Parking Solution for [University Campus]: Car Counting with OpenCV

ABSTRACT

Parking pains ultimately cost US drivers $95.7 billion annually due to the costs of search times, overpayment, and parking fines. These issues extend to college campuses, where parking can be especially troublesome. [University Name] is no exception. New dorms opened in 2018, but parking problems persisted. Though smart parking solutions are becoming more common, the smart parking market is dominated by companies that use expensive sensors and proprietary software. The objective of this research project was to investigate computer vision as an alternative smart parking solution for [Univ.]. Computer vision is underutilized in the smart parking industry, but it can be a more cost-effective and versatile method for capturing data. For this study, a car counting system was deployed at [Univ. Parking Lot] on the [Univ.] campus to monitor the lot occupancy count (number of cars/other vehicles in the parking lot). Data was successfully collected from 9:00am to 4:00pm for 7 days. This research is a contribution to open source, and it serves as a foundation for collecting and mining transportation data at [Univ.] to improve campus life.

KEYWORDS

opencv, computer vision, applied, smart parking, vehicle, mobilenet-ssd, object detection, object tracking

ACM Reference Format:

. 2020. Parking Solution for [University Campus]: Car Counting with OpenCV. In *published location placeholder*. ACM, New York, NY, USA, 6 pages. https://doi.org/doiplaceholder

1 INTRODUCTION

Finding a parking spot can take as long as 9 minutes on average, amounting to 17 hours per person each year in the US, and the economic consequences are significant. Fuel and time are wasted, traffic congestion and emissions increase, and air quality worsens. If parking is especially limited, drivers will avoid shopping, work, and even doctor's visits. And when parking is readily available, over-payment is still an issue. In the US, drivers typically overpay by 17 minutes per trip. Parking is understandably frustrating, so many drivers will also park illegally if they believe they can get away with it, leading to more fines. The culmination of long search times, over-payment, and parking fines is a $95.7 billion annual cost for US drivers alone [1]. Though parking is relatively understudied, its issues can be attributed to growing populations, dense urban areas, and limited space. Notable parking research suggests that free parking has created a tragedy of the commons problem, and carefully-priced paid parking solutions are the answer [18–20]. However, most of the US has sufficient free parking available. While the majority of drivers conclude there is not enough available parking, there are 3.5-8 spaces per person globally, and parking space occupancy can be as low as 50% in congested cities. There is not a parking problem, but a parking information problem that can be solved with technology, particularly smart parking solutions. Drivers are very interested in integrated apps or in-car navigation systems to receive information for where to park [1, 13]. Smart parking solutions will help solve parking pains, and the new opportunities for data collection will impact transportation as a whole. The lot occupancy rates, combined with time, work schedules, and other factors can be data-mined to establish trends. Urban planners and companies can use this information for further improving their transportation and parking management systems while providing users with even more helpful data.

This parking utilization research aims to provide [Univ.] with a simple, cost-effective technological solution for improving transportation services. An open source application was developed to monitor incoming and outgoing vehicles at [Univ. Parking Lot]. The main advantage of this DIY approach to smart parking is the price of the infrastructure. Services offered by mainstream vendors, such as Park Smarter by IPS Group, are very costly.

The remainder of this research paper is organized as follows. Section 2 describes relevant background information regarding smart parking and computer vision. Section 3 provides an overview of application development and testing. Section 4 illustrates the results. Sections 5 and 6 summarize the research outcomes in the context of a broader scope and outline future works.

2 BACKGROUND

2.1 Smart Parking

Smart parking vendors integrate sophisticated guidance systems, parking management systems, sensors, dynamic pricing, parking reservation, and other technology into one convenient product [13]. Therefore, the software used in this study is not a complete replacement for the services that smart parking vendors provide. The additional services beyond sensor-based vehicle and event recognition are out of the project scope.

Sensors are the most crucial part of a smart parking system because they are required for vehicle detection. Such sensors can either be intrusive or non-intrusive. Intrusive sensors are installed into holes on the road surface and include active infrared sensors and magnetic sensors. Non-intrusive sensors, on the other hand, can be easily installed and maintained. Examples of non-intrusive

ACMSE 2020, March 26-28, 2020, Tampa, Florida

ACM ISBN isbn placeholder...$price placeholder
https://doi.org/doiplaceholder

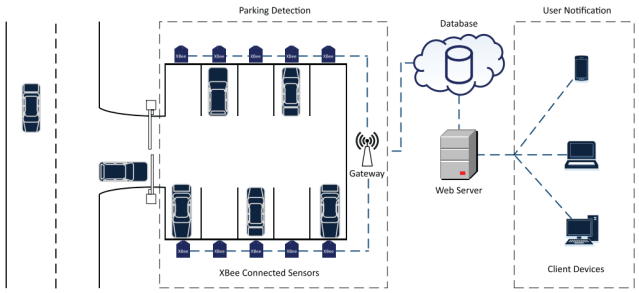

Figure 1: A simple smart parking model [3]

sensors are microwave radar, RFID, and image processing [5]. Non-intrusive sensors are often preferred due to increased accuracy and lower cost. IPS Group, the company that claims 90% of the smart parking market, uses microwave radar in their dome sensors.

The downsides of IPS and other smart parking vendors is the cost of the infrastructure. For example, a single street meter from IPS costs $900 for the housing and dome sensor. This figure does not include recurring costs and collection equipment for the entire installation. While the IPS dome sensors are the most accurate on the market, computer vision solutions for smart parking can still maintain high accuracy at a much lower cost [9, 12]. Most CCTV cameras cost less than $50, and they are able to monitor entire lots, making them potentially more cost-effective than the sensors and meters provided by mainstream vendors [21].

2.2 Image Processing and OpenCV

Digital image processing (DIP) is becoming more advanced and widespread as a result of the expanding library of available machine learning resources and techniques. Thanks to the cloud, hardware limitations can be easily circumvented, which means the average user can now deploy their own DIP applications. This "do it yourself" approach gives the end user more freedom and flexibility, allowing them to deploy and modify applications to suit their specific needs. In addition, the generated data is not tied to proprietary systems that limit one's ability to extract data and optimize the software. This is another disadvantage of mainstream smart parking solutions [2].

OpenCV (Open Source Computer Vision Library) is an example of one of the many free tools available for computer vision applications. It is a comprehensive library that contains a plethora of state-of-the-art computer vision and machine learning algorithms. Applications of these algorithms include facial recognition, object identification, motion tracking, object tracking and detection, image comparison, red-eye removal, and scenery recognition. OpenCV is cross-platform compatible and can be implemented in a variety of programming languages. Many Fortune 500 companies, such as Google, Microsoft, Sony, and Toyota, along with numerous startups employ OpenCV [11].

Image processing algorithms simplify the source image or video as much as possible to extract information. Usually, this begins with a conversion to gray scale, but color filters, gradients, or a combination of these can also be applied. Then the video frame or image is transformed further by edge detection, feature matching, or background subtraction algorithms [4].

2.3 MobileNet-SSD and dlib: Object Detection and Tracking

MobileNets simplify the information received from input images to meet the harsher resource constraints of less powerful machines such as cellular phones and Raspberry Pis. MobileNets filter and combine input channels to simplify them with depth-wise separable convolutions, a combination of depthwise and pointwise convolutions. First, depthwise convolutions filter input channels to reduce their spatial complexity. Then, pointwise convolutions take the reduced kernel and calculate a weighted sum to produce the final 1x1 kernel. This process speeds up convolutional neural net processing while maintaining a reasonable amount of accuracy. The number of computations can be up to 22x less than other models.

SSD (Single Shot Detector) is a combination of RPN (Region Proposal Network) and Fast R-CNN, a convolutional neural network used for object detection. The RPN component determines possible objects and feeds this information to Fast R-CNN (Region Proposal Convolutional Neural Network) to accomplish object detection. Even though SSD uses a fast implementation of R-CNN, it is still unsuitable for resource-constrained devices. To solve this issue, MobileNet can be used as the base layer. MobileNet-SSD, the resulting combination, takes advantage of the accuracy offered by Fast R-CNN while ensuring low-power usage [6–8, 14].

Dlib is another computer-vision library that provides machine learning algorithms with a focus on real-world applications. Dlib only requires NumPy, SciPy, and Scikit-Image. In this research, dlib was used for tracking.

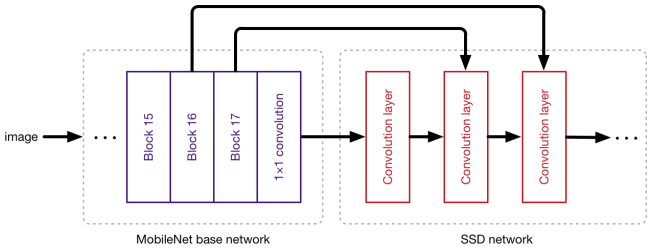

Figure 2: MobileNet extracts features and feeds the resulting high-level information to SSD. SSD uses high-level features along with low-level features to accomplish detection [7].

3 METHODOLOGY

3.1 Overview

For this study, a simple, open source client-server architecture, based on MobileNet-SSD and OpenCV, was developed and tested. The system was then deployed at [Univ. Parking Lot] on campus. A Raspberry Pi camera client was used to send incoming frames to the in-lab server. The server then applied object detection and tracking algorithms to register vehicles and events, which are either entries or exits. Based on whether cars were entering or exiting, the server updated the lot occupancy count accordingly. Data was collected from 9:00am to 4:00pm for 7 days.

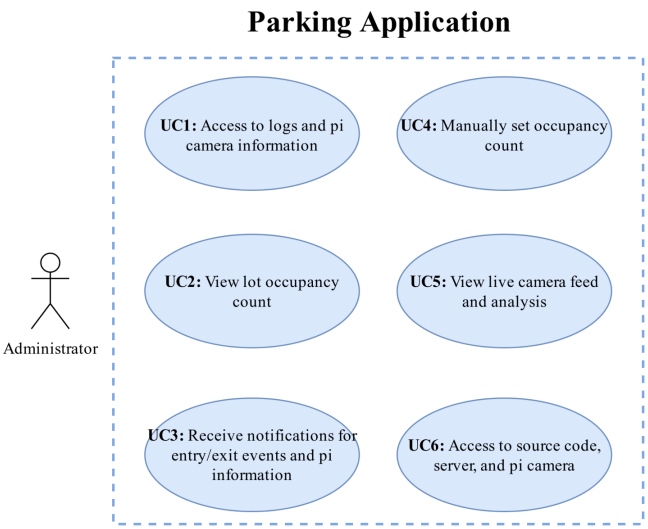

Figure 3: Use Case Diagram

The use case diagram (Figure 3) defines the relationship between the end-user (administrator) and the software. A requirements matrix was written for the project documentation to ensure traceability. The use cases are justified, because they fulfill the project's technical requirements. Primary and secondary use case scenarios were also listed in the documentation to help further define each use case's purpose.

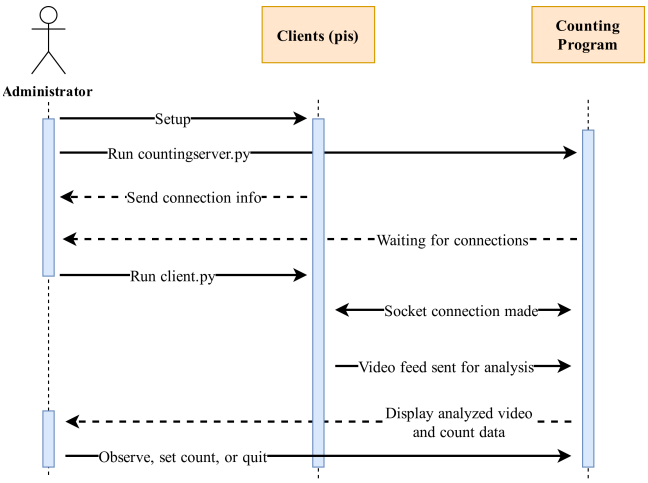

Figure 4: Sequence Diagram

The sequence diagram (Figure 4) illustrates how the software will operate. Actions and responses are represented by solid arrows and dashed arrows respectively.

The deployment diagram (Figure 5) is a high-level view of the overall architecture, and it shows the client-server setup. ZeroMQ is ideal for real-time frame transfer because of its low latency and high throughout (new frames arrive quickly). ZeroMQ was implemented

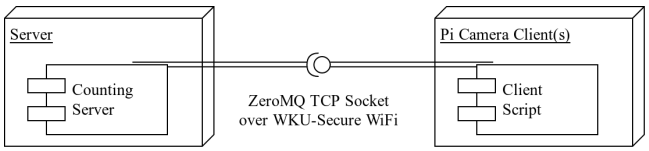

Figure 5: Deployment Diagram

through ImageZMQ, a specialized computer-vision implementation of the protocol for python [17]. The pi client provides frame data to the counting program over WiFi. The footage is analyzed server-side to determine the lot occupancy count.

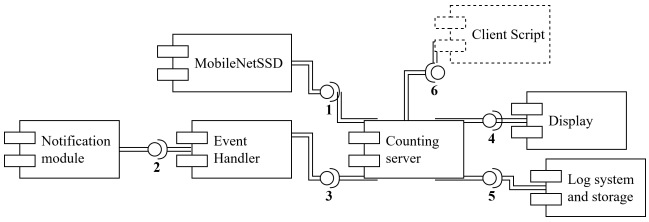

Figure 6: Component Diagram

The component diagram (Figure 6) details the relationships between different project modules. All the components are located on the server, with the exception of the client script. The connections in the diagram are labelled at each node. The nodes represent provided resources and accepted resources.

- 1. OpenCV loads the caffe and prototxt files found in the "mobilenetssd" directory to initialize the model used for object detection.
- 2. The Notification Module is a simple python script that uses Google's SMTP servers to send messages. The Event Handler module uses the notification module to send lot occupancy logs.
- 3. The Event Handler is integrated into the counting server script to manage lot occupancy data, notifications, and user interaction.
- 4. Incoming frames from the pi client are displayed along with the timestamp, client name, current vehicle count, vehicle centroids, and the current status (waiting, detecting, or tracking).
- 5. The Event Handler outputs log data that simple shell scripts organize. Written video footage is organized as well.
- 6. After establishing a TCP socket connection, the client script sends frames over ZeroMQ to the counting server for analysis.

The Event Handler is the only object-oriented module in the application. The Event Handler follows the factory pattern and the publish/subscribe pattern. The Event Handler was developed with scalability in mind. Each parking lot has its own data, so separate instances can be made. The Event Handler also imports the Notification Module in order to send periodic emails to notify administrators of occupancy updates.

3.2 Testing and Deployment

Before deployment, the server and pi client were tested in lab. Shellscripts were made to automate testing for sysbench, iperf, and pi power resource usage and temperature. Iperf was used to test the transfer rate from the pi to the server. The bandwidth ranged from 37.0Mbits/sec to 39.4Mbits/sec, which is more than enough to send frames over the network. Sysbench CPU, FileIO, and memory stress tests were run on both the server and client to ensure functionality. While running the counting program, the pi client used an upwards of 4.3 watts at 60.1°C. The in-lab test results indicated that both the server and pi client were ready for deployment.

Setup began around 8:45am each day. To start, the server-side counting program was initialized. Then, netcat scripts retrieved the pi's ip address. Netcat was needed due to recurring ip changes for the raspberry pi. The pi was remotely accessed to start the client script to send frames. The server-side counting program established a connection to the pi client to receive and analyze the incoming video feed. Setup happened twice each day, because the pi's batteries were only able to power it for a maximum of 2 hours and 45 minutes. The counting application was shutdown around 12:30pm to swap batteries and perform a reboot. Occupancy and video footage were cross-referenced at the end of the day to verify the results. The video footage was destroyed after occupancy data was analyzed to preserve the privacy of passing pedestrians.

After the pi client is correctly positioned and powered on, the remainder of the system setup requires command-line input from the administrator to specify which files to run, the directories to import modules from, and optional arguments to modify how frames are recorded and formatted.

Multiple threads exist in the application to allow for simultaneous processing. The manual count set function allows the administrator to initialize the lot occupancy count and adjust it as needed. It runs on a separate thread to prevent interference with frame processing. The notification system also runs on a separate thread to continuously publish log information.

The counting server component (Figure 6) and the client script used in this research were adapted from a combination of different programs for OpenCV real-time streaming, video storage, and pedestrian counting [10, 15–17]. Many changes were made to adapt the base programs into a vehicle counting application. Modifications include algorithmic improvements, increased parameterization, and the creation of a log management system.

3.3 Pi Camera Client

The camera was mounted to a short metal pole at the intersection between the [Univ. Parking Lot] entrance and the [Univ. Building] side road. The camera assembly is screwed to a pole mount. A waterproof electrical box contains the Raspberry Pi 3B+ and styrofoam padding to stabilize the pi camera. The pi camera was inserted into the black CanaKit pi case. The battery is attached to the bottom with velcro to allow for easy replacement. A hole was drilled into the side of the electrical box for the battery wire. Electrical tape was used to seal the battery wire hole. No water was found in the electrical box after rainfall. The pi remained in the electrical box for the month of October, and the pi withstood changes in humidity and temperature during the experiment.

Figure 7: The camera assembly was strapped to a metal pole on [Univ. Parking Lot] to monitor the entrance.

The original position for the pi camera was on a nearby light pole. The [Univ. Building] side road that merges into the [Univ. Parking Lot] entrance was in view, however. Frame cropping did not solve this issue, so the camera was relocated to a short metal pole on the sidewalk. The camera angle was then adjusted until only the [Univ. Parking Lot] entrance was in view. In the previous orientations and locations, the camera would register movement from cars that were not entering or exiting [Univ. Parking Lot]. Vehicles inside the parking lot and on nearby roads were detected, rendering the count data invalid.

3.4 Event Recognition Algorithm

```
1:  procedure (centroids[])                    ▷ Analyze vehicle positions
2:      current ← centroids[0]                        ▷ Last position (x,y)
3:      position ← current.location      ▷ Vehicle is on left or right
4:      direction ← current − centroids.mean()
5:      if direction is left & position is left then
6:          entry
7:      else if direction is right & position is right then
8:          exit
9:      end if
10: end procedure
```

Figure 8

A tracked vehicle's last recorded centroid was analyzed along with its position relative to the vertical centerline in order to determine entries or exits. The basis for this algorithm was adapted from pedestrian counting projects [10, 16].

The display alternates between "Waiting", "Detecting", and "Tracking" statuses. Every n frames, the detection algorithm runs. When

a vehicle is found, the centroids and bounding boxes are stored. Then, the counting program continuously tracks the object(s) for n-1 frames. Tracking is less computationally expensive, so it is used rather than detection for as long as possible [16].

Originally, vehicles at rest were occasionally recognized as entering or exiting. In response, a tolerance was added to the event recognition algorithm. The distance between the newest centroid and the previous has to be at least 20 pixels for the event to register. A possible shortcoming of this solution is that vehicles moving at very slow speeds will not trigger an entry or exit event due to the small distance between successive centroids. However, this was not a concern. All observed vehicles moved fast enough, even when backing out of the parking lot.

Figure 9: Registered exit event

4 RESULTS

Occupancy logs contain the vehicle count at a specific timestamp. The recorded timestamps and vehicle counts for each event were rounded to the nearest hour and subsequently averaged together for each day. The days of the week were grouped together according to the [Univ.] class schedule (MWF and TR). The data points in the graphs represent the mean of each day's average hourly vehicle counts. The mean was taken twice, because increased activity leads to more entries and exits (more data). Therefore, a single average of the hourly counts would have distorted the data to favor increased activity.

The measured car counts were compared to the actual number of vehicles in [Univ. Parking Lot] twice a day. The count was usually off by only 1-2 vehicles. The 1-2 extra or missing vehicles represents drift, defined as the deviance from the actual number of cars. For larger parking lots, this figure is not expected to change. It is assumed increased activity would lead to a proportional increase in false-positives and false-negatives, resulting in no overall change to the drift.

The [Univ.] class time frames for Monday, Wednesday, and Friday are 8:00-8:55, 9:10-10:05, 10:20-11:15, 11:30-12:25, 12:40-1:35, and 1:50-2:45. The time frames for Tuesday and Thursday are 8:00-9:20, 9:35-10:55, 11:10-12:30, 12:45-2:05, and 2:20-3:40. No apparent correlations to the [Univ.] class time frames were found for the current data set. However, activity does peak at midday for the Monday, Wednesday, Friday plot (Figure 10), which is to be expected. This trend was not observed for the Tuesday, Thursday plot (Figure 11).

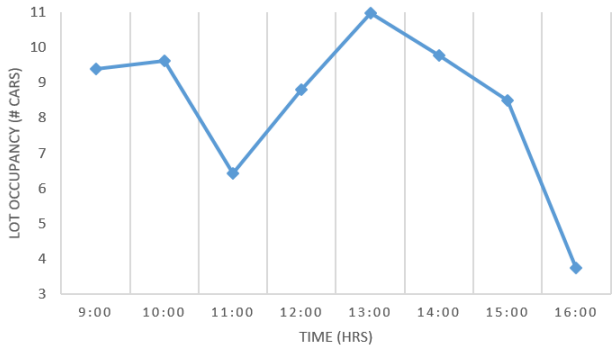

Figure 10: Average lot occupancy counts from 9am-4pm on Monday, Wednesday, and Friday.

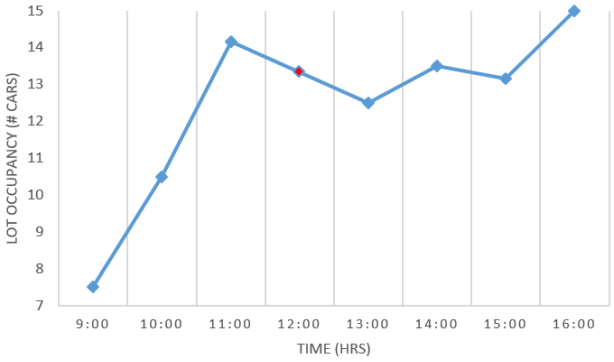

Figure 11: Average lot occupancy counts from 9am-4pm on Tuesday and Thursday. The data point for 12:00pm is red, because no data was collected for that time slot.

4.1 Limitations

4.1.1 Data Collection Procedure. The main obstacle to data collection were class schedule conflicts. Data was unavailable at 12:00pm on Tuesdays and Thursdays due to overlap with college courses, so maintenance could not be conducted.

Additionally, the inability to manually set the lot occupancy count from a mobile device allows for cars to enter and exit the parking lot during the short interval after the system is initially setup. This means the initial count may be off.

4.1.2 Detection and Tracking. Unless the footage is manually reviewed, there is no way to know whether or not every vehicle was detected. However, the registered events can be cross-referenced

to video footage at the corresponding timestamp. False positives were uncommon, but they still occurred for various reasons.

Vehicle centroids warped (changed position unexpectedly) whenever pedestrians passed in front of them. The pedestrian in question would take possession of the vehicle's centroid, resulting in incorrect event recognition. Centroids were also observed lagging behind their rightful vehicle, which led to a false negative (no event). To solve these issues, cameras could be placed directly above the parking lot entrances and exits to prevent object overlap. In addition, a more complex approach could be taken. Camera sensors could watch the entire parking lot, determining the lot occupancy count by observing spaces rather than motion at entrances and exits.

4.1.3 Scale. Though the counting application can receive multiple camera inputs and Event Handlers, the software as is cannot support multiple entrances and exits. Multiple parking lots, by extension, are also unsupported. More components would need to be created, and the counting program itself would need to be heavily modified in order to allow for this increased functionality. Additionally, new problems arise. A synchronization tool would be required for multiple entries and exits, because different objects would be modifying the same data (current lot occupancy count).

5 CONCLUSIONS

The deployment of the [Univ. Parking Lot] car counting application was successful. This research has shown the effectiveness of computer vision approaches for smart parking systems. Thanks to MobileNets, powerful computing resources are not required to setup a robust smart parking system [8]. The required software could easily run on the average office computer, requiring little to no overhead cost. Moreover, CCTV cameras are already present in many parking lots [13]. The computer vision approach for smart parking is therefore defined by the use of already available infrastructure rather than new, expensive installments.

This research adds to the growing body of works that represent the "DIY" trend of digital image processing [2]. New libraries and resources along with fewer barriers to entry have allowed for hobbyists and small teams to develop robust computer vision applications to suit their own needs.

6 FUTURE WORK

Extensions for this research include scaling to multiple entries and exits for various lot layouts and integrating the data into a web interface, such as i[Univ.]. Users could then view the real-time parking data they need. Switching from a Raspberry Pi-based camera system to a CCTV-based system would also be beneficial, because most businesses and campuses already have CCTV installations. Security systems are often contracted out, however, which is an obstacle. This is the case with [Univ.].

This research and other computer vision approaches to smart parking will be an asset to advancing the smart parking industry, making intelligent management systems more cost-effective. Computer vision-based systems also allow for more data, such as license plate numbers, to be collected. Such data can be combined with time, work schedules, and other factors to establish trends that will further improve parking management systems.

The open source angle of this research is a step in the right direction for opening up the smart parking industry. A major drawback of the current situation is incompatibility: businesses and cities cannot combine their parking data because they have different providers. These providers distribute proprietary software and technologies, which are not designed with cross-platform compatibility in mind. Instead, smaller entities and volunteers could develop their own open source management software and sensor programs. The end goal is to have independent cities and businesses, each with their own parking lots, openly contribute parking information and data, so drivers will have the best possible integrated solution that also protects their privacy. Scalability and compatibility are therefore the two most critical characteristics a future smart parking system needs to achieve in order to be successful.

REFERENCES

[1] Graham Cookson and Bob Pishue. 2017. The Impact of Parking Pain in the US, UK and Germany. *INRIX Research* July (2017). http://sevic-emobility.com/images/news/INRIX{_}2017{_}Parking{_}Pain{_}Research{_}EN-web.pdf

[2] Jose Garcia. 2018. *People Counting with OpenCV, Python & Ubidots.* ubidots.

[3] Robin Grodi, Danda B. Rawat, and Fernando Rios-Gutierrez. 2016. Smart parking: Parking occupancy monitoring and visualization system for smart cities. *Conference Proceedings - IEEE SOUTHEASTCON* 2016-July (2016), 1–5. https://doi.org/10.1109/SECON.2016.7506721

[4] Harrison. 2012. *OpenCV with Python Intro and loading Images Tutorial.* Python Programming.

[5] Khaoula Hassoune, Wafaa Dachry, Fouad Moutaouakkil, and Hicham Medromi. 2016. Smart parking systems: A survey. *SITA 2016 - 11th International Conference on Intelligent Systems: Theories and Applications* (2016). https://doi.org/10.1109/SITA.2016.7772297

[6] Matthijs Hollemans. 2017. Google's MobileNets on the iPhone. *machinethink.net* (2017).

[7] Matthijs Hollemans. 2018. MobileNet version 2. *machinethink.net* (2018).

[8] Andrew G Howard, Weijun Wang, Menglong Zhu, Tobias Weyand, Bo Chen, Marco Andreeto, Dmitry Kalenichenko, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. (2017). arXiv:arXiv:1704.04861v1

[9] IPS Group. 2016. Vehicle Detection Sensors. rev 1.003.2 (2016). http://www.ipsgroupinc.com/parking-meters/vehicle-detection.htm

[10] Caleb Ogbonnaya. 2019. Pedestrian Detection and Tracking - A People Counting Based Application Using Embedded OpenCV. *TopScholar* (2019).

[11] OpenCVTeam. 2019. About. (2019).

[12] Vijay Paidi, Hasan Fleyeh, Johan Håkansson, and Roger G. Nyberg. 2018. Smart parking sensors, technologies and applications for open parking lots: A review. *IET Intelligent Transport Systems* 12, 8 (2018), 735–741. https://doi.org/10.1049/iet-its.2017.0406

[13] Elena Polycarpou, Lambros Lambrinos, and Eftychios Protopapadakis. 2013. Smart parking solutions for urban areas. *2013 IEEE 14th International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2013* (2013). https://doi.org/10.1109/WoWMoM.2013.6583499

[14] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2016. Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks. 8828, c (2016), 1–14. https://doi.org/10.1109/TPAMI.2016.2577031

[15] Adrian Rosebrock. 2016. *Writing to video with OpenCV.* pyimagesearch.

[16] Adrian Rosebrock. 2018. *OpenCV People Counter.* pyimagesearch.

[17] Adrian Rosebrock. 2019. *Live video streaming over network with OpenCV and ImageZMQ.* pyimagesearch.

[18] Donald C. Shoup. 1997. The high cost of free parking. *Journal of Planning Education and Research* 17, 1 (1997), 3–20. https://doi.org/10.1177/0739456X9701700102

[19] Donald C. Shoup. 2006. Cruising for parking. *Transport Policy* 13, 6 (2006), 479–486. https://doi.org/10.1016/j.tranpol.2006.05.005

[20] Jos N. Van Ommeren, Derk Wentink, and Piet Rietveld. 2012. Empirical evidence on cruising for parking. *Transportation Research Part A: Policy and Practice* 46, 1 (2012), 123–130. https://doi.org/10.1016/j.tra.2011.09.011

[21] Michael Wilson. 2019. IPS Sample Quote.