Development and Testing of Algorithms for Vehicle Type Recognition and Car Tracking with Photo and Video Traffic Enforcement Cameras

S. M. Staroletov[a],*, M. A. Laptev[a],**, and D. V. Nekrasov[a],***

[a] *Polzunov Altai State Technical University, Faculty of Information Technologies, Department of Applied Mathematics, Barnaul, Altai krai, 656038 Russia*

** e-mail: serg_soft@mail.ru*

*** e-mail: m.laptev93@gmail.com*

**** e-mail: scannorone@gmail.com*

Abstract—The work is devoted to the research that was carried out within the framework of computer vision problems applicable to the analysis of images and video information with vehicles. We solve the problem of classifying vehicles. We analyze the drawbacks of Haar features and convolutional neural networks and test the obtained networks using the key point method; we construct an integral algorithm that includes several networks, and we further validate it on a large number of real photographs and types of vehicles. Next, we solve the task to develop a software framework for tracking vehicles by analyzing adjacent photographs from a video sequence. After that, we consider the tracking task in more detail. We analyze modern tracking algorithms using machine learning and describe our implemented tracker with support for the appearance of obstacles between the camera and a moving vehicle. As a result, we propose algorithms and open-source software that, after being configured for specific cameras, can be used in traffic analysis systems.

Keywords: classification, tracking, convolutional networks, integral algorithm, neural network testing, KCF, traffic camera

DOI: 10.1134/S1054661821020152

1. INTRODUCTION

Nowadays, in the service of traffic inspectorates, compute appliance systems for automatic recording of administrative offenses in the field of road safety have appeared. These systems allow one to monitor especially dangerous sections of roads and record various offenses: speed-limit violations, parking-rule violations, traffic-rule violations at intersections, etc. The identification of the vehicle with which the offense was committed is carried out by recognizing the symbols of the vehicle registration plate (a survey on recognition methods is given in [1] and examples of applicable algorithms in article [2]). Thus, if the registration plate belongs to another car, the offender will avoid liability. To solve the problem, the system of identification of the brand and model of vehicles, which partly solves this problem, is one of the topics of the present study. We believe that identifying the brand and model of vehicles from traffic enforcement cameras will allow detecting cars moving on public roads with deliberately forged numbers, which can be useful when searching for stolen cars and those that left the scene of an accident. With the development of artificial neural networks designed to distinguishing features in images, the process of determining the brand and model consists in training and subsequent operation of such a network [3, 4]. Using such a method, it is possible to obtain an accuracy higher than the methods of the SURF class [5] in an acceptable time, which is applicable to the analysis of images from servers of road compute appliance systems in the background process or upon given requests.

Consider the task of motion detection. This task is devoted to find moving objects in the video for further work with these objects. A review on the topic was published in [6, 7]. In papers [8, 9], methods for solving this problem are given in relation to video surveillance of the movement of vehicles. In such video-surveillance systems, the camera is static and does not change its angle, the background is inactive, and therefore background subtraction is the most effective method for detecting motion [10, 11]. As a result, we get a bounding rectangle of a moving object, or a pixel-by-pixel mask of the object. This approach will allow detecting traffic from traffic cameras and assessing vehicle speed.

To solve the problem of tracking objects in a video stream, the developer community currently uses algorithms based on classical machine-learning methods, such as building a linear classifier [12], support vector machines [13], and a random forest [14]. Compared to machine-learning methods based on convolutional neural networks [15], such algorithms do not require preliminary training, they build a classifier model

Received March 17, 2020; revised January 13, 2021; accepted January 16, 2021

based on one frame from a video stream and then update this model. In addition, they have high performance, which allows them to be used in real-time systems. For tracking purposes, we observe an important task to propose the modification of algorithms, which includes the appearance of short-term obstacles between the camera and a moving object, for example, overlapping one moving car with another. An overview on this topic is presented in [16].

This article summarizes several works performed by us in the study of cyber-physical systems [17], this time from the point of view of computer vision algorithms and self-teaching methods in relation to the analysis of the traffic situation. When writing this article, we are considering the key points of such systems, namely, motion detection, tracking and classification of vehicles, and implementing software that uses modern libraries, such as OpenCV, that clearly do not contain the required algorithms. We also set ourselves the goal of designing the software architecture in such a way that it was customizable and extensible. To do this, we apply software design patterns and follow the concept of software frameworks [18]. The main contribution of this work is the implementation and description of (1) an integral algorithm for determining the types of vehicles (2) a software framework for detecting motion, and (3) an add-on for the KCF tracking algorithm with overlapping.

The structure of this article is as follows: in Section 2, we consider issues of vehicle classification, including testing neural networks and developing an integral algorithm; in Section 3, we describe a software framework for motion detection and vehicle speed analysis by background subtraction; Section 4 deals with object tracking and the extension of the KCF algorithm with obstacles between the object and the camera, and Section 5 draws conclusions and provides links to our software.

2. SOLUTION OF THE VEHICLE CLASSIFICATION PROBLEM

2.1. Analysis of Data Coming from Road Computer Appliance Systems

At the disposal of centers for analyzing the road situation, there are stationary and mobile computer appliance systems for obtaining and processing traffic photo and video data produced by various independent companies. Therefore, each of the complexes has its own communication interface and method of storing information, for example, they can use MSSQL and PostgreSQL database management systems to store photos or save a photo and a digitally signed key along with information extracted from the photo to files. According to the results of our study, it was found that none of the available complexes provides an interface for working with a video stream. Thus, it is possible to work only with photographs taken in automatic mode. In this section, we only consider photographs in which it was possible to find a rectangle of the state vehicle registration plate (SRP). For example, according to the State Standard of the Russian Federation GOST R 50577-2018 [19], the length and width of the SRP for both passenger and cargo vehicles is 520 and 112 millimeters, respectively. Since the presence of a license plate in a photograph is mandatory and its physical size is standardized, it was customary to take the license plate as a basis and estimate the dimensions based on the size of the plate.

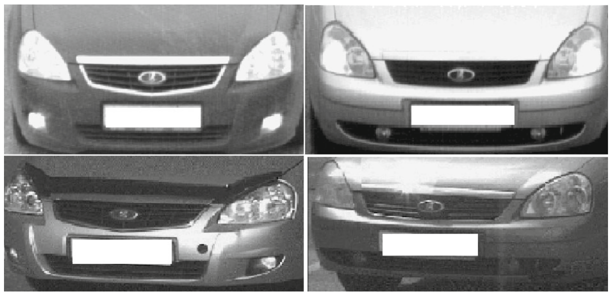


Fig. 1. Tutorial examples for Haar cascades.

2.2. Study of a Classifier Based on Haar Cascades

One of the methods of image recognition is the use of cascades of Haar features [20]. In the classical representation, the algorithm does not support multiclass classification so we applied a one-against-all approach to work with photos containing vehicles. That is, we trained a separate cascade for each vehicle class, and used positive examples of other classes as negative examples for each specific class. Thus, a well-functioning cascade should take images of other classes as the background. The multiclass classification consists in sequential search for cars in the image by different cascades.

For the experiment, we chose one desired class “Lada Priora” and 11 classes of cars, the photos of which were negative for the desired class. We selected cut-out images of the front part of Lada Priora cars as positive examples, an example is shown in Fig. 1. On the other hand, we used raw photographs of other vehicles from photo cameras as negative examples.

During the experiment, we revealed a large number of shortcomings of the considered algorithm: insufficient accuracy, a large number of false positives, as well as double positives. We suggest that the image of the whole car is too complex to be classified using a set of Haar cascade rectangles. Therefore, we made the decision to test the Haar cascades for a simpler task: classifying car nameplates. For this experiment, we prepared a sample containing ten classes of nameplates, 200 photos in each class. The experiment involved nameplates of the following makes: Nissan, Opel, Mazda, Honda, Volkswagen, Mercedes, Audi,

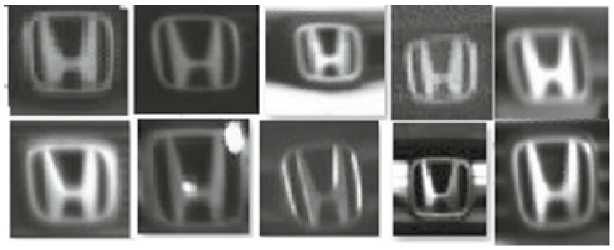


Fig. 2. Tutorial examples for the classification of nameplates with Haar cascades.

Toyota, Renault, Lexus. Examples from the training sample are presented in Fig. 2.

For each of the ten classes, we trained its own cascade. Then, to assess the classification accuracy, we sequentially recognized 1000 photos by each of the cascades. As a result of the experiment, the multiclassification accuracy was 65%. Here, accuracy should be understood as the ratio of correctly classified photos to the total number of photos, where a correctly classified photo is a photo in which the only image of the nameplate was found, and the class of the found nameplate is correct. A result of 65% for multiclassification is much more significant than a result of 66.8% for a binary classification; however, this accuracy is still not enough for use in real conditions.

2.3. Research of Artificial Neural Networks

The purpose of the first study of neural networks was to test their applicability for vehicle recognition. For this, we prepared a small data set (dataset) consisting of two parts. The first part is a training sample of 5000 photographs of five different classes. The second part is a validation set consisting of 2000 photographs of the same classes. Thus, for each of the five classes, we prepared 1000 photographs for training and 400 photographs for checking accuracy. All photographs were taken with real traffic enforcement cameras and are subject to the only modification: cropping of the black frame containing information about the offense.

For all experiments with convolutional neural networks, we used the Keras framework [21] with two different backends: Theano [22] and TensorFlow [23]. Physically, the calculations were carried out on an Nvidia GTX 970 GPU using the CUDA platform.

Currently, there are a large number of neural network architectures, among which we selected the most advanced and suitable for the task of image classification. The final list includes:

1. Inception V3 [24] from Google.

2. Resnet50 from Microsoft [25].

3. Xception from the creator of the Keras framework Francois Hall [26].

4. VGG from Oxford representatives [27].

The accuracy of the first experiment with Inception V3 was 89.2%; that is, 1784 of 2000 photographs of the validation sample were correctly recognized. We used the standard hyperparameters and the RMSprop optimizer. Thus, at that time it was the best result among all selected algorithms, and we continued experiments with this architecture. As a result of enumerating optimizers, batch sizes of simultaneous training, and the number of training epochs, we found the optimal parameters. The maximum accuracy was 96.9% in the case of training at least 80 epochs with the Adam optimizer and batch_size parameter equals to 16.

To experiment with the ResNet architecture, we chose the ResNet50 network, which contains 50 layers and fits into the memory of the GPU. Training took place on the same dataset; and we used the RMSprop optimizer. As a result of the experiment, the classification accuracy was 89%.

The Xception architecture is a popular modification of the Inception network, where some of the blocks are replaced with deep split convolutions. The author claims that such a network has a slightly higher recognition accuracy than the original Inception V3. However, in this particular case of black-and-white images from a traffic camera, Inception could not be surpassed. The final accuracy at 50 epochs with the Adam optimizer was 93.5%. At the same time, the training process looked similar to Inception, but the training time turned out to be 1.5 times longer.

VGG is a very deep convolutional network architecture from Oxford. The network was developed specifically for the ImageNet competition and has two implementations, VGG16 and VGG19, with 16 and 19 layers, respectively. Networks are very demanding on computational resources; therefore, using the available capacities, it was possible to start training VGG19 with a batch of only two photos and VGG16 with a batch of four photos. For training, we carried out several approaches with additional training for 30 epochs of each of the network; nevertheless, the recognition accuracy did not move from 20%.

Based on the results of the experiments, we chose convolutional neural networks of the ResNet and Inception architectures as the basic units of the integral algorithm described below. We developed a mechanism for determining not only the type of vehicle but also a specific brand using a single neural network.

2.4. Testing Networks Using Keypoints

To understand the principle of operation of neural networks and optimize the input data, we carried out an analysis of key points used by neural networks. This experiment is consistent with the method proposed in [28].

Experiment plan:

1. Choose one well-recognized photo, keep the recognition result.

2. Erase some of the information on the photo (put a black square on the image).

3. Recognize the resulting image using the same neural network.

4. Compare the recognition result with the original one. Fix the location of the square and the deviation of the result.

5. Nudge the square a pixel.

6. Repeat steps 2–5 for all possible areas.

7. Find the maximums among all deviations. The corresponding areas are the key points.

The study found that the deep neural network pays the most attention to the front of the car and almost completely ignores the background. Therefore, we considered inappropriate preliminary cropping and processing of photographs.

In addition, we trained a neural network to respond not to the brand of the car but to the body style. The result of correct recognition on the validation set is 47%. The reason for this result is that most of the photos in the sample contain only the front of the car. At the same time, the key points of the new network have practically remained the same. Thus, the neural network "tried" to recognize the brand anyway and then issue the body style corresponding to the brand. This experiment showed the inexpediency of separate recognition of the car body style.

2.5. Development of an Integral Algorithm

Cars are a difficult object to classify, even for humans. It often happens that cars have been reworked, or have traces of an accident. All this complicates the process of determining their brand. We have already noted that a modern approach to solving this problem is the use of convolutional neural networks. Each neural network trained on N classes has N outputs, on each of which is the probability of an example belonging to a specific class. To provide the user with a recognition result, the largest of these probabilities is selected. However, if two or more classes have the same or close probabilities, there is a great chance of making a mistake and giving the user an incorrect result. To solve this problem, we have developed an integral algorithm that combines several networks of different topologies at once and gives the final result based on the weighted opinion of these networks. The vehicle is characterized by an identifier, which consists of a tuple (1):

$$ID = (Mark \times Model \times Year \times Type) \tag{1}$$

Here, *Model* is the car model, *Mark* is the manufacturer's brand, *Year* is the year of manufacture or generation of the car, *Type* is the body style.

We denote a set of images of cars by *Img*. The considered integral learning algorithm consists of several subalgorithms for highlighting the key features of the vehicle. For the algorithm, it is necessary to obtain the target regions of the image depending on the region of interest (*ROI*) as well as the set of *Filters* necessary to accurately select these regions. Applying a chain of filters and an area of interest to the image, we get the region of interest as a composition:

$$Region_i = ROI_i o Filter_{i1} \ldots o Filter_{in} o Img \tag{2}$$

Each subalgorithm receives an image and a target region as input and, returns a set of numerical features (metrics) for the original image as a result.

$$Alg_i : Region_i \times Img \to (Metric_i)^* \tag{3}$$

In the next step, these metrics become the basis for training neural networks.

$$(Metric_i)^* \times ID \to Net_i \tag{4}$$

We propose to use the following metrics:

1. Binary track of the front or rear of the car, including headlights or lights, grille, bumper.

2. Distances from the license plate to the edges of the trunk or bonnet and headlights or lamps, given in centimeters based on the specific license plate size.

3. Binary trace of the manufacturer's logo.

4. Data on the year of manufacture, make, model, country of the car manufacturer.

The use of one network is impractical, since the classification problem is complicated by the similarity of cars and errors as a result of determining metrics from the original images.

During recognition, a neural network from the image returns several values of the found vehicle identifiers and the probability of the correctness of the following determination:

$$Net : Img \to (ID \times P)^* \tag{5}$$

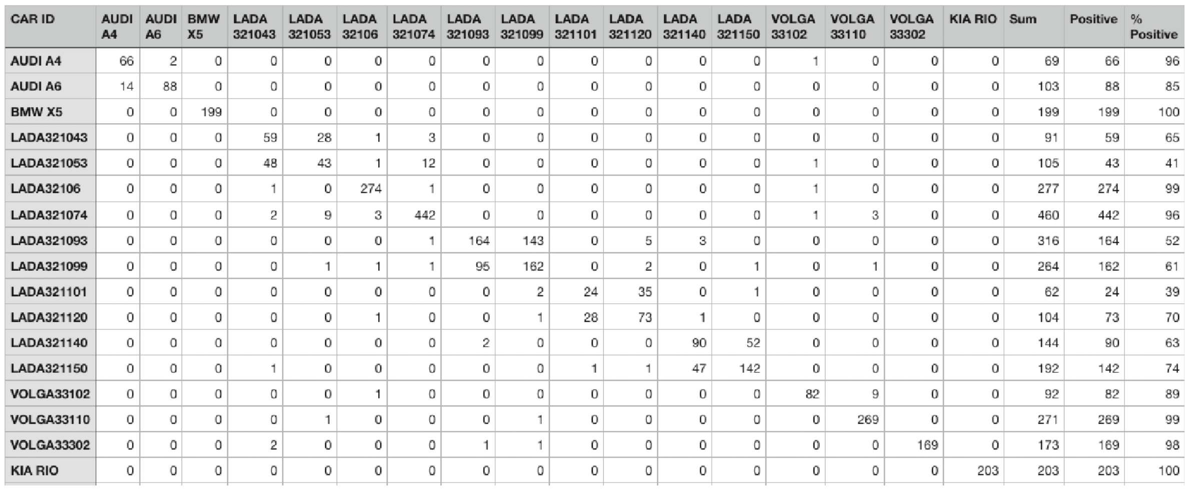
The integral algorithm uses several neural networks that work in parallel and return classes of machines and probabilities.

We use an integral weighted formula to determine the resulting class. Let several networks return some detection probabilities and identifiers:

$$\begin{gathered} Net_1 \to \{p_{11} \times id_1,\ p_{12} \times id_2, \ldots p_{1n} \times id_n\} \\ Net_2 \to \{p_{21} \times id_1,\ p_{22} \times id_2, \ldots p_{2n} \times id_n\} \\ \ldots Net_k \to \ldots, \{id_1, id_2, \ldots, id_n\} \in ID \end{gathered} \tag{6}$$

The set of identifiers returned by different networks is identical; therefore, by applying weighted, empirically selected coefficients w_i showing the weight of each network in the final integral algorithm, we get

$$\begin{gathered} ID \times P_w = \{id_1 \times (w_1{*}p_{11} + w2{*}p_{21} + \ldots w_k{*}p_{k1}), \\ \ldots,\quad id_1 \times (w_n{*}p_{1n} + w2{*}p_{2n} + \ldots w_k{*}p_{kn})\} \end{gathered} \tag{7}$$



CAR ID	AUDI A4	AUDI A6	BMW X5	LADA 321043	LADA 321053	LADA 32106	LADA 321074	LADA 321093	LADA 321099	LADA 321101	LADA 321120	LADA 321140	LADA 321150	VOLGA 33102	VOLGA 33110	VOLGA 33302	KIA RIO	Sum	Positive	% Positive
AUDI A4	66	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	69	66	96
AUDI A6	14	88	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	103	88	85
BMW X5	0	0	199	0	0	0	0	0	0	0	0	0	0	0	0	0	0	199	199	100
LADA321043	0	0	0	59	28	1	3	0	0	0	0	0	0	0	0	0	0	91	59	65
LADA321053	0	0	0	48	43	1	12	0	0	0	0	0	0	1	0	0	0	105	43	41
LADA32106	0	0	0	1	0	274	1	0	0	0	0	0	0	1	0	0	0	277	274	99
LADA321074	0	0	0	2	9	3	442	0	0	0	0	0	0	1	3	0	0	460	442	96
LADA321093	0	0	0	0	0	0	1	164	143	0	5	3	0	0	0	0	0	316	164	52
LADA321099	0	0	0	0	1	1	1	95	162	0	2	0	1	0	1	0	0	264	162	61
LADA321101	0	0	0	0	0	0	0	0	2	24	35	0	1	0	0	0	0	62	24	39
LADA321120	0	0	0	0	0	1	0	0	1	28	73	1	0	0	0	0	0	104	73	70
LADA321140	0	0	0	0	0	0	0	2	0	0	0	90	52	0	0	0	0	144	90	63
LADA321150	0	0	0	1	0	0	0	0	0	1	1	47	142	0	0	0	0	192	142	74
VOLGA33102	0	0	0	0	0	1	0	0	0	0	0	0	0	82	9	0	0	92	82	89
VOLGA33110	0	0	0	0	1	0	0	0	1	0	0	0	0	0	269	0	0	271	269	99
VOLGA33302	0	0	0	2	0	0	0	1	1	0	0	0	0	0	0	169	0	173	169	98
KIA RIO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	203	203	203	100

Fig. 3. Integral algorithm recognition matrix.

To speed up work with a large value of vehicle types for detection, we can use only a few maximum probabilities and the corresponding classes returned by neural networks.

After calculating the weighted probabilities for all vehicle identifiers, we select the identifier with the highest obtained probability:

$$id = Max_p\left(ID \times P_w\right) \tag{8}$$

To implement the integral algorithm in relation to the problem under consideration, we selected the networks that showed the best results on the validation set. The final implementation consisted of two InceptionV3 networks trained with different optimizers and a ResNet50 network. The result of each of the neural networks contributes to the final result with a coefficient fitted using linear regression.

2.6. Development and Testing of a Prototype Application That Implements the Integral Algorithm

Having thus decided on the architecture, we began on creating a working prototype. We prepared a sample of 100 000 practice and 25 000 validation photos from 115 of the most common car models. This sample was less balanced than the previous one and each brand had contributed a different number of photographs from 300 to 5000. In addition, there was no preprocessing and premoderation, for example, for overlapping photos or night shots. Thus, the conditions are as close as possible to real ones.

The accuracy of the resulting neural network on the validation set was 88%. To analyze the results, we built a recognition matrix. A fragment of such a matrix is shown in Fig. 3.

Analyzing the results, we see that the problem with the recognition of similar cars from the same manufacturer, in this case, especially of domestic production, becomes obvious. Usually, only the front part of the car is shown in the photo, so it is virtually impossible to distinguish, for example, Lada 21093 from Lada 21099. To solve this problem, we decided to combine indistinguishable brands into one class. Then we trained the network again on the resulting sample of 110 brands. As a result, we managed to achieve the result of 91.6%. This result can already be considered suitable for use in real conditions. The top five metric was 97.2%.

3. MOTION-DETECTION AND VEHICLE TRACKER FRAMEWORK

3.1. Formulation of the Problem

In this section, we move from the task of classifying vehicles by static images to the task of tracking them from traffic cameras.

The tracking task is to build the trajectory of the necessary objects on the input sequence of images. In the tasks of detecting motion and tracking objects, there are many different implementations, various characteristics, and parameters that can be varied depending on the problem being solved. So we propose to implement an object-oriented framework with which it would be possible to solve such problems. A framework is a software platform that defines the structure of a software system and facilitates the development and integration of various components of a large software project. One of the first proposals for such a merger was made by IBM in 1987 [29]. A framework differs from the concept of a library in that a library can be used in a software product simply as a set of subroutines of similar functionality without affecting the architecture of the software product and without imposing any restrictions on it, while the framework dictates the rules for building the architecture of the application, setting the default behavior at the initial stage of development. The framework can be extended and changed according to the specified requirements.

A motion-detection framework should be able to provide the following:

(1) work with streaming video or video stored on disk;

(2) set the area for motion detection;

(3) to detect motion, use the background and sequential image subtraction methods implemented in the framework, independently setting and changing some algorithm parameters, depending on the task (different filter chains, binarization threshold, etc.);

(4) highlight the detected object with a rectangle or outline;

(5) apply various filters for video frames, for example, such as median filter, blur and binarization;

(6) save chains of filters as presets;

(7) adapt the algorithm for the users by writing their own implementations of some methods; and

(8) implement interfaces for working with detected objects, for example, for object recognition.

To demonstrate the capabilities of the framework, we propose to implement the "Vehicle Tracker" application, which will count the number of cars by lanes, as well as their speed. In addition, the application should be able to select a motion-detection method and, in addition, provide flexible filter settings for the selected method.

3.2. Background Subtraction Method

The background subtraction algorithm implemented in the framework consists of four steps:

Step 1. Take the first frame of the video, which will later be the background.

Step 2. Find the absolute difference between the current frame and the background for each subsequent frame.

Step 3. Apply a chain of some filters for the obtained difference (the chain of filters depends on several factors, for example, the size of the detected objects, lighting conditions, etc.). After applying a chain of filters, one will get a binarized image, where the background is black and the foreground is white. The foreground objects will be our detected objects.

Step 4. Repeat steps 2 and 3 until the video ends or the user stops the algorithm.

Through our experiments, we selected the following chain of filters, which is applied to the absolute difference between the current frame and the background: binarization with an average threshold, median filter, blur, and binarization with a low threshold. This approach is consistent with the double binarization approach proposed in [30]. Binarization thresholds are algorithm parameters and are set in the application interface when calibrating for a specific camera that is the source of images.

3.3. Sequential Image Method

The sequential image algorithm implemented in our framework consists of the following steps:

Step 1. Take two consecutive frames of video.

Step 2. Find the absolute difference between frames.

Step 3. Apply a chain of filters to the resulting absolute difference between the two frames. We get a binarized image with selected moving objects.

Step 4. Repeat steps 2 and 3 until the video ends or the user stops the algorithm.

For the method of sequential images, we experimentally selected the following filter chain : binarization with a low threshold, blurring, and binarization with a low threshold.

3.4. Vehicle Tracker

To demonstrate an example of using the framework, we developed the "Vehicle Tracker" application. The application counts the number of vehicles in each lane as well as the speed of each vehicle. In addition, the tool provides the ability to change the motion-detection method while the program is running, as well as to edit filter chains for the selected method. Fig. 4 shows the operation of this application on a demo video, which is processed frame by frame.

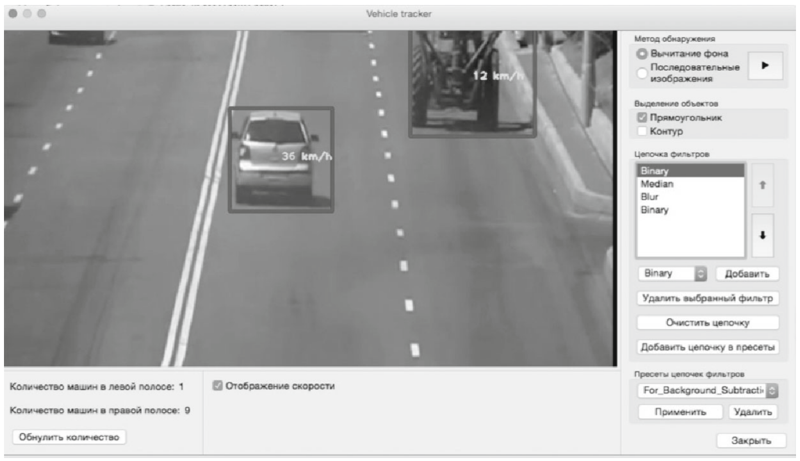
The cars were counted relative to the lower frame border. For each frame, we store how many cars are currently on the frame. For each new frame, we check whether the detected object has appeared at the border; if so, then we count their number and increase the number of cars in this lane by this value. This approach is consistent with the approach proposed in [31]. The number of stripes and their sizes are set during software configuration depending on the shooting parameters of a particular camera. Since we are counting the number of cars in lanes at the bottom of the frame, we do not need to account for perspective distortion. If the camera is hanging on the side of the road and one wants to count objects not on the lower boundary, then one can use the complicated model proposed in [32].

The speed is calculated according to the following algorithm:

(1) For the first frame with detected objects, remember the bounding boxes of each object.

(2) For subsequent frames with detected objects, generate a list of bounding rectangles for each object.

(3) For each bounding rectangle on the new frame, find the closest rectangle on the previous frame. The distance between rectangles is defined as the absolute



Метод обнаружения
[✓ Вычитание фона]
Последовательные изображения

Выделение объектов
☑ Прямоугольник
☐ Контур

Цепочка фильтров
Binary
Median
Blur
Binary

Binary

Пресеты цепочек фильтров
For_Background_Subtracti

Количество машин в левой полосе: 1
Количество машин в правой полосе: 9

☑ Отображение скорости

Fig. 4. Vehicle tracker application.

difference between the coordinates of the centers of the rectangles.

(4) If the distance between the rectangles is less than the specified threshold, then we find the position of the object of the previous frame on the current one. Now we need to convert the distance in pixels to meters. Knowing the width of the marking strip in centimeters and pixels, we can compose the proportion and find how many centimeters corresponds to one pixel.

(5) Knowing the frame rate, we find the time it takes to change frames, and then divide the distance traveled by the car, in meters, by the value of this time, and find the speed of the car between two frames.

The considered method makes it possible to determine the speed of vehicles from road cameras without using speed radars. The accuracy of determining the speed in this way depends primarily on the correct calibration of the algorithm according to the parameters of the camera frame rate and the conversion of pixels to centimeters. For the camera hanging directly over the lanes, when determining the speed of the car at the bottom of the screen, we can achieve almost absolute accuracy, which will decrease as the car moves away from the camera (without taking into account perspective distortions), but this is sufficient to save a certain speed value into the database.

4. KCF BASED OBJECT TRACKING WITH OBSTACLES

4.1. Analyzing Object-Tracking Problems

Over a long time, the object-tracking system based on the basic algorithms can degrade. To maintain the correct operation of such systems, approaches based on empirical observations of a specific system are used [9]. For example, in some systems where the camera is known to be static, background noise may be present due to slight changes in lighting or swaying trees. In this case, we used a technique based on the chains of filters applied for each frame of the video stream. However, over a long period of time, the background can change significantly, which will lead to errors in the tracking algorithm. In this case, reloading of the algorithm with the construction of a new background model can be used [33]. This approach has been implemented and described above.

Tracking algorithms based on machine-learning methods, in particular, on methods of object classification, help to avoid the above problem. These algorithms do not degrade their performance over time, since they use new information obtained from previous frames and rebuild the model online for the training stage. To build the most accurate models, deep-learning methods based on convolutional neural networks are sometimes used, however, these methods are not always suitable for tracking tasks, since their work requires large amounts of labeled data, which the developer of the tracking system may not have.

It should also be mentioned that most of the tracking algorithms are tailored for tracking one object. This fact does not allow us to use these algorithms to solve a number of problems, for example, car lane change, tracking football players during a football match, or constructing trajectories of products when they are sorted in an electronic color sorter [34]. It should also be mentioned that many tracking algorithms do not cope with obstacle handling: the temporary overlap of the tracking target by other objects. Examples of obstacles here are pedestrians blocking the tracking target, trees, road signs, passing cars, lamp posts, etc. Handling such obstacles is an important feature of the tracking algorithm. After the analysis, we formulate the following requirements for the object-tracking algorithm:

1. Real-time work (the ability to process frames received from the camera without gaps).

2. Unlimited tracking duration.

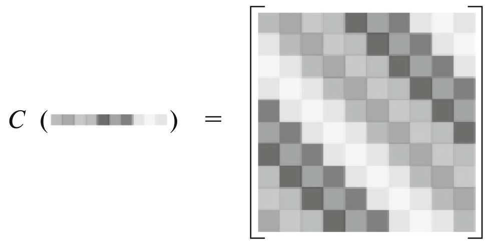


Fig. 5. Visualization of the circulant.

3. Using machine-learning algorithms to classify objects.

4. Lack of the pre-training phase.

5. The ability to track multiple objects in the frame.

6. Handling obstacles.

4.2. KCF Algorithm

One of the main methods for tracking objects in a video stream based on classical machine-learning algorithms is the use of a discriminatory classifier [35], which allows assigning an image fragment (patch) to one of two classes: object or background.

Each new detection of an object on new frames allows us to supplement and update the model. However, for the correct operation of the classifier algorithm, we need not only positive patches, which represent an object, but also negative ones, which must be classified by the algorithm as a background. The most commonly used negative examples are patches from different places in the image at different scales. Since there are practically an unlimited number of such patches per frame, modern algorithms based on the discriminating classifier are forced to use only a few such patches (from three to 15). In addition, these patches may overlap in the image, in which case they will have redundant information.

The authors of the KCF algorithm believe that the number of negative patches is the main factor hindering tracking efficiency. The main idea behind the KCF algorithm is to use several thousand patches without overlapping. The use of so many patches was made possible by the discovery that, in the field of signal processing (Fourier space), some machine-learning algorithms become simpler when more data is added by using specific models for translation. The circulant matrix [36] (Fig. 5) acts as such an analytical model.

The circulant is the bridge between popular machine-learning algorithms and classical signal processing algorithms. It is proposed to use a ridge regression tracker, which can be represented as the Kernelized Correlation Filter (KCF).

Comparing the complexity of the simple patch extraction algorithm and solving the regression problem, we can see that the solution to the ridge regression problem has a computational complexity of $O(n^3)$, since it is necessary for its solution to use the operation of matrix multiplication, which has the given asymptotics. On the other hand, all operations in equations based on linear regression, circular, and Fourier transform are elementary and have linear complexity $O(n)$, with the exception of the discrete Fourier transform, which has the asymptotics $O(n*\log(n))$. For data volumes used in practice, the transition to the signal processing space allows us to reduce the amount of stored data and also increases performance by several orders of magnitude.

When implementing the KCF algorithm, we used the material given in the original article describing this algorithm [37]. In addition, we reclaimed the source code of the OpenCV Tracking API as well as an open source implementation of the algorithm distributed under the BSD license [38].

4.3. Implementation of Tracking with Obstacle Handling

The KCF algorithm calculates an object detection score at every frame. If this estimate is less than the threshold, we can say that the object has disappeared from the frame or hidden behind an obstacle. The obstacle-processing algorithm is that, after the object has disappeared from the frame, we try to detect it in the next few frames in the vicinity where it disappeared. To do this, template matching algorithms are used to search for a template found in one of the previous frames [39]. To store templates from previous frames, we implemented a data structure, which is an array of length N, as well as methods for working with it: adding a template, extracting a template that was detected N frames ago, and a method for checking if the array is full. The parameters of this obstacle-processing algorithm are the following: the length of the queue, the number of frames on which the template matching should be performed, and the radius of the template search.

We can state that the Hough transform [40] can be used as the template matching algorithm. This method is designed to search for objects belonging to a certain class of figures using the voting procedure. The voting procedure is applied to the parameter space from which objects of a certain class of figures are obtained by the local maximum in the storage space, which is constructed when calculating the Hough transformation. The classical Hough transformation algorithm is associated with the identification of straight lines in the image, but the algorithm was later expanded with the ability to identify the position of an arbitrary figure and image templates.

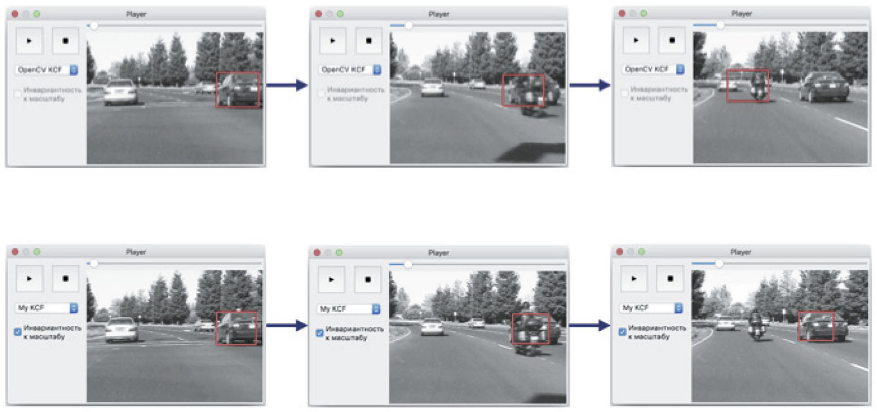


Fig. 6. Comparison of the operation of the standard KCF tracker (upper part) and the modified tracker with obstacle handling (lower part).

Our obstacle handling algorithm consists of the following steps:

Step 1. Go to a new frame.

Step 2. If the sign of object loss is set, then go to step 9, otherwise go to step 3.

Step 3. Add a patch to the queue.

Step 4. Call the detect() method to detect an object on the frame.

Step 5. If the method estimate is less than the threshold, then go to step 6, otherwise go to step 1.

Step 6. Set the sign of object loss from the frame.

Step 7. Initialize counter for counting frames without object.

Step 8. Get the patch from the queue.

Step 9. If the counter is less than the threshold, then go to step 10, otherwise go to step 13.

Step 10. Run the template matching algorithm.

Step 11. If the object is found within the specified radius, then go to step 12, otherwise go to step 1.

Step 12. Set the sign of object recovery, go to step 1.

Step 13. Stop the tracker.

As for test data to assess the quality of work, we used image sets from the Visual Tracker dataset [41], which meet the requirements for the presence of cars and some static camera. In particular, Fig. 6 shows an example of tracking a car when a motorcycle passes between it and the camera, while the camera is moving slightly. Tracking algorithms TLD (part of the OpenCV Tracking API) and the original KCF lose the vehicle, while the algorithm described above continues to track it.

5. CONCLUSIONS

In our studies, we developed and tested various methods of processing information from photo and video means of fixing the road situation. We implemented several software tools, some of the developments described in Section 2 are registered in the database of Federal Institute of Industrial Property [42]; the framework described in Section 3 is issued in [43]; algorithms presented in Section 4 are available in [44]. These solutions can be integrated into existing traffic control tools but may need to be brought to industrial use depending on the current tasks and their parameters.

FUNDING

The project was partially supported by the Innovation Promotion Foundation under the “Participant of the Youth Scientific and Innovative Competition” program, agreement 9546GU/2015.

CONFLICT OF INTERESTS

The authors declare they have no conflict of interests.

REFERENCES

1. S. Du et al., “Automatic license plate recognition (ALPR): A state-of-the-art review,” IEEE Trans. Circuits Syst. Video Technol. **23** (2), 311–325 (2012).
2. S. L. Chang et al., “Automatic license plate recognition,” IEEE Trans. Intell. Transp. Syst. **5** (1), 42–53 (2004).
3. I. Fomin, I. Nenahov, and A. Bakhshiev, “Hierarchical system for car make and model recognition on image using neural networks,” in *2020 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)* (IEEE, 2020), pp. 1–6.
4. Z. Alamgeer et al., “Review of car make & model recognition systems,” J. Appl. Technol. Innovation **2** (2), 9–16 (2018).
5. H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *European Conference on Computer Vision* (Springer, Berlin, Heidelberg, 2006), pp. 404–417.
6. P. N. Druzhkov and V. D. Kustikova, “A survey of deep learning methods and software tools for image classifi-

cation and object detection," Pattern Recognit. Image Anal. **26**, 9–15 (2016). https://doi.org/10.1134/S1054661816010065

7. S. Sivaraman and M. M. Trivedi, "Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis," IEEE Trans. Intell. Transp. Syst. **14** (4), 1773–1795 (2013).
8. V. D. Kustikova, I. B. Meyerov, and N. Y. Zolotykh, "Video-based vehicle detection method," Pattern Recognit. Image Anal. **24**, 588–592 (2014). https://doi.org/10.1134/S1054661814040117
9. V. D. Kustikova and V. P. Gergel, "Vehicle video detection and tracking quality analysis," Pattern Recognit. Image Anal. **26**, 155–160 (2016). https://doi.org/10.1134/S1054661816010156
10. M. Piccardi, "Background subtraction techniques: A review," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)* (IEEE, 2004), Vol. 4, pp. 3099–3104.
11. Y. Benezeth et al., "Comparative study of background subtraction algorithms," J. Electron. Imaging **19** (3), 033003 (2010).
12. J. F. Henriques et al., "Exploiting the circulant structure of tracking-by-detection with kernels," in *European Conference on Computer Vision* (Springer, Berlin, Heidelberg, 2012), pp. 702–715.
13. J. Ning et al., "Object tracking via dual linear structured SVM and explicit feature map," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 4266–4274.
14. A. Wang et al., "An incremental extremely random forest classifier for online learning and tracking," in *2009 16th IEEE International Conference on Image Processing (ICIP)* (IEEE, 2009), pp. 1449–1452.
15. Y. LeCun et al., "Backpropagation applied to handwritten zip code recognition," Neural Comput. **1** (4), 541–551 (1989).
16. S. Sivaraman and M. M. Trivedi, "Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis," IEEE Trans. Intell. Transp. Syst. **14** (4), 1773–1795 (2013).
17. S. Staroletov et al., "Model-driven methods to design of reliable multiagent cyber-physical systems," CEUR Workshop Proc. **2478**, 74–91 (2019).
18. E. Gamma, J. Vlissides, R. Helm, and R. Johnson, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1994).
19. *GOST* (State Standard) *50577–2018: License Plates for Vehicles. Types and Main Sizes. Technical Requirements.* http://docs.cntd.ru/document/1200160380.
20. P. Viola et al., "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* (2001), pp. 511–518.
21. A. Gulli and S. Pal, *Deep Learning with Keras* (Packt Publ. Ltd., 2017).
22. J. Bergstra et al., "Theano: A CPU and GPU math expression compiler," Proc. Python Sci. Comput. Conf. **4** (3) (2010).
23. M. Abadi et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," arXiv preprint (2016). arXiv:1603.04467
24. C. Szegedy et al., "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 2818–2826.
25. K. He et al., "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.
26. F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 1251–1258.
27. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint (2014). arXiv:1409.1556
28. M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision* (Springer, Cham, 2014), 818–833.
29. W. Harrison, "RPDE3: A framework for integrating tool fragments," IEEE Software **4** (6), 46 (1987).
30. Q. Chen et al., "A double-threshold image binarization method based on edge detector," Pattern Recognit. **41** (4), 1254–1267 (2008).
31. B. Tamersoy and J. K. Aggarwal, "Counting vehicles in highway surveillance videos," in *2010 20th International Conference on Pattern Recognition* (IEEE, 2010), pp. 3631–3635.
32. R. Sanchez-Matilla, F. Poiesi, and A. Cavallaro, "Online multi-target tracking with strong and weak detections," in *European Conference on Computer Vision* (Springer, Cham, 2016), pp. 84–99.
33. H. Pistori et al., "Mice and larvae tracking using a particle filter with an auto-adjustable observation model," Pattern Recognit. Lett. **31** (4), 337–346 (2010).
34. N. A. Shmalko et al., "Method for cleaning amaranth seeds from impurities," Food Process.: Tech. Technol. **46** (3), 114–120 (2017).
35. Y. Li and J. Zhu, "A scale adaptive kernel correlation filter tracker with feature integration," in *European Conference on Computer Vision* (Springer, Cham, 2014), pp. 254–265.
36. P. J. Davis, *Circulant Matrices* (Am. Math. Soc., 2013).
37. J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," IEEE Trans. Pattern Anal. Mach. Intell. **37** (3), 583–596 (2015).
38. C++ Implementation of KCF Tracker. https://github.com/joaofaro/KCFcpp.
39. F. Jurie et al., "Real time robust template matching," in *The 13th British Machine Vision Conference (BMVC '02)* (Cardiff, UK, 2002), pp. 123–132.
40. J. Illingworth and J. Kittler, "A survey of the Hough transform," Comput. Vision Graphics Image Process. **44** (1), 87–116 (1988).
41. Y. Wu, J. Lim, and M. H. Yang, "Online object tracking: A benchmark," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2013), pp. 2411–2418.

42. M. A. Laptev and S. M. Staroletov, "A program for preprocessing images from photofixing cameras and for recognizing vehicle types," Certificate of State Registration of a Computer Program No. 2017660999 (2017).

43. D. Nekrasov and S. Staroletov, A Framework for Building Motion Detection and Tracking Systems (2015). https://doi.org/10.5281/zenodo.3696547

44. D. Nekrasov, Self-Learning Object Tracking Software (2018). doi 10.5281/zenodo.3696522

Sergey Mikhailovich Staroletov, born 1984, graduated from the Polzunova Altai State Technical University in 2006, defended his Ph.D. thesis in the specialty 05.13.11 in 2011, currently works at AltSTU as an associate professor of the Department of Applied Mathematics, interests: software engineering, formal verification of programs, logic, new directions in artificial intelligence, cyberphysical systems, operating systems, and system programming. He has more than 50 Russian science citation index publications and more than 10 publications in selected peer-reviewed journals as well as 2 textbooks with ISBN. The best young teacher of AltSTU 2014.

Mikhail Alexandrovich Laptev, born 1993, master's student at AltSTU in the direction of 09.04.04 "Software Engineering." Winner of the "Participant of the Youth Scientific and Innovative Competition" competition (UMNIK) of the Innovation Promotion Foundation. Interests: machine learning, data science.

Dmitry Viktorovich Nekrasov, born 1993, master's student at AltSTU in the direction of 09.04.04 "Software Engineering." Interests: image processing, tracking.