Real Time Speed Estimation: Final Report

1st Bolat Tleubayev
Robotics and Mechatronics department
Nazarbayev University
Astana, Kazakhstan
bolat.tleubayev@nu.edu.kz

2nd Zhanel Zhexenova
Robotics and Mechatronics department
Nazarbayev University
Astana, Kazakhstan
zhanel.zhexenova@nu.edu.kz

***Abstract*—This document presents the work done within the framework of the ROBT 310 Image Processing class. Our group worked on the Real Time Speed Estimation from monocular RGB camera. In this work we used various Computer Vision and Image Processing techniques to implement the system.**

***Index Terms*—Image Processing, Computer Vision**

I. Introduction

The aim of this project is to create a system that is capable of speed estimation of cars on road. This is particularly important project to study as it may be useful for road traffic control, proper establishment of which may save peoples' lives and decrease the number of car accidents.

Similar system, SERGEK[1], is already used on roads in Astana to control the car flow and penalize drivers, who exceed maximum speed limit or create dangerous situations on road. The system helped to solve more than 500 crimes in Astana, reduce the numebr of offences by 30% and the number of accidents by 18% [1].

For the project we decided to use a video recording from the camera mounted on the crossroad of Moores Mill Road and East University Drive in the city of Auburn, Alabama, USA[2].

Capabilities and applications of the system are broad and this project has numerous potential uses.

II. Literature Review

Before starting to work on this problem we made some literature review. Besides the papers that are available on Google Scholar[3] we relied on various web-sources in building the skeleton of the system.

The first paper we considered was written by Temiz, M. S., S. Kulur, and S. Dogan[2]. This paper presents a solution to a problem similar to our project. It is implemented using C++ and OpenCV for Real-time speed estimation from monocular video. The accuracy is that was obtained by researchers was +/- 1-2 km/h.

Then we found a paper done by Kumar, Pankaj, et al.[3]. This paper presents well-written description for a more complex analysis of the road traffic, i.e. the behavior of drivers on the road using monocular camera. One of the future applications of the project may involve studies of a deeper traffic analysis.

Another paper that helped us was written by Suzuki and Abe [4]. The algorithm that is proposed by the authors is also used for function in OpenCV[4] library. From this paper we extracted several insights on Contour Extraction algorithm which plays significant role in car tracking.

We also got numerous hints from the textbook written by Gonzalez and Woods[5]. Especially we found algorithms for Moore's pixel following algorithm, which is an essential part of Countour Extraction algorithm.

We also referred to OpenCV Tutorials[5] in order to get more familiar with strategies and techniques used in video processing.

III. Work Done

The project could be separated to several parts. Firstly, we need to segment cars from the background. Secondly, we need to be able to track a single object while it is within the Region of Interest (ROI). Thirdly, we need to map the movement of the car on the image with real units of measurements.

A. Car segmentation

For this part of the project we have developed three approaches and tried their combinations.

The first method to consider is Accumulative Difference Image. We decided to check the method by subtracting consecutive frames from each other and extracting the difference. After that, we threshold the image and apply morphological operations such as Erosion and Dilation. You can see the input and output images below.

[1]http://sergek.kz
[2]https://www.youtube.com/watch?v=J8YXGIPkLNYt=93s
[3]https://scholar.google.com
[4]https://opencv.org
[5]https://opencv-python-tutroals.readthedocs.io/en/latest/index.html

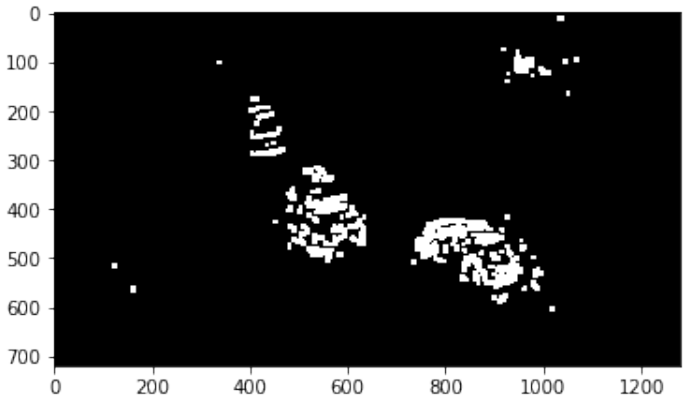

Fig. 1. Difference of two images after morphological operations

As it could be seen from images above, the result is not satisfactory, as even after performing morphological operations on binary output image, we still end up with numerous segments that are not suitable for further analysis.

The second method was the difference of each frame from an image of a clear road. We took a reference frame with no cars on it and subtracted it from each frame. Then, we applied thresholding to create binary image. You can see the result below.

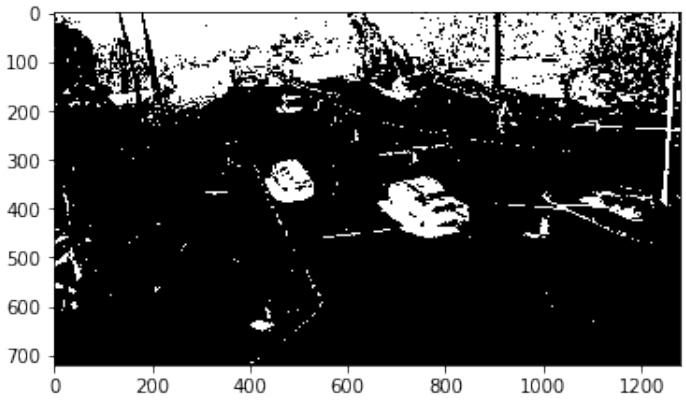

Fig. 2. Difference of a frame from reference after morphological operations

As you can see from the image above, the result is not satisfactory either, as future morphological operations could result in interference with topmost white part of the binary image.

The third approach is to use RGB thresholding to focus only on the pixels that have colour of the asphalt, then we obtain binary image of the road and objects that have colour different from the asphalt.

For that purposes we used two approaches. Firstly, we played with HSV threshold in order to get an idea of what kind of boundaries interest us. Then we converted from HSV colorspace to RGB. Secondly, we found the histogram of a part of the road in order to get more precise boundaries. You can see the histogram below.

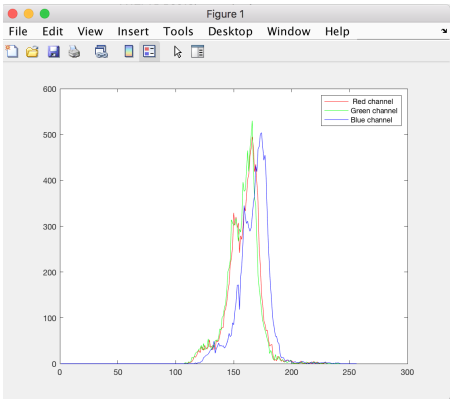

Fig. 3. Histogram of the asphalt

After RGB thresholding we used two images to define the Region of Interest (ROI) in each frame, so that we could focus on the main lane. You can see the ROI images below.

Fig. 4. Helper image, left lane

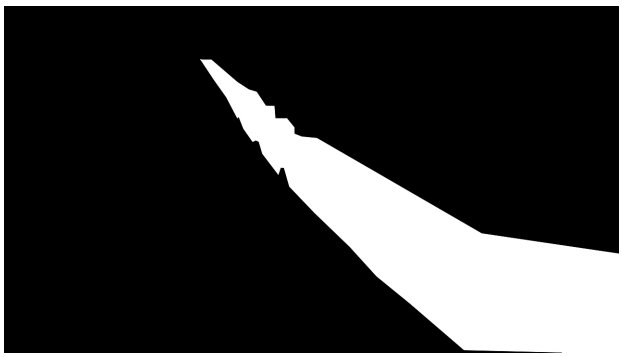

Fig. 5. Helper image, right lane

Subtraction of the thresholded image from helper images yields the following image.

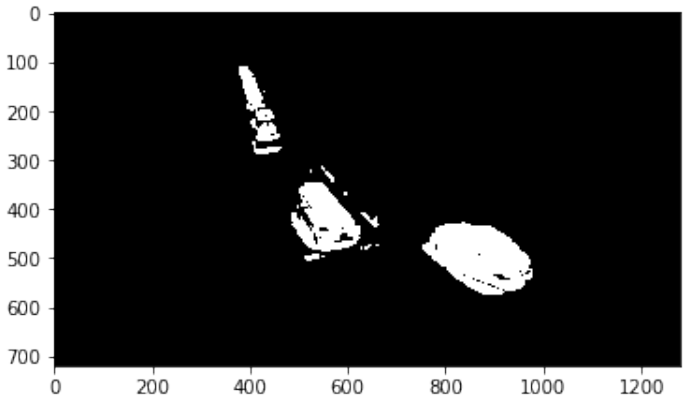

Fig. 6. Output image

As it could be seen such image is much clearer than that obtained by Accumulative Difference Image. Moreover, that method allows us to track stationary cars, or cars which moved insignificantly on consequent frames. It also worth mentioning that Accumulative Difference Image may not detect most pixels on the car as the car moving on the image has the same colour.

In order to close gaps within the car boundary we applied several iterations of morphological operations such as Erosion and Dilation.

B. Tracking

After obtaining the binary image, with cars having white colour and the rest is black colour, we may write logic for tracking one specific object. In the video we may have three cases. For that purposes we store contours, obtained by cv2.findContours() function, from current and previous frame.

The first case when there is an object on the current frame near the object on a previous frame, i.e. when the car is on both frames we continue to track it.

The second case when there is no object on the current frame, but there was on a previous frame, then stop tracking the car. This may be redundant as we discard the tracked object after every new frame.

The third case case is when there was no object on the previous frame, but there is on a current, i.e. when new car enters the Region of Interest.

You can see the sample image of tracked objects below.

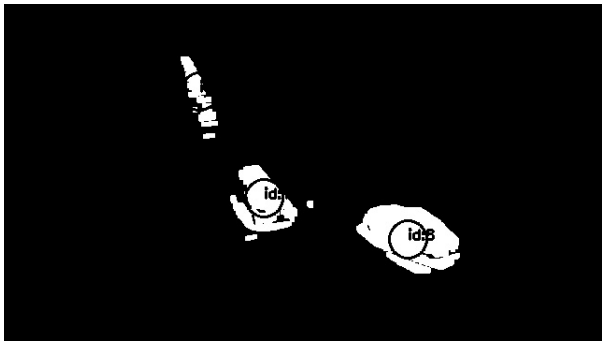

Fig. 7. Tracking

C. Mapping

After performing all calculation we will need to map the motion of cars on the image to motion of cars in real life. This part of the project was implemented using Homography estimation in order to properly map the motion on the image to real-life. Homography calculation allowed us to rescale the road so that we could eliminate the effect of perspective error.

For this part we also took measurements of the actual crossroads using Google Maps[6]. You can see the sample measurement below.

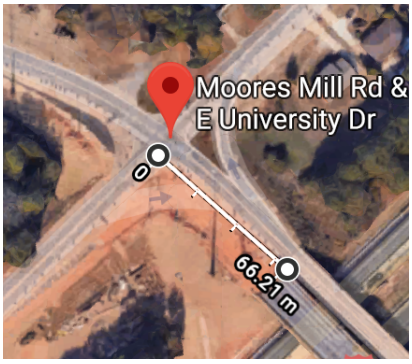

Fig. 8. Map measurements

The next step for speed estimation was calculation of the actual speed. Firstly, we found that the video was recorded at 30 Frames Per Second (FPS). Then we divided the motion of the centroid of the car over the FPS rate and multiplied by the total distance of 60 meters. Finally, we manually calculated that is usually takes about 10 seconds for a car to go through 60 meters of the ROI, i.e. the average speed is about 6 m/s, this information helped us to tune the code for better precision.

Performing all above mentioned steps allowed us to obtain such output.

[6]https://www.google.com/maps

Fig. 9. Final output

The speed is measured in meters per second. Circles represent the centroids of tracked cars.

IV. Functions implementation

A. Erosion

After obtaining binary image, we get a noisy input, that is not acceptable for feeding to speed estimation algorithm. Therefore, we have implemented functions for morphological operations. One such function is erosion function. It is used to shrink outermost contour of relatively large objects and totally remove relatively small objects. It is useful for removal of outliers that may occur due to lighting change or camera slight movement.

Our implementation of the function takes as input a binary image and a kernel, in other words we have implemented variable kernel erosion function.

B. Dilation

Another morphological operation that is used in this project is dilation. It is an operation that expands contours of objects. It is useful to undo the effect of erosion for relatively large objects and to fill small gaps that may occur due to RGB threshold algorithm malfunction.

Our implementation of the function takes as input a binary image and a kernel, in other words we have implemented variable kernel dilation function.

C. Nearest Neighbour Interpolation

In order to reduce computational cost of the video, each frame is resized before the further processing. For this we implemented Nearest Neighbour Interpolation algorithm which takes the value of the neighbouring input pixel is translated to the value of the output pixel. It works with both RGB and binary image.

Function takes image and its new x and y dimensions as input and outputs the resized image.

However, in current implementation we are using OpenCV's resize function, because even with *jit* it requires more time for computation which slows our algorithm drastically.

D. Contour Finding

In order to differentiate between different objects we needed an algorithm that would give different numbers to different objects. For this part of the project we got valuable insights from Suzuki and Abe[4], we also used numerous techniques from the book by Gonzalez and Woods[5].

The algorithm utilizes Moore pixel following algorithm. Firstly, as it finds leftmost and topmost white pixel of the object, the pixel location is recorded and labeled b, the background pixel to the left is stored as c. Then the algorithm goes in clockwise direction around b in the 8-neighbourhood starting from pixel c. As it hits the next white pixel, its location is also recorded and stored as b, the background pixel that was checked before hitting the pixel b is stored as c. This process continues until algorithm reaches its original starting pixel. You can see the illustration of the algorithm below.

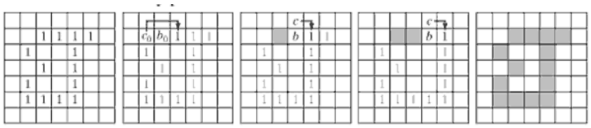

Fig. 10. Moore pixel following. Retrieved from Gonzalez and Woods [5]

After running the algorithm on the binary image we obtain an image with labeled objects. The centroids of labeled objects are calculated by taking the average of all boundary pixel locations.

E. Numba

In our implementation we had significant amount of nested loops, this created a problem of computational efficiency. In order to boost the speed of the algorithm we decided to use open-source Python compiler Numba[7]. For our case, Numba was able to drop the frame processing from several seconds to several milliseconds. Even though we still process frames slower than the FPS rate on the video, this compiler significantly boosts the execution of the code.

F. Homography

Another concept that brought us to mapping the real road to 2D image was Homography. Firstly, we have calculated a homogeneous transformation matrix, where both rotation and translation of the road in the frame were considered. To do so, we solved for h the linear equation $Ah = b$, where A, h, b are given as following:

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1^{'} & -y_1x_1^{'} \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1x_1^{'} & -y_1x_1^{'} \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2^{'} & -y_2x_2^{'} \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2x_2^{'} & -y_2x_2^{'} \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3^{'} & -y_3x_3^{'} \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3x_3^{'} & -y_3x_3^{'} \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4^{'} & -y_4x_4^{'} \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4x_4^{'} & -y_4x_4^{'} \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x_1^{'} \\ y_1^{'} \\ x_2^{'} \\ y_2^{'} \\ x_3^{'} \\ y_3^{'} \\ x_4^{'} \\ y_4^{'} \end{pmatrix}$$

[7]http://numba.pydata.org/

Then we mapped pixels from the real road to the 2D image by multiplying each pixel location by homogeneous roto-translational matrix obtained by calculating $h = A/b$. For simplification purposes last element of the homography matrix h_{33} was set to 1. Thus, we obtained two equations that give us new location of the pixel:

$$x_{new} = \frac{h_{11}x_{old} + h_{12}y_{old} + h_{13}}{h_{31}x_{old} + h_{32}y_{old} + 1} \quad (1)$$

$$y_{new} = \frac{h_{21}x_{old} + h_{22}y_{old} + h_{23}}{h_{31}x_{old} + h_{32}y_{old} + 1} \quad (2)$$

Then the beginning and the end of the road were mapped to the measurements from Google Maps. The real distance between points used for homography is approximately 60m, dividing by image width we found correspondence between real dimensions and image dimensions. Below you can see the original and transformed images of the road.

Fig. 11. Original clear road

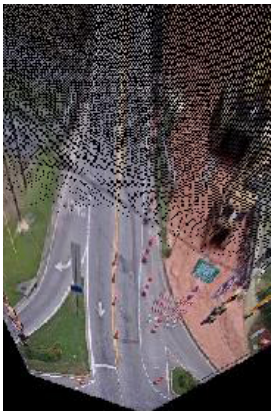

Fig. 12. Road with respect to Homography

G. Speed Estimation

Finally, we had to calculate the speed of the car from the 2D RGB image. This was done in several steps. Firstly, we have implemented the tracking algorithm for cars, which was able to identify an object through all frames, where it appeared. Then the algorithm calculated centroids of tracked objects in order to decrease error that might occur due to cars overlapping. Then we calculated the change of centroid location in consecutive frames. Having the interframe location difference converted into the real dimensions using homography matrix we multiplied it by the FPS rate of the video to get the speed of the car.

However, the uncertainty rate is high due to the several reasons such as car detection, image resolution and gaps in homography due to the image resolution. Because of the detection problem speed of the vehicle could change drastically and unpredictably, to tackle this problem we put limitations so that if the speed will accidentally drop lower than some value or will be too high the algorithm will take the average value of this vehicle's previous speed values.

V. Limitations

There is a room for improvement of the project due to following limitations of current implementation.

A. Computation efficiency

An issue that can be resolved is computational efficiency of the code. It is necessary to find a way to boost the performance of functions. This is necessary in order to create the real-time system without lags. Currently, every frame requires more time than it is being fed by the recording.

B. Cars segmentation

Currently, our main method of car segmentation is RGB threshold. This method however, is error prone. As it sometimes mistakenly removes white or gray cars due to their color similarity to the asphalt. One possible solution may be use of Machine Learning techniques for car recognition, unfortunately we did not have enough time to study this approach. So, current segmentation technique causes improper contour extraction, which leads to drastic changes in centroid location, which in turn adds more error to speed estimation.

References

[1] Uatkhanov, Y. (2019) Sergek has helped solve more than 500 crimes in Astana. [online] The Astana Times. Available at: https://astanatimes.com/2018/04/sergek-has-helped-solve-more-than-500-crimes-in-astana/ [Accessed 20 Feb. 2019].

[2] Temiz, M. S., S. Kulur, and S. Dogan. "Real time speed estimation from monocular video." International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS) 39 (2012): B3.

[3] Kumar, Pankaj, et al. "Framework for real time behavior interpretation from traffic video." (2005).

[4] Suzuki, Satoshi, Keiichi Abe. "Topological structural analysis of digitized binary images by border following." Computer vision, graphics, and image processing 30.1 (1985): 32-46.

[5] Gonzalez, Rafael C., and Richard E. Woods. "Digital image processing." (2002).