An IoT Transport Architecture for Passenger Counting: A Real Implementation

Janine Kniess[1], Julio Cezar Rutke[1], William Alberto Cruz Castañeda[1]
Graduate Program in Applied Computing, Santa Catarina State University, Brazil[1]
janine.kniess@udesc.br[1], rutke.julio@gmail.com[1], williamalberto.cruz@gmail.com[1]

***Abstract*—A smart urban mobility considers the technological-based development in transport systems, services and their use to improve the movement of people and goods inter or intra-cities. Cities increasingly face problems caused by traffic and need to find solutions to improve mobility and to reduce congestion and pollution. In this paper, we present a Passenger Counting Scheme into buses. Embedded counting algorithms based on Computer Vision are proposed to count passenger arrivals and departures in real time. Experiments in a real scenario expose that the proposed scheme achieved a high percentage of success in passenger count.**

***Index Terms*—Smart Transportation, Passenger Counting Algorithm, Urban Mobility, Smart Cities**

I. Introduction

Sustainable urban developments depend on the successful management of urban growth, including infrastructure, as well as, basic services. The smart city concept integrates Internet of Things (IoT) paradigm to improve city operations and services into six dimensions: environment, governance, economy, people, living and mobility [1]. Among the aforementioned dimensions, mobility systems are interesting to study, due to the lack of information, such as unavailability of routes, passengers traveling on each route, estimated arrival time, and poor distribution and overcrowding, and, they can be shaped into a smart mobility system. According to [2], the main objectives of smart mobility are the reduction of pollution, traffic congestion, noise pollution and transfer costs, increase mobility safety and transfer speed. Therefore, we propose a Passenger Counting Scheme for urban public buses. This scheme is composed of counting algorithms, that involve identification, tracking and counting functions, as well as an aggregation algorithm, that manages strategies for data transmission.

The counting scheme is embedded in micro-controllers that were installed on a real public bus. Passenger images were obtained through cameras, subsequently, stored and processed in micro-controllers. We developed one algorithm for detection and tracking based on Computer Vision. The passenger counting algorithm identifies who enter/leave in the bus, and follows their route while in the camera range. It developed a counting module to be attached to the bus door. Afterward, the Counting module transmits the processed information via wireless network to an aggregator module fixed in the bus. The Aggregator module is responsible for unifying information from all ports and sending them to a network outside the bus. Internet of Things (IoT) protocols, such as, COAP [3] and MQTT [4] were used to reduce the use of resources as memory and processing. Transmission tests with technologies 802.11n and GPRS were carried out, aiming to evaluate the potential of these technologies in sending data on the bus between micro-controllers and also to an external network. The paper is organized as follows. Section 2 describes related work. Section 3, describes the embedded in-route counting scheme as well as the computer vision-based mechanisms for counting passengers. Section 4, presents the results obtained from the implementation in a real scenario. Finally, Section 5, presents the conclusions and future works.

II. Related Work

Counting people approaches can be classified according to the following properties: Position and Depth, Person Size in Picture, Detection and Tracking and Processing and Aggregation. Yahiaoui et al. [5] proposed counting passengers in buses through stereo vision position using two cameras. The similarity in grayscales between two images sought to remove the background, identify similar neighboring pixels and detect edges to recognize passengers. The framework of Mukherjee et al. [6] detects and tracks passengers in a train station. Detection is performed through cameras located on the top of station entryways. The system recognizes passengers by the circular shapes of their heads.

Boreiko et al. [7], presented a system for passenger counting and tracking on buses that counts through two cameras (ceiling and above the doors). A Raspberry Pi put together recorded images along with GPS location and transmits the data to an external server via a 3G modem. In Escolano et al. [8] cameras are installed at the bus doors. The system is based on estimation of optical flow to detect and monitor the entry and exit of passengers. In [9], an Automatic Bus passenger Counting System (iABACUS) is presented. In iABACUS the passenger count is based on the detection of Wi-Fi signatures coming from mobile phones. Data are transferred to the cloud through Wi-Fi at the bus station. Unlike the related works, the Passenger Counting Scheme presented in Section III performs the counting of passengers in an embedded system fully loaded on the bus. We use cameras to capture passengers images. Among the detection and tracking algorithms we chose those with the best accuracy. So for detection we used HOG [10] and Haar Likes [11] classifiers and for tracking we developed one (III-A).

III. Passenger Counting Scheme

The scheme for detecting and counting passengers has been specified with embedded devices to provide support for counting flow on each route of the bus, as well as, to provide these information to stakeholder. The overall passenger counting scheme with four layers is illustrated in Figure 1. The Perception Layer recognizes and collect data from the environment through sensors and cameras. The Collected Data Treatment Layer algorithms identify individuals in the images and count them as they pass through the door. The Communication Layer manages communication interfaces as well as transmission with external networks. The Application and Data Storage Layer brings together information handling and presentation of the processed data. Their implementation can be in the cloud or in an external server.

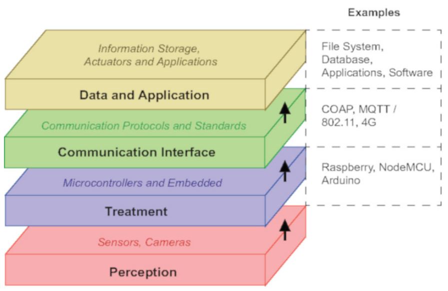

Examples
Information Storage, Actuators and Applications
Data and Application
File System, Database, Applications, Software
Communication Protocols and Standards
Communication Interface
COAP, MQTT / 802.11, 4G
Microcontrollers and Embedded
Treatment
Raspberry, NodeMCU, Arduino
Sensors, Cameras
Perception

Fig. 1: Counting Scheme Layers.

A. Counting and Aggregation: Algorithm

The algorithm for detection and counting was organized as represented in Figure 2: Door activation and Detection, Passenger counting, Aggregation, Communication and Application. The system begins with the door activation signal (open) coming from the Perception layer. Afterwards, when the output signal (close) is detected, the counting module acquires and transfer images to the Collected Data Treatment layer. If the count is equal to zero, the image is deleted. If the count is greater than zero, the Communication Interface layer transmits to the Aggregation module the number of passengers. Lastly, the aggregation module receives from the Communication Interface layer an update message, and replies an acknowledgment message notifying the process completion.

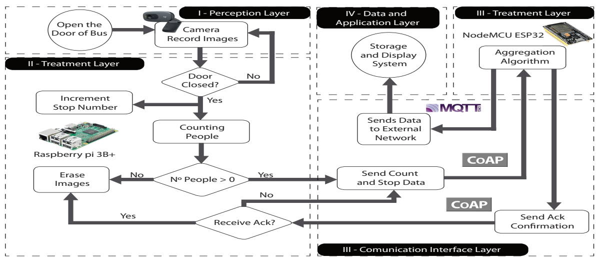


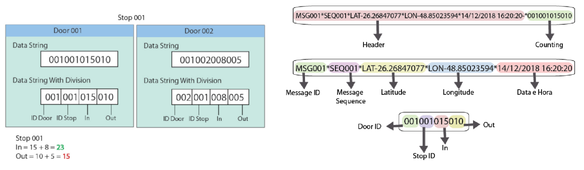
Fig. 2: Flowchart Counting Algorithm.

The counting approach (Algorithm 1) establishes three main functions: identification, tracking and counting. Identification includes passenger detection using Haar Likes [11] or HOG [10] classifiers. Tracking and Counting involves trajectories from individuals that cross a boundary line. Initially, the algorithm defines a vector that will store the positions of the identified passenger, stores information about him and performs the count. The algorithm set up limits of maximum and minimum size to avoid possible false positives. A counting and distance boundary line among objects for tracking purposes is defined. Frame acquisition, analysis and video processing (lines 5 to 14) are performed frame by frame to convert them in grayscale. Afterward, a classifier, defined on an external computer, is trained and retrieves a vector containing identified passengers in frames.

Algorithm 1: Passenger Accounting

```
1  classifier = getCascadeClassifier('classifier.xml');
2  Begin
3      video = captureimage('videocaptured.avi');
4      nFrames = 0;
5      for all frames to be analyzed do
6          frame = readNextFrame(video);
7          nFrames $+ =$ 1;
8          if $nFrames < 1$ then
9              previous_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY);
10         end
11         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY);
12         people = classifier.getPeople(gray, 1, 5, cv2.CASCADE_SCALE_IMAGE, (SZ_LIMIT1,
             SZ_LIMIT1), (SZ_LIMIT2, SZ_LIMIT2));
13         findPeople(people);
14     end
15 Function findPeople(people)
16     for each $people \in people[]$ do
17         rectangle_center = [people.getX + people.getWidth/2, people.getY + people.getHeigth/2];;
18         for each $p \in people\_list[]$ do
19             distance = sqrt((p.getX - rectangle_center.getX) ** 2 + (p.getY - rectangle_center.getY)
                 ** 2);
20             if $lowest\_distance > distance$ then
21                 lowest_distance = distance;
22                 closest_index = p.index;
23             end
24         end
25         if $lowest\_distance > distance\_threshold$ then
26             closest_index = None;
27         end
28         if $closest\_index\ != None$ then
29             if $lowest\_distance < distance\_threshold$ then
30                 updatePosition(rectangle_center);
31                 count(limit_line1);
32             end
33             else
34                 new_person = rectangle_center;
35                 people_list.append(new_person);
36             end
37         end
38     end
39 end
```

The function *findPeople()* (lines 16 to 26) sets up a vector and performs the tracking, establishing central points in rectangles to mark passengers. It analyses records detected in frames, and accomplishes a comparison with a list of previous recordings of movements to find out if the frame is already added. Thus, the vector stores a recognized passenger and a whole list of identified individuals in the counting process. The Euclidean distance (line 28) is calculated for each vector regarding each passenger to identify the closest. If the distance among the list and the vector is greater than one threshold, it means that there is no equal passenger record. Also, an update function compares if the record has an equivalent position. With this position, one function compares the central points for a possible identified passenger and seeks to recognize the boundary limit. If in a previous frame the point is below the boundary line and in the current frame the point is in the same



(a) (b)

Fig. 3: Message Payload (a), Message Header (b).

line or above, an entry is accounted. Otherwise, a leaving is accounted.

There is communication between the Counting module coupled in the bus doors to the Aggregator module. The aggregation algorithm operates in the Treatment layer and it encompasses strategies to manage counting data and transmit them. Moreover, manipulates a list of bus stops in a route. When the route starts, the list is empty and is incremented at each bus stop. The primary objective of the aggregation algorithm is to transmit the count data in string format. The images are analyzed and processed directly in the counting module of the door micro-controllers. Figure 3a illustrates the message structure including 3 digits variables. As depicted in Figure 3b, the header includes: message identifier (MSG001), sequential message number (SEQ001), latitude, longitude and date-time of the stop. In Figure 3a, the body includes: port identification, stop identification, number of passenger entries, and number of passenger exits. The stop identifier receives the value 0 (zero) when the bus finishes the route. The system uses location data obtained from a GPS in the micro-controller to identify the end of the route.

The Aggregation algorithm is required to add entries and exits of several ports, as well as, to take the total number of passengers at each stop. In addition, identifies whether the received stop already exists or not. If no stop identifier is found on the list, the received identifier is inserted in the list. Thus, the algorithm sends a confirmation message to the counting module, which notifies that the data has been received. When the Counting module completes processing and sends the data to the Aggregator module, it awaits confirmation (ACK count). At that moment, the counting module can delete the videos. If this confirmation does not return within a period of time, the system assumes that communication has been lost and data is forwarded. In order to ensure counting reliability and transmission integrity, the data stored along the route are send as messages to an external server to find possible divergences or message loss.

IV. Experiments and Results

Most of the public transport companies in Brazil implement an integrated ticketing system that works as follow: Passengers access a bus terminal through a ticket reader. Thus, they can take any bus entering by the back or central doors. When a bus arrives at a new terminal, they can change buses without a new payment. We installed a counting scheme composed by cameras and micro-controller on a bus to capture passengers flow and perform the count [12]. The following hardware were used in a real scenario: Raspberry Pi 3b+, NodeMCU ESP32 with a GPRS module, and a monocular webcam Logitech HD C270 with a resolution of 1280x720 pixels. The packages were captured using the Wireshark [13]. The confidence interval for the results is 95%. The camera was positioned in the zenith position above the door with the camera facing the ground. Videos were recorded, with a rate of 30 frames per second. Table I presents a description of the collected videos.

TABLE I: Classification: Collected Videos

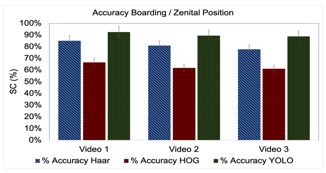
Videos	Passengers	Diurnal	Nocturnal	Position	Direction
Video 1	27	✓		zenith	Embarking
Video 2	21	✓		zenith	Embarking
Video 3	18	✓		zenith	Embarking
Video 4	25		✓	zenith	Embarking
Video 5	20		✓	zenith	Embarking
Video 6	28	✓		zenith	Disembarking
Video 7	25	✓		zenith	Disembarking
Video 8	30	✓		zenith	Disembarking
Video 9	27		✓	zenith	Disembarking
Video 10	22		✓	zenith	Disembarking

A total of 1400 positive and negative images were used. Each positive image was compared to all negatives ones, which generated a mapping of 1400 vectors used for training. Negative images were obtained from same videos, but with clippings of areas where people are not, such as handrails, stairs, floors, and also backpacks and hats. In the training which was carried out in 10 stages, 5000 combinations were applied among positive images and 2500 combinations among negative images. Table II presents the processing time into the micro-controller applying Haar Likes [11] with an Adaboost classifier [14] and HOG [10]. The potential of the algorithm was evaluated with the Deep Learning technique, *YOLOV3* [15]. In the HOG technique, the standard OpenCV training was used with a SVM classifier. The processing time with HOG was higher when compared with Haar. HOG performs a pixel by pixel calculation to find the histogram of the image and detects the edges of the heads, while Haar optimizes this search by calculating the integral of an image. Meanwhile, at the YOLOV3, the video processing time is higher than Haar and HOG because it uses 53 convolutional layers that require a large processing power.

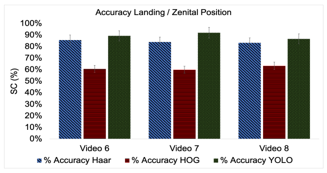
Figure 4 presents the success percentage rate (number of passengers entering/leaving the bus *SC*) related to diurnal boarding and landing. HOG presented a *SC* of 72% on average for videos 1, 2 and 3 with a false positive o (passengers counted by the system wrongly) of 7% on average. With Haar Likes the *SC* was around 85.0% with an average of 3% of false positives. For YOLOV3 was on average 92% without false positives. Figure 4b presents results concerning the diurnal disembarkation. For Haar the *SC* was 86% and 3.33% are false positives. High number of false positives occurred because the passenger carries a backpack or enters quickly on the bus. For HOG the low percentage, is due to the applied classifier that uses a standard training provided by the OpenCV. The processing time training is very high, making it unfeasible to use it an embedded micro-controller in real time. For YOLOV3 the success percentage rate (diurnal landing) was 92%.

TABLE II: Counting Results: Processing Time (s)

Videos	Passengers	Frames	Haar	Hog	Yolov3
Video 1	27	949	99	1780	2102
Video 2	21	444	43	737	893
Video 3	18	387	37	703	843
Video 4	25	867	85	1439	1715
Video 5	20	431	42	756	864
Video 6	28	634	61	1048	1298
Video 7	25	524	52	884	1012
Video 8	30	843	81	1355	1611
Video 9	27	732	70	1196	1479
Video 10	22	412	38	722	835



(a)

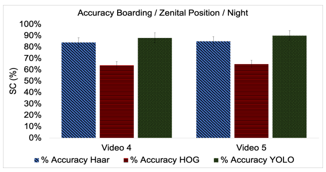


(b)

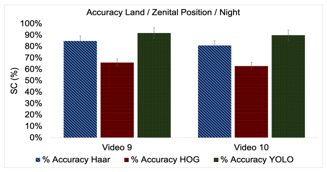
Fig. 4: Diurnal Boarding (a), Diurnal Landing (b): Zenith Position.

The Figures 5a and 5b depict success percentage rate regarding nocturnal boarding and landing. With Haar the *SC* was around 85%, both without false positives. In nocturnal disembarkation for videos 9 and 10 the *SC* value was 86.81% on average, with 4.0% of false positives. The *SC* with HOG for nocturnal embarkation was on average 66%. With YOLOV3 it was on average 89% (video 4 and 5) both of them without false positives. For nocturnal disembarkation, YOLOV3 presented on average 91% of success rate to videos 9 and 10, both of them without false positives. It was observed that the short distance among passengers and the camera negatively affected the metric *SC*.

In the experiments the counting data was sent between the aggregator implemented in NodeMCU ESP32 with an AI Thinker card that supports GPRS technology and the counting module at the Raspberry PI. The Aggregator was implemented as an access point to allow the Counting Modules to connect to it. A COAP server was implemented in the Aggregator to receive messages regarding the count. The aggregator module communicates with an external server via GPRS. Each experiment was repeated 30 times. Data transmission among counting was deployed between aggregator and external server with MQTT protocol using QoS level 1. Different number of packets (1, 5, 10 and 15) were transmitted. Figure 6a depicts the delay between the Count and the Aggregator modules (inside the bus) over IEEE 802.11. Regarding delay, the average was 664ms to transmit one packet, 674ms (five packets), 682ms (ten packets) and 692ms (fifteen packets). It is because the Aggregator module needs to send acknowledgment messages to the counting module, generating a message queue. Regarding packet loss, when ten packages were transmitted the packet loss was 0.17% and with fifteen packages, 0.18%. The losses occurred due to interference such as, high number of people standing, other WiFi signals in the area.

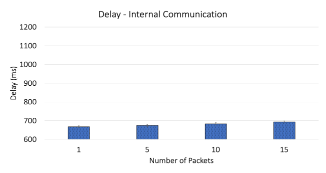


(a)

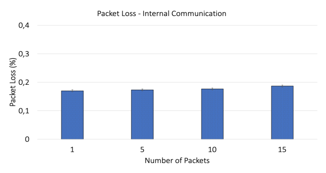


(b)

Fig. 5: Nocturnal Boarding (a), Nocturnal Landing (b): Zenith Position.

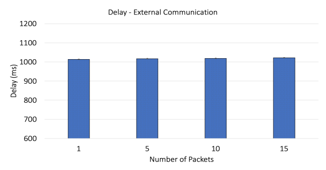


(a)

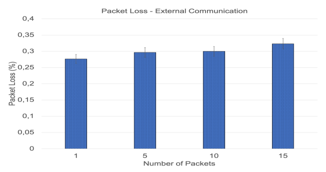


(b)

Fig. 6: Delay (a), Packet Loss (b): Counting to Aggregator Module (802.11).



(a)



(b)

Fig. 7: Delay (a), Packet Loss (b): Aggregator Module to Server (GPRS).

Figure 7a shows results of delay between the Aggregator module and an external server. The average transmission of one packet was 1013ms, 1017ms (five packets), 1019ms (ten packets) and 1022ms (fifteen packets). The average delay was greater when compared to the results of the delay shown in Figure 6a. This behavior is justified by the use of the TCP protocol and the MQTT with QoS 1. For packet loss, the transmission of fifteen packages was around 0.3%.

V. Conclusion

The Counting algorithm implements computational vision techniques to support passenger detection and counting, for later, with the aggregation algorithm, join and compute the other doors of the same bus at each stop. Results show that the Haar technique brought optimum results after training with an Adaboost classifier reaching 86%. HOG technique proved to be time-consuming for real-time processing environment. Also, the percentage of success presented by YOLOV3 was the highest, however, the time to process the counting information was very long. It was concluded that further studies can be done to improve the percentage success rate while preserving good processing time results. For example, the count can be more accurate if the cameras are equipped with a depth sensor, such as kinects.

References

[1] S. P. Mohanty, U. Choppali, and E. Kougianos, "Everything you wanted to know about smart cities: The internet of things is the backbone," *IEEE Consumer Electronics Magazine*, vol. 5, no. 3, pp. 60–70, July 2016.

[2] C. Benevolo, R. P. Dameri, and B. D'Auria, "Smart mobility in smart city: Action taxonomy," *ICT Intensity and Public Benefits: Switzerland: Springer International Publishing [doi¿ 10.1007/978-3-319-23784-8_2]*, 2016.

[3] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7252.txt

[4] G. Hillar, *MQTT Essentials - A Lightweight IoT Protocol*. Packt Publishing, 2017. [Online]. Available: https://books.google.com.br/books?id=40EwDwAAQBAJ

[5] T. Yahiaoui, L. Khoudour, and C. Meurie, "Real-time passenger counting in buses using dense stereovision," *Journal of Electronic Imaging*, vol. 19, no. 3, p. 031202, 2010.

[6] S. Mukherjee, B. Saha, I. Jamal, R. Leclerc, and N. Ray, "Anovel framework for automatic passenger counting," in *Image Processing (ICIP), 2011 18th IEEE International Conference on*. Brussels, Belgium: IEEE, 2011, pp. 2969–2972.

[7] O. Boreiko and V. Teslyuk, "Structural model of passenger counting and public transport tracking system of smart city," in *Perspective Technologies and Methods in MEMS Design (MEMSTECH), 2016 XII International Conference on*. Lviv, Ukraine: IEEE, 2016, pp. 124–126.

[8] C. O. Escolano, R. K. C. Billones, E. Sybingco, A. D. Fillone, and E. P. Dadios, "Passenger demand forecast using optical flow passenger counting system for bus dispatch scheduling," in *Region 10 Conference (TENCON), 2016 IEEE*. Singapore, Singapore: IEEE, 2016, pp. 1875–1878.

[9] M. Nitti, F. Pinna, L. Pintor, V. Pilloni, and B. Barabino, "iabacus: A wi-fi-based automatic bus passenger counting system," *Energies*, vol. 13, no. 6, 2020. [Online]. Available: https://www.mdpi.com/1996-1073/13/6/1446

[10] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. San Diego, CA, USA, USA: IEEE, 2005, pp. 886–893.

[11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. Kauai, HI, USA, USA: IEEE, 2001, pp. I–I.

[12] J. C. Rutke and J. Kniess, "Architecture with internet of things for counting passengers in urban bus," in *XLIV Latin American Computer Conference, CLEI 2018, São Paulo, Brazil, October 1-5, 2018*. IEEE, 2018, pp. 735–743. [Online]. Available: https://doi.org/10.1109/CLEI.2018.00093

[13] C. Sanders, *Practical Packet Analysis, 2nd Edition: Using Wireshark to Solve Real-world Network Problems*, ser. No Starch Press Series. No Starch Press, 2011. [Online]. Available: https://books.google.com.br/books?id=Zl6LBAAAQBAJ

[14] X. Li, L. Wang, and E. Sung, "Adaboost with svm-based component classifiers," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 5, pp. 785–795, 2008.

[15] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.