ParkMaster: An in–vehicle, edge–based video analytics service for detecting open parking spaces in urban environments

Giulio Grassi
Sorbonne Universitès, UPMC, LIP6

Paramvir Bahl
Microsoft Research, Redmond

Kyle Jamieson
Princeton University; University College London

Giovanni Pau
Sorbonne Universitès, UPMC, LIP6

ABSTRACT

We present the design and implementation of ParkMaster, a system that leverages the ubiquitous smartphone to help drivers find parking spaces in the urban environment. ParkMaster estimates parking space availability using video gleaned from drivers' dash-mounted smartphones on the network's edge, uploading analytics about the street to the cloud in real time as participants drive. Novel lightweight parked-car localization algorithms enable the system to estimate each parked car's approximate location by fusing information from phone's camera, GPS, and inertial sensors, tracking and counting parked cars as they move through the driving car's camera frame of view. To visually calibrate the system, ParkMaster relies only on the size of well-known objects in the urban environment for on-the-go calibration. We implement and deploy ParkMaster on Android smartphones, uploading parking analytics to the Azure cloud. On-the-road experiments in three different environments comprising Los Angeles, Paris and an Italian village measure the end-to-end accuracy of the system's parking estimates (close to 90%) as well as the amount of cellular data usage the system requires (less than one megabyte per hour). Drill-down microbenchmarks then analyze the factors contributing to this end-to-end performance, as video resolution, vision algorithm parameters, and CPU resources.

CCS CONCEPTS

• **Information systems** -> *Mobile information processing systems*; • **Human-centered computing** -> *Ubiquitous and mobile computing systems and tools*;

SEC '17, San Jose / Silicon Valley, CA, USA
 978-1-4503-5087-7/17/10…$15.00
DOI: 10.1145/3132211.3134452

KEYWORDS

Fog computing, Edge computing, Mobile Systems, Visual Analytics

1 INTRODUCTION

Urban driving can be challenging and stressful, with the task of searching for parking spaces one of the key reasons. For example, a 2007 study in San Francisco [45] shows that in one of that city's commercial districts a driver spends on average 6.5 minutes to find a parking spot after reaching the destination, adding about 3.3 km to the trip. The same study shows that every day in one Los Angeles commercial district alone, cars searching for a parking space generate more than 3,500 vehicle-miles of travel (VMT)—more than a coast-to-coast US journey. Hence simply steering drivers to the closest available parking spot would address a significant portion of the traffic congestion and pollution present in big cities, with the added benefits for drivers of increased convenience and saved time.

While there are several systems that let drivers see parking garage availability [48], providing such information for roadside parking spots is more challenging because they are often not clearly delineated with road markings. Recent rumors [4] announced Google Maps may provide drivers with statistical information about how difficult parking is at destination area, information that however should be based on historical patterns and should have a coarse granularity (the probability of finding a spot is deemed Easy, Medium or Limited). Focusing on real–time data instead, some cities, such as San Francisco [44], are currently installing sensors on parking meters to detect cars' presence. These solutions however entail significant infrastructure deployment, with an associated cost. Other prototype systems use image recognition techniques to detect cars' presence in parking lots by using cameras placed on top of buildings or poles [19, 52, 57]. While feasible for single parking lots, these systems suffer the same infrastructure costs. [31, 35, 36, 47] instead exploit user's smartphone in order to track the owner's actions (parking the car or leaving the spot). However, those system are unaware of other

Figure 1— ParkMaster **deployed in a car's windshield.**

drivers (who may not use these systems) and thus require a high penetration rate to maintain a reasonable accuracy. [35] tries to overcome such a limitation by designing a nonuser's actions prediction model to estimates the effects of nonusers to parking availability. However, such a model focuses on large parking lots and not on the more challenging on-street parking scenario. The ParkNet System [34] takes a complementary approach, adding ultrasonic sensors on cars to flag empty roadside parking bays. While ParkNet achieves high accuracy using mobile sensors, it again requires the installation of additional hardware on cars, posing deployment and cost issues. Nonetheless, the core idea of the vehicles themselves operating as a sensor network to collect data about parking availability remains promising. Such a collection system would indeed facilitate coverage over a large geographical area, without the burden associated with adding roadside infrastructure.

There are three key technology trends also changing the problem space:

(1) Image processing has made significant accuracy advances in accuracy and speed.

(2) Mobile devices' processing capacity and camera quality have both been steadily increasing over recent years, making now feasible to run more advanced image–based machine learning algorithm on smartphones.

(3) Cloud services are being restructured to push latency-critical parts of their functionality out to the edge.

Viewed together, these three trends suggest a new design point: a system that leverages users' smartphones as sensors to capture parking information at the network's edge in real-time, reducing the cost for cities to only that required for the cloud service.

In this paper, we demonstrate that such an edge-based sensor system is feasible with nowadays hardware: we present the design and implementation of ParkMaster, a system that integrates with users' smartphones to address the dilemma of parking without the need for additional infrastructure, either in the urban environment or on the vehicle. Results from two major cities in the US and Europe (Los Angeles and Paris), and a small European village show that ParkMaster achieves an overall end-to-end accuracy close to 90% with a negligible overhead in mobile cellular data consumption.

ParkMaster uses the cameras on drivers' phones to sample the presence of cars at road-side parking spots from the driver's vehicle itself. It features two main components. Firstly, the ParkMaster *app*, which runs on the in-car edge — the driver's smartphone — performs real-time visual analytics. Secondly, the ParkMaster *cloud service* maintains a real-time database summarizing the number of available parking spaces on each road and provides client support for location services. While the user is driving, with the smartphone placed on the windshield as shown in Figure 1, ParkMaster captures video with the phone's camera and, locally processing frames in real-time, looks at the road-side parking spaces.

The most natural approach to parking-space detection may seem to be searching for empty spaces, as humans do. Hence we have investigated edge detection [13] algorithms on the smartphone to detect and measure free parking spaces themselves. However, while feasible in controlled environments when the background is uniform, this approach quickly becomes more challenging in real cities where road and background scene have an arbitrary appearance. Furthermore, there are no guarantees that an empty stretch of shoulder is a legal parking spot, and detailed maps with exact parking space coordinates (with meter–level accuracy) are unavailable in most locations. Also, smartphone absolute localization accuracy is too imprecise to reliably identify parking spots. We further contrast ParkMaster with this approach in Section 2.1.

We therefore take the approach where the smartphone detects the parked cars themselves. The smartphone then estimates the location of each detected vehicle, streaming this information to the cloud over Wi-Fi or the cellular interface. In order to detect the presence of a parked car at the roadside, ParkMaster needs to recognize and track the parked car as it moves across the frame of the driving car's camera. Existing motion tracking algorithms [17, 28, 55, 60] excel at this when the camera is still or almost still and objects are moving against background, but to the best of our knowledge, do not focus on our scenario where both background and object are moving across the frame together.

Having made the above design choice of detecting parked cars, in order to estimate free parking space, ParkMaster's cloud service relies on an additional information feed: the number of parking slots per road. This data is nowadays available in many cities, for example the cities of Seattle and Paris maintain this data online. At a high level, ParkMaster estimates the parking availability of each *road segment* (the smallest piece of street connecting two intersections) as the difference between its parking capacity and the number of parked vehicles that the cloud estimates by aggregating the parking analytic data that the smartphones upload. Finally, ParkMaster's cloud assists the smartphones in their own localization process by providing GPS data correction. Since the only hardware ParkMaster requires is users' off-the-shelf

smartphones, ParkMaster benefits from the individual deployment of each user's smartphone.

Contributions. This paper contributes the following:

(1) We demonstrate the feasibility of running image–based machine learning techniques at the edge, on today's smartphones, to capture valuable information about the surrounding environment in real time without any human intervention, when battery power is not a concern. In order to do so we tackle the available parking location problem designing, building, and deploying a complete system.

(2) We propose a novel, lightweight tracking algorithm for car detection that fuses speed estimates of the car with vision processing of the video stream to "de-duplicate" multiple detections of the same parked car in a drive, while the background scene moves with the parked car (unlike most object tracking algorithms).

(3) We design and implement a localization algorithm to estimate the location of a parked car in a single frame, without requiring stereo vision or any input from the user, instead relying on camera calibration against well-known objects in the driving environment. Our localization algorithm is lightweight and thus can efficiently run on a smartphone.

Roadmap. The rest of this paper is organized as follows. Section 2 describes the design of ParkMaster, while Section 3 summarizes its implementation. A performance evaluation containing microbenchmarks and a real-world evaluation of ParkMaster on the streets of two major cities and a village follows in Section 4. ParkMaster achieves a 90% average end-to-end accuracy, the result of meter-level video based localization of cars from the phone's camera. A sensitivity analysis of ParkMaster's design parameters explores optimal camera and video parameters to use given the smartphone's available processing power, and shows a tradeoff between false positive car detections and missed cars. Section 5 surveys related work, and we conclude in Section 6.

2 DESIGN

This section describes the ParkMaster design in detail. Starting with functionality on the edge, we first explain how ParkMaster recognizes (§2.1) and localizes (§2.2) vehicles parked on the road. Continuing with functionality located in the cloud, we then describe how ParkMaster counts parked cars and free spots on a road (§2.3). We begin with the main goals of our design, followed by a high-level design summary to place each part into context.

Design goals. ParkMaster has the following goals:

(1) Since computation at the edge is limited, we avoid overly-expensive computation on the mobile.

(2) Since real-world conditions and camera limitations may obscure line of vision or create visual ambiguities, we aim to design a system that can identify a car given a limited number of frames in which the car is visible.

(3) Operating within the constraints of the first design goal, since smartphone-cloud data constitutes a cost for the users, we design a system that leverages edge processing to limit the amount of data uploaded.

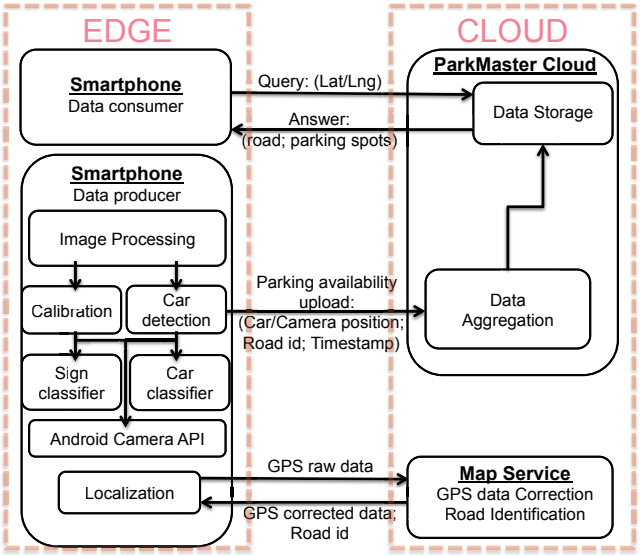


Figure 2— ParkMaster **architecture.**

Non goals. Phone power management: Here we assume that user powers the smartphone from the car's electrical system.

Design summary. We present ParkMaster's overall architecture in Figure 2. Edge-side software running on the smartphone begins with the *camera calibration* phase, processing the video stream looking for road signs. For each frame containing a road sign of known height and size, ParkMaster computes its real-world coordinates (relative to the camera) and records the relationship between these relative real-world coordinates and the sign's coordinates in the smartphone's two-dimensional camera field of view. As soon as ParkMaster collects enough samples, it runs the calibration algorithm described in Section 2.2, to derive a rotation and translation vector that captures the phone's three-dimensional orientation. Once calibration concludes, ParkMaster loads a car classifier and starts searching for cars parked on the right side of the road, parallel to the street (Section 2.1). On detecting a parked car, ParkMaster extracts its frame coordinates, computes the parked car's real-world coordinates (relative to the camera) and stores both sets of coordinates and the time of detection on the phone: we refer to these data together as the *parked car analytics* (Section 2.2). Simultaneously, ParkMaster records data from the phone's GPS, accelerometer, magnetometer and gyroscope sensors, leveraging Google Maps' "snap to road" functionality [21] to obtain even better phone localization

accuracy. Whenever the user enters a new road ParkMaster retrieves the samples collected from the previous road segment and computes the camera's position (in real-world coordinates) at each detection time. This estimate, together with the relative real-world coordinates of each parked car, yields latitude-longitude coordinates of the vehicles. If the sample is deemed valid, ParkMaster uploads the parked car analytics stored at the edge to the cloud, adding the car's real-world latitude and longitude, a road identifier, and the time each sample was detected. When the cloud receives a phone's update it runs the DBSCAN [18] clustering algorithm in order to count the number of parked cars on the road segment (using cars' coordinates to discern between different vehicles). Finally, ParkMaster subtracts this estimate from the number of parking spots to compute the number of free parking spaces.

In the future, as new data arrives about road conditions, the cloud discards old data in favor of more recent information provided by other users. When a driver queries the ParkMaster cloud looking for a parking spot, the latter provides the computed per road-segment parking availability of the area, together with its timestamp, which allows drivers to evaluate the freshness of ParkMaster's information.

In contrast with [31, 35, 36, 47], ParkMaster collects data about the surrounding parked cars — not about users actions — thus it doesn't need to run in every vehicles nor to make additional efforts in guessing non-users' behavior.

2.1 Detecting parked cars

Works like [26] have shown the feasibility of running image–based basic techniques like color filtering and edge detection on smartphones to detect particular events while driving, like traffic light status. ParkMaster takes a step further and applies machine learning techniques, the Viola-Jones feature-based cascade classifier [30, 56], to detect complex objects — cars — in the phone camera's video stream, in real time.

During *one-time offline training*, the classifier learns about the object's features through sets of positive and negative examples of the object of interest. Once the training terminates, the classifier's *online detection* (on the smartphone) analyzes the video stream looking for the features listed by the classifier in a multi-stage process. For each object that reaches the final stage, the classifier provides the *frame coordinates* of a bounding box containing the object, as shown in Figure 3.

To search for larger or smaller examples of the object of interest, at each stage the features of interest are scaled, based on a *scaling* parameter. To reduce the number of spurious object detections, a candidate part of the image is considered detected only if there are at least k adjacent detections in its immediate vicinity (*i.e.*, one to two pixels): k is referred to as the *minimum neighbors threshold*.

ParkMaster must operate in a variety of light levels, *i.e.* sunny days *v.* cloudy days; time of the day *i.e.* mornings, afternoons, evenings; and direction of the light. Ideally ParkMaster could use a specialized classifier and/or different detection settings for each situation. However, to reduce system complexity, we chose to use a single classifier trained in for a wide range of conditions. As shown in Section 4, we have obtained good results running ParkMaster in three different countries across a wide variety of ambient conditions.

Figure 3— Parked car detection: frame samples indicating bounding box of detected cars.

Discussion. In order to find empty parking spots the most natural approach is to look for empty spaces. However, such approach has two limitations: first, a map with the coordinates of each parking spot (with meter–level accuracy) is required to determine if the empty stretch of shoulder is a legal parking spot, information that is rarely available. Second, smartphone localization accuracy doesn't allow to properly identify a spot (a few meters error in the localization may translate in referring to the wrong stretch of shoulder in a hypothetical parking coordinates database). In contrast ParkMaster relies in a coarser and easier to get knowledge, the number of parking spots per road, which is already available online for instance for cities like Seattle and Paris[1], and doesn't need a precise object localization; instead it simply requires that estimated locations of neighboring cars don't completely overlap — they don't need to match any specific location in the map.

2.2 Localizing parked cars

It is likely that the same vehicle is detected multiple times in subsequent frames. Thus, to accurately count the number of parked cars, ParkMaster has to determine when a vehicle has been already detected in previous frames, and count it only once. Analyzing subsequent frames, for instance computing pixel–based image difference, to track a vehicle, is highly CPU intensive and doesn't easily handle differences among subsequent detections of the same car (while the driver is approaching the parked vehicle, the view of the car and the background changes). Thus, with its primary design goals in mind, ParkMaster aims to estimate each detected cars' latitude-longitude position in the real world, and then decide whether two detected cars are in fact the same car based on their computed coordinates.

As described in the previous section, the detection algorithm only provides the position of the parked car in the coordinates of the camera's frame. In the remainder, we refer

[1] See web6.seattle.gov/sdot/seattleparkingmap/ or opendata.paris.fr.

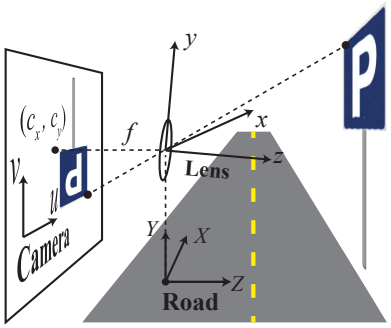


Figure 4— Coordinate systems used in ParkMaster's design: the calibration process maps objects in the camera coordinate system to the real-world road coordinate system through the lens coordinate system.

to a car's position in the camera's two-dimensional coordinate system (u, v) with *camera* coordinates (u_0, v_0). In order to compute the corresponding latitude-longitude in the real world ParkMaster must map from camera coordinates to a three-dimensional *road* coordinate system (X, Y, Z), whose X-Z plane is parallel with the road and whose origin is located on the road under the camera as shown in Figure 4. This is the classic *camera calibration* problem, which we now describe.

2.2.1 Camera calibration. Camera calibration uses the classical *pinhole camera model*, which treats the camera's aperture as a point in space. Although this model does not account for various factors such as lens distortion, coordinate discretization, and blurring, it does provide a good first-order approximation to the camera's projection. Camera calibration amounts to estimate two types of parameters. The first type, *intrinsic camera parameters*, characterizes unchanging optical properties of the camera: *focal length* f, the distance between the pinhole and the film, and offsets of the axes from the origin in the film c_x and c_y. The second type, *extrinsic parameters*, describes camera's location in the world via a translation vector $\mathbf{t}$ and the direction in which the camera points via a *rotation matrix* $\mathbf{R}$. In contrast with the intrinsic parameters, the extrinsic parameters $\mathbf{R}$ and $\mathbf{t}$ change as the camera points in different directions.

In order to separate the effects of the intrinsic and extrinsic parameters, we introduce a new coordinate system (x, y, z), dubbed *lens* coordinate system, whose z-axis is aligned with the camera lens. Now the mapping from lens to camera coordinate systems captures the intrinsic camera properties, while the mapping from road to lens coordinate systems captures all extrinsic camera properties (camera tilt, rotation, and elevation) except the GPS coordinates of the camera: these we process as part of smartphone localization (§2.2.3).

Intrinsic parameter calibration. To find the focal length f and image center (c_x, c_y) parameters ParkMaster simply queries the Android camera API. Then, it can translate between the lens and camera coordinate systems with the following relationships of the pinhole camera model:

$$x = (u - c_x) \cdot z/f \qquad \text{and} \qquad y = \left(v - c_y\right) \cdot z/f. \quad (1)$$

Extrinsic parameter calibration. To find the rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$, ParkMaster calibrates the camera while driving—without user input—using objects of well-known sizes that can be easily found on the road. In both the United States and Europe, road signs have a consistent height $h_{\text{sign}}^{\text{road}}$ (measured in road coordinates) and elevation from the road $Y_{\text{sign}}^{\text{road}}$ mandated by the highway code.

ParkMaster uses a separate cascade classifier trained on a sign of interest to search for road signs during a "start-up phase". Every time the sign classifier detects a sign, it returns the camera coordinates of the detected sign (u_0, v_0), as well as the height of the detected sign in camera coordinates $h_{\text{sign}}^{\text{camera}}$. These, along with the intrinsic camera parameters described above, allow us to compute the camera's range to the sign Z_0 as a function of the sign's height in camera coordinates:

$$Z_0 = h_{\text{sign}}^{\text{road}} \cdot f / h_{\text{sign}}^{\text{camera}} \quad (2)$$

To avoid using the results of calibration before the calibration itself is over, we approximate $X_0 \approx x_0$, and take $Y_0 = Y_{\text{sign}}^{\text{road}}$. This gives us a pair of (camera, road) coordinates for the single sign detection. After L detections, the list

$$\left[\left((u_0, v_0), (X_0, Y_0, Z_0)\right), \ldots, \left((u_{L-1}, v_{L-1}), (X_{L-1}, Y_{L-1}, Z_{L-1})\right)\right] \quad (3)$$

of (camera, road) coordinate pairs can be passed to OpenCV [39], a computer vision library, which calculates the extrinsic parameters $\mathbf{R}$ and $\mathbf{t}$ based on a global Levenberg-Marquardt optimization algorithm that minimizes projection error [11, 62].

Notice that in order to estimate Z_0 and X_0, we have assumed that the X and Y axes of the road and the lens coordinate systems are aligned. Not having this requirements satisfied, which is most likely to happen in a realistic scenario (as drawn in Figure 4), introduces error in the calibration. The misalignment is unlikely to be large in realistic driving settings as the driver wants to see the phone's display, and thus can be tolerate. In order to verify this empirically, we intentionally placed the smartphone not always perfectly aligned with the road axis in our experimental evaluation (§4.1).

Once $\mathbf{R}$ and $\mathbf{t}$ are computed, they are valid till the smartphone remains in its original position. Currently ParkMaster assumes the driver won't remove the phone from its holder while driving. However, if this happens, it is straightforward to pause the car detection, reset $\mathbf{R}$ and $\mathbf{t}$ and start again the calibration phase once the phone is back in place (this event could be triggered manually by the driver or automatically by the phone's sensors). Minor changes in the phone's position, due for instance to car vibration, have instead a negligible and usually temporary effect on $\mathbf{R}$ and $\mathbf{t}$ (the phone holder

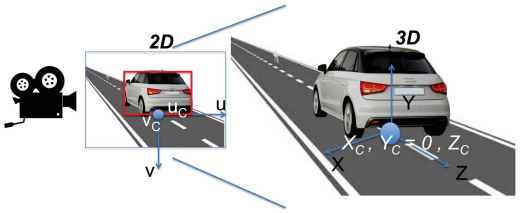


Figure 5— Extracting the lower-middle point of a detected car's bounding box in the camera coordinate system for processing in the road coordinate system, to yield a final GPS location estimate of the parked car.

stabilizes the device) and thus ParkMaster doesn't try to compensate for such events. In the experimental evaluation (§4.1) we used a standard phone holder without any additional actions to improve phone's stability, thus the results reported include the effects of vibrations, potholes ...

2.2.2 Car localization. After calibration, ParkMaster is ready to localize parked cars. However, camera parameters do not suffice to find a unique position in road coordinates given a set of camera coordinates, since a coordinate in the camera frame corresponds to multiple points in the road frame. In order to find a unique road coordinate, we exploit a natural constraint of our scenario: cars elevation from the ground is always zero, *i.e.*, $Y = 0$. Thus, supposing car detection has just detected a parked car at camera coordinates (u_c, v_c), our task is to determine X_c and Z_c, the two dimensional location of the car in the road coordinate system. To this end, consider the relationship between the road and lens coordinate systems:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathbf{R}^{-1} \cdot \left(\begin{bmatrix} x \\ y \\ z \end{bmatrix} - \mathbf{t} \right). \tag{4}$$

With Equations in 1 we can compute (x_c, y_c), the (x, y) position of the car in the lens coordinate system. Substituting in $(x \leftarrow x_c, y \leftarrow y_c, Y \leftarrow 0)$ and the calibrated values of $\mathbf{R}$ and $\mathbf{t}$ into Equation 4 yields a linear system of three equations in three unknowns (X, Z, z), which has a unique solution (X_c, Z_c) for the car's road-coordinate location.

2.2.3 Smartphone localization. Now that we have a mean of mapping car locations between camera and road coordinate systems, for each frame where a car is detected, ParkMaster extracts the lower-middle point of the bounding box (Figure 5) and computes its road coordinates. This measure, together with the coordinates of the smartphone, allows ParkMaster to estimate the absolute position of the detected vehicle and compare it with the coordinates of subsequently-detected cars.

Nowadays, every smartphone is equipped with a GPS sensor. To improve its accuracy, ParkMaster periodically sends raw GPS data to Google Maps ("snap-to-road") API service, which supports filtering and correction of the GPS traces: it removes or corrects points out of the road, interpolates coordinates based on street layout and identifies the road segment on which the user is driving. This information flows into the next stage of ParkMaster's processing, counting parked cars.

2.3 Counting parked cars

Now that it has an approximate GPS location for each parked car detection, ParkMaster relies on car's approximated coordinates to track its location between successive frames and estimate the number of parked cars. Since the localization process is prone to error, we can't do a mere coordinate comparison, because (1) the detection algorithm doesn't always pick the same point of a car; (2) the localization process is affected by inaccuracy, due to calibration imperfection and camera distortion; and (3) the smartphone location given by GPS and snap-to-road is not always accurate.

Most likely, whenever ParkMaster detects a car multiple times, it gets a set of coordinates close to each other (a couple of meters). Thus, in order to distinguish among different cars, ParkMaster runs a Density Based Spatial Clustering of Applications with Noise (DBSCAN[18]) algorithm on all the coordinates collected on a segment of road, clustering nearby points into estimates of a single car's location. Knowing the maximum number of cars that a given road can accommodate and the number of estimated cars on that road, ParkMaster estimates the parking availability.

Preliminary studies have been done in order to choose the proper clustering algorithm: among the approaches which don't require prior knowledge of the total number of clusters, we evaluated Affinity Propagation [20], Mean Shift [16] and DBSCAN. In the counting cars problem these techniques seem equivalent, with DBSCAN that slightly outperforms the others in CPU time. It must be noticed, however, that due to the small number of samples (few points per car), in most of the cases the difference in performance is negligible.

Discussion. The estimate of the maximum number of parking spaces a road can host may affect the accuracy of ParkMaster. Indeed, while sometimes road-side parking is marked with bays, in others they are left unmarked, making the maximum number of parked vehicles on the road merely an estimate. At the moment ParkMaster does not take any additional action to cope with the uncertainty of unslotted areas, but we are confident that with more sophisticated techniques (*i.e.*, measuring space between parked cars) we will be able to improve accuracy in free parking scenarios.

2.3.1 Heuristics. In order to increase accuracy ParkMaster applies the following heuristics during the clustering phase:

Sample distance. Increasing the camera-vehicle distance generally increases the error of the parked car's localization, which may lead to unreliable estimation (§4.4). Thus ParkMaster discards samples estimated to lie further than a certain

threshold distance, determined empirically in §4.1.1.

Single-element cluster. Today's smartphones can't process every video frames in real-time (§4.1.3). Furthermore, cars are not always detected, even when clearly visible. As a result, some vehicles are detected only once. To decrease the number of misses, ParkMaster conservatively considers elements that don't belong to any DBSCAN cluster as individual parked vehicles. We note here that while this approach will rarely result in spurious parked vehicle detections, our experimental evaluation shows that it has an overall benefit to accuracy.

Cluster size. Consecutive cars may be merged into the same cluster if the points in the overall dataset are dispersed and the clustering process fails to discriminate between them. In this case ParkMaster relies on the "size" of the cluster to evaluate the number of vehicles. If the maximum distance $d_{\max}$ among two points in the same cluster is bigger than a certain threshold *maxClusterSize*, ParkMaster splits the cluster into n smaller clusters, where $n = d_{\max}/maxClusterSize$.

Driving cars. Even though the detection algorithm is trained with pictures of vehicles capturing the back and part of the side of a car, it may happen that the classifier detects vehicles that are driving in the opposite direction (only the front is visible) or that are driving in front of the camera (only the back is visible). To mitigate these cases, the classifier focuses only on the lower-right part of the video, where cars parked on the right side of the road usually appear. Reducing the area of interest also allows the classifier to process only a portion of the frame, which reduces the per-frame processing time.

A special case is represented by multi-lane roads, where vehicles on the right may be moving cars. In this case, car detection must be enabled only when the user is driving on the rightmost lane. While at the moment ParkMaster assumes the user is always doing so, we believe vision-based [40] or sensors-based [6, 7, 10, 15, 43] solutions for lane detection can be easily integrated with ParkMaster to trigger the detection only when the user is driving on the right lane.

Samples while turning. A vehicle parked on the left side of the road may fell in the lower-right part of the video, resulting in false detections (Figure 6). Since GPS sometimes fails to discern these cases, ParkMaster also samples phone's magnetometer, accelerometer and gyroscope, and, when approaching an intersection, discards the sample if the user is turning *i.e.* the current orientation of the phone differs from the average orientation the phone had on that road.

3 IMPLEMENTATION

Edge. We have implemented ParkMaster on three mid-range to high-end Android phones: a Samsung Galaxy S4 GT-19505 running Android 4.4.2, a Samsung Galaxy S6 edge+ running Android 5.1.1, and an LG Nexus 5 with Android 6.0.1. The

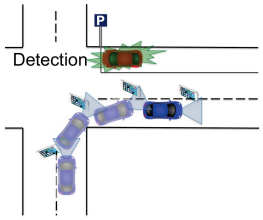


Figure 6— Car parked on the opposite side of the street.

Galaxy S4 and the Nexus 5 are equipped with a Quad Core CPU and 2 Gbytes RAM, while the Galaxy S6 is equipped with two Quad Core CPUs and 4 Gbytes RAM. All smartphones are equipped with at least an eight megapixel main camera as per the "full" specifications [1–3].

For cascade classifier-based car detection on the phone, we use OpenCV 2.4, an open-source library for image processing available for Android. The classifier (made available online [2]) has been trained with 1,202 positive samples and 719 negative samples, resulting in 20 stages. Currently, ParkMaster focuses on parallel parking only, thus we train the classifier using pictures capturing the back and the side of the vehicle as positive examples.[3]

Cloud. We have implemented ParkMaster cloud services in Azure, using Azure Mobile Service for cloud backend functionality and Azure mobile app client and Node.js SDK for authentication and interaction between clients and the in-cloud database (MongoDB) used for data collection. For GPS data correction and road identification instead the Google Snap Road API has been used. The API is available on Android as part of the Java Client library for Google Maps.

4 EVALUATION

In this section we present a comprehensive performance evaluation of ParkMaster. We begin with single-driver, real-world experiments that exercise ParkMaster's entire processing pipeline in "on the road" scenarios in metropolitan and rural environment (§4.1).We then discuss data coverage and freshness (§4.2). Finally, we drill down into ParkMaster's design, explaining how we have tuned parameters in the car detection algorithm (§4.3) and car localization algorithm (§4.4).

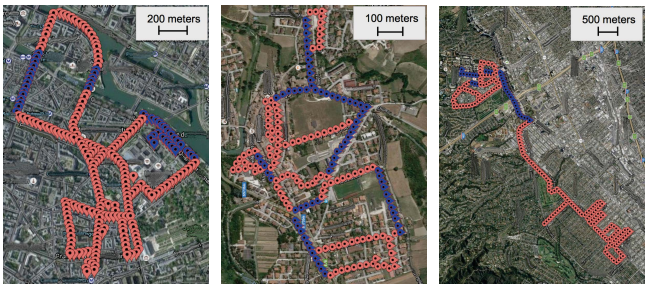
4.1 Road-based experiments

We experiment in both metropolitan (Los Angeles and Paris) and rural environments (Sant'Angelo in Vado, a small village in Italy). In Paris cars are typically parked 30–40 cm. from each other, while in Los Angeles they are usually separated by larger space and in the European village they are typically spaced by a half-meter or more, and sometimes isolated (*i.e.*,

[2] Classifier and training set available at www.github.com/grassig/parkmaster.
[3] It is possible to build a classifier for each type of parking (*i.e.*, angled or head-in parking) and then, knowing the type of parking spot on a given road, use the appropriate classifier.

Place	Distance	Unique dist.	Slots	Parked cars
EU-village	17.9 km.	3.3 km.	1,381	710 cars
EU-city	38.4	7.9	3,527	2,892
USA-city	41.0	19.3	3,268	2,294
Totals	97.3	30.5	8,176	5,896

Table 1— Experiments on the road – covered distance (only roads with at least one parking spot have been accounted).



(a)— EU-city. Paris (France). **(b)— EU-village. Sant'Angelo In Vado (Italy).** **(c)— US-city. Los Angeles (USA).**

Figure 7— Experiments on the road. Red points indicate streets with parking spaces, blue points indicate roads with no legal parking.

a few solitary parking spots on a long road). In addition, we experiment in different weather conditions: our tests span a period of ten days between December 2015 and March 2016, between 11 a.m. and 6 p.m. Typical weather includes a clear sky, dawn, partly cloudy conditions, and cloudy with light rain. In total, as Table 1 shows, we drove for 97 km. on roads that have at least one parking spot, covering a 30.5 km. path. We report a total of 5,896 parked cars and 2,280 available parking spaces[4]. Figure 7(a) shows these roads in a map view all three environments.

We experiment using the three phones described in Section 3 and drive three different cars (with different dashboard heights from the ground). In Paris we used the Galaxy S4 and S6 respectively for 41% and 59% of the time, while we used the Nexus 5 for all in-village experiments and we equally used the Galaxy S6 and Nexus 5 in Los Angeles.

On each run, we place the smartphone as shown in Figure 1, slightly changing its position (i.e. few tens of cm) and orientation each time (i.e. few degrees). We run the calibration phase on a traffic sign, and then begin driving (on the rightmost lane), respecting local speed limits: 30 kph (Paris), 30 kph and 50 kph (Sant'Angelo in Vado), 40 kph and 48 kph (Los Angeles). On average, during each run we drive for 30 minutes. GPS sampling frequency is one sample per second.

[4]The amount of parking slots each road–segment can host has been computed manually, by on-the-field observations.

		Ground truth: **Car**	Ground truth: **No car**
ParkMaster	**Car**	Correct detection (TPR)	Spurious detection (FPR)
	No car	Missed car (FNR)	Correct non-detection (TNR)

Table 2— Confusion matrix for accuracy evaluation.

4.1.1 Car detection accuracy. Before looking at ParkMaster's entire pipeline, we begin with the detection phase in isolation, analyzing the effects of the heuristics described in Section 2.3.1. We empirically set the threshold of the above heuristics after preliminary experiments: we set the maximum sample distance to 7 m. (ParkMaster discards samples at larger distances) and the maximum cluster size to 6 m. (ParkMaster splits larger clusters into smaller groups). These values maximize detection accuracy which, however, was largely insensitive to their exact settings, varying less than 10% with a ± 1 m. difference.

Accuracy metrics. In order to ascertain ground truth (the number of parked cars and empty parking spots) an additional camera records each experiment and we visually tally these quantities after the drive. We evaluate in terms of the *true positive rate* (TPR) and *false positive rate* (FPR), as shown in Table 2. TPR is the ratio of the number of cars that ParkMaster detects in at least one frame to the ground-truth number of parked cars, while FPR is the ratio of the number of spuriously-detected cars ParkMaster reports to the sum of spurious detection and correct empty parking space detection. If the same car is detected multiple times, we count it once. Note that whenever ParkMaster detects a car which is not parked or which is parked on the left side of the road, we count the sample as a spurious detection (false positive).

Results. Figure 8(a) shows the results. The heuristics of §2.3.1 slightly decrease the true positive rate. Indeed they sometimes fail discarding samples that are correct. In contrast they are strongly beneficial for the false positive rate. Manual data inspection shows that the heuristics are able to discard a good fraction of (1) running cars (based on road coordinates); (2) cars parked on the other side of the road when the user is turning (based on GPS and azimuth); (3) cars crossing intersections (based on GPS) and (4) non-vehicle object misclassified as cars that don't lay on the ground (based on their higher estimated distance to the camera).

4.1.2 End-to-end accuracy. We now measure end-to-end accuracy, from detection on the smartphone to clustering and counting in the cloud.

Metrics. In order to evaluate end-to-end, we extend the above concepts in order to take into account the counting process. We consider true positive when a car parked on the right side of the road appears among the vehicles ParkMaster counted,

while we consider as false positive all the non-cars, vehicles parked on the left side of the road and driving cars that ParkMaster counts as parked cars. Furthermore, whenever ParkMaster counts the same parked vehicle multiple times, we consider the first sample as true positive and all the others as false positives. In addition, we introduce the *accuracy with compensation* metric: the final outcome of ParkMaster is an estimation of the number of parked cars, which includes both correct samples and counting error—spurious detections "compensate" misses in the count for a specific road. Thus, for each road i, we compute $error_i$ as the difference among the number of parked cars (ground-truth) and the number of vehicles ParkMaster considers as parked. We define the *accuracy with compensation* as:

$$accuracy = 1 - \frac{\sum_{i=0}^{\#roads} |error_i|}{\#parked\,cars}. \quad (5)$$

During a single experiment, it may happen that we drive several times on the same road; nevertheless, we consider each pass independent from the others—the error from one pass is not mitigated by later pass on the same road.

Finally, we evaluate ParkMaster accuracy when determining if there is space to park on a road in terms of *Positive* and *Negative Predictive Value*. Defining positive samples as occurrences of roads with at least one available parking spot and negative samples as occurrences of roads without free space, the *Positive Predictive Value* is the ratio of correct positive estimation (ParkMaster correctly estimates there is enough space) to the number of positive samples. Similarly, *Negative Predictive Value* is the ratio of correct negative estimations (ParkMaster correctly estimates the road is full) to the total number of negative samples.

Results. Figure 8(b) shows true and false positive rates and accuracy with compensation. In general, in rural environments ParkMaster shows better performance compared to urban scenarios. Indeed, in the first we usually have cars parked at larger distance, which overcomes eventual inaccuracies in the car localization estimates. In contrast, in Paris spaces among vehicles are limited and the counting process can tolerate a smaller error in the localization. Intermediate performance characterizes the experiments in USA, which indeed presents a higher density of cars than rural environments but a less chaotic parking displacement than the European city. Nevertheless, with its almost free of costs approach, ParkMaster always shows a satisfactory close to 90% accuracy (notice that the city of San Francisco considers their sensors-on-parking-meter deployment effective and starts paying for the service when accuracy is higher than 70% [5]).

Figure 8(c) shows Positive and Negative Prediction values. Similarly to Figure 8(b), rural environments show the highest accuracy. Anyhow in most of the cases (from 87% to 98%) ParkMaster successfully classifies roads with empty parking spots. Lower accuracy is reported for negative prediction.

	Classifier	**B. B.**	**GPS**	**Loc.**	**F. P.**
EU-city	24.3%	29.2%	26.3%	8.0%	12.2%
EU-village	29.6	29.8	14.4	7.0	19.2
US-city	24.5	27.8	25.1	17.1	5.6

Table 3— On-road experiments: Error analysis.

Indeed when a street is full or almost full, a single error may lead to a misclassification of the road status.

Figure 9 shows the effect of the error compensation on the accuracy of car counts. In particular, Figure 9(a) shows the error per road while Figure 9(b) shows the same error in percentage points (on the number of parking spots on the road). In both cases, the error is reported as absolute value and with a sign (a negative error means missing cars, while positive values mean ParkMaster overestimates the number of parked cars).

Error analysis. Inaccuracies can be caused by several factors. In particular, we define the following type of errors:

(1) **Classifier:** the classifier detects something that does not correspond to a car,

(2) **B. B.:** Inaccurate bounding box—the classifier captures only the upper part of the vehicle,

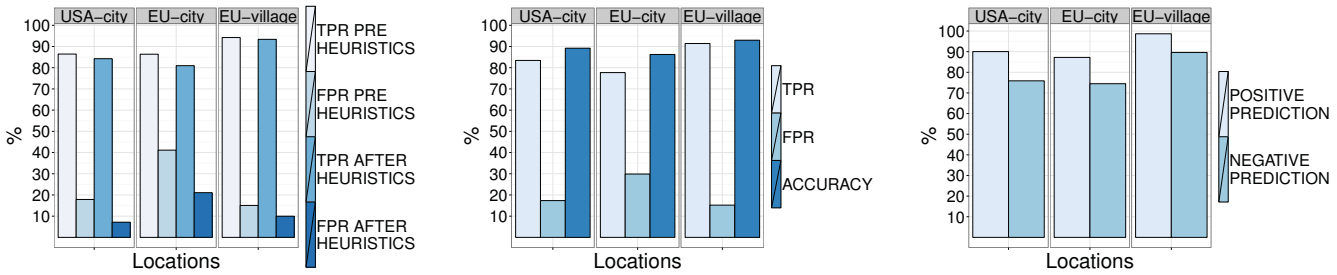
(3) **GPS:** ParkMaster fails to properly count cars due to GPS inaccuracies *i.e.* phone's GPS doesn't report any movement while car is moving (or the opposite), or GPS reports large distance among two consecutive samples (5-10 meters) while it's clear from the recorded video that user's movement is considerably smaller (1 meter or less),

(4) **Loc.:** ParkMaster miscounts vehicles because the clustering process is not able to associate samples with the corresponding car *i.e.* a car has been counted more than once or multiple cars are merged into the same cluster,

(5) **F. P.:** ParkMaster has failed to discard driving vehicles or cars parked on the left side of the road (false positive).

Table 3 shows the frequency of those errors during the experiments on–the–road. With the help of logs and videos we manually evaluate each erroneous sample and pick the best fitting error category. Table 3 confirms the remarkable impact that inaccuracy of bounding box and phone's GPS have on the car counting process. In particular, between the three scenarios, it shows a major impact of GPS error in the two cities. The difference is mainly due to higher buildings elevation (especially in the European case) and a more frequent *stop-and-go* car mobility caused by traffic lights and stops which characterize urban scenarios [14].

It must be noticed that, while the locations have different characteristics, all the parameters used in the aforementioned heuristics, in the clustering process etc. are constant. Adapting some of those parameters based on the location (*e.g.* based on the typical distance among parking spots) may reduce the error and thus further increase the system accuracy.

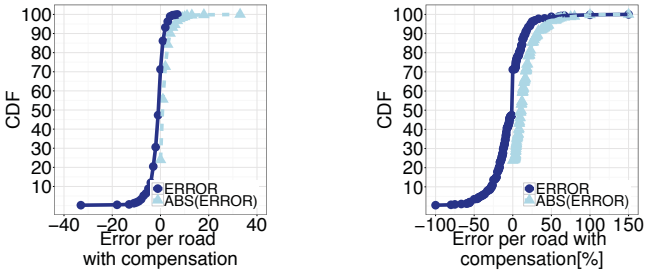


(a)— Detection on the road: True and False Positive rates before and after applying the §2.3.1 detection heuristics.

(b)— ParkMaster end-to-end accuracy in real-world driving experiments on the road.

(c)— Can we park there? Positive (there is at least one spot) and negative (road is full) prediction accuracy.

Figure 8— On the road experiments



(a) **(b)**

Figure 9— ParkMaster end-to-end counting accuracy in real-world driving experiments on the road.

4.1.3 Processing rate. On todays phones ParkMaster doesn't keep up with video speed but drops frames: on average during the experiments Samsung Galaxy S4, S6 and Nexus 5 processed respectively 5.75, 10,3 and 6.75 frames per second. Despite the large gap between the two Galaxy phones, if we compare ParkMaster performance with the two phones in similar conditions (we run few experiments with both phones at the same time), the gap reduces: in those experiments the TPRs are 83.4% in the Galaxy S6 and 76.8% in the S4, while the FPRs are 34.4% and 27.8% respectively. Drilling down the processing costs of ParkMaster, we identified in the car detection phase the main burden. Indeed, having only this phase running, the number of frames per seconds processed by the phones doesn't change: the Galaxy S4 and S6 respectively processed on average 5.78 (against 5.75 with the entire pipeline) and 10.44 fps (against 10.3).

4.1.4 Data usage. By design, ParkMaster concentrates most of the computation at the edge and relies on the cloud only for data aggregation. The amount of information exchanged by the two parts is therefore negligible. During all the on-road experiments (about 8 hours driving), the phones exchanged 6.1 megabytes[5] with cloud and Google's snap-to-road service, divided as follows: (1) 67 bytes for each sample uploaded by the phone; (2) 16 bytes for each GPS coordinate sent to Google's snap-to-road service; (3) 47 bytes for each GPS coordinate corrected by Google's snap-to-road Service.

While ParkMaster required the upload of only 6.1 megabytes of data to the cloud, we estimate that in a fully cloud-based approach that requires the entire video to be uploaded for processing, the same 8 hours drive would generate approximately 4.8 gigabytes of video (with a 720x480 resolution and same phones). This estimate demonstrates that, even if the cloud has "infinite" processing resources which may increase the accuracy of the system (Section 4.3 evaluates the effects of processing limitations on the car detection phase), the amount of data that needs to be transfered over the network is too high to be sustainable. A study of the benefits of a mixed approach, where video is sometimes processed at the edge and sometimes uploaded to the cloud, is left for future work.

4.1.5 Fusion of parked cars analytics. One of ParkMaster's strength resides in being able to run in COTS smartphones, which potentially enables every driver to participate in data collection. As a consequence, the cloud may receives information about the same road from multiple drivers. While an extended study of multi–user data fusion is out of the scope of this paper, we evaluate through preliminary experiments how an extended version of ParkMaster may reduce the error when users sample the same road in a short period of time (ParkMaster's outcome is likely to vary each time because of different user's speed, camera's properties, phone's characteristics ...). In particular, we look into the single-element cluster case (object detected only once), which constitutes a considerable portion of false positives.

In this extended version of ParkMaster, the cloud, after receiving new data, instead of overriding older samples, assigns weights to each parked car detection, indicating its confidence. Whenever data from different users matches the same spot, samples are merged and their resulting weight is increased. In order to take into consideration time, the cloud periodically decreases these weights. Whenever a single-element cluster is

[5]The negligible HTTP/TCP overhead has not been accounted.

uploaded, the cloud accepts it only if other users have recently reported a car at the same location — only if the resulting weight is higher than a certain threshold. To cope with GPS inaccuracy, the cloud may adjust traces of different users in order to match the position of the parked cars analytics.

In order to give a preliminary assessment of this approach, we place two smartphones (Samsung Galaxy S4 and S6) in the same car and we repetitively drive along the same path (950 m. path for a total of 11.4 km. in the Paris) to emulate multiple-users' activities in a short period of time.

Although such a data fusion approach is quite rudimental, 35% of the false positives are identified and removed, with a small price in terms of true positive rate reduction (2.5% of true positives are erroneously classified as not-parked-car and removed), which would further increase the accuracy of ParkMaster reported in Figure 8 (which have been obtained without data fusion). We leave for future work a more extensive study of data fusion *e.g.* external information like traffic reports (*i.e.* Waze), or historical parking data, may be used to evaluate parked-cars-analytics validity over time (once we detect a parked car, how long will it remain parked?).

4.2 Data pertinence to the parking search

As for any crowd-based system, the number of users collecting data is a crucial factor for the success of ParkMaster. Data coverage and freshness are indeed vitals to make ParkMaster information of any pertinence to the drivers looking for parking and strongly depend on the number of data collectors. In order to evaluate this, we utilizes results reported by [34]. Such a work, as previously mentioned, addresses the parking problem by installing additional hardware on cars. While the mean is different, the data collection model is similar: running cars collect data about parking availability in the area.

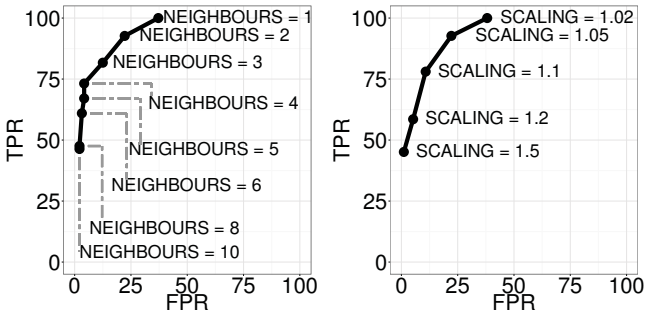
[34] analyzes taxis' traces collected over a month in the city of San Francisco [50] and estimates that with the only 536 cabs reported in the traces, in the downtown area of San Francisco, 80% of the road (dubbed cells in [34]) is on average visited with an inter-visit interval of under 10 minutes. Such a result shows that with a small subset of vehicles collecting data a significant data freshness can be achieved.

Moreover, it provides insights on how ParkMaster could be initially deployed: the system could run on taxis or other public vehicles, which has been shown to guarantee adequate data freshness, and then, given the low cost for the users (in contrast with [34]), expand to other drivers, for instance with the incentive of more parking queries for non-free-loaders.

4.3 Tuning car detection

We now drill down into ParkMaster's design, explaining how we have tuned parameters in the car detection algorithm.

We record a five-minute video using a phone while driving. Afterwards, we run the detection algorithm on a server, at first processing every frame of the video (30 fps), and then artificially skipping frames to emulate the edge limitations.



(a)— *neighbors* threshold (*scale* = 1.05). **(b)— *scale* parameter (*neighbors* = 2).**

Figure 10— Tuning parameters of a computationally-unconstrained car classifier (1920 × 1080 resolution).

Resolution	TPR	FPR
1920 × 1080	92.7%	22%
1280 × 720	91.5	21
720 × 480	91.4	15
640 × 480	86.5	13
320 × 240	76.8	4.2

Table 4— Car detection with no computational limits: Impact of changing video resolution with Viola-Jones classifier parameters *scale* = 1.05 and *neighbors* = 2.

4.3.1 Tuning video parameters for accuracy. We first measure car-detection true and false positive rates at 30 fps while varying input resolution and the classifier parameters.

Viola-Jones classifier parameters. Figure 10(a) shows the impact of varying the *neighbors* threshold with a 1920 × 1080 resolution video and a *scale* = 1.05 (*i.e.*, scaling in steps of 5%) when the Jones-Viola classifier processes the video at full 30 fps rate. The ROC curve shows the tradeoff between fewer spurious car detections but more missed cars with a high *neighbors* (high detection confidence) and more detections (true and false) with low *neighbors*. Based on this data, ParkMaster sets *neighors* = 2.

Figure 10(b) shows the impact of varying the *scaling* parameter with the same resolution video and *neighbors* = 2. We see a similar tradeoff, with the classifier missing cars when its scaling search step is coarse (*scaling* = 1.5). Based on this data, ParkMaster sets *scaling* = 1.05 to make a reasonable tradeoff between misses and spurious detections.

Video resolution. Decreasing the video resolution reduces the FPR, but, at the same time, leads to lower TPR (see Table 4). While with resolutions higher than 720 × 480 the variation in TPR is relatively small, decreasing further the resolution causes a more consistent drop in the number of TP.

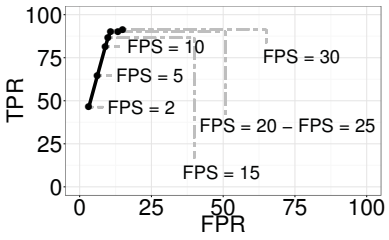


Figure 11— Detection with smartphone limitations: Impact of processing rate with video resolution 720×480, classifier parameters *scale* = 1.05 and *neighbors* = 2.

Resolution	Scale	Cropping ↕	Cropping ↔	Processing rate S4	Processing rate S6
720×480	1.05	½	½	5.78 fps	10.4 fps ★
720×480	1.05	full	full	1.18	4.11
720×480	1.02	½	½	2.30	5.22
1280×720	1.05	½	½	2.08	5.90
1920×1080	1.05	½	½	0.90	3.45

Table 5— Sensitivity analysis of video parameters on phone performance (processing rate) measured in frames per second. ★ indicates the parameters ParkMaster uses.

4.3.2 Tuning video parameters for performance. Computational power at the edge is increasing, but still limited. We thus consider what performance level (as measured in frames per second the phone processes) we require. Figure 11 shows many missed cars at 2 fps and diminishing gains past 10–15 fps, because the classifier already has enough frames in which it may detect the car.

Table 5 summarizes a sensitivity analysis on performance, measured in frames per second the Samsung Galaxy S4 and S6 can process when only car detection is active. Since ParkMaster aims to detect only vehicles parked on the right side of the road, it can focus on a smaller part of the image, the bottom-right part of the frame (½ cropping in both horizontal and vertical directions). We see that *scale* factors smaller than 1.05 significantly reduce the frames per second the smartphone can process. The data shows that with parameters chosen for ParkMaster (denoted by the ★ symbol), the Samsung S6 can process 10.4 fps, *i.e.*, one frame every 40 cm at 15 kph or one frame every 1.3 m. at 50 kph, removing frame rate as a limiting factor for car detection.

4.4 Measuring car localization error

In this section we probe the causes of inaccuracy in the localization process. Firstly, we evaluate the effects of camera distortion and calibration: we place an easily-recognizable object (a box with a high contrast color and well defined edges) at the end of a corridor, calibrating the smartphone with a copy of a road sign. Then, we place the phone at well-known positions in the corridor and we let it compute the relative position of the object to the camera. Due to the simplicity of the object, bounding box inaccuracies are close to zero.

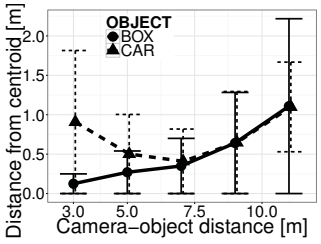


Figure 12— Localization error versus ground-truth distance from the object of interest, for an easily-recognized object indoors, and real cars in a parking lot.

Afterwards we measure the effects of bounding box misplacements — a classifier recognizing only parts of an object. We run a second, similar experiment in a parking garage detecting and localizing a real car.

In both experiments as the camera changes range to the object of interest, we compute the estimated location of the object (as camera-object distance plus camera position). Afterwards we compute the centroid of the set of samples collected at different locations and we measure the average distance between each sampled coordinate and the centroid: this essentially measures dispersion in the location estimate, which we refer to below as *localization error*. If the points are too dispersed, ParkMaster won't be able to track a car among subsequent frames because points belonging to cars close to each other will overlap. Figure 12 shows the resulting localization error for both experiments. As expected, getting farther from the box increases the localization error, because a miscalculation in the estimation of the camera extrinsic parameters (*i.e.* wrong orientation) increases its effects at large distance. Similarly for the car experiment, which however presents a peak in the error at small distance, due to bounding box error: the classifier detected only part of the vehicle (*e.g.* only the upper part of the car), which translates in a further away estimated location in street coordinates.

5 RELATED WORK

Car detection. *Advanced Driver Assistance Systems* use computer vision, among other techniques, to alert drivers of potential dangers in the surrounding area [49]. Several mechanisms have been developed to detect and track cars from a moving vehicle using an in-vehicle camera [9, 22, 38]. ParkMaster uses Viola and Jones classifier for its simplicity and computational speed.

Object tracking. Object tracking methods can be used in order to count cars. In [17, 28, 55, 60] the authors use background subtraction techniques to facilitate object tracking. This approach is however impractical in ParkMaster's scenario, where the camera is moving. [8, 17, 27, 29, 46] apply

multi-object tracking-by-detection algorithms, which rely on large temporal video windows to discern among subsequent objects. In contrast in our scenario the same vehicle may be detected in few frames only. Finally [23, 42] combine particle filters with object detector for Markovian tracking-by-detection, which [12] uses to track pedestrians. The tracker initialization however requires the same object to be detected few times in subsequent frames, which again is not always feasible in ParkMaster's scenario. Furthermore, the processing time of [12] is not negligible.

Object/obstacle localization. Stereo vision is commonly used to detect obstacles and estimate their relative location in the scene using simple triangulation [37, 51, 59]. Stereo vision comes at the cost of two cameras facing a scene at the same time, not available in the majority of COTS smartphone. In [33] authors use different pictures of the same scene taken by one camera in different locations to emulate two virtual cameras and reconstruct a 3D model of the scene. This approach however is very CPU intensive. Similarly Wedel *et al.* [58] propose the use of multiple pictures taken by a single camera to estimate the distance of a vehicle from a moving car. In order to be accurate the algorithm needs several samples, which may not be always available in ParkMaster case. Furthermore, both [33, 58] require an accurate localization of the camera, which may be hard to achieve with smartphone's sensors while moving on a car.

Camera calibration and coordinate system mapping. Another class of single-camera based methods for object localization consists in mapping camera's two-dimensional coordinate system to the three-dimensional road coordinate system. [53] introduces a pattern-based camera calibration, which has been used by [24] for obstacle avoidance on smartphones (through chessboard camera calibration). However, to calibrate the phone, the driver has to stand in front on the car with a chessboard every time the phone is placed on the dashboard. In contrast in ParkMaster, given the triggering action — the phone has been placed — calibration runs on–the–go without user intervention. [25, 41, 54] exploit Inverse Perspective Mapping (IPM) to remove perspective effects. Such techniques however require to know the height of the camera and it's angle to the road [32], which may change at each run. Similar assumptions are made by [61], which needs camera's height to compute car-camera distance.

6 CONCLUSION

This paper has described ParkMaster, the first system to combine advanced vision algorithms, edge computing on mobile devices, and the cloud to build a zero-overhead parking space availability solution for dense, vibrant urban environments that costs drivers and the city next to nothing. On–the–road experiments run in uncontrolled environments (city of Paris, Los Angeles and a small Italian village) validate ParkMaster approach, whose accuracy in estimating parking availability reaches 90%. We believe ParkMaster and similar technologies will push the envelope of what is possible in the smart cities of tomorrow, with a key-role played by the edge in extracting vital information about the urban environment.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 1617161.

REFERENCES

[1] http~://www.gsmarena.com/samsung_i9505_galaxy_s4-5371.php.

[2] http~://www.gsmarena.com/samsung_galaxy_s6_edge+-7467.php.

[3] http~://www.gsmarena.com/lg_nexus_5-5705.php.

[4] Google maps tells how bas the parking is at your destination. http://mashable.com/2017/01/18/google-maps-finds-parking-spots/#LafUADIWnaq3.

[5] Parking sensor performance standards and measurement. http://sfpark.org/wp-content/uploads/2011/09/SFpark_SensorPerformance_v01.pdf.

[6] H. Aly, A. Basalamah, M. Youssef. Lanequest: An accurate and energy-efficient lane detection system. *Pervasive Computing and Communications (PerCom), 2015 IEEE International Conference on*, 163–171. IEEE, 2015.

[7] H. Aly, A. Basalamah, M. Youssef. Robust and ubiquitous smartphone-based lane detection. *Pervasive and Mobile Computing*, **26**, 35–56, 2016.

[8] S. Avidan. Ensemble tracking. *IEEE TPAMI*, **29**(2), 261–271, 2007.

[9] M. Betke, E. Haritaoglu, L. S. Davis. Multiple vehicle detection and tracking in hard real-time. *Proc. of IEEE Intelligent Vehicles Symposium*, 351–356, 1996.

[10] C. Bo, X.-Y. Li, T. Jung, X. Mao, Y. Tao, L. Yao. Smartloc: Push the limit of the inertial sensor based metropolitan localization using smartphone. *Proceedings of the 19th annual international conference on Mobile computing & networking*, 195–198. ACM, 2013.

[11] J. Bouguet. MATLAB calibration tool: http~://www.vision.caltech.edu/bouguetj/calib_doc/.

[12] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, L. Van Gool. Robust tracking-by-detection using a detector confidence particle filter. *IEEE Conf. on Computer Vision*, 1515–1522, 2009.

[13] J. Canny. A computational approach to edge detection. *IEEE TPAMI*, (6), 679–698, 1986.

[14] R. Carisi, E. Giordano, G. Pau, M. Gerla. Enhancing in vehicle digital maps via gps crowdsourcing. *WONS*, 27–34. IEEE, 2011.

[15] D. Chen, K.-T. Cho, S. Han, Z. Jin, K. G. Shin. Invisible sensing of vehicle steering with smartphones. *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, 1–13. ACM, 2015.

[16] D. Comaniciu, P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE TPAMI*, **24**(5), 603–619, 2002.

[17] N. Dalal, B. Triggs. Histograms of oriented gradients for human detection. *IEEE conf. CVPR*, vol. 1, 886–893, 2005.

[18] M. Ester, H. Kriegel, J. Sander, X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD*, 1996.

[19] T. Fabian. An algorithm for parking lot occupation detection. *CISIM*, 165–170. IEEE, 2008.

[20] B. J. Frey, D. Dueck. Clustering by passing messages between data points. *Science*, **315**(5814), 972–976, 2007.

[21] Google snap to road. https://developers.google.com/maps/documentation/roads/snap.

[22] U. Handmann, T. Kalinke, C. Tzomakas, M. Werner, W. Seelen. An image processing system for driver assistance. *Image and Vision Computing*, **18**(5), 367–376, 2000.

[23] S. Ioffe, D. Forsyth. Human tracking with mixtures of trees. *Proc. IEEE Conf. ICCV*, vol. 1, 690–695, 2001.

[24] R. Itu, R. Danescu. An efficient obstacle awareness application for android mobile devices. *IEEE Conf. ICCP*, 157–163, 2014.

[25] R. Jiang, R. Klette, T. Vaudrey, S. Wang. New lane model and distance transform for lane detection and tracking. *CAIP*, 1044–1052. Springer, 2009.

[26] E. Koukoumidis, L.-S. Peh, M. R. Martonosi. Signalguru: leveraging mobile phones for collaborative traffic signal schedule advisory. *Proceedings of the 9th international conference on Mobile systems, applications, and services*, 127–140. ACM, 2011.

[27] X. Lan, D. P. Huttenlocher. Beyond trees: Common-factor models for 2d human pose recovery. *IEEE Conf. ICCV*, vol. 1, 470–477, 2005.

[28] N. D. Lawrence, A. J. Moore. Hierarchical gaussian process latent variable models. *Proc. of conf. on Machine learning*, 481–488. ACM, 2007.

[29] B. Leibe, E. Seemann, B. Schiele. Pedestrian detection in crowded scenes. *IEEE Conf. CVPR*, vol. 1, 878–885, 2005.

[30] R. Lienhart, J. Maydt. An extended set of haar-like features for rapid object detection. *Proc. Conf. on Image Processing*, vol. 1, I–900. IEEE, 2002.

[31] S. Ma, O. Wolfson, B. Xu. Updetector: Sensing parking/unparking activities using smartphones. *Proceedings of the 7th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, 76–85. ACM, 2014.

[32] H. A. Mallot, H. H. Bülthoff, J. Little, S. Bohrer. Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological cybernetics*, **64**(3), 177–185, 1991.

[33] J. G. Manweiler, P. Jain, R. Roy Choudhury. Satellites in our pockets: an object positioning system using smartphones. *MobiSys*, 211–224. ACM, 2012.

[34] S. Mathur, T. Jin, N. Kasturirangan, J. Chandrasekaran, W. Xue, M. Gruteser, W. Trappe. Parknet: drive-by sensing of road-side parking statistics. *MobiSys*, 123–136. ACM, 2010.

[35] A. Nandugudi, T. Ki, C. Nuessle, G. Challen. Pocketparker: Pocketsourcing parking lot availability. *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 963–973. ACM, 2014.

[36] S. Nawaz, C. Efstratiou, C. Mascolo. Parksense: A smartphone based sensing system for on-street parking. *Proceedings of the 19th annual international conference on Mobile computing & networking*, 75–86. ACM, 2013.

[37] V. D. Nguyen, T. T. Nguyen, D. D. Nguyen, J. W. Jeon. Toward real-time vehicle detection using stereo vision and an evolutionary algorithm. *VTC Spring*, 1–5. IEEE, 2012.

[38] D. Noll, M. Werner, W. Von Seelen. Real-time vehicle tracking and classification. *Proc. of the Intelligent Vehicles Symposium*, 101–106. IEEE, 1995.

[39] OpenCV: http~://opencv.org/.

[40] F. Ren, J. Huang, M. Terauchi, R. Jiang, R. Klette. Lane detection on the iphone. *International Conference on Arts and Technology*, 198–205. Springer, 2009.

[41] M. Rezaei, M. Terauchi, R. Klette. Robust vehicle detection and distance estimation under challenging lighting conditions. *Transactions on Intelligent Transportation Systems*, 2015.

[42] E. Seemann, B. Schiele. Cross-articulation learning for robust detection of pedestrians. *Pattern Recognition*, 242–252. Springer, 2006.

[43] Y. Sekimoto, Y. Matsubayashi, H. Yamada, R. Imai, T. Usui, H. Kanasugi. Lightweight lane positioning of vehicles using a smartphone gps by monitoring the distance from the center line. *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, 1561–1565. IEEE, 2012.

[44] SFPark: http~://sfpark.org/.

[45] D. C. Shoup. Cruising for parking. *Transport Policy*, **13**(6), 479–486, 2006.

[46] L. Sigal, M. J. Black. Measure locally, reason globally: Occlusion-sensitive articulated pose estimation. *IEEE Conf. CVPR*, vol. 2, 2041–2048, 2006.

[47] S. Soubam, D. Banerjee, V. Naik, D. Chakraborty. Bluepark: tracking parking and un-parking events in indoor garages. *Proceedings of the 17th International Conference on Distributed Computing and Networking*, 33. ACM, 2016.

[48] Streetline Networks: http~://www.streetlinenetworks.com/.

[49] Z. Sun, G. Bebis, R. Miller. On-road vehicle detection: A review. *IEEE TPAMI*, **28**(5), 694–711, 2006.

[50] Yellow cab of san francisco, location dataset. http://cabspotting.org/.

[51] G. Toulminet, M. Bertozzi, S. Mousset, A. Bensrhair, A. Broggi. Vehicle detection by means of stereo vision-based obstacles features extraction and monocular pattern analysis. *IEEE Transactions on Image Processing*, **15**(8), 2364–2375, 2006.

[52] N. True. Vacant parking space detection in static images. *University of California, San Diego*, 2007.

[53] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, **3**(4), 323–344, 1987.

[54] S. Tuohy, D. O'Cualain, E. Jones, M. Glavin. Distance determination for an automobile environment using inverse perspective mapping in opencv. *Proc. ISSC*, 2010.

[55] R. Urtasun, D. J. Fleet, P. Fua. 3d people tracking with gaussian process dynamical models. *IEEE Conf. CVPR*, vol. 1, 238–245, 2006.

[56] P. Viola, M. Jones. Rapid object detection using a boosted cascade of simple features. *Proc. of IEEE Conf. CVPR*, vol. 1, I–511. IEEE, 2001.

[57] C. Wah. Parking space vacancy monitoring. *Projects in Vision and Learning*, 2009.

[58] A. Wedel, U. Franke, J. Klappstein, T. Brox, D. Cremers. Realtime depth estimation and obstacle detection from monocular video. *Pattern Recognition*, 475–484. Springer, 2006.

[59] T. A. Williamson. *A high-performance stereo vision system for obstacle detection*. Ph.D. thesis, Carnegie Mellon, 1998.

[60] B. Wu, R. Nevatia. Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors. *International Journal of Computer Vision*, **75**(2), 247–266, 2007.

[61] C.-W. You, N. D. Lane, *et al.* Carsafe app: alerting drowsy and distracted drivers using dual cameras on smartphones. *MobySys*, 13–26. ACM, 2013.

[62] Z. Zhang. A flexible new technique for camera calibration. *IEEE TPAMI*, **22**(11), 1330–1334, 2000.