Street Parking Detection

Sergio Patricio Figueroa Sanz, James Tran, Chung Yu Wang

We present a computer vision-based street parking detection. Our system detects vacant parking spots in between the T-shaped parking markers that are commonly used to indicate parking spots on the sides of streets. We detect these T marks using a template-based sliding window approach that convolves HOG features. Using these T marks, we can find parking regions in the image, and compute color histograms to determine whether the parking regions are vacant. Our result achieved 69.3% recall and 92.6% precision. We conclude by analyzing the performance of our system on our mobile platform, the Nvidia Shield Tablet.

1. Introduction

Finding a parking spot is an unpleasant part of the driving experience. We propose a street parking detection system using a dash-mounted mobile device to automatically detect empty parking spaces that can be broadcasted to all users in system. Using computer vision techniques directly running in the device, we will analyze live camera footage to estimate and highlight potential parking regions.

The goal of our system is to help people find available parking spots. With our system, we hope to allow drivers to avoid averting their attention to find parking, and thereby increasing safety. Furthermore, with the advent of self-driving technology, cars will need to park themselves, a task which relies on automatically finding an empty parking space. If a lot of people were to use such a parking detection system, then we could gather the parking data onto a server that keeps track of available parking spots users drive past.

Current systems for parking detection are often static in nature. For example, many cities are implementing networked parking space trackers that can cost hundreds of dollars per parking spot in addition to continuous fees [1] [2]. Other systems use mounted surveillance cameras to analyze parking lots and detect potential parking spots [3]. While these systems are effective for determining whether predefined parking spaces are vacant, they cannot be readily adapted to a car-level mounted camera moving down a road.

More dynamic systems involve using satellites or drones to image streets to find potential parking. Satellite imaging is powerful and can sweep over large areas at a time, yielding a lot of information that can be extracted [7], but the images are rarely constantly updated and are thus not suitable for real-time parking detection. The drone method involves sending a drone to scout ahead and find parking spots to direct drivers towards [6]. This system has the advantage of finding parking spots well in advance, but sending out a drone in an urban area may not always be feasible.

2. Related Works

In this section, we will briefly review two prior papers on similar topics in Section 2.1 and discuss our key contributions in Section 2.2.

2.1 Review of Prior Work

Parking Lot Surveillance Camera

Many parking detections have been done for static parking lots as described in [3], [9] and [10]. Most of these systems fix the surveillance camera at a high angle and use human intervention to mark the regions of interest (ROI) to track the changes of the scene. With these ROI being marked, [3] builds 32-bin color histograms for each ROI with the luminance removed from the L*a*b* color space. Such a feature is then fed into either SVM or kNN to classify if the ROI is vacant or not. Both classification approaches result in around 90% accuracy. [3] also attempts to build a bag of words and tries to match the new features with the set of words. However, such an approach is not quite viable as the detector isn't able to constantly pick up the similar interest points. Although these systems enjoy some successes, they tend to suffer from problems like occlusion due to their fixed camera position.

Automatic Lane Detection

Previous work in automatic lane detection in real-time has also been done, though focused primarily on driving assistance applications (or even self-driving cars). Nonetheless, being able to detect lanes is pertinent to our problem, given that the T marks are essentially a special kind of lane marker. [4] provides heuristics to reduce search space for lane detection, a useful concept to us.

The proposed approach consists of a simple yet effective real-time lane detection algorithm that foregoes any complex features. What the authors propose is filtering the image with both a low-pass filter and a Canny filter to obtain a set of proposed points that could lie in edges. After some thresholding of these points to reduce them to a set of edge points, Hough voting is used to find proposed lines.

These lines are then filtered down to remove false positive lanes using a clever heuristic to constrain the search space. This heuristic takes into account that, when on the road, lanes tend to be vertical (in the image) and that they have certain ranges of angles depending on which side of the image they are on. Only the lines that fit the proposed heuristics are then clustered together into lanes. Detected lanes are subsequently used to help detection in the next video frame as another refinement of the search space.

2.2 Key Contributions to Related Work

Most of the parking detection developments are restricted to the surveillance camera model. This approach is not appropriate for the market as this technology heavily relies on the cooperation of the parking lot owner, who may not find the need to deploy such system. Furthermore, a fixed camera has a limited field of view, increasing the overall system cost if the parking lot is of multi-level or if the parking lot is a long strip similar to the case of a street parking. What we have achieved here tackles both problems. The cost is on the end users who will be motivated to deploy the system and potentially at no cost to them due to built-in front facing cameras in future cars.

Previous work related to actual street parking detection was done in [11]. Their approach was to use a high frequency line scanning camera facing side ways to detect the vacant spots. Their frame rate was at 7kHz while the car is driving at 10km/hr. Such approach requires a specialized camera and vehicle. This does not fit the today's consumer

behavior. Our approach of using a front facing camera will be able to cut down the need of the high video frequency and allow the driver to drive at a normal speed.

3. TECHNICAL DETAILS

Our pipeline for detecting street parking is as follows. We begin by reducing the search space by removing regions where parking spots are not likely to occur. We then find the locations of T marks based on the similarity scores via template matching in HOG space and further refine the bounding box detection via Hough line voting and saturation masks. Afterwards, we identify the locations of parking spots and use color histograms to classify whether or not a spot is vacant. Finally, we output the result back to the display.

3.1 Search Space Reduction

To run the application on the mobile device efficiently, we have to reduce the search space by removing the dashboard regions and the left part of the moving scene where no parking spots will likely exist.

Although a simple static mask may perform well in some scenarios, we created a dynamic mask, which can more flexibly adapt to wherever the dashboard lies in the video frame. This mask is created based on the changes in luminance from one frame to the other. Any pixel that is changing is marked as an important pixel. The assumption is that given the mobile device is mounted on the dashboard, the dashboard will not move relative to the camera and so the pixels covering it will not be marked as important. To be more robust, the system allows some small changes to be ignored in order to account for changes in lighting. Thus, any pixels that change from one frame to another beyond a certain threshold are marked as important.

After a few initial frames, the importance mask is created but it has regions of mixed important and unimportant pixels. If we were to apply this mask to our video frames as it is, we would produce noisy images, negatively affecting our algorithm. Thus, we further process the importance mask to produce a solid region of interest that corresponds to the camera frame's view of the street. The heuristic applied here is simple: assume the region of interest starts at the top of the camera frame and ends where we find, in the importance mask, the first row with no pixels marked as important (which should correspond to the beginning of the dashboard). This heuristic produces a solid rectangular region of interest that we can then extract to run the rest of our algorithm on.

As a final note, it is worth mentioning that, since the importance mask is additive and no pixel, once marked, is removed, it is convenient to stop the procedure and stop updating the mask after some time. In our own implementation, we simply stop updating the mask once it covers more than 50% of the video frame.

3.2 HOG Template Matching

Building our own HOG

Due to limitations of the current HOG Descriptor in OpenCV on cell sizes, block sizes and number of orientation bins, we implemented our own version and extended it so as to make the entire feature descriptor more flexible. We also adopted data structures that will facilitate our subsequent convolution step.

After testing with a few different parameters, we found that using an 8x8 pixel cell size and 9 bins for gradient

histogram has enough granularity to detect the closest two T marks without too much computational burden. Therefore, we used those values for our HOG feature implementation.

Generating Templates

We collected over 18 raw template T marks, each with slightly different perspective, shape, and noise level as our positive templates. For each of them, we built a pyramid of varying scales and aspect ratios. Because these raw templates were collected directly from the video, the perspective and rotation will not vary much from the actual input video and no variation on those parameters are built into the pyramid. However, due to the HOG feature descriptor's granularity, the additional raw template T shapes did not increase the detection rate. Hence, we only used the clearest and closest raw template we had.

Sliding Window Convolution on HOG

With one raw template, we created a 20-level pyramid of different scales and aspect ratios. We calculated and stored the HOG for the templates before the start of the input video. Upon receiving an input frame, we then calculated the HOG for the frame and convolved each template's HOG with the frame's HOG in a sliding window fashion. We kept only the candidate bounding boxes that are above an absolute similarity score and at least 80% of the similarity score of the highest similarity score in the frame. Then, we used non-max suppression based on IoU (Intersection of Union) to remove noise. Finally, we would only keep the top 5 candidate bounding boxes per frame, given that we can only see at most 3 to 5 T shapes per frame.

HOG Template Refinement

Upon testing, we noticed a significant amount of noise in the raw templates in regions where there aren't supposed to be any edges from the T shape. These noisy gradients significantly increased the false positive rate. Hence, we increased the contrast and removed the saturation of the templates before finding the HOG features. Figure 1 shows the HOG feature of the template with and without these preprocessing. Although our actual video inputs will not go through the same preprocessing procedure, this is done since our templates are collected from an uncontrolled environment with noisy background.

3.3 Bounding Box Refinement: Hough Line

We used the fact that T marks would lie in a straight line to remove noises. Although using a line with a fixed location and angle could potentially avoid the use of Hough voting, it was nonetheless an interesting approach to practice what we have learned. Hough voting also informs us how confident it is in identifying a line via the number of votes it received.

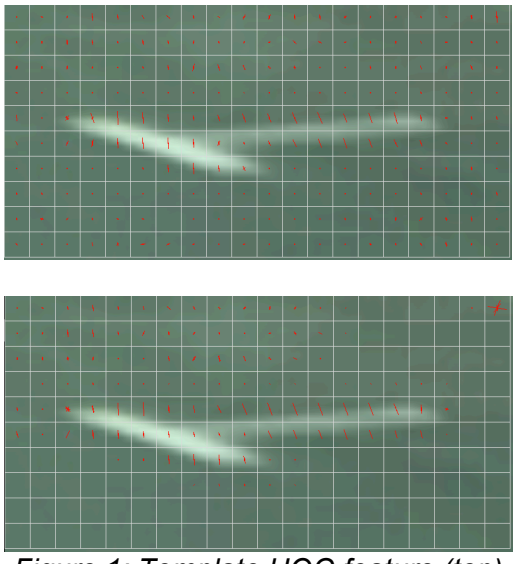

Figure 1: Template HOG feature (top) without and (bottom) with increased contrast and saturation removal

Since our current HOG template matching approach has not been able to find more than two true T bounding boxes per frame, we used the bounding boxes from previous frames as shown in Figure 2. This approach is valid given that the direction of travel does not change drastically and therefore the true bounding boxes in the previous frames would be aligned with the current true bounding boxes. We used the center points of the past 100 candidate bounding boxes to conduct the voting. In general, the more previous bounding boxes we keep, the more robust the system is to noise. Upon identifying the top voted line, we apply a Gaussian distribution along the line and increase the score of bounding boxes based on the probability of the bounding boxes lying on that line based on distance to the line.

Finally, we only apply the score adjustment if the line has more than 40 votes. In the case when there are genuinely no T shapes, then ideally the noise would be random and all lines would have fewer votes, and hence no score adjustment is justified.

3.4 Bounding Box Refinement: Saturation and Value Mask

To further refine our bounding boxes for the T marks, we used a heuristic based on whitish color of the T shapes. We created a mask based on the pixel's saturation and brightness values using the image's HSV space. Such a mask would be on for regions with low saturation (below 10%) and high brightness value (above 40%). To handle potential noise, we conduct dilation on the mask before applying the mask on to the bounding boxes. Figure 3 shows a sample mask. For the given bounding boxes, we only include those bounding boxes whose centers have an on bit in the mask.

Figure 2: (top) Current frame's bounding box, (bottom) Past 100 bounding boxes

3.5 Parking Spot Region of Interest

Given the locations of the T marks, we then identify regions of parking spots (vacant or not). For every region between every pair of the T marks that satisfy the dimension and orientation of a parking spot, we will identify it as a parking spot.

As cars can be parked at slightly different locations within the parking spot, after identifying a parking spot, we only use middle 80% of the region between the two T marks for vacancy detection. This avoids obtaining pixel values from cars that are either in front or behind the current spot while still retaining enough data for vacancy detection.

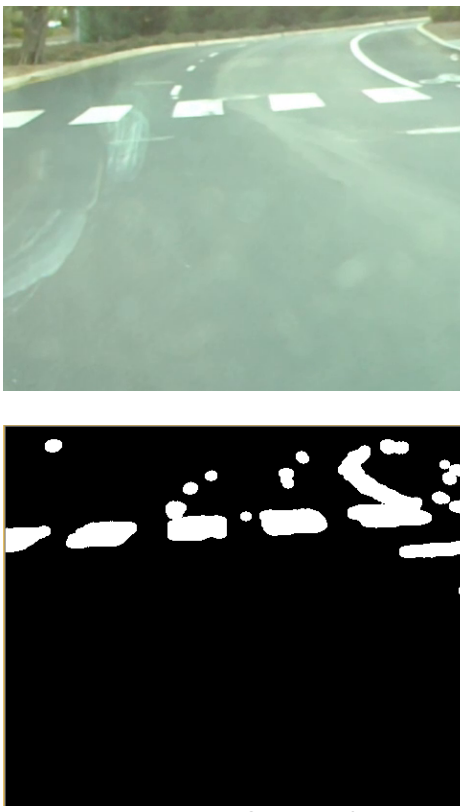

Figure 3: (top) Current frame (bottom) Dilated mask

3.6 Parking Spot Vacancy Classification

While there are many ways to classify if a parking spot is vacant or not, including more involved car detection algorithms, we take after the work in [3] and build a color histogram of the region of interest.

Initially, we considered using the RGB color space to compute our histograms, with the idea being that an empty parking spot should be mostly asphalt and thus would have a very uniform color. This would mean that the color histogram would have only one peak per channel in RGB. While this is generally true, we found that, in our experiments, results were somewhat inconsistent.

In order to produce more robust results, we used the HSV color space, where hue and saturation can provide useful information. Particularly, there are two essential observations that helped us in the task of classification:

- First, hue summarizes the information that was originally obtained from the RGB channels. If our histogram were to have hue values concentrated in an area, then this indicates a less varying color distribution.
- Second, saturation is a good measure of the amount of white/gray colors in an image. If most of the saturation values were to be in the lower bins of our histogram, this means most pixels are either white or gray.

A good simple classification method is to classify the histograms that have a single peak in hue and a single peak in the low ranges for saturation as vacant. When such distribution pattern occurs, the ROI has a similarly colored area that is mostly white or gray, and hence likely to be asphalt. In contrast, a parking spot that does not satisfy this would mean that we have more varying colors in the area, meaning that there is something in it, in all likelihood a car.

Thus, considering all of the above, we implemented HSV color histograms with 16 bins for each of the channels for each parking spot identified as shown in Figure 4. Subsequently, we simply use the heuristic mentioned above to classify whether a parking spot is vacant or not.

While the heuristic defined works well in many scenarios, there is room for improvement to produce more robust results if we, instead of using a simple test for each histogram, were to implement some learning algorithm to better classify vacancy.

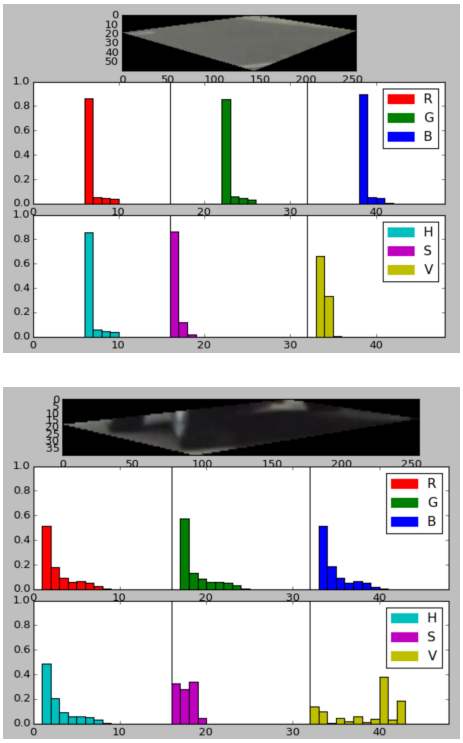

Figure 4: (top) Histogram for a vacant parking spot (bottom) Histogram for an occupied parking spot

3.7 Other Technical Attempts

During the development of the T mark detection, we tried numerous other technical pursuits. Here are a few that were not used in our final pipeline.

Contrast Normalization

In order to narrow down our search space, we tried to isolate the whitish regions by increasing the contrast in a frame by thresholding out the lower intensity colors in the image and spreading out the remaining values. After this, we removed the saturation in the image to get a purely black and white image. This yielded some results, but it required picking a very accurate threshold, and this value proved to be too sensitive to lighting conditions like glare from the sun. If the threshold value is too high, we would be removing T marks from the scene, while if it is too low, we would not get a very useful result.

EM based Segmentation

Based on the Leafsnap paper [12], we considered an EM strategy based on HSV color space in order to segment the image into two areas: the whitish regions (where the T marks are) and everything else. Since Leafsnap's EM is essentially a refined K-means, we decided to segment our image into two clusters using K-means on the saturation and value channels of HSV space. Unfortunately, the black/gray ground is too similar to the white T marks while the trees and sky are too different from the ground, so we cannot clearly differentiate the T marks using this method.

Locality Based Bounding Box Refinement

To reduce false positive bounding boxes, we noticed certain false positive boxes appear only in a single frame. We implemented a localization method that only keeps bounding boxes appearing at roughly the same location for two or more frames in a row. However, this method proved to be not as useful as expected since a lot of false positives appeared in the same location for more than one frame, and furthermore, some true positive boxes jitter around from frame to frame, meaning that our localization threshold could not be very strict.

4. EXPERIMENTS

This section discusses some results on our detection performance and runtime performance.

4.1 Detection Performance

We drove down Bowdoin Lane at Stanford University and took some video to test our system on. Figure 5 shows an example of a successfully detected parking spot, indicated by the green marker.

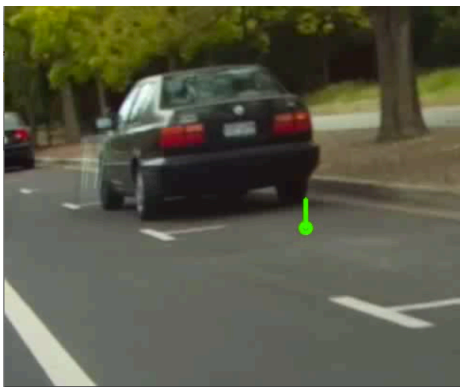

Figure 5: Vacant parking spot detected

Our algorithm achieved a 69.3% recall and 92.6% precision on this road. The high precision is desirable since if our system detects a vacant parking spot, we want it to truly be a vacant parking spot. In contrast, our system has a much lower recall, primarily due to the failure in detecting two closest T marks concurrently in one frame due to noises and thresholding. When such is the case, we currently do not pass it on to subsequent vacancy classification pipeline. To minimize such dependency on having both T marks detected in the same frame, we may use previous frame's T mark locations to estimate current frame's T mark locations.

Finally, the system had trouble finding faraway parking spots, not detecting them until they got closer. For example, Figure 5 shows how our system does not detect the far parking spot. Part of the reason is because our templates better suit close T marks compared to far ones, but more importantly, the far away T marks are too small to have enough HOG features to distinguish them from noise. We need to rely more on contextual clues in order to detect those far marks. The Hough lines helped partially but could not fully solve the problem.

4.2 Performance Analysis

We ran the application on the NVIDIA SHIELD Tablet (Nvidia Tegra K1 192 core Kepler GPU, 2.2 GHz ARM Cortex A15 CPU, 2GB RAM) and saw average runtimes of 390ms - 500ms per frame. We also recorded the amount of processing power it required to run and the amount of memory it used. The results are shown in the graphs that follow.

Before the app was run, the processor had 20% utilization for all running processes on the device. However, once the application started, the processor had 78% utilization. Of this percentage, the breakdown of processor use was as delineated below:

Process	CPU Utilization
Parking Detection (user)	44%
Parking Detection (kernel)	8%
Camera (user)	9%
Camera (kernel)	5%

We included the camera use into the processor utilization for our program since our app necessitates the camera and was the only one running at the time that used it.

On the other hand, when we look at the memory use for the application on device, we get the following results: before starting the application, we have about 25% free memory and about 30% memory in use by programs. When running our parking detector, we see a decrease in the free memory and the memory used by all programs increases to around 51%. Given that the only difference between the two memory

profiles we observed was our application, we can easily attribute the cause of the 21% memory use increase to Parking Detection.

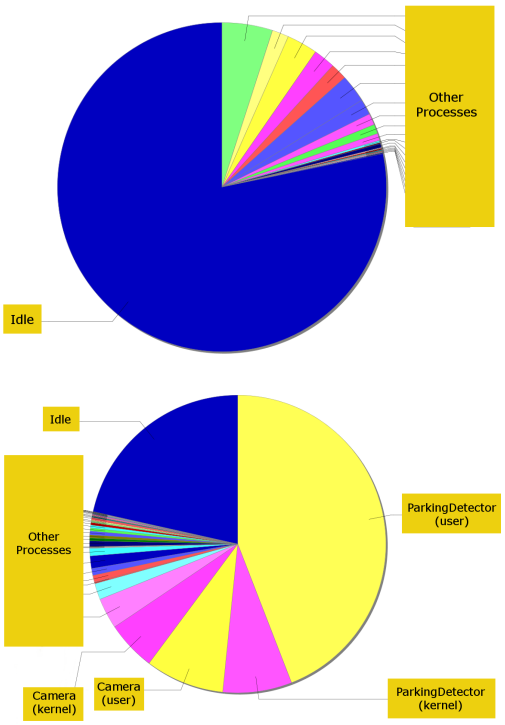

Figure 6: CPU usage (top) without and (bottom) with the app running

In addition to the two tests on mobile above, we profiled the C++ code running on a desktop in order to find bottlenecks in our algorithm. Our main method took 61% of the runtime (most of the remaining 39% came from open_cv's video reading code used to set up our tests). Of this 61%, 52% came from finding the bounding boxes using HOG with the other 9% being everything else.

Of the 52% spent computing the bounding boxes, 21% came from computing HOG histogram orientations. This part of the code relies heavily on C++'s atan2 function, and trigonometric functions can be very expensive. This would be a prime area for improvement; there are many fast approximations to the arctangent that could potentially work. The other big components of the 52% were the sliding window convolution (15%), the bounding box filtering described in section 3.4 (6%), and HOG gradient computation (5%).

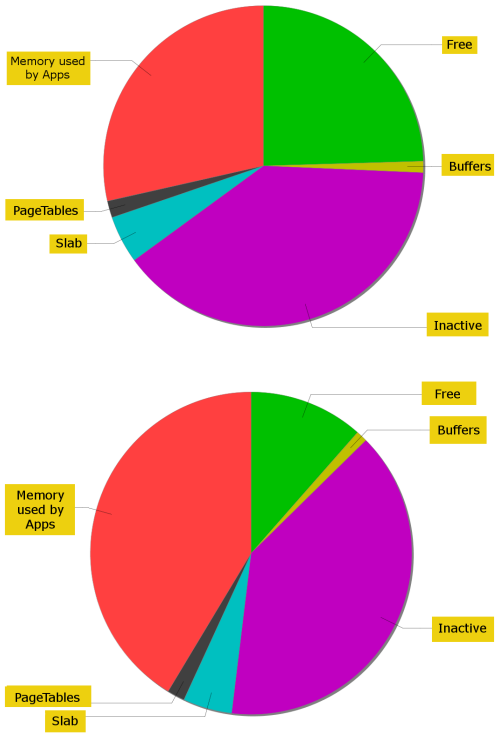

Figure 7: RAM memory usage (top) without and (bottom) with our app running

Thus, the HOG computation is the most expensive part of our pipeline. This is expected since we compute costly orientations and then do a sliding window approach to template matching. A faster approach could use masks early in the pipeline to narrow down the region we need to do the HOG calculations on. Furthermore, instead of HOG, faster descriptors like binary descriptors could potentially be useful.

5. Conclusion

This project demonstrates that using a pipeline similar to the one we propose can obtain high precision in identifying vacant parking spots. Although there are several challenges in identifying the T shapes, including blurriness, exposure and shape

variations making the T mark detection not entirely robust, we still managed to produce results that are still highly useful to drivers.

There are still a lot of unexplored areas in this project for future work. With our current pipeline, it would be worth training a SVM model that will classify if a bounding box contains a T mark based on the current HOG feature, instead of using a hard threshold on the 1D similarity score. Although potentially even slower, it might be worth trying CNN to identify the T marks or even just the cars with shallow layers of CNN network. What we have learned about reducing the search spaces using Hough voting and color masking would be tremendously useful in the case of CNN.

With this project, we hope that the future parking detection paradigm would shift from the current fixed camera model to mobile cameras. This is the ultimate goal and we are extremely excited to push the technology in this direction.

6. REFERENCE

[1] John, Jeff St. "Cisco, Streetline Team Up on Smart, Networked Parking." Greentechmedia. Greentech Media, 4 Dec. 2012. Web. 09 May 2015.

[2] Ross, Valerie. "Looking for Parking? Check Your Phone." Popular Mechanics. Popular Mechanics, 16 Feb. 2011. Web. 09 May 2015.

[3] True, Nicholas. “Vacant Parking Space Detection in Static Images.” http://cseweb.ucsd.edu/classes/wi07/cse190-a/reports/ntrue.pdf

[4] Buczkowski, M., and Stasinski, R. “Automatic Lane Detection” http://www.pwt.et.put.poznan.pl/PWT_2012/PWT%202012_3349.pdf

[5] Zitnick, C. Lawrence, and Piotr Dollár. Computer Vision–ECCV 2014. Springer International Publishing, 2014. 391-405. Topic — Object Proposals.

[6] Guha, Auditi. "Where’s My Car Again? Plan Uses Drones to Find Parking Spots." Washington Times. The Washington Times, 28 Feb. 2015. Web. 09 May 2015.

[7] Skybox Imaging http://www.skyboximaging.com/technology

[8] Xiang, Yu, et. al. “Data-Driven 3D Voxel Patterns for Object Category Recognition.” http://cvgl.stanford.edu/papers/xiang_cvpr15_3dvp.pdf

[9] Wah, Catherine. “Parking Space Vacancy Monitoring.” http://cseweb.ucsd.edu/classes/wi09/cse190-a/reports/cwah.pdf

[10] Wu, Qi and Zhang, Yi. “Parking Lots Space Detection.” http://www.cs.cmu.edu/~epxing/Class/10701-06f/project-reports/wu_zhang.pdf

[11] Hirahara, K.; Ikeuchi, K., "Detection of street-parking vehicles using line scan camera and scanning laser range sensor," *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE* , vol., no., pp.656,661, 9-11 June 2003

[12] N Kumar, PN Belhumeur, A Biswas, DW Jacobs. “Leafsnap: A computer vision system for automatic plant species identification.” ECCV 2012.