Visual Sensor for Smart Parking

Alex Macdonell, Jonathan Lobo

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

Dec 19, 2011

Technical Report No. ECE-2011-01

Contents

List of Figures

List of Tables

1 Introduction

Finding a parking spot in a garage or metropolitan setting can prove to be quite an issue. No one likes to circle around parking lots looking for empty spots that may or may not exist. This process could become a lot simpler if there were a method to determine which spots are vacant and alert drivers of these vacancies. Current solutions to this problem involve entrance/exit sensors which merely help to let drivers know that there are vacancies, not where they are. Other proposed solutions include installing magnetic sensors underneath each parking spot to determine whether there is a vehicle parked. This method costs significant amounts of money and is thus not ideal. Instead, we propose a solution that uses parking lot surveillance cameras and a video algorithm to determine whether a spot is vacant or occupied. This method would be able to identify the status (vacant/occupied) of each spot and would not increase cost significantly as most parking lots are already outfitted with surveillance cameras. This solution is based on the comparison of region covariance matrices as in [1], and has different implementations for night and day conditions in order to better address changing lighting conditions. In this report we will first briefly discuss other work that has been in this area and then move on to a more detailed description of the problem. We will then discuss the theoretical underpinnings of the solution, present some results from initial testing, and draw some conclusions.

2 Related Work

The detection of vehicles in both static images and in video has seen a great deal of research in the past couple of decades. In the early 1990’s, an automated system to detect vehicles in video streams based on image processing techniques, known as Autoscope, was developed [2]. This system detected vehicles by comparing the intensity values of newly captured pixels with the value of those pixels for the background image only, accumulated over a wide range of conditions. More sophisticated methods were explored as well, including a generic approach which can detect any unique object in a scene by forming a dictionary of intensity differences across various regions of the image [3]. The only drawback of this approach is that a large training set of positive and negative examples is required to ensure the accuracy of the algorithm. Recently, robust algorithms using hidden Markov models have been employed to detect and track vehicles in video streams [4]. These algorithms are unsuitable for application in parking space occupancy detection, however, as they depend on temporal information from the video to compare to a vehicle motion model [4].

Non-traditional techniques for occupancy detection of parking spaces have also been well explored in recent years. In 2007, a method of tracking parking space occupancy was proposed using Vehicular Ad-hoc Networking (VANET) [5]. This solution has the unique advantage that it would be usable in any parking lot or structure with no installation cost, but requires that VANET equipment be available and operating correctly in all vehicles that wish to park. A system using a surveillance camera to detect parking space occupancy was developed by a student at UCSD in 2007, using comparisons of color histograms between captured camera frames and training data to make occupancy decisions [6]. A more complex system using surveillance cameras was developed in 2009 and uses a classifier based on fuzzy c-means clustering to make robust occupancy decisions in a variety of lighting conditions [7]. This system was designed specifically to lower error-rates for detections in outdoor parking areas where lighting conditions are extremely variable [7], an issue that we will also address in the solution presented here.

3 Problem Statement

The problem addressed by our solution is a problem of detecting whether or not a parking space is occupied or empty by performing analysis on frames extracted from a camera that has that parking space in its field of view. This is not a problem of classification, and our solution does not make decisions about the type of object that is occupying the parking space. The following assumptions better define the operation and scope of our solution:

1. A sequence of images will be extracted from a static, color camera with a fixed zoom. Minimal camera jitter is acceptable, but large movements will cause the solution to fail.
2. The locations of parking spaces in the view of the camera are known to the system, and will not be automatically extracted by our solution. This constraint targets the situation where the camera is mounted in a parking garage or other location where it is monitoring a fixed set of parking spaces that can be manually located in the camera feed during installation.
3. The system has knowledge of the current time of day, which is necessary to distinguish night frames from day frames. It is assumed that frames taken from the camera will have the time of day attached to them as metadata or as part of the frame itself, as is the case with the camera used for testing the solution presented here.
4. Only a sufficiently large object in a parking space will cause the space to be labeled as "occupied". The solution was not designed to detect arbitrarily small objects under the assumption that the presence of these objects in a parking space is a transient event.

In addition to these requirements, it is of note that the testing of our solution was done using side views of parking spaces only. This is not necessarily a requirement of the solution, but its performance under other conditions is unknown.

4 Solution

Our solution revolves around the comparison of the "contents" of parking spaces with spaces that are known to be empty. Specifically, we seek a distance metric between an occupied space and an empty space that can be compared against a threshold value to yield a binary detection result. The threshold value is established from training data that includes a large number of "distance" values computed between empty spaces as well as between empty and occupied spaces. This distance metric is established via the region covariance approach, described in detail in [1], which is also the source of the summary in section 4.1. The region covariance method was applied to both day and night frames, with the night frames undergoing additional pre-processing which is described in further detail in section 4.2.

4.1 Region Covariance Overview

In order to establish a distance metric between parking spaces in our solution, we followed the region covariance approach described in [1]. This approach starts with selecting a set of image descriptors to form a “feature vector” at each pixel in the image. In selecting features to include in the feature vector, it is important to use only the ones which contain unique descriptive information about the scene. Using additional features that contain little information adds to the complexity of the problem and does not help to distinguish the image being described from other images. For our solution we selected a four-element feature vector containing the position of the pixel being examined and the gradients of the luminance of the image (Y from the YC_BC_R color space) at that pixel, shown as

$$\{\overrightarrow{\boldsymbol{z}_k}\}_{k=1\ldots n} = \begin{bmatrix} x_n \\ y_n \\ dY_n/dx \\ dY_n/dy \end{bmatrix},$$

where *n* represents a pixel number from a serialized version of the image obtained from scanning the image horizontally. The gradients were obtained using the Sobel operator which is shown below [3]:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

We had originally considered the inclusion of pixel chrominance values and/or second derivative values in our feature vector, but we found that these items lack sufficient information in our images to be useful. We also considered using various other kernels to take the image gradients and tested several, but found that the Sobel kernel performed well under the noise that is present in our images.

We now describe a parking spot image by computing the 4x4 covariance matrix of our feature vectors [1]

$$\mathbf{C} = \frac{1}{n-1}\sum_{k=1}^{n}(\boldsymbol{z}_k - \mu)(\boldsymbol{z}_k - \mu)^T$$

where $\boldsymbol{\mu}$ is the mean of the feature vector entries across all of the pixels. The distance metric between two covariance matrices can now be established through the following calculation:

$$\rho(\mathbf{C}_1, \mathbf{C}_2) = \sqrt{\sum_{i=1}^{n} \ln^2 \lambda_i(\mathbf{C}_1, \mathbf{C}_2)}$$

where $\{\lambda_i(\mathbf{C}_1, \mathbf{C}_2)\}_{i=1\ldots n}$ are the generalized eigenvalues of $\mathbf{C}_1$ and $\mathbf{C}_2$ [1]. All future references to the "distance" between images or the distance metric refer to the results of this calculation, $\rho(\mathbf{C}_1, \mathbf{C}_2)$.

4.2 Night Detection Considerations

At night the camera is extremely susceptible to jitter and lack of luminance. In order to obtain frames with enough "sharp"-ness, four sequential frames are averaged together.

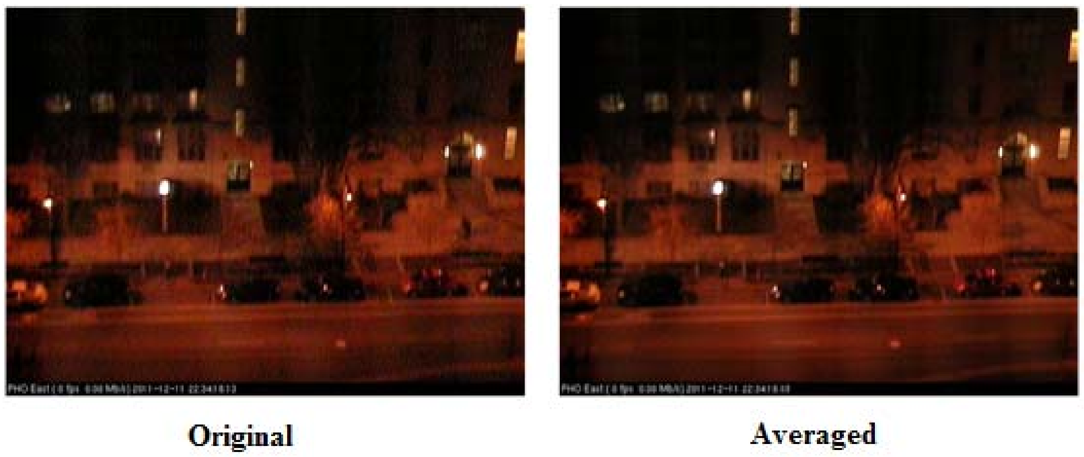

Figure 1: Example of the effect of frame averaging on nighttime frames

Since our distance is determined by the gradient (essentially edge detection), it is necessary to have significant contrast between the asphalt and the car. This can be a problem in the absence of light. Fortunately, any light that does fall on parking spaces is pretty constant as it generally due to street lamps.

In order to enhance the differences between asphalt and vehicles, contrast stretching is implemented. It was observed that vehicles generally have a darker luminance (15<L<30) than the asphalt (45<L<60). Using the contrast stretching formula below, a larger distance was observed between a vacant and occupied spot through preserving the darker luminance values pertaining to a car and stretching that of the asphalt.

$$v = \begin{cases} \alpha u, & 0 \leq u < a \\ \beta(u-a) + v_a, & a \leq u < b \\ \gamma(u-b) + v_b, & b \leq u < L \end{cases}, where\ a = 40, b = 100, \alpha = 1, \beta = 2$$

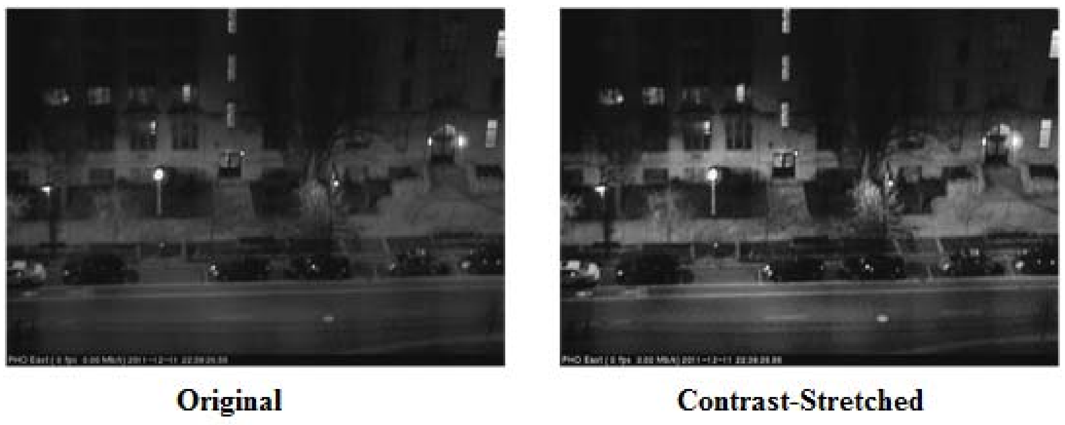

Figure 2: Example of the effect of contrast-stretching on nighttime frames

By building a dictionary of empty spots, an observed spot can be compared to each empty spot by the distance metric. From there, the distance can be compared to a threshold to create a decision. It is important to note that the frames in the dictionary of empty spots must also be contrast stretched in order for accurate detection.

4.3 Implementation

In order to establish the presence of cars or other obstructions in camera frames of parking spaces, we followed the approach shown below:

1. Obtain four sequential frames from the camera and average these frames together. This averaging helps suppress transient events in the daytime frames and helps to reduce noise in the nighttime frames.
2. Convert the averaged frame from the RGB color space into the YC_BC_R color space and extract only the Y component for further use. We ignore the chrominance values here because the chrominance images are very noisy and contain little descriptive information. Additionally, we are not concerned with the color of obstructions that may be present in the parking spaces.
3. If the sequence of frames were collected during the nighttime, perform contrast stretching on the averaged frame (as in section 4.2). Note that this step invokes the assumption that the time of day is known for each set of frames.
4. Break up the frame into sub-images representing each of the parking spots in the frame. This is done manually with the assumption that the parking spot locations are known beforehand.
5. Generate covariance matrices for each of the sub-images using the process from section 4.1.
6. Compute the "distance" between the covariance matrix for each sub-image and the covariance matrices of a training set consisting of a large number of empty spaces.
7. Take of the mean of the "distances" computed in step 6 for each sub-image and compare to a threshold value. If the "distance" is above the threshold, the space is then labeled as occupied, if it is below the threshold, the space is labeled as empty.

Our implementation was done entirely in MATLAB using built-in functions when available (source code available in Appendix 1). In order to determine the threshold used for the decision in step 7, we generated a histogram of all empty spot-to-empty spot and empty spot-to-occupied spot distances from our training set. We then hand-selected a threshold from this histogram in order to minimize the chances of getting a missed detection, as that is the most undesirable scenario for a driver that is searching for parking. More on this process including details of the training set and a sample histogram is available in section 5.

5 Experimental Results

For our experiment we used “Commonwealth Avenue parking meters” view from the PTZ2 camera at Boston University, mounted on the top of the Photonics Center building. A typical daytime scene from this camera with detection results can be seen in figure 3. Parking spaces that are detected as “occupied” are surrounded by a red bounding box, whereas those detected as “unoccupied” are surrounded by a green bounding box. This figure shows that the solution works for different-colored vehicles, an issue that is explored more in section 5.1.

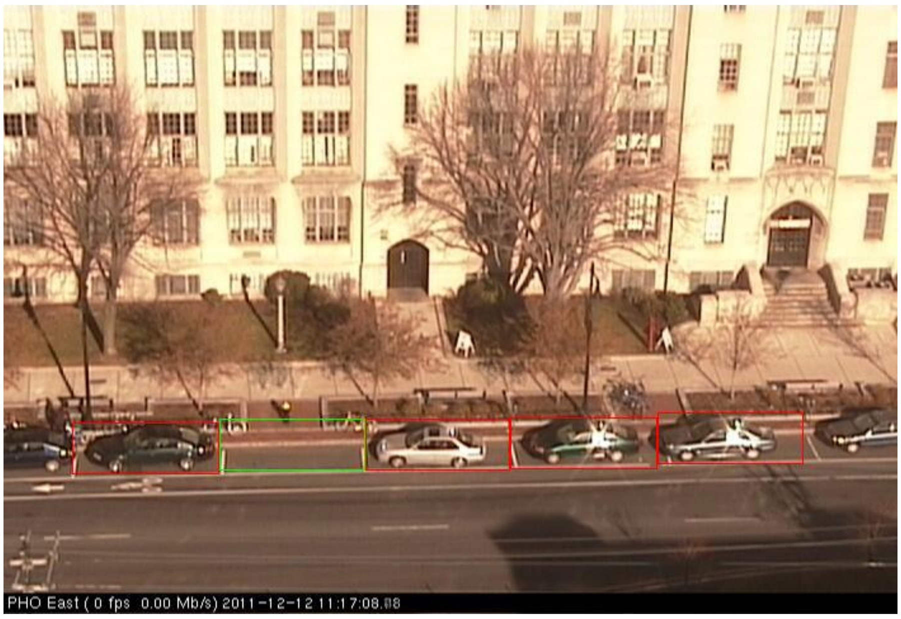

Figure 3: Example of daytime scene used in experiments

From these scenes we segmented out the five parking spaces shown manually and extracted the luminance images of each. An example luminance image for the center parking space is shown in figure 4.

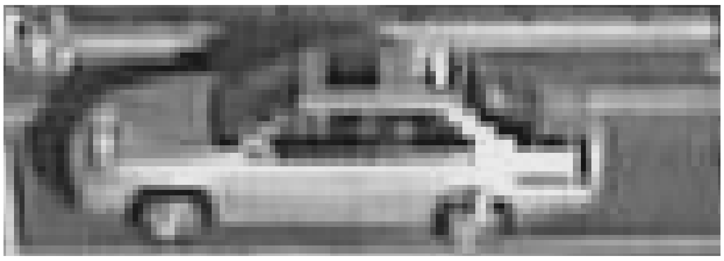

Figure 4: Example of daytime luminance image used for processing

5.1 Image Features

We used the horizontal and vertical image gradients from the Sobel kernel as well as pixel position information for our features. The gradients from the Sobel kernel were quite distinct for well illuminated scenes with obstructions that have clear changes in luminance across their surface (most vehicles). Examples of the gradient images for the vehicle shown in figure 4 are shown in figures 5 and 6 below.

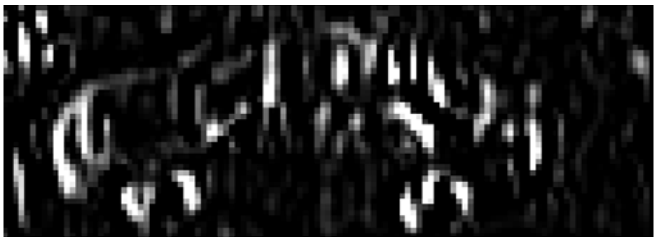

Figure 5: Example of horizontal gradient image

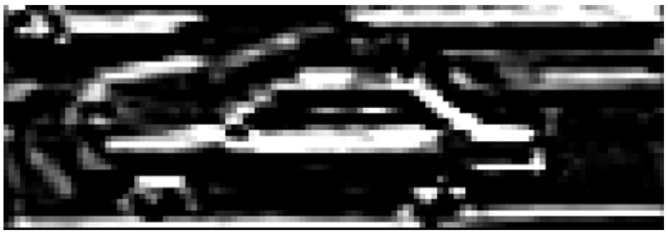

Figure 6: Example of vertical gradient image

As can be seen in these figures, the gradients from the obstruction are clearly distinguishable and separate them well from those of an empty parking space, shown below in figures 7 and 8.

Figure 7: Example of horizontal gradient image from an empty spot

Figure 8: Example of vertical gradient image from an empty spot

The use of gradients is not always the best choice, as occasionally obstructions will have very little change in luminance over their surfaces. This occurs most often in dark-colored cars, as the change in luminance between the paint and the glass of the windshield or windows becomes minimal. An example of this issue is shown in figures 9, 10, and 11.

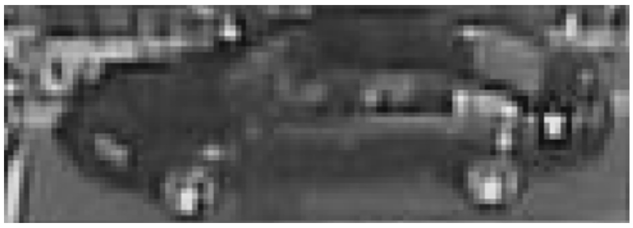

Figure 9: Luminance image from dark-colored vehicle

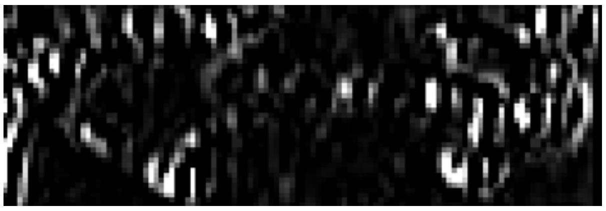

Figure 10: Horizontal gradient image from dark-colored vehicle

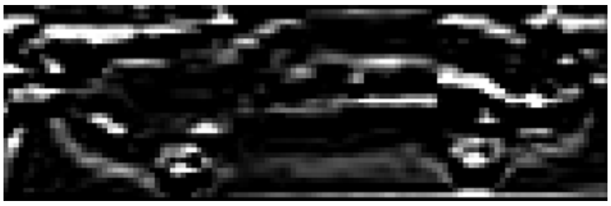

Figure 11: Vertical gradient image from dark-colored vehicle

We found that despite the limited amount of large-magnitude pixels in the gradient images for these dark-colored vehicles, the computed distance between these images and those for empty spaces was still, in general, sufficiently large.

5.2 Training Set and Threshold Determination

For our daytime training set we used a series of four-frame averaged images of 67 empty parking spaces and 268 occupied parking spaces. The occupied spaces contained some variety of obstructions which overwhelmingly consisted of sedan-style cars due to the particular set of parking spaces we were observing. We then computed the "distances" between the empty spots and themselves and then the occupied spots and the empty spots, and generated the histogram shown in figure 12.

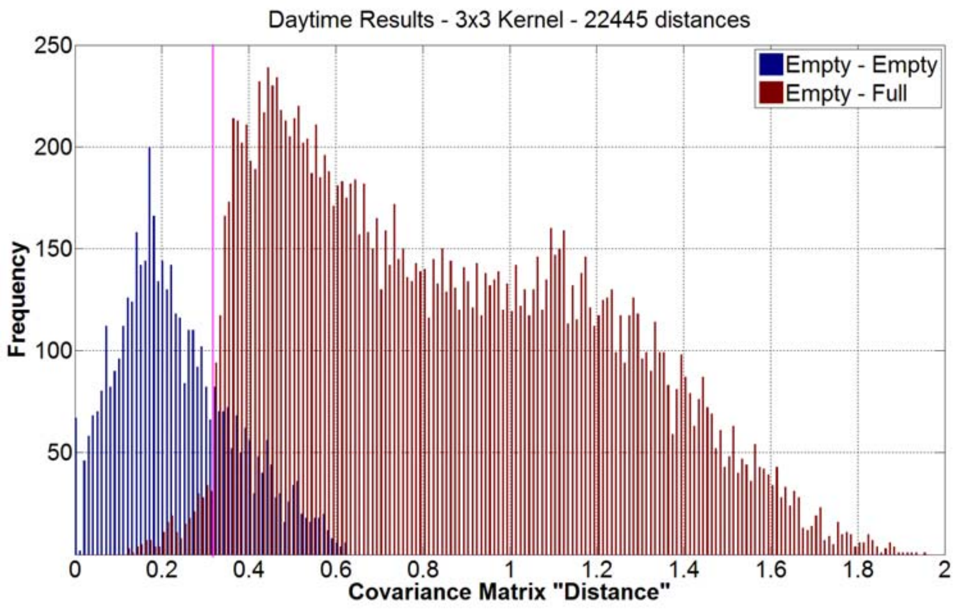

Figure 12: Histogram of distances used for computation of daytime threshold

From this histogram a threshold was selected manually in order to limit the chances of a missed detection while simultaneously minimizing false positives, and is indicated in the figure by a magenta line. This threshold for the daytime data ended up being a "distance" of 0.31. A similar process for threshold determination was used for the contrast-stretched nighttime images, resulting in a nighttime threshold of 0.70.

5.3 Day Results

Daytime results for a series of 67 four-frame averaged images are shown below in table 1. These images contained five parking spaces as can be seen in figure 3.

Parking Spot	Total Decisions	Correctly Detected	CD %	Missed Detection	MD %	Falsely Detected	FD %
1	67	60	89.55%	7	10.45%	0	0.00%
2	67	66	98.51%	0	0.00%	1	1.49%
3	67	66	98.51%	1	1.49%	0	0.00%
4	67	64	95.52%	3	4.48%	0	0.00%
5	67	64	95.52%	3	4.48%	0	0.00%

Table 1: Daytime results for 67 averaged images

From the table it can be seen that the system performed well in the daytime, with only one false detection and several missed detections. Some of these missed detections can be attributed to unrecoverable situations, such as the one presented in figure 13.

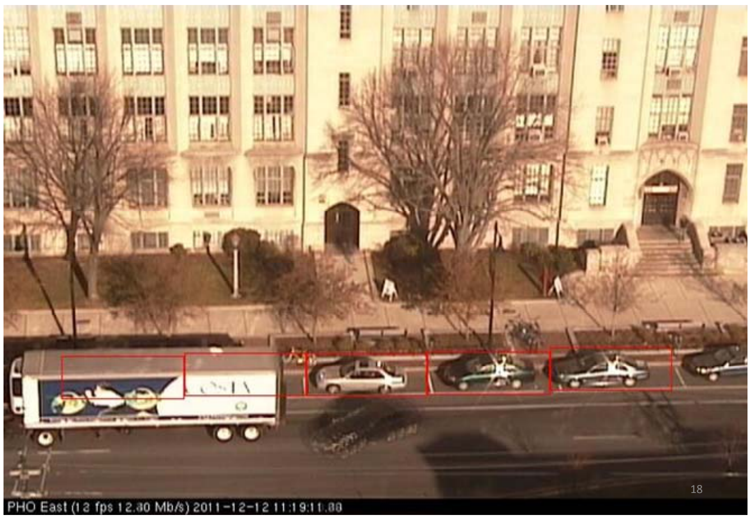

Figure 13: Unrecoverable situation causing missed and false detections

One persistent daytime problem involved the auto-gain control present on the PTZ2 camera used. This system was capable of generating large luminance shifts in the images when a shadow fell across the parking region, invalidating the static threshold value and causing detection failures. Figure 14 shows an example of a failure of this type, which we were unable to combat in our current implementation.

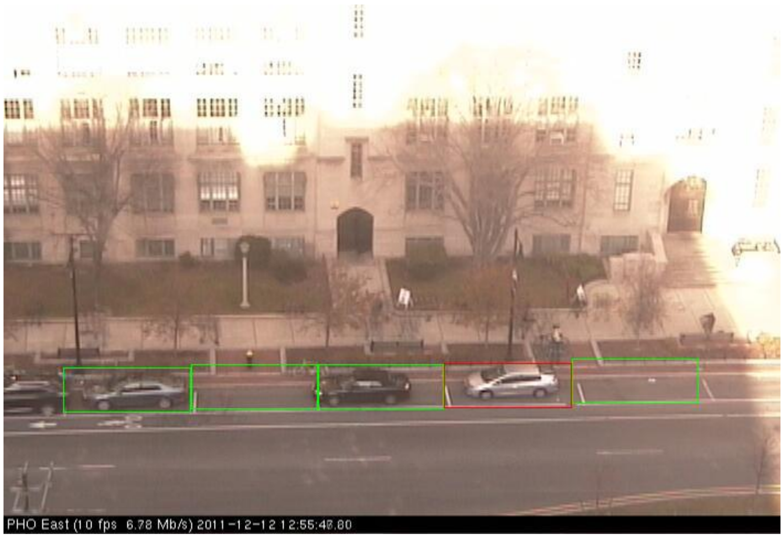

Figure 14: Camera automatic gain control causing failures

The frame-averaging process to suppress transients was, in general, successful, as can be seen from the rejection of a moving biker as an obstruction in figure 15. This process was also used to great effect in the nighttime results presented in the next section.

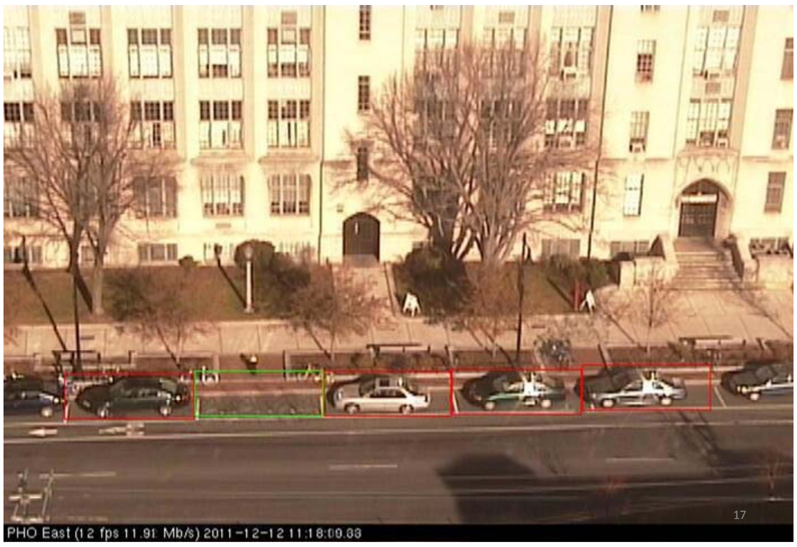

Figure 15: Rejection of transient event via averaging

5.4 Night Results

By running the distance metric on hundreds of night frames and creating a histogram of the distances between vacant-vacant and vacant-occupied, a decision rule was generated. For distances greater than 0.7, an occupied spot was detected, and for spots with distances less than 0.7, a vacant spot was detected.

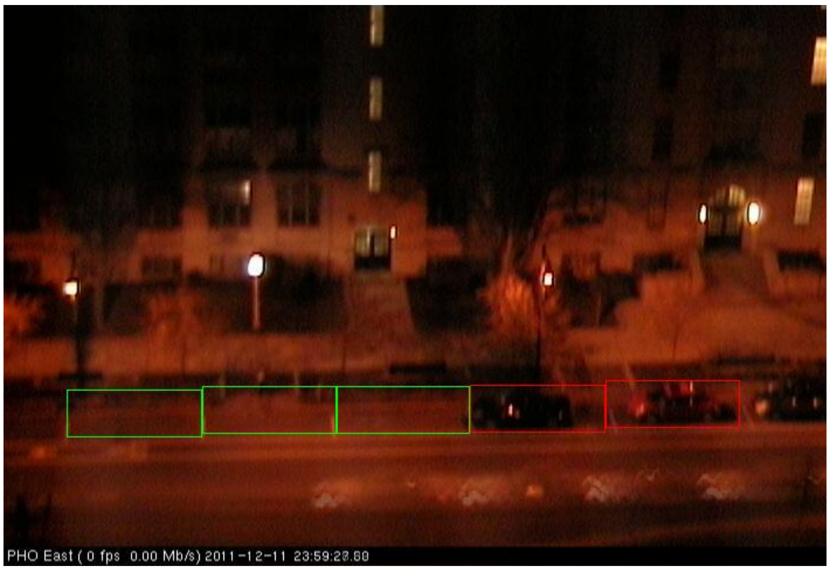

Figure 16: Night detection scene

In general this detection method worked quite well, as it correctly detected either occupied or vacant spots nearly 100% of the time. In the isolated false detection in parking spot 1, it appears that a person is walking through the spot at the time of the frame capture. While a car was not present and the spot would be vacant, a car is technically unable to park there as a person is standing there.

Additionally, the three missed detections in parking spot 4 were due to a darkly colored car that seemed to match the luminance values of the asphalt. As such, the distances between the observed frame and the dictionary of empty spots were smaller than expected leading to the missed detection. This problem could be solved by simply lowering the threshold value or modifying the contrast stretching formula. Results from 67 nighttime images are shown in table 2.

Parking Spot	Total Decisions	Correctly Detected	CD %	Missed Detection	MD %	Falsely Detected	FD %
1	**67**	**66**	**98.51%**	**0**	**0.00%**	**1**	**1.49%**
2	**67**	**66**	**98.51%**	**0**	**0.00%**	**1**	**1.49%**
3	**67**	**67**	**100.00%**	**0**	**0.00%**	**0**	**0.00%**
4	**67**	**64**	**95.52%**	**3**	**4.48%**	**0**	**0.00%**
5	**67**	**67**	**100.00%**	**0**	**0.00%**	**0**	**0.00%**

Table 2: Nighttime results for 67 averaged images

6 Conclusions

We have proposed and implemented a solution to the problem of detecting the occupancy of a parking spot from frames taken from a surveillance camera. The results show that region covariance is an effective tool to generate a distance metric between images of this kind, creating an efficient way to form detection decisions. In addition, we have shown that contrast stretching and averaging can help to enhance night-time images, reducing the effect of low luminance and jitter on the detection process.

There are, however, many ways to improve this solution so that it will better deal with the large amount of variability present in the outdoor scenes we observed. First, feature vectors with more sophistication could be generated to better distinguish similar scenes. Though we considered and rejected several additions to our feature vector structure, many other descriptors exist that may include useful information. Second, the Sobel operator used to find image gradients in this implementation may be non-ideal, as other kernels might better represent gradients commonly found in parking spot obstructions. Finally, the solution would benefit from a more complex dictionary to be used for daytime decisions that would be able to dynamically respond to changing lighting conditions and to changes imposed by the camera's automatic gain control system.

References

[1] O. Tuzel, F. Porikli, and P. Meer, "Region Covariance: A Fast Descriptor for Detection and Classification," *Lecture Notes in Computer Science*, vol. 3952, 2006.

[2] P.G. Michalopoulos, "Vehicle detection video through image processing: the Autoscope system," *Vehicular Technology, IEEE Transactions on*, vol. 40, no. 1, pp.21-29, Feb 1991.

[3] C. Papageorgiou and T. Poggio, “A Trainable System for Object Detection,” *International Journal of Computer Vision*, vol. 38, no. 1, pp.15-33, June 2000.

[4] A. Jazayeri, Hongyuan Cai, Jiang Yu Zheng, and M. Tuceryan, "Vehicle Detection and Tracking in Car Video Based on Motion Model," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 12, no. 2, pp.583-595, June 2011.

[5] R. Panayappan, J. M. Trivedi, A. Studer, and A. Perrig, “VANET-based approach for parking space availability,” *Proceedings of the fourth ACM international workshop on Vehicular ad hoc networks,* September 10, 2007, Montreal, Quebec, Canada

[6] N. True, "Vacant Parking Space Detection in Static Images," University of California, San Diego, La Jolla, CA

[7] H. Ichihashi, A. Notsu, K. Honda, T. Katada, and M. Fujiyoshi, "Vacant parking space detector for outdoor parking lot by using surveillance camera and FCM classifier," *Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on,* pp.127-134, August 20-24, 2009.

Appendix 1 – MATLAB Code

```
% Function to get "distance" metric between a dictionary empty parking spot
% and a query parking spot using the covariance matrix technique, as in:
%
% O. Tuzel, F. Porikli, and P. Meer, "Region Covariance: A Fast Descriptor
% for Detection and Classi?cation," in Computer Vision – ECCV 2006,
% ISBN: 978-3-540-33834-5.
%
%
% INPUTS:  dict,query:   equal sized images in double format, where:
%                           dict = dictionary image to compare against
%                           query = query image
%
% OUTPUT:  dist: "distance" metric between query and dictionary images
%
% Alex Macdonell and Jon Lobo
% EC520 Term Project

function dist = getQueryDist(dict,query)

% ---Uncomment for sobel edges (also change vector creation code below)----
% Perform sobel edge detection and convert result to a binary matrix where
% pixels on an edge are 1 and off an edge are 0.
%
% SOBEL_THRESH = 125;
% bQueryEdge = sobel(query, SOBEL_THRESH) == 255;
% bDictEdge = sobel(dict, SOBEL_THRESH) == 255;
%
-----------------------------------------------------------------------
--

% Calculate horizontal and vertical image gradients
GRADIENT_KERNEL = [-1 -2 -1; 0 0 0; 1 2 1];
% GRADIENT_KERNEL = [-1 0 1];
hGradDict = conv2(dict,GRADIENT_KERNEL, 'same');
vGradDict = conv2(dict,GRADIENT_KERNEL','same');
hGradQuery = conv2(query,GRADIENT_KERNEL, 'same');
vGradQuery = conv2(query,GRADIENT_KERNEL','same');

% Create sets of x and y positions for vectors
Xs = repmat((1:size(dict,2)),size(dict,1),1);
Ys = repmat((1:size(dict,1))',1,size(dict,2));

% Serialize positions and gradients and create vectors for covariance calc
zDict(:,:,1) = Xs(:);
zDict(:,:,2) = Ys(:);
% Using sobel edges:
% zDict(:,:,3) = bDictEdge(:);
% Using luminance only:
% zDict(:,:,3) = dict(:);
% Using gradients:
zDict(:,:,3) = hGradDict(:);
zDict(:,:,4) = vGradDict(:);
```

```
zQuery(:,:,1) = Xs(:);
zQuery(:,:,2) = Ys(:);
% Using sobel edges:
% zQuery(:,:,3) = bQueryEdge(:);
% Using luminance only:
% zQuery(:,:,3) = query(:);
% Using gradients:
zQuery(:,:,3) = hGradQuery(:);
zQuery(:,:,4) = vGradQuery(:);

% Calculate covariance matrices
CovQuery = cov(squeeze(zQuery));
CovDict = cov(squeeze(zDict));

% Calculate generalized eigenvectors
eigCov = eig(CovQuery,CovDict);

% Calculate the "distance" metric
dist = sqrt(sum(log(eigCov).^2));
```

```
% Grab frame from camera
% frame = imread('http://ptz2-iss.bu.edu/jpg/image.jpg','jpg');
frame = imread('..\Images\Day_Images\frames_color67.jpg','jpg');
% Convert image to YCbCr color space
frameYCbCr = rgb2ycbcr(frame);

% Separate individual parking spaces
% ps1 = double(frameYCbCr(300:340,33:130,1));
% ps2 = double(frameYCbCr(300:340,147:244,1));
% ps3 = double(frameYCbCr(300:340,261:358,1));
% ps4 = double(frameYCbCr(300:340,492:589,1));
% ps5 = double(frameYCbCr(300:340,607:704,1));

ps1 = double(frameYCbCr(330:370,55:170,1));
ps2 = double(frameYCbCr(327:367,170:285,1));
ps3 = double(frameYCbCr(327:367,284:399,1));
ps4 = double(frameYCbCr(326:366,400:515,1));
ps5 = double(frameYCbCr(322:362,515:630,1));

% Calculate "distance" metrices for other spaces
distps1 = getQueryDist(ps2,ps1)
distps3 = getQueryDist(ps2,ps3)
distps4 = getQueryDist(ps2,ps4)
distps5 = getQueryDist(ps2,ps5)
```

```
function decision = getNightDecision(query, dict)

% Visual Sensor for Smart Parking - EC520 Project
% Alex Macdonell and Jon Lobo
frame = imread(query);
path(path,'.\mmread');
```

```
alpha = 1;
beta = 2;
[r c d] = size(frame);
frameYCbCr = rgb2ycbcr(frame);
threshA = 40;
threshB = 100;
Va = threshA*alpha;
Vb = beta * (threshB  - threshA) + Va;
gamma = (255-Vb)/(255-threshB);
for a = 1:r
    for b = 1:c
        if frameYCbCr(a,b,1) < threshA
            frameYCbCr(a,b,1) = alpha * frameYCbCr(a,b,1);
        elseif ((frameYCbCr(a,b,1) <threshB ) && (frameYCbCr(a,b,1) >=
threshA ))
            frameYCbCr(a,b,1) = beta * (frameYCbCr(a,b,1) - threshA) + Va;
        elseif (frameYCbCr(a,b,1) >= threshB)
            frameYCbCr(a,b,1) = gamma * (frameYCbCr(a,b,1) - threshB) + Vb;
        end
    end
end

% Separate individual parking spaces
ps(:,:,1) = double(frameYCbCr(330:370,55:170,1));
ps(:,:,2) = double(frameYCbCr(327:367,170:285,1));
ps(:,:,3) = double(frameYCbCr(327:367,284:399,1));
ps(:,:,4) = double(frameYCbCr(326:366,400:515,1));
ps(:,:,5) = double(frameYCbCr(322:362,515:630,1));

dist = zeros(5,120);
avg_dist = zeros(1,5);
for i=1:5
   for j = 1:120
        fn = [dict '\empty' num2str(j) '.tif'];
        EmptySpot = double(imread(fn,'tif'));
        dist(i,j) = getQueryDist(EmptySpot,ps(:,:,i));
%         fprintf('%d,%d\n',i,j);
   end
   avg_dist(i) = mean(dist(i,:));
end

% [r c] = size(frameYCbCr(:,:,1));
% [r c d] = size(frame);
% % decision = zeros(480,704,3);
% decision(:,:,1) = frameYCbCr(:,:,1);
% decision(:,:,2) = frameYCbCr(:,:,1);
% decision(:,:,3) = frameYCbCr(:,:,1);
decision = frame;
for i = 1:5
    if avg_dist(i) <0.7 %vacant
        for j = 1:r
            for k = 1:c
                if (i == 1)
                    ylow = 330;
                    yhi = 370;
                    xlow = 55;
                    xhi = 170;
                    if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 0;
                    decision(j,k,2) = 255;
                    decision(j,k,3) = 0;
                    end
                elseif (i == 2)
                    ylow = 327;
                    yhi = 367;
                    xlow = 171;
                    xhi = 285;
                    if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 0;
                    decision(j,k,2) = 255;
                    decision(j,k,3) = 0;
                    end
                elseif (i == 3)
                    ylow = 327;
                    yhi = 367;
                    xlow = 286;
                    xhi = 399;
                    if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 0;
                    decision(j,k,2) = 255;
                    decision(j,k,3) = 0;
                    end
                elseif (i == 4)
                    ylow = 326;
                    yhi = 366;
                    xlow = 400;
                    xhi = 515;
                    if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 0;
                    decision(j,k,2) = 255;
                    decision(j,k,3) = 0;
                    end
                elseif (i == 5)
                    ylow = 322;
                    yhi = 362;
                    xlow = 516;
                    xhi = 630;
                    if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 0;
                    decision(j,k,2) = 255;
                    decision(j,k,3) = 0;
                    end
                end
            end
        end
    else
         for j = 1:r
            for k = 1:c
                if (i == 1)
                    ylow = 330;
                    yhi = 370;
                    xlow = 55;
                    xhi = 170;
                    if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                        decision(j,k,1) = 255;
                        decision(j,k,2) = 0;
                        decision(j,k,3) = 0;
                    end
                elseif (i == 2)
                    ylow = 327;
                    yhi = 367;
                    xlow = 171;
                    xhi = 285;
                    if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                        decision(j,k,1) = 255;
                        decision(j,k,2) = 0;
                        decision(j,k,3) = 0;
                    end
                elseif (i == 3)
                    ylow = 327;
                    yhi = 367;
                    xlow = 286;
                    xhi = 399;
                    if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                        decision(j,k,1) = 255;
                        decision(j,k,2) = 0;
                        decision(j,k,3) = 0;
                    end
                elseif (i == 4)
                    ylow = 326;
                    yhi = 366;
                    xlow = 400;
                    xhi = 515;
                    if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                        decision(j,k,1) = 255;
                        decision(j,k,2) = 0;
                        decision(j,k,3) = 0;
                    end
                elseif (i == 5)
                    ylow = 322;
                    yhi = 362;
                    xlow = 516;
```

```
                 xhi = 630;
                 if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                     decision(j,k,1) = 255;
                     decision(j,k,2) = 0;
                     decision(j,k,3) = 0;
                 end
             end
         end
      end
  end
end
decision = uint8(decision);
```

```
function [decision,master_dists,master_dist_count] =
getDayDecision(query, dict, master_dist_count, master_dists)

% Visual Sensor for Smart Parking - EC520 Project
% Alex Macdonell and Jon Lobo
frame = imread(query);
frameYCbCr = rgb2ycbcr(frame);
[r c d] = size(frame);

% Separate individual parking spaces
ps(:,:,1) = double(frameYCbCr(330:365,65:160,1));
ps(:,:,2) = double(frameYCbCr(327:362,180:275,1));
ps(:,:,3) = double(frameYCbCr(327:362,294:389,1));
ps(:,:,4) = double(frameYCbCr(326:361,410:505,1));
ps(:,:,5) = double(frameYCbCr(322:357,525:620,1));

for i=1:5
   for j = [1,3:44,46:69]
        fn = [dict,num2str(j-1),'.jpg'];
        EmptySpotIm = rgb2ycbcr(imread(fn,'jpeg'));
        EmptySpot = double(EmptySpotIm(327:362,180:275,1));
        dist(i,j) = getQueryDist(EmptySpot,ps(:,:,i));
        master_dists(master_dist_count,1) = dist(i,j);
        if(i == 2)
            master_dists(master_dist_count,2) = dist(i,j);
        else
            master_dists(master_dist_count,3) = dist(i,j);
        end
        master_dist_count = master_dist_count + 1;
   end
   avg_dist(i) = mean(dist(i,:));
end

decision = frame;
for i = 1:5
    if avg_dist(i) < 0.225 %vacant
        for j = 1:r
            for k = 1:c
                if (i == 1)
                ylow = 330;
                yhi = 370;
                xlow = 55;
                xhi = 170;
                if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 0;
                    decision(j,k,2) = 255;
                    decision(j,k,3) = 0;
                end
            elseif (i == 2)
                ylow = 327;
                yhi = 367;
                xlow = 171;
                xhi = 285;
                if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 0;
                    decision(j,k,2) = 255;
                    decision(j,k,3) = 0;
                end
            elseif (i == 3)
                ylow = 327;
                yhi = 367;
                xlow = 286;
                xhi = 399;
                if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 0;
                    decision(j,k,2) = 255;
                    decision(j,k,3) = 0;
                end
            elseif (i == 4)
                ylow = 326;
                yhi = 366;
                xlow = 400;
                xhi = 515;
                if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 0;
                    decision(j,k,2) = 255;
                    decision(j,k,3) = 0;
                end
            elseif (i == 5)
                ylow = 322;
                yhi = 362;
                xlow = 516;
                xhi = 630;
                if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 0;
                    decision(j,k,2) = 255;
                    decision(j,k,3) = 0;
                end
            end
        end
    end
else
    for j = 1:r
        for k = 1:c
            if (i == 1)
                ylow = 330;
                yhi = 370;
                xlow = 55;
                xhi = 170;
                if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 255;
                    decision(j,k,2) = 0;
                    decision(j,k,3) = 0;
                end
            elseif (i == 2)
                ylow = 327;
                yhi = 367;
                xlow = 171;
                xhi = 285;
                if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 255;
                    decision(j,k,2) = 0;
                    decision(j,k,3) = 0;
                end
            elseif (i == 3)
                ylow = 327;
                yhi = 367;
                xlow = 286;
                xhi = 399;
                if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 255;
                    decision(j,k,2) = 0;
                    decision(j,k,3) = 0;
                end
            elseif (i == 4)
                ylow = 326;
                yhi = 366;
                xlow = 400;
                xhi = 515;
                if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 255;
                    decision(j,k,2) = 0;
                    decision(j,k,3) = 0;
                end
            elseif (i == 5)
```

```
                ylow = 322;
                yhi = 362;
                xlow = 516;
                xhi = 630;
                if (((j == ylow) && (k<=xhi) && (k>=xlow)) || ((j == yhi)
&& (k<=xhi) && (k>=xlow)) || ((k == xhi) && (j<=yhi) && (j>=ylow)) || ((k
== xlow) && (j<=yhi) && (j>=ylow)))
                    decision(j,k,1) = 255;
                    decision(j,k,2) = 0;
                    decision(j,k,3) = 0;
                end
            end
        end
     end
   end
end
decision = uint8(decision);
```

```
% avg_build = zeros(67,5);
for i = 26:92
   fn = ['Empty Spots\frames_done_color' num2str(i) '.jpg'];
   decision= getNightDecision(fn,'StretchEmptySpots');
   disp(i)
   resultsfn = ['Results\frame' num2str(i) '.jpg'];
   imwrite(decision,resultsfn,'jpg');
end

%
% plot(26:92, avg_build);
% legend('Spot 1', 'Spot 2', 'Spot 3', 'Spot 4', 'Spot 5');
%
% hold on
% plot(26:92, 0.7*ones(1,67),'k-','LineWidth',3);


master_dist_count = 1;
master_dists = [0,0,0];

for i = [45,70,71]%[1,3:44,46:69]
   fn = ['..\Bad_Day_Ims\frames_done_color',num2str(i-1),'.jpg'];
   dict = '..\Day_Images_avg\frames_done_color';

[decision,master_dists,master_dist_count]=getDayDecision(fn,dict,master
_dist_count,master_dists);
   disp(i-1)
   resultsfn = ['..\DayResults_avg\' num2str(i-1) '.jpg'];
   imwrite(decision,resultsfn,'jpg');
end
```