars which we have already started tracking. In this situation the tracks locations in the subsequent frames are estimated by merging the predictions from the Median Tracker and the Kalman filter. In the case in which we receive a detection for a car which is tracked we need to update the vehicle features stored in this track. This way we are always updating the model of the car that the track stores to the last detection. We need to make this update because we want to always know what is the most recent aspect of the car. By doing the update only when the Detector provides a track with the next detection we are trustring the Detector.

*Find Detection Match*

The *Merging Detection to Track* stage in the algorithm

Next we will present several methods of evaluating this distance that we have implemented and tested.

*Modeling the Track*

How do we model a track:

How do we describe the identity of a track

Contains history of points

Contains only a final location

Contains position velocity and direction

Lucas-Kanade Forwarding

Merging Predictions

Matchers

Define what is the concept of matchers in our solution.

LBP

(LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. Due to its discriminative power and computational simplicity, LBP texture operator has become a popular approach in various applications. It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination variations. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings. The basic idea for developing the LBP operator was that two-dimensional surface textures can be described by two complementary measures: local spatial patterns and gray scale contrast. The original LBP operator (Ojala et al. 1996) forms labels for the image pixels by thresholding the 3 x 3 neighborhood of each pixel with the center value and considering the result as a binary number. The histogram of these 28 = 256 different labels can then be used as a texture descriptor. This operator used jointly with a simple local contrast measure provided very good performance in unsupervised texture segmentation (Ojala and Pietikäinen 1999). After this, many related approaches have been developed for texture and color texture segmentation.

The LBP operator was extended to use neighborhoods of different sizes (Ojala et al. 2002). Using a circular neighborhood and bilinearly interpolating values at non-integer pixel coordinates allow any radius and number of pixels in the neighborhood. The gray scale variance of the local neighborhood can be used as the complementary contrast measure. In the following, the notation (P,R) will be used for pixel neighborhoods which means P sampling points on a circle of radius of R. See Fig. 2 for an example of LBP computation.

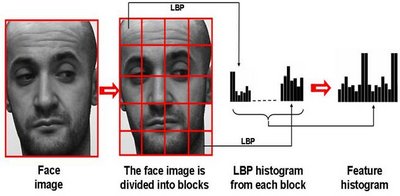
In the LBP approach for texture classification, the occurrences of the LBP codes in an image are collected into a histogram. The classification is then performed by computing simple histogram similarities. However, considering a similar approach for facial image representation results in a loss of spatial information and therefore one should codify the texture information while retaining also their locations. One way to achieve this goal is to use the LBP texture descriptors to build several local descriptions of the face and combine them into a global description. Such local descriptions have been gaining interest lately which is understandable given the limitations of the holistic representations. These local feature based methods are more robust against variations in pose or illumination than holistic methods.

The basic methodology for LBP based face description proposed by Ahonen et al. (2006) is as follows: The facial image is divided into local regions and LBP texture descriptors are extracted from each region independently. The descriptors are then concatenated to form a global description of the face, as shown in Fig. 4.

This histogram effectively has a description of the face on three different levels of locality: the LBP labels for the histogram contain information about the patterns on a pixel-level, the labels are summed over a small region to produce information on a regional level and the regional histograms are concatenated to build a global description of the face.

-f

Pictures Many pictutres



NCC (disadvantages, pictures with fail => track and detection over same vehicle but still fail, too much locality)

Position (good for discarding fast detections)

Merging detection to track

We maintain a track as the last detection that has been merged with the track

Gerally a feature extraction algorithm is used to obtain the describing vector from

A detection we call this entity a matcher , match a track to a detection. Track module uses different matchers to determine if the the track can be merged with the detection. Matches provide a score for a track with a detection.

Matching can be done based on any feature that you can think of be it position or color.

We use a matcher based on position to quickly discard vehicles which are not in the neighbourhood of a track, on the trajectory of a track and the overlapping area.

This is not enough, a new detection can be merged in an old track which is a new vehicle.

That is why we need to extract more discriminative feature to specifically determine if the features on the track and on the detection match. We need another matcher working with more complex features which can uniquely identify the vehicle in the scene.One may use color features but there it is not discriminative enough. An simple method is to

Matchers:

* State diagram
* How to merge predictions
* Features
  + NCC
  + LBP