

Measuring Traffic Dynamics at the Edge

Midiendo la dinámica de tráfico vehicular en el Edge

Medindo a dinâmica do tráfego de veículos na Edge

Luis Gerardo León-Vega^{1*}, Jorge Castro-Godínez¹

Received: Oct/4/2021 • Accepted: Jan/14/2022 • Published: Nov/1/2022

Abstract

This work aims to measure the impact of approximate computing on a case study of traffic dynamics metering on a System-on-Chip edge computing device. Firstly, the study proposes a baseline implementation of the metering system in C++. To analyze the application in detail, study profiled the baseline using a built-in instrumented profiler, presenting the overall performance of each of its parts. During the hotspot analysis, some parts had optimization opportunities exploitable by multi-threading and approximate computing techniques, particularly frame skipping, which is inspired by the loop perforation approximate technique. The first optimization employed was multi-threading, which led to a 1.32x speedup on the application without introducing errors in the metrics. Then, the meter was optimized by using frame skipping. This work demonstrated that adaptatively modifying the number of frames skipped improved the error in the final metrics compared to keeping it fixed. In terms of performance, the frame skipping brought the overall speed up to 1.76x. Approximate computing, in particular, frame skipping, managed to contribute up to 25% of the overall speedup, managing to accelerate the meter from 8.7 frames per second to 15 fps in the most critical case in exchange for some numerical error on the final metrics.


Keywords: traffic dynamics meter; edge computing; approximate computing

Resumen

El objetivo de este trabajo es medir el impacto de la aplicación de técnicas de computación aproximada sobre un caso de estudio de implementación de un medidor de la dinámica de tráfico vehicular en una unidad computacional basada en un sistema embebido. El estudio parte de una implementación inicial del medidor hecha en C++. Para el análisis de la aplicación, la implementación inicial se perfila con un perfilador empotrado en el mismo código, que muestra información detallada de cada una de las partes. Durante el análisis de consumo, se encontraron partes optimizables con paralelismo a nivel de hilos y técnicas de computación aproximada como omisión de recuadros, que es una técnica inspirada en la perforación de lazos. La primera optimización realizada fue la implementación multihilo, que logró acelerar la aplicación 1.32 veces sin introducir errores en los resultados. Posteriormente, la implementación fue optimizada con la omisión de recuadros. Durante el desarrollo de este trabajo, se demuestra que modificar el número de recuadros omitidos de forma dinámica en tiempo de ejecución mejora considerablemente

*Corresponding author

Luis Gerardo León-Vega, ✉ lleon95@estudiantec.cr,  <https://orcid.org/0000-0002-3263-7853>

Jorge Castro-Godínez, ✉ jocastro@tec.ac.cr,  <https://orcid.org/0000-0003-4808-4904>

¹ Electronics Engineering School, Instituto Tecnológico de Costa Rica, Cartago, Costa Rica.



el error introducido comparado a mantener constante el número de recuadros omitidos. La combinación de ambas optimizaciones concluyó en una implementación 1.76 veces más rápida, donde la aplicación de computación aproximada mediante omisión de recuadros contribuyó hasta en un 25% sobre el total de la mejora, acelerando el medidor de 8.7 recuadros por segundo a 15 en el escenario más crítico a cambio de la introducción de errores numéricos.

Palabras clave: Medición de dinámica vehicular, computación en el *Edge*, computación aproximada.

Resumo

O objetivo deste trabalho é medir o impacto da aplicação de técnicas de computação aproximada em um estudo de caso da implementação de um medidor de dinâmica de tráfego de veículos em uma unidade computacional baseada em um sistema embarcado. O estudo começa a partir de uma implementação inicial do medidor feita em C++. Para a análise da aplicação, a implementação inicial é perfilada com um perfilador incorporado no próprio código, que exibe informações detalhadas sobre cada uma das partes. Durante a análise de consumo, foram encontradas peças otimizáveis com paralelismo em nível de thread e técnicas de computação aproximada, como o salto de quadros, que é uma técnica inspirada na perfuração de laço. A primeira otimização realizada foi a implementação multithread, que conseguiu acelerar a aplicação em 1,32 vezes sem introduzir erros nos resultados. Posteriormente, a implementação foi otimizada com o salto de quadros. Durante o desenvolvimento deste trabalho, mostra-se que modificar a dinâmica do número de salto de quadros em tempo de execução melhora consideravelmente o erro introduzido em comparação com a manutenção constante do número de salto de quadros. A combinação de ambas as otimizações resultou em uma implementação 1,76 vezes mais rápida, onde a aplicação da computação aproximada por meio do salto de quadros contribuiu com até 25% da melhoria total, acelerando o medidor de 8,7 quadros por segundo para 15 no cenário mais crítico, em troca da introdução de erros numéricos.

Palavras-chave: Medição da dinâmica de veículos, computação de borda, computação aproximada.

Introduction

As many other countries, Costa Rica faces a severe traffic congestion problem, particularly in the great metropolitan area (GAM). The growing number of vehicles causes dense traffic jams on main roads, and it has increased by 40 % of the mobilization time along the GAM in the last five years.

Finding an integral solution to this problem requires automatic mechanisms to estimate traffic dynamics. A Traffic Dynamics Meter (TDM) is an application used to determine traffic parameters, such as average speed, traffic density, and occupancy (Hall, 1992). A TDM is commonly implemented

using cloud computing: remote cameras capture video streams sent over the net to distant servers to be processed (Zhu, Yu, Wang, Ning, & Tang, 2019). The information a TDM provides can help define traffic optimization approaches, for instance, by adapting traffic lights intervals and enabling reversible and exclusive lanes for public transport. Figure 1 depicts a TDM fed with a live video of a road.



Figure 1. Example of a Traffic Dynamics Meter (TDM). This TDM detects vehicles (blue zone), estimates traffic flow (green zone), and determines traffic density (red zone), as traffic speed is computed alongside these zones. This road feed corresponds to Soquel Ave in Santa Cruz, California. (California Department of Transportation (Caltrans)). Note: Caltrans Streaming Video Locations. Caltrans. Retrieved (2018).

As technology moves forward, it is no longer foreseen to entirely depend on remote servers for data-intensive processing (Yu, et al., 2017). **Edge computing** has emerged to offload the computational stress away from the centralized cloud by distributing and transferring computation to individual smart nodes (Yu, et al., 2017), bringing the processing where the information is obtained. For instance, IoT nodes are expected to perform partial, or even complete, machine learning applications despite their computing power and energy constraints (Samie, Bauer, & Henkel, 2019). In the context of a TDM, edge nodes could perform all sensing and processing required and transmit only relevant information to a centralized server in the cloud.

Contribution: This work describes a case study of a video-based TDM implemented on an SoC-based edge computing platform. Improving its performance supposed exploiting approximate computing, a novel design paradigm for error-tolerant applications, and thread execution.

Background

Diverse approaches have been proposed to implement TDMs based on different data acquisition techniques, algorithms, and system architectures. One approach is based on big data (Zhu, Yu, Wang, Ning, & Tang, 2019). The main idea is to capture and exploit data from different sources, such as video cameras, on-board car sensors, on-road sensors, and even social networks (D'Andrea, Ducange, Lazzarini, & Marcelloni, 2015). All acquired data is uploaded to the cloud, where a group of servers and algorithms perform computations to obtain traffic dynamics estimations.

Other approaches simplify the complexity of the TDMs by just using on-road sensors, such as cameras, light and ranging detection (LIDAR), ultrasonic, and acoustic sensors. For instance, TDMs based on computer vision have been reported in the literature about this topic (Giridharan, Kadaieaswaran, Arunprath, & Karthika, 2017). Taking advantage of the current cameras used for reporting traffic conditions, especially in newscasts, measuring the traffic density at intersections is possible.

For the case of video-based TDMs, four relevant traffic metrics have been modeled and reported in the literature (Hall, 1992): average speed (\bar{u}_t), traffic flow (q), density (λ), and occupancy (o).

Average speed

It is the average speed of the vehicles during a time-lapse:

$$\bar{u}_t = \frac{1}{N} \sum_{i=1}^N u_i$$



where

$$u_i = \frac{dx}{dt} = \frac{x_2 - x_1}{t_2 - t_1}$$

Here N is the number of sampled vehicles, and x_j is the physical position at the t_j time.

Flow rate

It represents how many vehicles are crossing an imaginary line (or limit) per sample period T , defined as:

$$q = \frac{N}{T}$$

Traffic density

It is the number of cars per unit of length:

$$\lambda = \frac{N}{L}$$

where L is the physical distance measured or estimated from the images.

Occupancy

It is the average time in which a vehicle is analyzed:

$$o = \sum_{i=1}^N \frac{(t_{2,i} - t_{1,i})}{N}$$

where $t_{1,i}$ and $t_{2,i}$ are the times when the i -th vehicle enters and exits from the analysis zone, respectively.

A Traffic Dynamics Meter

This work proposes a TDM using data from videos, particularly from free-to-access

camera feeds from roadsides, to estimate the previously defined traffic metrics. Figure 2 depicts the main stages of this TDM. The *Retriever* stage takes each video frame, converts it to grayscale, and applies filtering and morphological operators. According to the road and camera feed, this stage uses predefined masks to define regions of interest in the scene, particularly for detection and tracking. The *Detector* stage identifies vehicles in the detection zone using Haar-like features (Viola & Jones, 2001) and produces an array of detected vehicles. For such an array, the *Tracker* stage provides a new tracker for each vehicle in the scene. As depicted in Figure 2, the number of trackers changes dynamically as the total number of vehicles being tracked changes. Also, trackers for the previously detected vehicles are refreshed with each new video frame. This tracking is performed using MOSSE filters (Bolme, Beveridge, Draper, & Lui, 2010). Finally, the *Meter* stage collects new vehicle positions and extracts metrics for each one, and it calculates the global flow, density, occupancy, and average speed.

Baseline Performance

The TDM proposed was implemented using C++ and OpenCV (version 3.3.0); it was deployed on a Zynq XC7Z020, an off-the-shelf SoC available in the ZedBoard

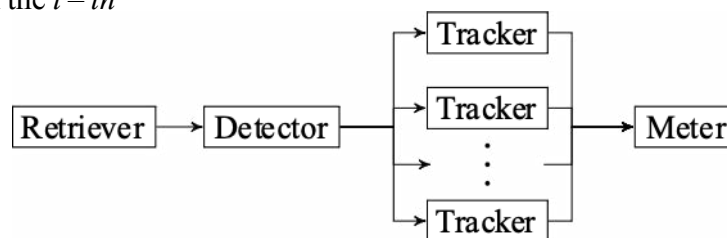


Figure 2. Block diagram of the main components in our proposed TDM.

Note: Own source.



development platform. We evaluated the performance of this TDM with video samples from three different test scenarios: two camera feeds from roads of San José, Costa Rica, here referred to as *General* and *Multipiazza* (Ministerio de Obras Públicas y Transportes, n.d.); and a third camera feed from Soquel Ave, in Santa Cruz, California, here referred as *SoquelAve* (California Department of Transportation (Caltrans)). Samples of these camera feeds are presented in Figures 3.a, 3.b, and 3.c, respectively. Each scenario presents different traffic dynamics, which leads to different detection and tracking effort. *General* represents a low traffic density with high-speed vehicle flow. On the other hand, *SoquelAve* presents a high density with moderate speed. *Multipiazza* is an intermediate scenario.

An execution profile of the TDM is presented in Figure 1.d for these three test scenarios. For this baseline performance

evaluation, the queue of detected vehicles was processed in serial. Considering a requirement of 30 fps, the current execution time required per frame is above this constraint for the three scenarios. In the experimentation, the time required by the *Retriever* stage is almost constant for these scenarios. However, the *Detector* and *Tracking* stages consume most of the execution time, ranging from 27 % to 79 %, and from 13 % to 225 % of the entire processing time per frame, respectively. As depicted, the required time for actually estimating the traffic dynamic metrics is negligible.

Considering these scenarios, the detection effort in *General* is caused by false positives during the first stages of the detection. However, its tracking requirement is smaller with respect to the other two scenarios due to a lower traffic density. In *SoquelAve*, more vehicles are detected and tracked along the region of interest, which

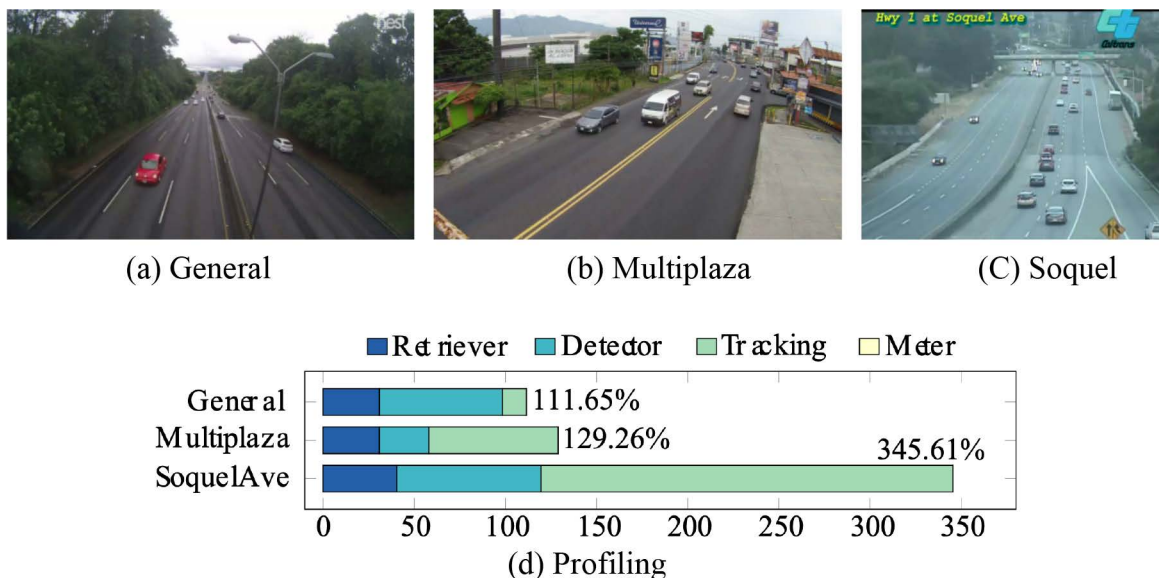


Figure 3. Test scenarios are used in the scope of this work and the execution profile of TDM for them. The execution time for each stage of the TDM presented is relative to a frame period (1/fps). A frame rate of 30 fps is defined as a performance constraint.

Note: Cámaras Viales CR. Cámaras Viales CR. Retrieved (2018).



is reflected in a high execution time required for each frame. *Multiplaza* has the least detection, but more tracking effort compared to *General*. In this case, the detection discards more elements early but tracks more vehicles due to a higher density.

The implementation of the proposed TDM deals with several implementation challenges, for instance, rotation, geometric transformations, scaling of the vehicles, and light variances in the scene. This was considered when selecting object detection and tracking algorithms. Haar-like features were more suitable for object detection than Local Binary Patterns (LBP). Although LBP were faster than Haar-like features (2.3x), its detection accuracy was lower (about 5 %).

Object tracking techniques are employed to complement the detection and make it more robust to catch false negative cases. Thus, once the application detects a vehicle, the *Tracker* is in charge of analyzing it in the following frames. One requirement for object tracking is transformation invariance. Adaptive-kernel trackers are used for their capability to adapt to the object transformation over time if the object does not change suddenly. The MOSSE filter adapts its kernel according to a valid detection, which helps the *Tracker* catch most of the transformations of the object during the video sequence. If multiple objects are detected, the same number of trackers are required, leading to linear growth in the computational requirement.

Optimizations

To improve the performance of our TDM, we explored two techniques at the software level: frame skipping and multi-threaded execution for object tracking.

Frame skipping: In dense traffic, vehicles do not move as fast as they do on a free road, affecting the traffic density. Inspired

by *loop perforation* (Sidiroglou-Douskos, Misailovic, Hoffmann, & Rinard, 2011), an approximate computing technique, and considering the traffic density metric, we propose to save execution time by skipping frames to be processed. We explored two approaches for frame skipping. The first considers a *fixed* number of frames to be skipped in-a-row. The second adjusts this number dynamically, considering the traffic density metric as a reference. The number of frames f to be skipped with this *dynamic* approach can be defined as:

$$f(\lambda) = \frac{F_{max} - 1}{\lambda_{max} - \lambda_{min}} \cdot \lambda + \frac{\lambda_{max} - F_{max}}{\lambda_{max} - \lambda_{min}}$$

where λ is the last known density, and λ_i are the minimum and maximum traffic density values reported for a video stream. The F_{max} is the maximum number of frames that can be skipped without exceeding a given maximum error with a *fixed* approach. We use relative error RE as error metric, calculated as follows:

$$RE = \frac{|(M_{accurate} - M_{approximate})|}{M_{accurate}} \times 100\%$$

where M corresponds to one of the traffic metrics. For each scenario, the accurate value for each metric was manually determined.

Both frame skipping approaches impact the accuracy of the estimations performed by our TDM, as depicted in Figure 4 for the case of *SoquelAve*. For the *fixed* approach, 3 and 2 frames were skipped for the *Detector* and *Tracker* stage, respectively. For the *dynamic* one, F_{max} was selected, such as $RE < 20\%$ for all the metrics. As shown, even the baseline implementation is not entirely accurate, and it presents errors with respect to a golden estimation. The *fixed*



approach improves the average speed and occupancy estimations regarding the baseline; however, it significantly increases the error for flow and density. The *dynamic* approach improves the metrics estimations concerning the baseline, except for average speed.

It is interesting to notice that approximate implementations of our TDM can produce better traffic metrics estimations. To understand this, it is required to analyze the effects of a tracking failure within the detection zone. Under certain conditions, such as vehicle rotation or light variance, the trackers cannot find the car and refresh themselves. This results in trackers stuck within the detection zone, affecting the detection of new vehicles. This issue happens less frequently when applying a frame skipping technique since some cars are detected later than the baseline, enhancing the tracking. It is worth mentioning that density and flow metrics depend on the total vehicles tracked and the speed and occupancy of the tracking itself.

Multi-Thread Object Tracking: As previously discussed, a new tracker is required each time the *Detector* stage finds a vehicle. Each tracker performs several operations: frame preprocessing, correlation

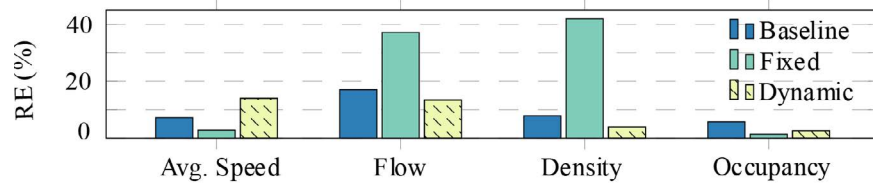


Figure 4. Effect of frame skipping in the accuracy of traffic dynamic metrics. These results correspond to the SoquelAve scenario, which is the worst-case scenario.

Note: Own source.

using FFT, Peak-to-Sidelobe Ratio (PSR) computation, and filter update. This last step is only executed if $PSR > th$, where $th = 5.7$ in the MOSSE filter implementation of OpenCV.

The computing effort spent by the filter update is comparable to the other previous steps performed during the tracking. Hence, the multi-thread optimization consists in using an additional *thread*, from the standard C++ library to compute the filter updates passed by a queue as soon as the thresholding has been applied and keeping the other steps of the *Tracking* stage on the main thread. Thus, the correlation and the filter update work in a pipeline manner, enhancing the time-to-solution for each tracker.

Evaluation

Figure 5 highlights the performance improvement obtained by executing our TDM with multi-thread object tracking

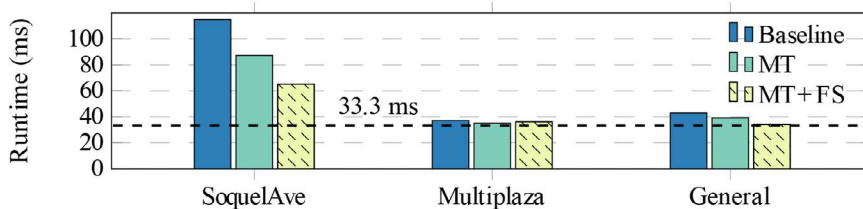


Figure 5. Runtime optimization for our TDM implementation with multi-thread object tracking (MT) and frame skipping (FS).

Note: Own source.

(MT) and frame skipping (FS). For the case of FS, we have used the *dynamic* approach as, in general, it provides metric estimations with smaller errors.



As it can be noticed, for the *Multiplaza* and *General* scenarios, using both optimizations (MT + FS), the execution time of the TDM is near the 33.3 ms per frame (noted in the graph) required to achieve a 30 fps throughput. For such scenarios, near 5 % and 25 % performance improvement was experienced, respectively, compared to the baseline performance of the TDM.

For the *SoquelAve* scenario, a more significant improvement was achieved. Compared to the baseline implementation a 1.32 x speedup improvement was achieved using the MT optimization. Considering MT + FS, the speedup reached was 1.76 x. Despite this performance improvement, the maximum throughput reached is about 15 fps, far below the desired 30 fps constraint. This is due to the high vehicle density found in this scenario, which, as previously discussed, increases the computational requirements for detection and tracking.

A detailed analysis of the optimized TDM when processing the *SoquelAve* scenario shows that the individual execution time of the *Retriever*, *Detector*, and *Tracker* stages are 20.4 ms, 17.1 ms, and 26.3 ms, respectively. This means that none of the individual stages of our TDM reaches more than 33.3 ms of the per-frame available execution time if a 30 fps throughput is required. The entire TDM can be further modified to adopt a pipeline implementation in a daisy chain fashion, in which each stage can be allocated to an individual computing engine. This will enable us to get higher throughput by sacrificing a small initial latency of 3 frames.

Conclusions

In this work, we proposed a TDM based on video feeds from road cameras

running on an SoC-based edge computing device. We tested our TDM with three different road scenarios, each of them with different traffic metrics.

With the optimizations proposed, our TDM delivered a performance near to 30 fps for scenarios with low and moderate traffic. For dense scenarios, our final TDM improved 76 % with respect to the baseline implementation but achieved a 15 fps throughput.

Further optimizations will consider using a low-power multi-core platform in which each stage of the TDM can be deployed to a single-core.

Acknowledgments

This work was possible thanks to the Costa Rica Institute of Technology (ITCR), particularly the Mobility Program for students and professors.

Conflict of Interest

The authors declare no competing interests.

Author contribution statement

All the authors declare that the final version of this paper was read and approved.

The total contribution percentage for this paper's conceptualization, preparation, and correction was as follows: L.L.V., 55 %; and J.C.G., 45 %.

Data availability statement

The data supporting the results of this study will be made available by the corresponding author, L.L.V., upon reasonable request.



References

- Bolme, D. S., Beveridge, J. R., Draper, B. A., & Lui, Y. M. (2010). Visual object tracking using adaptive correlation filters. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2544-2550. IEEE. <https://doi.org/10.1109/CVPR.2010.5539960>
- California Department of Transportation (Caltrans). (n. d.). Soquel Ave. Santa Cruz, California, United States of America. <https://cruz511.org/drive/traffic-conditions/traffic-cameras/>
- D'Andrea, E., Ducange, P., Lazzarini, B., & Marcelloni, F. (2015). Real-Time Detection of Traffic From Twitter Stream Analysis. *IEEE Transactions on Intelligent Transportation Systems*, 16, 2269-2283. IEEE. <https://doi.org/10.1109/TITS.2015.2404431>
- Giridharan, E. N., Kadaieaswaran, M., Arunprath, V., & Karthika, M. (2017, February). Big Data Solution for Improving Traffic Management System with Video Processing. *International Journal of Engineering Science and Computing*, 7(2), 4606-4609.
- Hall, F. D. (1992). *Traffic Stream Characteristics*. Federal Highway Administration. <https://www.fhwa.dot.gov/publications/research/operations/tft/chap2.pdf>
- Ministerio de Obras Públicas y Transportes. (n. d.). Cámaras viales CR. <https://www.camarasvialescr.com>
- Samie, F., Bauer, L., & Henkel, J. (2019). From Cloud Down to Things: An Overview of Machine Learning in Internet of Things. *IEEE Internet of Things Journal*, 6(3), 4921-4934. <https://doi.org/10.1109/JIOT.2019.2893866>
- Sidirogrou-Douskos, S., Misailovic, S., Hoffmann, H., & Rinard, M. (2011). Managing Performance vs. Accuracy Trade-Offs with Loop Perforation. *19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 124-134. <https://doi.org/10.1145/2025113.2025133>
- Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 511-518). IEEE. <https://doi.org/10.1109/CVPR.2001.990517>
- Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J., & Yang, X. (2017). A Survey on the Edge Computing for the Internet of Things. *IEEE Access*, 6, 6900-6919. <https://doi.org/10.1109/ACCESS.2017.2778504>
- Zhu, L., Yu, F., Wang, Y., Ning, B., & Tang, T. (2019). Big Data Analytics in Intelligent Transportation Systems: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 383-398. <https://doi.org/10.1109/TITS.2018.2815678>



Measuring Traffic Dynamics at the Edge (Luis Gerardo León-Vega • Jorge Castro-Godínez) *Uniciencia* is protected by Attribution-NonCommercial-NoDerivs 3.0 Unported (CC BY-NC-ND 3.0)