Software Impacts 20 (2024) 100636

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

RouteRecoverer: A tool to create routes and recover noisy license plate number data

Alberto Durán-López [a], Daniel Bolaños-Martinez [a,*], Luisa Delgado-Márquez [b], Maria Bermudez-Edo [a]

[a] *Computer Science School and Research Centre for Information and Communication Technologies (CITIC-UGR), University of Granada, C/ Periodista Daniel Saucedo Aranda S/N, 18071 Granada, Spain*
[b] *Department of Applied Economics, University of Granada, Campus of Cartuja, 18011 Granada, Spain*

ARTICLE INFO

Keywords:
Data recovery
Vehicles routes
License Plate Recognition
Smart villages
Internet of Things

ABSTRACT

License Plate Recognition (LPR) sensors often fail to detect vehicles or to identify all plate numbers correctly. This noise results in missing digits or an incomplete route of a vehicle, for example, missing one node (LPR camera) in the route. Addressing these issues, RouteRecoverer creates the route followed by a vehicle while efficiently recovering absent LPR plate digits, and filling gaps in routes. For example, when a vehicle is detected by LPR A and C, with the only route between them being B, our tool seamlessly retrieves the missing information, improving the data output.

Code metadata

Current code version	*v 1.0*
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2024-78
Permanent link to Reproducible Capsule	
Legal Code License	MIT License
Code versioning system used	*git*
Software code languages, tools, and services used	Python and Tkinter GUI
Compilation requirements, operating environments & dependencies	PIL, networkx, pandas, plotly, textdistance and python3-tk (Tkinter)
If available Link to developer documentation/manual	https://github.com/SmartPoqueira/RouteRecoverer/blob/main/README.md
Support email for questions	albduranlopez@ugr.es

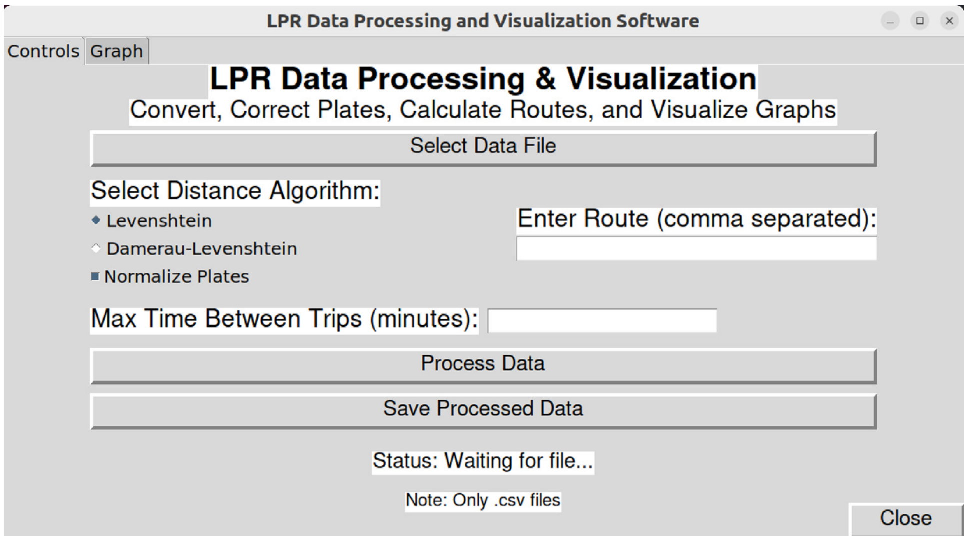
1. Introduction

Nowadays, cities are evolving towards smarter environments, adapting to the demands of their inhabitants, and seeking innovative solutions to enhance the quality of life [1]. This advancement has been propelled by the Internet of Things (IoT) paradigm and the widespread deployment of sensors and cameras in urban settings [2]. These devices leverage real-time data and automation to optimize various aspects of urban life, ranging from resource management to mobility and security [3,4]. Among the devices in the realm of smart cities are License Plate Recognition (LPR) systems for vehicular traffic detection and monitoring [5,6]. These systems utilize high-resolution cameras and machine learning algorithms to capture and analyze the license plate numbers of vehicles traversing urban roads, providing valuable data for traffic management, movement pattern analysis, or vehicle monitoring [7–9].

However, despite technological advancements, LPR systems currently face limitations [10]. In many cases, they encounter error rates in both license plate detection, stemming from inaccuracies in character recognition algorithms [11]; and vehicle detection, completely missing the detection of a vehicle. This may result in some vehicle positions not being tracked, making retrieving the vehicle's route difficult. These deficiencies are aggravated in environments where device connectivity is compromised due to poor coverage, such as in rural areas [12], or temporary system downtime caused by an excessive number of people connected to the same network simultaneously.

DOI of original article: https://doi.org/10.1016/j.inffus.2023.102164.
* Corresponding author.
E-mail addresses: albduranlopez@ugr.es (A. Durán-López), danibolanos@ugr.es (D. Bolaños-Martinez), luisadm@ugr.es (L. Delgado-Márquez), mbe@ugr.es (M. Bermudez-Edo).

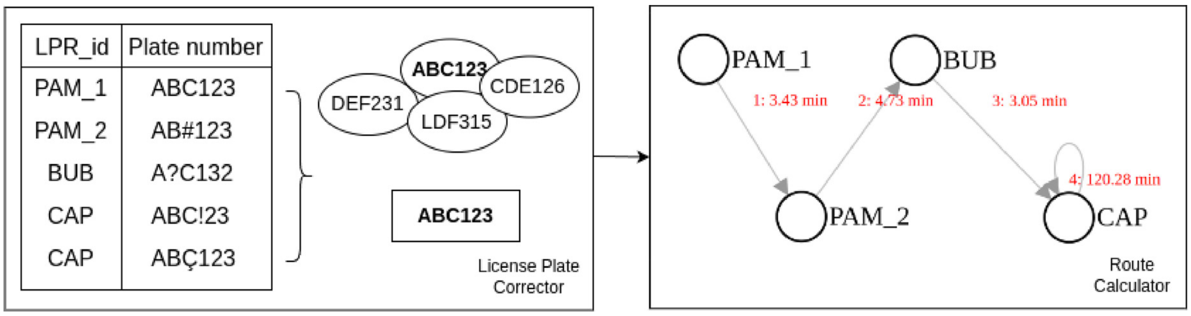
https://doi.org/10.1016/j.simpa.2024.100636
Received 14 March 2024; Received in revised form 22 March 2024; Accepted 25 March 2024


LPR Data Processing and Visualization Software
Controls
Graph
LPR Data Processing & Visualization
Convert, Correct Plates, Calculate Routes, and Visualize Graphs
Select Data File
Select Distance Algorithm:
Levenshtein
Damerau-Levenshtein
Normalize Plates
Enter Route (comma separated):
Max Time Between Trips (minutes):
Process Data
Save Processed Data
Close

Status: Waiting for file...

Note: Only .csv files

Fig. 1. LPR data processing and visualization software.



LPR_id	Plate number
PAM_1	ABC123
PAM_2	AB#123
BUB	A?C132
CAP	ABC!23
CAP	ABÇ123

Fig. 2. Example of some corrections on erroneous plate number detections of the plate number ABC123 in the left-hand side, and the plot of the route in the right-hand side.

Most efforts have focused on optimizing license plate detection algorithms to reduce recognition errors [13–15]. However, end users, most of the time, do not have access to the detection algorithms and have to deal with the outputs of the detection. In this context, we introduce RouteRecoverer, an LPR data recovery software designed to make the most of available information. Our software can retrieve a high percentage of missing plate digits. It also creates the routes the vehicles follow in an area by adding each node, or LPR camera, in the temporal order of detection of said vehicle. After that, our tool imputes missing nodes in a route. This approach enhances the effectiveness and reliability of LPR systems, thus contributing to the ongoing development of smart cities and mobility optimization.

2. RouteRecoverer overview

RouteRecoverer has two tabs (Fig. 1). The first tab is *Controls* and has all the application functionalities. The second tab, *Graph*, plots all the routes as in the right-hand side of Fig. 2. In the tab *Controls*, the user has to upload a Comma-Separate Values (CSV) file with the button "Select Data File". The CSV file should have a first column with the LPR identification and a second column with the plate numbers, see Table 1. After that, the user has to select the distance algorithm, which will compare a plate number with missing digits to other plate numbers in the CSV file. The tool in its current version has two distance algorithms. The user can also select a previous normalization step, which cleans special characters (see Section 3, for details). Optionally, the user can also enter routes. For example, if a vehicle to go from LPR-A to LPR-C necessarily has to pass through LPR-B, and similarly with route LPR-X, LPR-Y, LPR-Z, the user can write the two routes as: *LPR-A, LPR-B, LPR-C; LPR-X, LPR-Y, LPR-Z*. These routes will be used to input missing values. For example, if LPR-A and LPR-C have detected a vehicle but LPR-B has not, the RouteRecoverer will input the missing value.

Table 1
Example of an input file. The file is a CSV with three columns: the detected plate number, the camera (identification) that has detected the plate number, and the timestamp of the detection.

num_plate	LPR_id	timestamp
ZYW535	PAM_1	2023-03-19 14:08:00
ABC123	PAM_1	2023-03-19 15:57:00
ABC123	PAM_2	2023-03-19 16:00:25
ABC123	BUB	2023-03-19 16:05:09
ABC123	CAP	2023-03-19 16:08:12
ABC123	CAP	2023-03-19 18:08:29
ZYW535	PAM_1	2023-03-19 20:17:26
ZYW535	PAM_2	2023-03-19 20:21:43
ZYW535	BUB	2023-03-19 20:25:00
...	...	...

The user can also optionally personalize the maximum time between trips in minutes. This last parameter indicates the maximum allowed time (in minutes) between consecutive camera detections to consider them part of the same trip. If the time difference between two consecutive detections by two cameras of the same vehicle exceeds this threshold, the latter detection is treated as the start of a new trip. Essentially, it is used to differentiate between separate visits or trips made by the same vehicle. For example, if our threshold is 1 day (in minutes: 1440), if a vehicle is detected by different cameras in the area in one day and the next detection of that vehicle is in 20 days, we will start a second trip (meaning a second graph for this vehicle). After that, the user can process the data by pressing the corresponding

Table 2
Example of an output file. It is a CSV with five columns: the plate number, the route, the times between two nodes (or cameras), the start time of the route, and the finish time of the route.

num_plate	route	times	entry_date	exit_date
ZYW535	['PAM_1', 'PAM_1', 'PAM_2', 'BUB']	[369.44, 4.29, 3.28]	2023-03-19 14:08:00	2023-03-19 20:25:00
ABC123	['PAM_1', 'PAM_2', 'BUB', 'CAP', 'CAP']	[3.43, 4.73, 3.05, 120.28]	2023-03-19 15:57:00	2023-03-19 18:08:29
BPN277	['CAP', 'PAM_2', 'BUB', 'BUB', 'PAM_1']	[3.7, 4.43, 915.33, 4.57]	2023-03-19 16:21:00	2023-03-20 7:49:01
EHK389	['PAM_1', 'PAM_2', 'PAM_2', 'PAM_2']	[3.2, 820.36, 354.38]	2023-03-20 8:48:00	2023-03-21 4:25:56
NCQ886	['PAM_1', 'PAM_2', 'BUB', 'PAM_2', 'CAP', 'BUB']	[4.26, 4.47, 4.45, 4.98, 4.91]	2023-03-20 12:20:00	2023-03-20 12:43:04
ZTP678	['PAM_1', 'CAP', 'PAM_1', 'PAM_1', 'PAM_1']	[3.95, 3.79, 216.67, 166.15]	2023-03-20 14:06:00	2023-03-20 20:36:33
RZX028	['CAP', 'CAP', 'PAM_2', 'PAM_1', 'PAM_2']	[316.66, 4.26, 4.47, 3.81]	2023-03-20 14:45:00	2023-03-20 20:14:12

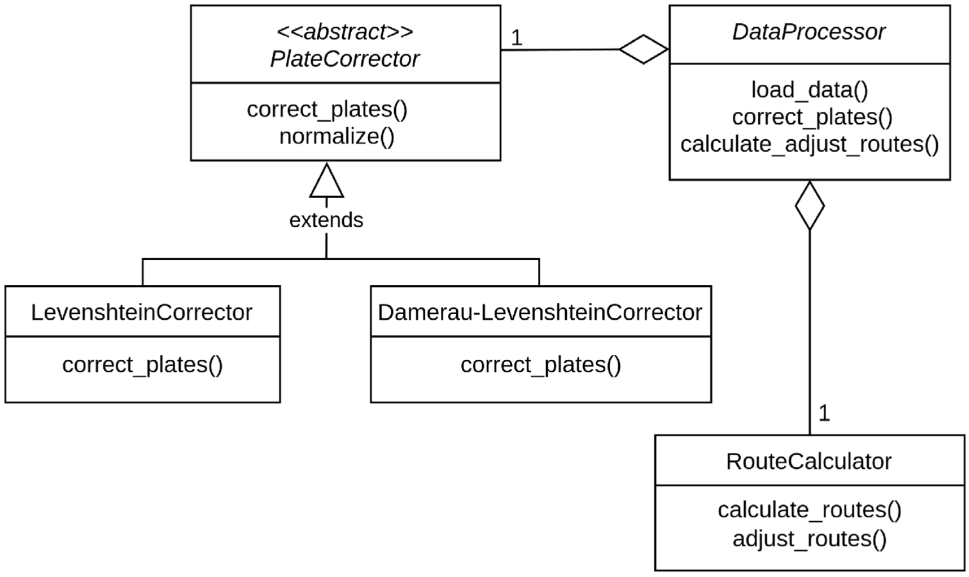


Fig. 3. UML class diagram of the software design.

button. RouteRecoverer then will recover all the plate numbers with missing values that it can. For example, in the left-hand side of Fig. 2 RouteRecoverer has recovered the plate number ABC123 from different erroneous plate numbers, such as AB#123, being # a missing value. Then, RouteRecoverer creates the routes followed by each vehicle, as in Table 2, which is the output file. In Tables 1 and 2 we have highlighted the detection of the vehicle ABC123 by different LPRs and the calculated route. The user can save the output file with the button "Save Processed Data". All the routes of all vehicles can be seen in the tab *Graph*. The route followed by vehicle ABC123 in the example looks like the right-hand side of Fig. 1.

3. Software description

The system is based on a modular architecture that implements the Strategy design pattern, allowing the dynamic selection of the correction (or distance) algorithms at runtime. The software's structure consists of various classes that interact with one another, as illustrated in the UML diagram shown in Fig. 3. The *PlateCorrector* abstract class defines a contract for plate correction algorithms. This design simplifies the incorporation of future license plate correction methods. In our tool, we have developed two different concrete implementations of this abstract class: *LevenshteinCorrector* and *DamerauLevenshteinCorrector*, providing various strategies for addressing the variability in correcting errors in license plates. These strategies include several combinations of the presence/absence of the normalization and the use of the different correction algorithms alone or in combination. We have implemented two correction algorithms, one based on the Levenshtein distance and the other on the Damerau–Levenshtein distance. These distances are defined as follows:

1. **Levenshtein Distance:** The Levenshtein distance is a metric measuring the edit distance between two strings. It is defined as the minimum number of single-character edit operations required to transform one string into another. The allowed edit operations are insertions, deletions, and substitutions of a single character. Formally, for two strings a and b, with lengths $|a|$ and $|b|$ respectively, the Levenshtein distance, denoted as $dist_{a,b}(i,j)$, is defined recursively as follows:

$$dist_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j)=0, \\ \min \begin{cases} dist_{a,b}(i-1,j)+1, \\ dist_{a,b}(i,j-1)+1, \\ dist_{a,b}(i-1,j-1)+\mathbb{I}(a_i \neq b_j) \end{cases} & \text{otherwise.} \end{cases}$$

Here, $dist_{a,b}(i,j)$ represents the distance between the first i characters of a and the first j characters of b. The operations within the min function correspond to the costs associated with deletion, insertion, and substitution, respectively. The indicator function $\mathbb{I}(a_i \neq b_j)$ is 0 when $a_i = b_j$, indicating no cost for substitution in this case, and 1 otherwise, indicating a substitution cost.

2. **Damerau–Levenshtein Distance:** The Damerau–Levenshtein distance extends the concept of the Levenshtein distance by including transpositions among its set of allowable edit operations, offering a more precise approach to measuring similarity. In particular, it considers the cost of transpositions of adjacent characters in addition to the operations considered in the Levenshtein distance. It is defined as:

$$dist_{a,b}(i,j) = \begin{cases} \text{previous definition} & \text{for the base and recursive cases,} \\ dist_{a,b}(i-2,j-2)+1 & \text{if } i,j>1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j. \end{cases}$$

We implemented an optional method in the class *PlateCorrector* to normalize plates before running the distance algorithm. The normalize method removes non-alphanumeric characters erroneously introduced by the License Plate Recognition (LPR). This method enhances the correction methods robustness. After plate correction, the *RouteCalculator*, not only calculates the routes, but also optimizes them, ensuring coherence and data accuracy by including missing steps in the route of a vehicle. For example, suppose one vehicle is detected by sensors A and C (route A-C), but the only route that links A to C is passing through sensor B. In that case, our *RouteCalculator* will infer that said vehicle has followed the route A-B-C. Algorithm 1 shows the complete functionality of our tool.

Algorithm 1: Algorithm for Correcting Plates Number and Calculating Routes

```
Data: $filepath$, $MAX_T$, $distanceAlgorithm$: Corrector,
      $routeCalculator$: RouteCalculator
Result: Processed $data$
1  Load and Prepare Data:
2  Load and sort $data$ from $filepath$
3  Correct Plates Number:
4      foreach $plate\_number$ to correct in $data$ do
5          if normalize then
6              $plate\_number \leftarrow$ Normalize($plate\_number$)
7          Find $minDistance$ using $distanceAlgorithm$
8          Assign closest match based on $minDistance$
9  Remove invalid Plates Number entries
10 Calculate Routes:
11     foreach $plate\_number$ in unique $plate\_numbers$ of the $data$ do
12         Initialize the $route$ with the first occurrence of the plate
            number
13         foreach $node$ grouped by $plate\_number$ do
14             $\Delta t \leftarrow TimeDifference(node, previousNode)$
15             if $\Delta t > MAX_T$ then
16                 $route \leftarrow route \cup PreviousNode$
17                 Reinitialize $route$ with $node$
18             $route \leftarrow route \cup node$
19         Routes $\leftarrow$ Routes $\cup$ $route$
20 Adjust Routes:
21     foreach $route$ in all $Routes$ of the $data$ do
22         Initialize $refinedRoute$ and $refinedTimes$
23         foreach $node$, $nextNode$ in consecutivePairs($route$) do
24             Apply $insertions$ between $node$ and $nextNode$
25             Distribute $times$ across inserted nodes
26         $route \leftarrow$ Update $route$ with [$refinedRoute$, $refinedTimes$]
27 return Processed $data$
```

4. Tool validation

RouteRecoverer was used in a study to cluster vehicle behaviors in a rural area [7], using a LPR dataset [16]; recovering erroneous plate numbers, calculating routes, and assigning missing values in the routes. Out of 18,955 erroneously registered license plates, our tool corrected around 88%. In particular, Table 3 shows the results utilizing a combination of the implemented distance algorithms. The first configuration used only Levenshtein, the second used a previous normalization, the third used only Damerau–Levenshtein, and the fourth used a previous normalization. As expected, we can appreciate from the results that the fourth configuration is the one recovering the most plate numbers. We can observe that the normalization step before Levenshtein recovers more plate numbers than the extension of Levenshtein (Damerau–Levenshtein) alone. The results indicate high effectiveness in error correction and accuracy in license plate retrieval, suggesting the feasibility of the software for mobility studies, urban planning, tourism, and socio-demographic analysis using LPRs.

Table 3
Total corrected license plates for each of the algorithms implemented in our software.

	Number of corrections
Levenshtein	16 616
Normalized Levenshtein	16 626
Damerau–Levenshtein	16 621
Normalized Damerau–Levenshtein	**16631**

5. Impact

The correction and recovery of lost license plates are valuable to many groups in society; to technical groups, and to interdisciplinary teams. Our system has an impact on multiple dimensions, as detailed below:

- **Data Scientists and Researchers**: Our software helps analyze traffic data more accurately. By correcting errors and recovering missing detections, data scientists can be confident in the integrity of the data, leading to more robust conclusions in those studies related to vehicular mobility behavior and patterns.
- **Toll collection processes:** LPRs enable seamless and automatic deduction of toll fees from drivers' accounts using their license plate numbers. A prime illustration is the Dartford Crossing's Dart Charge system[1] utilized in United Kingdom. Here, LPR technology is seamlessly integrated to automatically deduct toll fees as vehicles cross, without the need for toll booths. This not only eliminates the need for manual payment but also contributes to reducing congestion, showcasing how enhanced data recovery in LPR systems can streamline toll-collection processes and enhance overall transportation efficiency.
- **Parking management practices:** With improved capabilities, authorities can swiftly identify vehicles parked in prohibited areas or those exceeding designated time limits. For instance, in the city of London, Transport for London utilizes LPR systems to enforce parking regulations and alleviate congestion by swiftly identifying and penalizing vehicles parked in restricted zones. This illustrates how advancements in data recovery within LPR systems can significantly enhance parking management efficiency and effectiveness.
- **Traffic management:** By ensuring the retrieval of lost data, it enables the generation of accurate and reliable information regarding vehicle movements. This information can be used for optimizing traffic flow, mitigating congestion, and ultimately enhancing transportation efficiency. For instance, in Norway, the Norwegian Public Roads Administration utilizes LPR systems[2] integrated with advanced data recovery software to monitor and analyze traffic patterns in real-time. This allows for proactive measures to be taken, such as adjusting traffic signal timings or implementing traffic diversions, to alleviate congestion and improve the overall flow of vehicles.
- **Law enforcement:** These improvements enable swift and accurate identification of stolen vehicles and those involved in criminal activities. For example, the Los Angeles Police Department's utilization of LPR technology[3] showcases how such advancements allow for the rapid tracking of suspects and vehicles implicated in criminal incidents based on license plate data.

1 https://www.rac.co.uk/drive/advice/legal/the-dartford-crossing-charge/
2 https://intrada.q-free.com/2020/05/26/norway-tolling/
3 https://www.lapd.com/blog/protecting-officers-and-public-lpr-technology

6. Limitations and future work

Although the presented software achieves great results in recovering missing values in LPR systems, it has limitations when dealing with route corrections. We have optimized the route recovery function for smart villages with only a few LPRs and potential routes between two cameras. A scalability test should be necessary for smart cities with numerous LPRs, different intersections, and several potential paths between two cameras. Additionally, in the future, we will add new distance metrics as implementation classes of the abstract class *PlateCorrector*.

CRediT authorship contribution statement

Alberto Durán-López: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis. **Daniel Bolaños-Martinez:** Writing – review & editing, Investigation, Conceptualization. **Luisa Delgado-Márquez:** Writing – review & editing, Investigation, Conceptualization. **Maria Bermudez-Edo:** Writing – review & editing, Supervision, Resources, Project administration, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This publication is part of the R&D&i Project Ref. C-SEJ-128-UGR23, co-financed by the Regional Ministry of University, Research and Innovation, Spain and by the European Union under the Andalusia ERDF Program 2021–2027, and also by the project "Thematic Center on Mountain Ecosystem & Remote sensing, Deep learning-AI e-Services University of Granada-Sierra Nevada" (LifeWatch-2019-10-UGR-01), which has been co-funded by the Ministry of Science and Innovation, Spain through the FEDER funds from the Spanish Pluriregional Operational Program 2014–2020 (POPE), LifeWatch-ERIC action line. The project has also been co-financed by the Provincial Council of Granada.

References

[1] D. Saba, Y. Sahli, B. Berbaoui, R. Maouedj, Towards smart cities: challenges, components, and architectures, in: Toward Social Internet of Things (SIoT): Enabling Technologies, Architectures and Applications: Emerging Technologies for Connected and Smart Social Objects, Springer, 2020, http://dx.doi.org/10.1007/978-3-030-24513-9_15.

[2] B.P.L. Lau, S.H. Marakkalage, Y. Zhou, N.U. Hassan, C. Yuen, M. Zhang, U.-X. Tan, A survey of data fusion in smart city applications, Inf. Fusion 52 (2019) 357–374, http://dx.doi.org/10.1016/j.inffus.2019.05.004.

[3] M.L.M. Peixoto, A.H. Maia, E. Mota, E. Rangel, D.G. Costa, D. Turgut, L.A. Villas, A traffic data clustering framework based on fog computing for VANETs, Veh. Commun. 31 (2021) 100370, http://dx.doi.org/10.1016/j.vehcom.2021.100370.

[4] R.P. Centelles, F. Freitag, R. Meseguer, L. Navarro, S.F. Ochoa, R.M. Santos, A lora-based communication system for coordinated response in an earthquake aftermath, Multidiscip. Digital Publ. Inst. Proc. 31 (1) (2019) 73, http://dx.doi.org/10.3390/proceedings2019031073.

[5] M. Spanu, M. Bertolusso, G. Bingöl, L. Serreli, C. Castangia, M. Anedda, M. Fadda, M. Farina, D.D. Giusto, Smart cities mobility monitoring through automatic license plate recognition and vehicle discrimination, in: 2021 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB, IEEE, 2021, pp. 1–6, http://dx.doi.org/10.1109/BMSB53066.2021.9547163.

[6] K. Arthi, S. Suchitra, A. Shobanadevi, K.S. Babu, Vehicle monitoring system detection using deep learning technique, in: Recent Trends in Computational Intelligence and Its Application: Proceedings of the 1st International Conference on Recent Trends in Information Technology and Its Application (ICRTITA, 22), CRC Press, 2023, p. 61, http://dx.doi.org/10.1201/9781003388913-8.

[7] D. Bolaños-Martinez, M. Bermudez-Edo, J.L. Garrido, Clustering pipeline for vehicle behavior in smart villages, Inf. Fusion 104 (2024) 102164, http://dx.doi.org/10.1016/j.inffus.2023.102164.

[8] W. Yao, C. Chen, H. Su, N. Chen, S. Jin, C. Bai, Analysis of key commuting routes based on spatiotemporal trip chain, J. Adv. Transp. 2022 (2022) http://dx.doi.org/10.1155/2022/6044540.

[9] W. Yao, J. Yu, Y. Yang, N. Chen, S. Jin, Y. Hu, C. Bai, Understanding travel behavior adjustment under COVID-19, Commun. Transp. Res. 100068 (2022) http://dx.doi.org/10.1016/j.commtr.2022.100068.

[10] J. Shashirangana, H. Padmasiri, D. Meedeniya, C. Perera, Automated license plate recognition: A survey on methods and techniques, IEEE Access 9 (2020) 11203–11225, http://dx.doi.org/10.1109/ACCESS.2020.3047929.

[11] N. Watcharapinchai, S. Rujikietgumjorn, Approximate license plate string matching for vehicle re-identification, in: 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS, IEEE, 2017, pp. 1–6, http://dx.doi.org/10.1109/AVSS.2017.8078538.

[12] P. Flores-Crespo, M. Bermudez-Edo, J.L. Garrido, Smart tourism in villages: Challenges and the alpujarra case study, Procedia Comput. Sci. 204 (2022) 663–670, http://dx.doi.org/10.1016/j.procs.2022.08.080.

[13] Y. Wang, Z.-P. Bian, Y. Zhou, L.-P. Chau, Rethinking and designing a high-performing automatic license plate recognition approach, IEEE Trans. Intell. Transp. Syst. 23 (7) (2021) 8868–8880, http://dx.doi.org/10.1109/TITS.2021.3087158.

[14] C.-L. Chang, P.-J. Chen, C.-Y. Chen, Semi-supervised learning for YOLOv4 object detection in license plate recognition system, J. Imag. Sci. Technol. 66 (4) (2022) http://dx.doi.org/10.2352/J.ImagingSci.Technol.2022.66.4.040404, 040404–1.

[15] H. Nguyen, A high-performance approach for irregular license plate recognition in unconstrained scenarios, Int. J. Adv. Comput. Sci. Appl. 14 (3) (2023) http://dx.doi.org/10.14569/IJACSA.2023.0140338.

[16] D. Bolaños-Martinez, M. Bermudez-Edo, J.L. Garrido, B.L. Delgado-Márquez, Spatio-temporal dynamics of vehicles: Fusion of traffic data and context information, Data Brief 110084 (2024) http://dx.doi.org/10.1016/j.dib.2024.110084.