UAS4T competition: Solution using `pandas` and kernel density estimation

Danial Hasan Uroš Kalabić

I. Introduction

This paper describes our entry into the UAS4T competition at the 2020 Intelligent Traffic Systems Conference.

The competition asks us to analyze 13.55 minutes of traffic data and return traffic measurements. Specifically, the competition is concerned with three separate road sections in the city of Athens. The data collected was obtained using a drone and includes data on 2307 vehicles, which are categorized as on of "Motorcycle", "Car", "Medium", "Heavy", "Taxi", or "Bus". The relevant sections of road are given in Fig. 1 and colored green, red, and yellow. The green section concerns vehicles going south on 28is Oktovriou; the red section concerns vehicles going west on Leof. Alexandras; the yellow section concerns vehicles turning east from 28is Oktovriou onto Leof. Alexandras. The competition asks us to find the longest queue over the timespan of the data in each of the regions and specify the following:

1) Length of longest queue
2) Lane of longest queue occurrence
3) Initial and final coordinates of longest queue
4) Time of longest queue occurrence
5) Time of spillback (only relevant for yellow region)

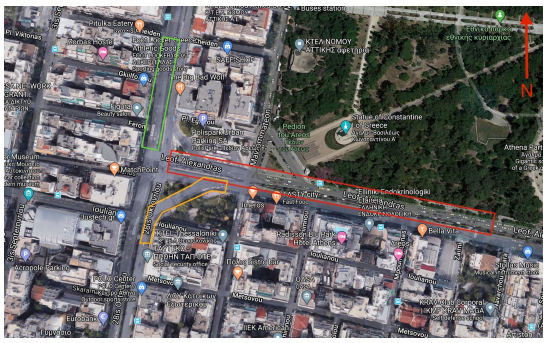

Fig. 1. Regions of Athens considered

Our code is in Python and uses standard data analysis tools, available in the `pandas` and `sklearn` packages. We submitted our code in the format of a Jupyter notebook, to make it readable and easier to judge. The structure of this paper is a summary of the code and follows its structure. The code itself contains more detailed information and figures.

This work was supported by Mitsubishi Electric Research Laboratories.

D. Hasan is with the Division of Engineering Science, University of Toronto, ON M5S 1A4, Canada `danial.hasan@mail.utoronto.ca`

U. Kalabić is with Mitsubishi Electric Research Laboratories, Cambridge MA, 02139, USA `kalabic@umich.edu`

II. Solution

We approached this problem by finding the solution for the green, red, and yellow regions, in that order. Our approach for each new region built on previous results. In this section, we describe how we found our solution, region-by-region, from least to most difficult.

A. Preprocessing

As with any data analysis project, we start by processing and cleaning up the data. The main tool we use for this purpose is the `pandas` package for Python. We load all the trajectory data into a Pandas dataframe and begin cleaning. We note that the latitude and longitude coordinates have a precision to the 6th significant digit, as is usual in mapping. We also note that the scale of the region is very small relative to the Earth, so we are able to use a flat-Earth approximation, which allows us to treat lat-lon coordinates as Euclidean.

Our first step is to plot the data and visualize it. We do this by using the `PIL.Image` package, which is standard for constructing images. As shown in Fig. 2, the result clearly shows the existence of roads, and even lanes.

Our belief is that data should define lanes, so that we are not limited by *a priori* beliefs on lane definitions, since vehicles cannot be expected to perfectly follow traffic demarcations. For this reason, we plot the data of stopped vehicles to see how stopped vehicles form queues. The result seems strange on first glance, as it seems that vehicles congregate around stop lights in irregular patterns. In analyzing the data, we came to believe that the irregularity was due to the presence of motorcycles, which are small enough to split lanes and overtake larger vehicles in congested situations. This belief seems to be true and is supported by the plot in Fig. 3, which shows that the irregular behavior is almost exclusively, if not always, exhibited by motorcycles. This realization guides our first important design choice: the removal of motorcycles because they do not participate in the creation of queues.

Filtering out all motorcycles IDs from the dataframe, the data becomes easier to work with. Queues are clearly defined and we can begin the analysis.

B. Green region

We begin by providing a user-defined control region, a polygon containing relevant data given in Fig. 3.

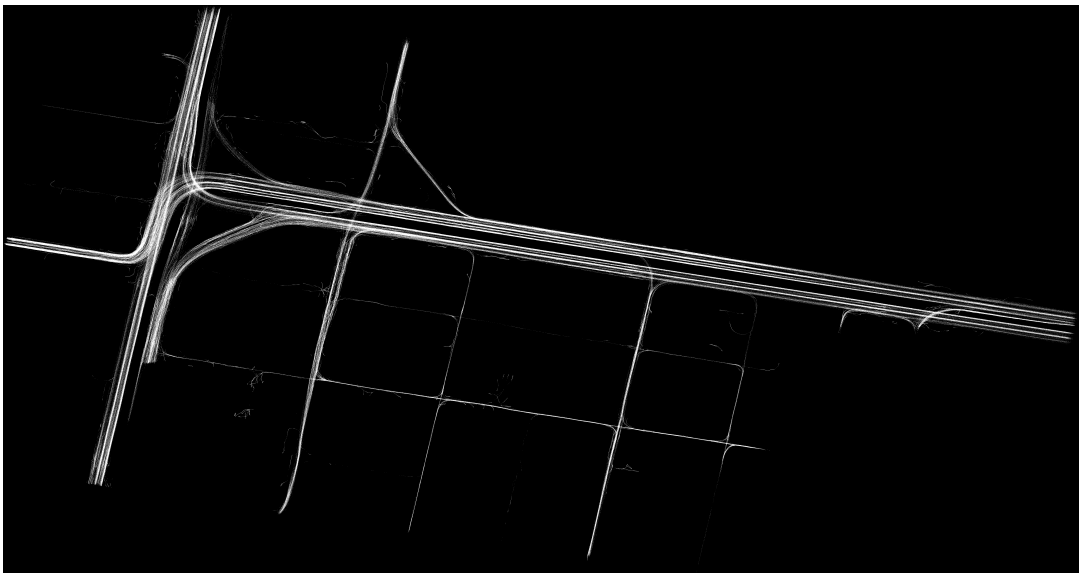

Fig. 2. Visualization of all traffic data

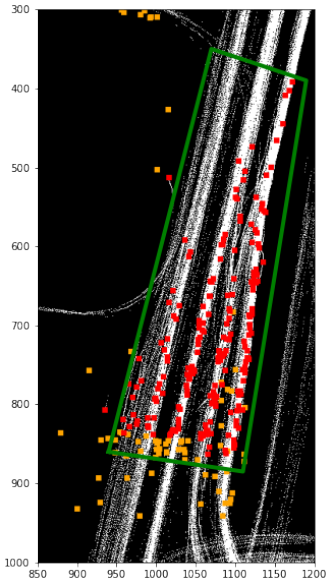

Fig. 3. Plot of green region with larger vehicle trajectories (white), stopped non-motorcycle vehicles (red), stopped motorcycles (orange)

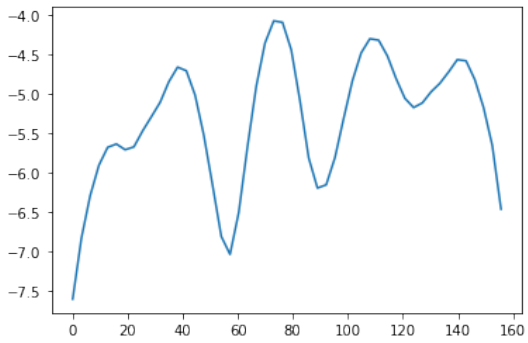

Fig. 4. KDE plot for green region

In the control region, we first determine the direction of traffic by taking the mean of the approximate vehicle headings in the region. The approximate headings are obtained by filtering out stopped vehicles and then taking the difference in positions N frames ahead and N frames behind. With this direction vector, we rotate the position data, adding two new columns to the dataframe: L, corresponding to length along direction of traffic, and W, corresponding to width perpendicular to the direction of traffic.

Finding which lane a vehicle occupies is a clustering problem. Strictly speaking, it is a non-Euclidean 2D problem; non-Euclidean because there is no concept of distance between vehicles in separate lanes. To sort vehicles into lanes, we reduce the problem by projecting onto the subspace perpendicular to the direction of traffic and then finding the demarcations between lanes using kernel density estimation (KDE). KDE is a method used to find demarcations in data by setting demarcations to be the minimum of the estimated kernel density and has shown to be effective in clustering traffic data into lanes [1]. We implement KDE using the package `sklearn` and provide results in Fig. 4, where we show a plot of estimated density for the green region showing clear minima, which we take to be lane demarcations.

There are three local minima, implying the existence of five lanes. Labeling the data according to their KDE-defined cluster, we obtain the plot shown in Fig. 5. The plot shows clear lane demarcations. The demarcations are not straight, which is unsurprising given that the data is never perfect, but we consider this to be acceptable because vehicles sort themselves within lanes when stopped.

Lane 5 (yellow in Fig. 5), is a partial lane. This is because Lane 4 splits into two. We therefore consider Lane 4 and 5 to be the same lane. Given this clustering, we search for the longest lane by finding the largest difference between along-track distances. The solution we find gives us a queue

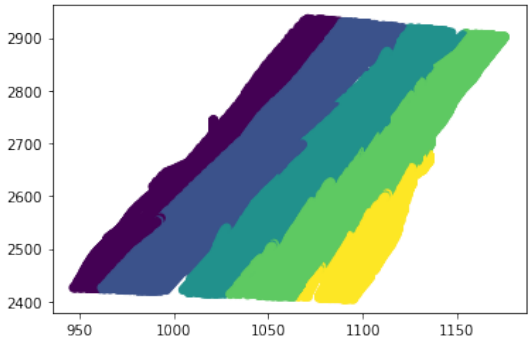

Fig. 5. Lanes in green region

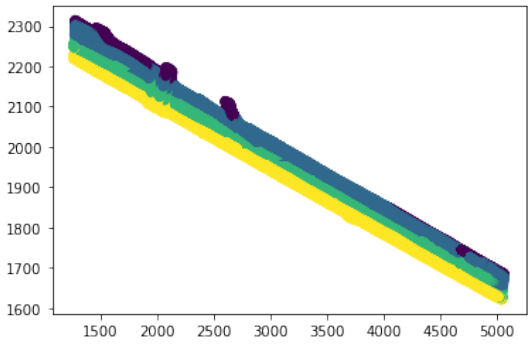

Fig. 6. Lanes in red region

in which a vehicle has not come to a complete stop, so we then search for the first occurrence where that vehicle comes to a stop and provide this as our answer.

C. Red region

Our approach to finding the answer for the red region begins by following the same procedure as that for the green. We follow the same procedure of automatically determining a traffic direction vector and using KDE to obtain the lane clustering given in Fig. 6.

This time, because the road segment under consideration is so long, we find the existence of multiple queues. To ensure that we do not overestimate the length of queue, we only consider vehicles to be part of a queue if they are closer than 200 units of length to each other. We find the longest such queue and provide this as our answer.

D. Yellow region

The yellow region is more difficult than the rest because it involves spillback. The region includes a diagonal road component which flows into the eastbound road. We therefore split the yellow region into two, as shown in Fig. 7, and begin by sorting into lanes according to KDE, as in the other two cases. This results in a nonsensical result, as shown in Fig. 8, because the vehicles clearly follow their own lane-making rules. Specifically, the figure shows that the queues in the latter yellow region curve down and are not straight.

We therefore take a slightly different approach for finding the longest queue. We follow the same procedure as for the other regions in finding the longest queue in the former

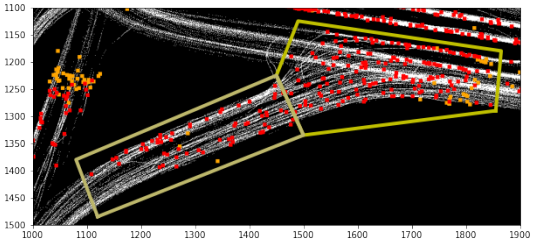

Fig. 7. Plot of yellow regions with larger vehicle trajectories (white), stopped non-motorcycle vehicles (red), stopped motorcycles (orange)

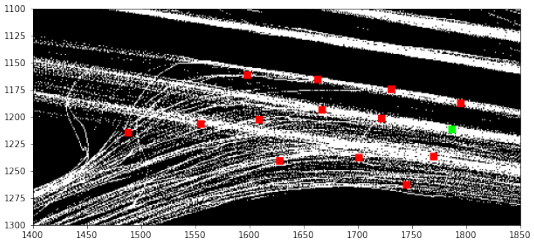

Fig. 8. Snapshot of vehicles in the latter yellow region with stopped vehicles (red) and moving vehicles (green)

region, but this time defining our own directions of traffic based on shape of the relevant control region. We also simplify the analysis by filtering out all time instances where there is a moving vehicle in either region, justifying this because the region is compact and doesn't include multiple queues in the same lane. We then concatenate the two results to find the longest queue, since both regions contain three lanes, and provide this as our answer. We also find the first time of spillback from the latter region into the former.

III. Summary

We have provided answers to the questions posed by the competition. Our approach was general across all three regions. It consists of removing all motorcycle trajectories from the data, since motorcycles do not participate in queue formation, and the performing, region-by-region:

1) Defining a control region
2) Determining a direction of traffic
3) Clustering vehicles into lanes using KDE
4) Sorting vehicles by along-track position
5) Measuring sorted vehicles to find longest queue

In summary, the approach consists of straightforward queries and manipulations of data, coupled with KDE. The code is implemented in a Jupyter notebook, using only standard data analysis tools. It takes about 5.0 minutes to run on a Ryzen 2700x machine at 4.0Ghz, with 16GB RAM, and no GPU usage.

References

[1] E. R. I. A. C. M. Uduwaragoda, A. S. Perera, S. A. D. Dias, "Generating lane level road data from vehicle trajectories using kernel density estimation," in *Proc. IEEE Conf. Intell. Transp. Syst.*, The Hague, Netherlands, 2013, pp. 384–391.