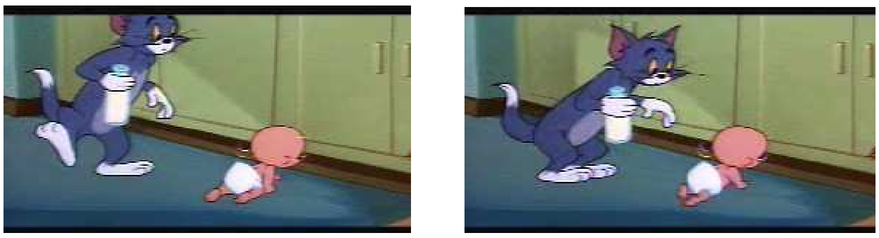
# *Motion Detection*

**Final project by**

***Neta Sokolovsky***



## Introduction

The goal of this project is to recognize a motion of objects found in the two given images. This functionality is useful in the video processing and video analysis. Thus, our base assumption is that the two given images are quite similar to each other, i.e. the motion that objects can do in two consecutive frames is small.

Moreover, finding objects motion can contribute to objects recognition. We use the fact that background cannot move, to separate objects from the background. In additional, we can investigate a video segment generated by surveillance camera for tracing objects over the time.

The general idea of our approach is that object movement causes all its pixels to move in this direction. Thus, the main steps of our project are to recognize pixels belonging to the same object and using a "vote" approach to determine the object motion.

## Approach and Method

In this section we describe the main steps of our approach. In our scenario, the input consists of two images denoted as *first* and *second* images (in video segment these can be two consecutive frames). First, for a pixel in the first image we define all its possible locations in the second image (after motion or at the same place). Since we assume small motion, all these locations form a small region around this pixel. Second, for each object we identify the pixels that belong to its boundaries. Finally, the motion vector of the object is determined by pixels movement using accumulating technique.

* Next we describe construction and initialization steps of our approach.

### Smooth filter

Our approach works on grayscale images (if it is necessary we convert the input images). To reduce the effect of the noise in the image we apply the smooth filter on the two images. This helps to avoid recognition of false objects generated in result of minor changes of the color over the object surface. The smooth filter is a convolution of the image and the quadratic matrix of ones. The size of the matrix can change (a parameter in our implementation) according to the required smooth level.

### Graph construction and image connection

In this step we describe how to represent the image as a graph *G(V, E)*. The nodes *V* of the graph correspond to the pixels of the image. For every two neighbor pixels in the image represented by nodes *u* and *v*, there is an edge *(u,v)* in *E*. For each node *v* we call all its adjacent nodes as *close neighbors* of *v*. Each input image is represented as a graph as described above.

Next, we build a connection between the two images in the following way (the connection is directed from the first image to the second image). For a pixel $p$ in the first image we denote $R_p$ as the small region around $p$ in the second image. The region $R_p$ represents all possible new locations of $p$. Therefore, we connect (by edges) the node representing $p$ in the first graph to all the nodes (in the second graph) representing pixels in $R_p$. All these nodes are called *far neighbors* of the node representing $p$. The maximal movement determines the region size.

Note that the close neighbors of the pixel (in the first or second image) are all its adjacent pixels in the same image. On the other hand, the far neighbors of the pixel (defined only for the first image) are all the pixels in the small corresponding region in the second image (see Figure 1).

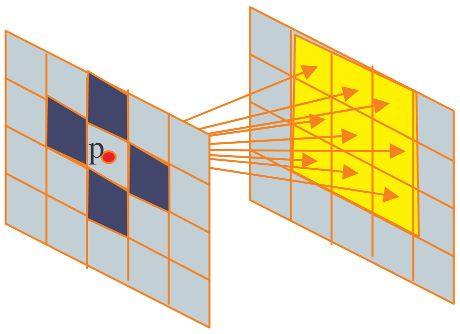


Figure 1. Blue nodes in the first image are close neighbors of node $p$, yellow nodes in the second image are far neighbors of $p$.

* Next we describe the edge detection and perceptual organization steps.

## Edge detection

Let us denote the nodes on the image boundary and on the objects boundaries as *fixed* nodes. We say that node is on the object boundaries if at least one its close neighbors has a different color. We define an edge *(u,v)* as *collapsible* if nodes $u$ and $v$ have similar color and exactly one of them is a fixed node. The edge-collapse operation of an edge *(u,v)*, where $u$ is a fixed node, connects all the close/far neighbors of $v$ to the close/far neighbor-lists of $u$ and deletes $v$ from the graph (see Figure 2).

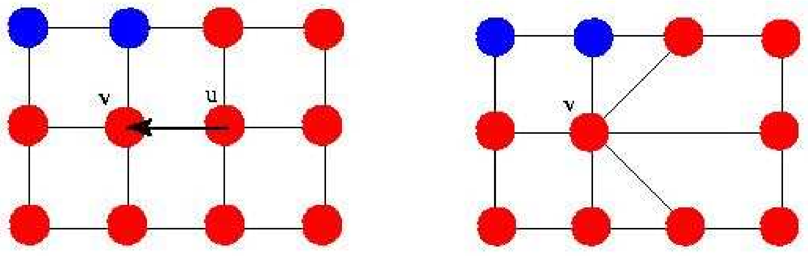


Figure 2. Edge collapse of *(u, v)* where $v$ is a fixed node (left – before collapsing, right – after collapsing).

In this step we pick the shortest edge from the all collapsible edges and collapse it. We repeat this operation while there is a collapsible edge. Since we choose the shortest edge, the fixed nodes "pull" the inner nodes of the object in a balanced way. At the end of this process the graph contains only nodes that represent the pixels on the objects boundaries (see Figure 3).

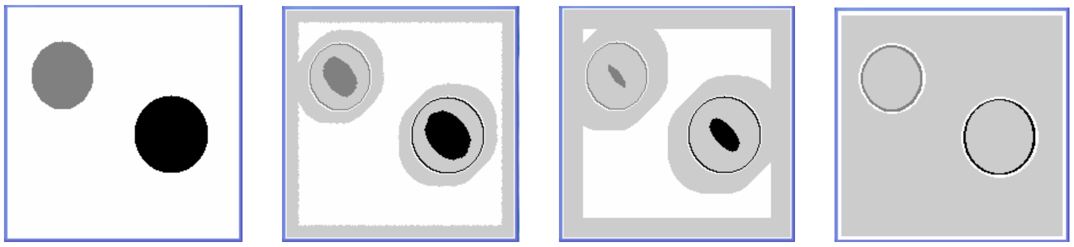


Figure 3. The process of edge detection of the image. The pixels whose corresponding nodes are collapsed are shown in light gray color.

## Perceptual organization

Now, we organize the nodes, found by edge detection, into objects by using color similarity and distance closeness. This process based on the idea of Canny edge detection. We start from the first unlabeled node *v* and declare a new object. For each close neighbor *u* of node *v*, if *v* and *u* have a similar color and an edge *(v,u)* is short (the length of *(v,u)* is smaller than a predefined threshold), we add *u* to the object and recursively check its close neighbors. If there are no nodes to add, we declare a new object and repeat the previous procedure, until all the nodes are labeled(see Figure 4).

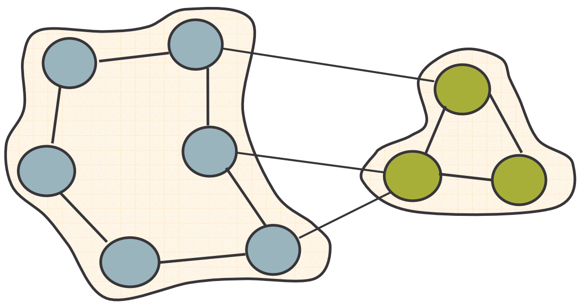


Figure 4. Organization of pixels into objects.

* Next we describe the object motion detection step.

## Motion vector

Each object consists of the nodes representing the pixels lying on its boundary. Each node holds a list of its far neighbors – all the possible movements for this pixel (we refer to the far neighbors with a similar color). We would like to find the motion of the object which is determined by the major motion of its pixels. Thus, we generate a list of vectors which is the joining of all motion vectors of all pixels and we calculate their "main" direction. We implement two methods to find a motion vector – clustering and accumulation.

## Clustering

In this method we divide all vectors into clusters, and determine the motion vector of the object as the center of the largest cluster. We compute the clusters in the following way: in the initialization, each vector forms a cluster. Then, we pick the two closest clusters, and if the union of these clusters is legal (not too large), we merge them. We repeat this procedure until there are no available clusters to merge. The center of the cluster that includes the maximal number of vectors is the motion vector of the object (see Figure 5).

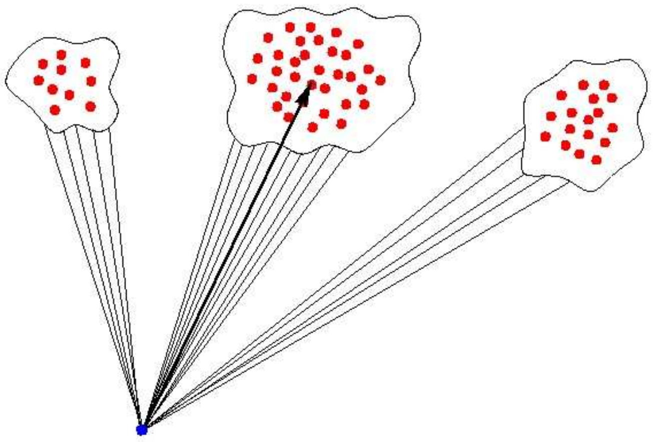


Figure 5. Vector clustering.

## Accumulation

This method is similar to the Hough transform for line detection. Each motion direction has a bin (initialized by zero). Each node "votes" for each possible direction (increases by one the value of the appropriate bin). The object motion vector is the vector with the maximal value in its bin. We implement also a variation of this method in which each node has only one vote equally divided between all its possible vectors.

Searching for a bin with the maximal value can be done in several ways:

- Global maximum – find the maximal value among all bins.
- Region maximum – find the value, such that the sum of all the values in its surrounding small region, is maximal.
- Weighed region maximum – find the value, such that the weighted sum of all the values in its surrounding small region, is maximal. The weights in the region are defined in the following way. The region center gets the maximal weight, the farer (from the center) the bin the smaller weight it gets.

## Results

We have implemented this project in C++ and tested it on different images. The test images can be divided into three groups: simple images, cartoons, and real images. First, we present the results of the edge detection step. Figure 6-9 show (a) original color image, (b) image of the found edges, where each edge has its grayscale color, and (c) image of the found edges shown by white color.

Next, we present the results of motion detection step on different images. Figure 10 and 11 show the result of motion detector for simple images: (a) first original image, (b) second original image, and (c) motion directions of the objects.

Figures 12 and 13 show the result of motion detector for cartoons images, while Figure 14 shows the result of motion detector for real images: (a) original images, (b) images of edges, (c) image of motion vectors.

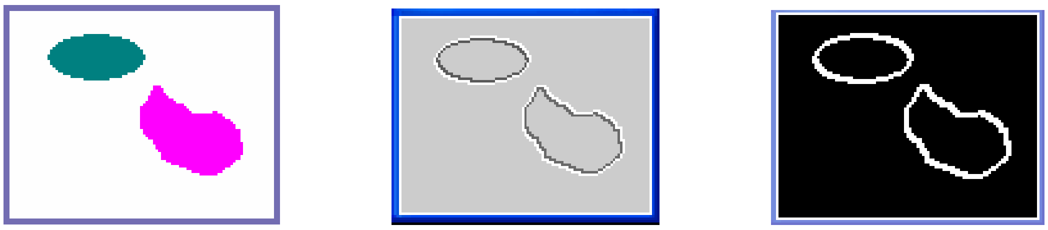
We have implemented two methods for motion vector calculation: clustering and accumulation, and three ways to find the major value. After running several tests we have found that the results are quite similar for all the methods. There is one obvious disadvantage of the clustering method – its high runtime comparing to the accumulation method.

## Conclusions

We have developed and implemented an approach to detect motion vectors of objects in images. This approach is useful to find a small motion that occurs in two consecutive frames of video segment. We have found that for simple images (synthetic or cartoon) we get good results. However, for more complicated (real) images we get false vectors in addition to the real motion vectors (as a result of the noise and the nature of our approach). Our approach is sensitive to changes in the object shape. We have not found significant advantage between clustering and accumulation methods, except in running time, where accumulation is faster.

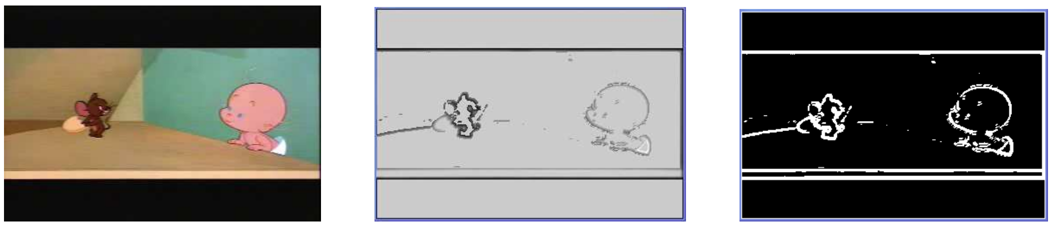
## References

A.K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. ACM Computing Surveys, 31(3):264-323, September 1999.



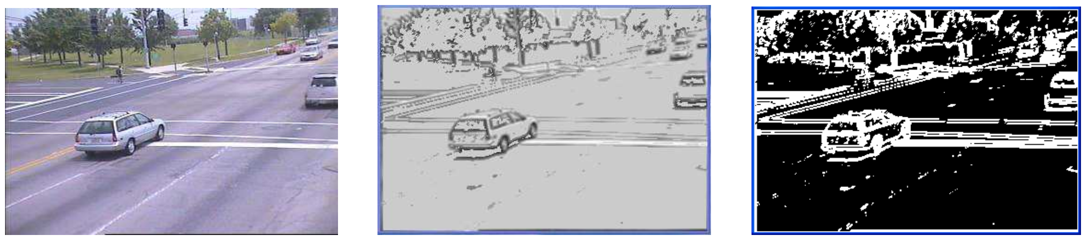
(a) (b) (c)

Figure 6. Edge detection of simple image of parts.



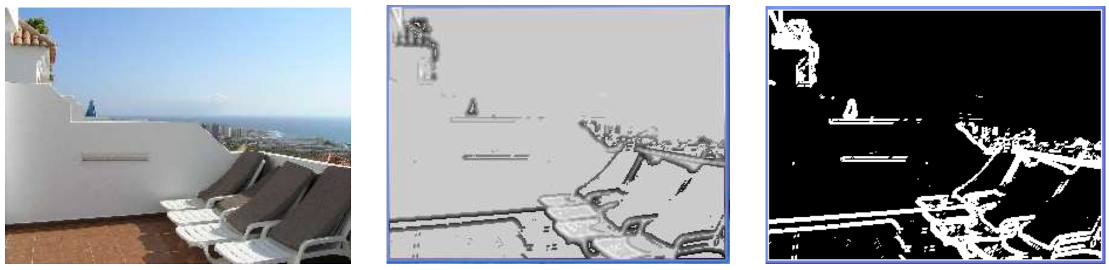
(a) (b) (c)

Figure 7. Edge detection of cartoon image.



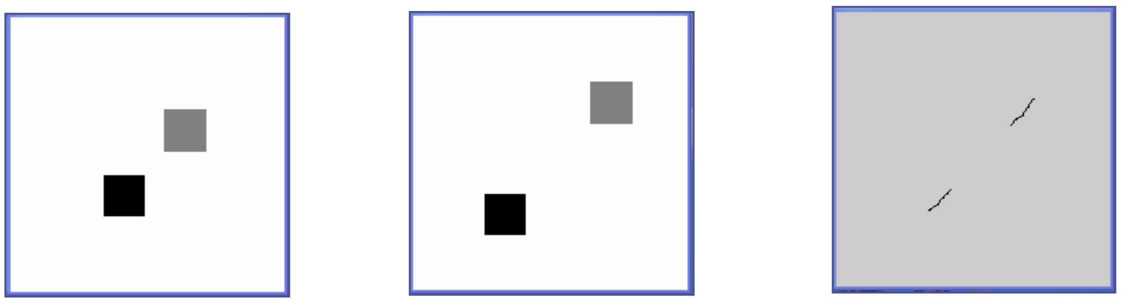
(a) (b) (c)

Figure 8. Edge detection of real image of a car.



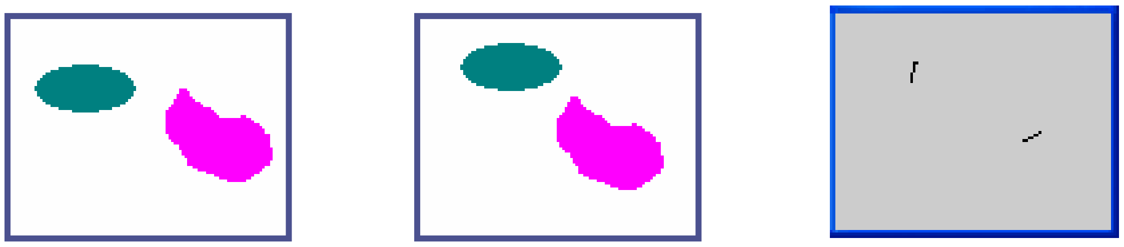
(a) (b) (c)

Figure 9. Edge detection of real image of a balcony.



(a) (b) (c)

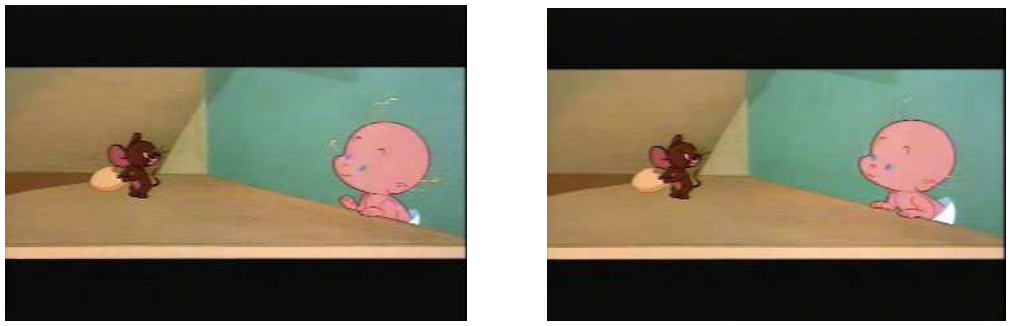
Figure 10. Motion detection for a simple image with two squares.



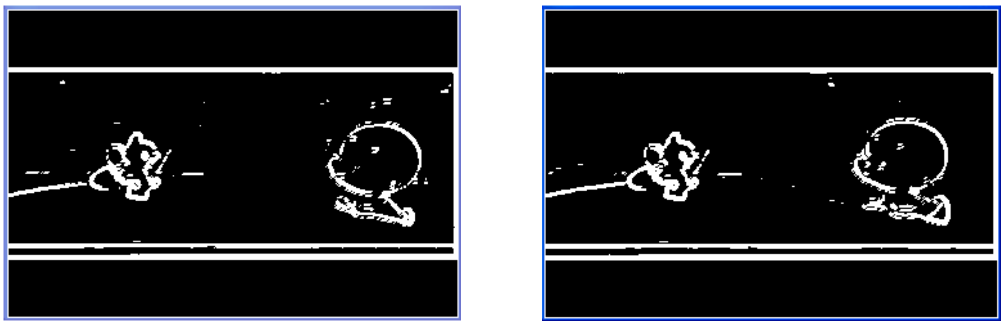
(a) (b) (c)

Figure 11. Motion detection for a simple image.

(a)



(b)



(c)

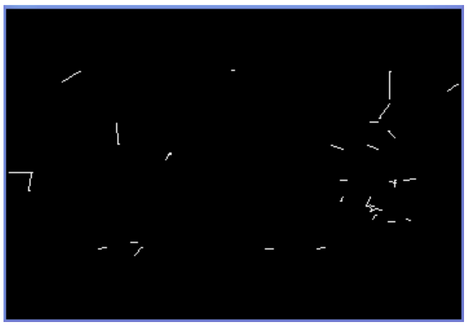
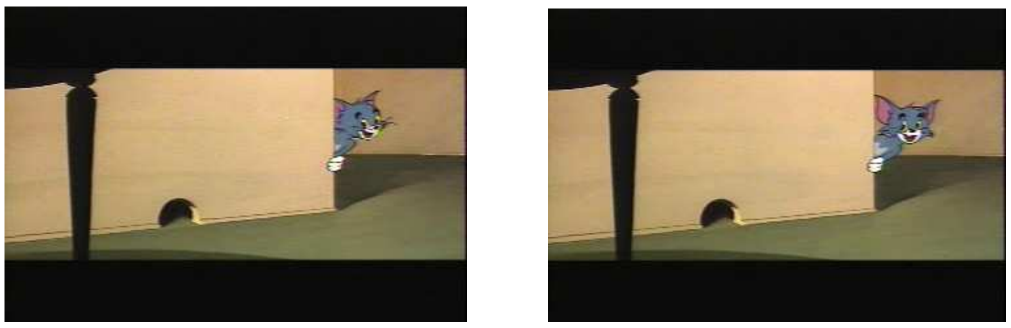
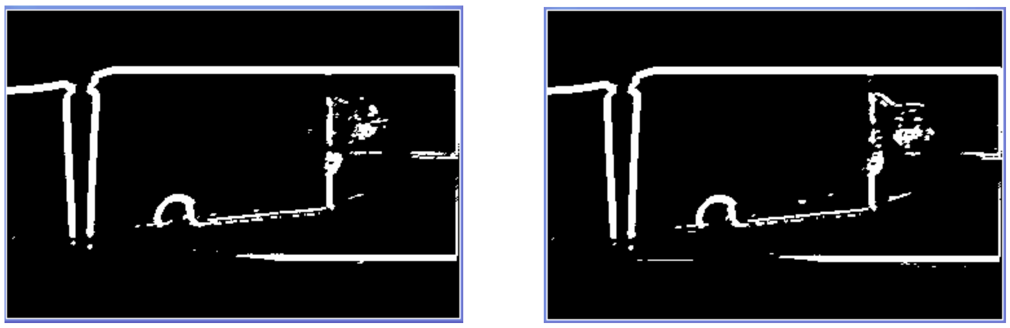


Figure 12. Motion detection for a cartoon image of baby.

(a) 

(b) 

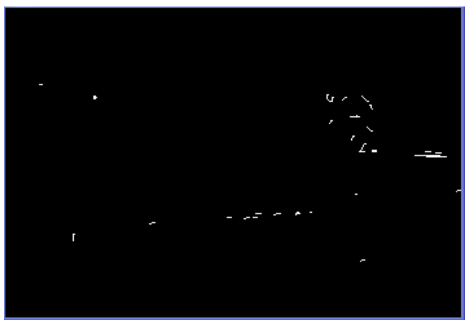
(c) 

Figure 13. Motion detection for a cartoon image of cat Jerry.

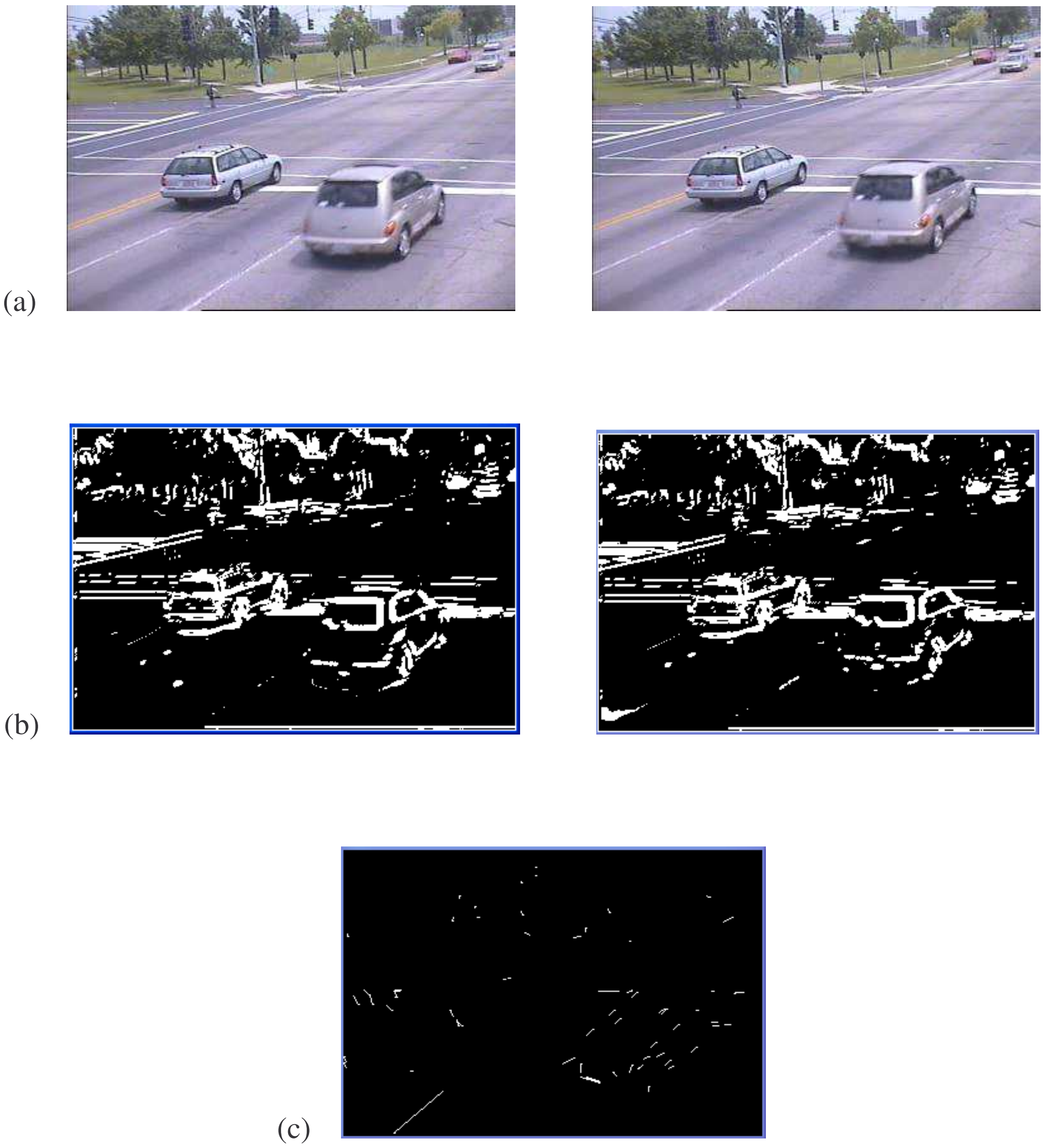


Figure 14. Motion detection for a real image of cars.