Abstract

Finding a parking space in a busy parking lot is complicated. The traditional approach involves driving through the entire lot, examining each space to determine its usability. This approach has some shortcomings that make it unusable in large or extremely busy lots. In this paper, we examine an approach to automating the detection of parking spaces that are about to be vacant—in other words, the detection of cars that are about to leave the space they are in. We examine the accuracy of tracking moving people and vehicles and how this information could be used to display vacancy information at the parking lot entrance.

1 Introduction

1.1 Motivation

The task of finding a parking space in an urban environment is complicated. Compared to rural or suburban environments, the number of spaces and their location is limited. In a busy location, people waiting to park can possibly wait several minutes before finding any type of space—much longer if they want a closer space. The act of finding a space involves driving around the entire parking lot multiple times, examining each space to determine its vacancy. This approach is not optimal, for several reasons:

Use of non-renewable resources The act of examining all the spaces in a parking lot by car uses valuable gasoline and diesel fuel. This fuel could be used for other purposes, such as delivering goods across the country. Unnecessary burning of fossil fuels increases the operating cost of the vehicle.

Inefficient use of time Every minute spent looking for a space is a minute that could be used running other errands or spending time with family.

Traffic considerations Cars that are needlessly driving around a parking lot increases the traffic inside the lot. This reduces the rate at which cars leave the lot, compounding the problem.

From the above, it is obvious that a better approach is needed. We propose a computer vision solution to the problem of parking space vacancy detection, in which video taken from a camera
(or other source) is analyzed and appropriate actions taken depending on the movement of objects within the camera’s field of vision. The process could be extended to broadcast a listing of parking spaces that are about to become vacant to a conspicuous location in the parking lot.

1.2 Previous Work

There is at least one project that has already attempted to tackle this problem. UbiPark[2], designed by students at UCSD, is designed to detect vacant parking spaces versus occupied spaces. It relies only on the presence or lack thereof of a car in a particular space.

Zhao and Nevatia tackled the human tracking problem in [4]. In particular, they worked on tracking multiple humans at once, something that this system would have to implement to be useful in busy situations. There are other approaches to human motion capture and analysis at [3].

Finally, the general problem of object detection has been discussed in Bergboer, Postma and van den Herik’s paper[1]. Their approach relies on still images, but in theory, such an approach could be extended to video as well.

2 The Person/Car Tracking Algorithm

The algorithm is split up into several sections.

2.1 Image Differencing

In order for the other algorithms used by the system to function, we need to be able to determine which portions of the frame have changed, relative to a certain frame or combination thereof. In our system, the components of each pixel of the current frame are subtracted from the components of each pixel in the previous frame. This produces the amount of difference in each component. Then, to visualize the magnitude of change across all components, the Euclidian distance, \( \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2} \) (where \( x \) corresponds to the current frame and \( y \) corresponds to the previous frame) is used. This distance is filled in for all three components of a result image, producing a grayscale RGB image.

This leaves the question of what each component should contain. In typical graphics applications, RGB is used. RGB places the magnitude of red, green and blue in each pixel. This also implicitly encodes information other than the color of each pixel, such as luminiscence. In contrast, L*a*b* separates color data from luminiscence data, allowing an application designer to determine the amount of movement in an image simply from the degree of the color change. As a result, the L*a*b* color space is used in this project.

Once the grayscale image is produced from the image differencing process, extraneous data—very small changes of a pixel or few in area—are removed by smoothing the image using a Gaussian blur. The kernel used for this purpose is a 7x7 matrix. This effectively eliminates very small changes from the image, while leaving larger areas usable in later algorithms.

2.2 Blob Detection

Blob detection (also known as connected component labeling) is the next step in the pipeline. This takes the grayscale image produced in the previous section and attempts to find the bright areas. We use a library in OpenCV[5] for this purpose, called cvBlobsLib. Once blobs are detected, the library determines their size and shape and encodes this information in a structure for later use.

To account for intermittent differences that result in many smaller blobs, we merge blobs in the same frame that are close together into one larger blob. Merging blobs requires a pass over all the blobs in the frame to determine which blobs are close together. This calculation must be done on the order of \( n^2 \) times, where \( n = \text{number of blobs} \). Because of this, we replaced euclidian distance calculation with box calculation. The center of each box has a box drawn around it. If any of these boxes overlap, the structures of each blob are merged and both blobs are considered as a single blob. This process is repeated until every blob has been checked against every other blob.
2.3 Object Typing

In order to take appropriate action, it is necessary to know what type of object a particular blob represents in an image. From our own perception of the world around us, we realized that cars tend to have a low height to width ratio, while humans have a larger height to width ratio. A 1:1 height/width ratio is a square.

To make this decision as quickly as possible, we take the minimum/maximum x and y coordinates of each blob and calculate the width and height of the blob (using $|x_{\text{min}} - x_{\text{max}}|$ for width and $|y_{\text{min}} - y_{\text{max}}|$ for height) and diving through. If the ratio is less than one, it is a car. Otherwise, it is a human. The designation is made in such a way that a blob is categorized as a person by default unless otherwise calculated.

2.4 Motion Tracking

Once we have a list of blobs in a frame, the next step is to correlate them with already detected movement. We start with a structure called a Worm. This structure stores the size and type of a blob, as well as each individual coordinate that the object was seen in during its movement.

When the application is initially started, the list of worms is empty. For each blob detected in the current frame, every worm in the list is examined. In particular, the last known position of the object and its type is retrieved. Using Euclidian distance, the worm that is the closest to the center of the current object (within a certain threshold) is selected. If there are no suitable objects, a new Worm is created with the center of the blob as the first coordinate. In other words, this is a modified k-nearest neighbor algorithm, where $k$ in this case is 1.

2.5 Detecting Potential Vacancies

Although this paper does not concentrate on detecting vacancies, we speculate that given the algorithm described above, the process of detecting parking space vacancies is simple. If a person worm’s last coordinate is within a certain distance from a car worm’s first coordinate, then it can be concluded that the parking space is being vacated. Likewise, the number of frames since a person blob’s last movement can be used to determine the probability that a space will open up within a given amount of time. Performing heuristics such as these is beyond the scope of the paper.

2.6 Implementation

The software used in this project was implemented using the OpenCV computer vision library and the C++ programming language. These two technologies were chosen for their built-in features; for C++, the STL allowed access to pre-written classes and objects for basic data types, while OpenCV provided basic image manipulation and blob detection. We were able to focus on the workings of our algorithm instead of having to deal with pixel-by-pixel image manipulation.

3 Experimental Method

Several variables are measured in this project. One important variable is type classification accuracy. Does our algorithm properly detect whether a blob is a person or a car? How often does it make this detection properly versus improperly or not at all? Another important variable is the accuracy of the actual object detection. Does it detect objects that it should? Finally, does our algorithm track objects correctly during their entire lifetimes?

To test these, we filmed several minutes of footage from the fifth floor of UCSD’s Applied Physics and Mathematics building (AP&M). This footage contained both cars moving around in the parking lot facing the building, as well as people walking along the walkways in front of the building. This footage was run through the algorithm repeatedly, tweaking various parameters as needed.

Then, three more videos were taken from an empty parking lot (Regent’s lot). One video contained a dark colored vehicle parking in a space, followed by a person walking around outside and then returning to the car and driving away. A second video contained the same footage, but with a white
Table 1: Number of objects and their detection accuracy

<table>
<thead>
<tr>
<th></th>
<th>actual</th>
<th>detected</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13</td>
<td>110</td>
<td>746</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>12</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 2: Number of objects and their detection accuracy after improvement

<table>
<thead>
<tr>
<th></th>
<th>actual</th>
<th>detected</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13</td>
<td>9</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>5</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>0.6</td>
</tr>
</tbody>
</table>

car. The final video contained both cars in the same video next to each other. This final video was
designed to test the algorithm’s behavior with multiple objects that might conflict with each other.

At the conclusion of a run of the application, the application produces two Comma Separated Values
(CSV) files: one that contains each object and its classification, and another that contains each
coordinate of each object. These results are then plotted and compared to the expected (manually
classified) results for each object.

4 Results

The algorithm detected several objects well in the 5th video taken from AP&M. It detects moving
objects well. Keeping track of those objects from frame to frame is more problematic. Table 1 shows
the number of objects, number of objects detected and percentage of superfluous detections.

Object lifetime for each of the videos varied widely. Lifetime is calculated by the number of frames
that an object exists. At each frame, the algorithm attempts to join objects from the previous frame
with objects in the current frame. Figures 1 through 6 contain the lifetimes of the detected objects
for the videos taken at AP&M, and the corresponding trajectories of the 3 longest lived objects.

Object tracking improved by a great deal when objects were tracked independently of their classifi-
cation. In the previous runs, classification was done once on an object at the object’s birth. Every
time the object’s classification changed, it was regarded it as a new object. Now the object is tracked
based on position only. The classification of an object can change throughout the life of an object.
Table 2 shows improvement over the previous method.

Figure 1 through 6 show the object lifetimes for each video, and the corresponding trajectories of
the top 3 longest lived objects.

Figures 7 through 12 show the new histograms of the objects in each video after improving the
tracking. They also show the trajectories of the 3 longest lived objects.

In Figure 13, the blue box behind the white truck driving to the left should be red. The truck is
classified as a person in this frame. Objects had odd shapes after filtering and amalgamation steps.
However, everything else in the frame is correctly labeled (if we consider the biker as a vehicle).

In Figure 14, there is a vehicle on the right hand side of the video. It is correctly labeled as a vehicle.
Behind it is another box indicating a vehicle object on nothing. Sometimes the object tracking
leaves behind a box. this was originally because we expect to track cars going to parking spots and
stopping. Our object tracking depends on motion. If we lose track of an object, we assume that
it has merely stopped moving. There are cases like this one which show we need to improve this assumption.

In Figure 15, the blue box in the horizontal middle of the frame is tracking the person walking to the left of it. Sometimes the boxes lagged behind the object they were tracking. This is due to the object not moving fast enough and causing enough difference to be recognized. The box will move to keep up with the person eventually.

5 Conclusion

The objects detected in the video switched between being categorized as cars and people. Every time this happened, the algorithm could not identify that the object was the same object as an object in a prior frame, even if the x and y values for its center were within the threshold. Because of this, the number of objects detected is much greater than the actual number of objects, even in short videos. The lifetime of each object is very short to the point of being unusable. We will have to modify the algorithm to keep tracking an object even if the type changes.

After viewing the algorithm at work on the videos taken on saturday in Regents Parking, we realized that we are going to have to modify them. The objects captured in Regents Parking appear much larger than the objects viewed from AP&M. As a result, each object has many boxes drawn on it. There are also many boxes floating around in the ambient noise of the videos. As a result, we have found that our algorithm was dependent on the objects being filmed from a certain distance. We had not considered this while writing the algorithm.

We found that our algorithm needs work keeping objects linked together through the entire sequence of the film. It also needs help in keeping each object classified. Improving the classification will probably improve tracking because tracking currently depends on the class of the object being tracked. The algorithm can also be more forgiving when the object changes class.

6 References

Figure 2: Trajectories of objects in video 1
Figure 3: Video 2 lifetime of objects, in number of frames.


Figure 4: Trajectories of objects in video 2
Figure 5: Video 3 lifetime of objects, in number of frames.
Figure 6: Trajectories of objects in video 3
Figure 7: Video 1 lifetime of objects (after changes), in number of frames.
Figure 8: Trajectories of objects in video 1 (after changes)
Figure 9: Video 2 lifetime of objects (after changes), in number of frames.
Figure 10: Trajectories of objects in video 2 (after changes)
Figure 11: Video 3 lifetime of objects (after changes), in number of frames.
Figure 12: Trajectories of objects in video 3 (after changes)
Figure 13: Screen capture from video 1
Figure 15: Screen capture from video 1