VISUAL SENSOR FOR SMART PARKING

Luis Caro Campos, Gregary B. Prince

12/15/2011

Boston University

Department of Electrical and Computer Engineering

Technical Report No. ECE-2011-02

BOSTON UNIVERSITY

VISUAL SENSOR FOR SMART PARKING

Luis Caro Campos, Gregary B. Prince

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

12/15/2011

Technical Report No. ECE-2011-02

Summary

This report describes results of our investigation of an image processing algorithm for the detection of on-street parking vacancies completed within a fall 2011 graduate course "Digital Image Processing and Communication" (EC520). Our project fits in a broader activity on smart parking currently conducted at Boston University. The system we investigated starts by acquiring images from a rooftop camera facing occupied and vacant parking spots along Commonwealth Avenue on BU campus. After acquiring the images, a k-NN classification algorithm is applied to region covariance matrices of features in order to classify potential vacant and occupied parking spaces. The system model is based on a feature vector containing spatial coordinates, color chromas, and spatial change of illumination for each region of interest. A GUI was developed to facilitate system training and validation. Two methods of validation were applied. One involved a leave-part-out cross-validation (LPOCV) approach which was repeated ten times (10-fold cross-validation), the results of which were between 90% and 97% successful classification rate. The other validation was performed in real time on newly-acquired images with results ranging between 85.5-85.75% for pre-sunrise conditions and 88.0-89.5% for post-sunrise conditions. In conclusion, the proposed system performs fairly well but may be able to be improved by incorporating different features into the system model or applying a more mature classification method like a support vector machine or neural network.

Contents

List of Figures

List of Tables

1 Introduction

In certain circumstances, finding a vacant parking space becomes a time consuming task for drivers that can often arise in traffic congestion. This situation can become problematic in well-traveled areas, especially at peak traffic intervals. For this reason, it is useful to be able to inform drivers of availability of vacant parking spaces in real time, or as close to real time as possible. In some scenarios such as indoor parking lots, human operators have to monitor several cameras at the same time to perform this task, which can become expensive and laborious. An unsupervised parking system should be able to provide the number and location of vacant parking spaces, by monitoring changes in occupancy [WuX06].

In recent years, researches have proposed different solutions to the problem. According to the authors in [ITR09] we can classify smart parking systems into two categories, depending on the nature of the sensors: intrusive and non-intrusive. Intrusive sensors typically require invasive installation procedures on each parking space. In some cases to reduce costs, sensors are installed at the entry and exit ways of parking lots, which does not enable locating vacant spots, yet provides a simple counter relative to total parking spots available. In the second category, we can include systems that analyze the video signal from a camera pointed at several parking spaces.

A visual surveillance solution to the problem requires real-time interpretation of the image sequence captured by the camera in order to detect vacant parking spaces. This poses several challenges, as the camera cannot be mounted perpendicularly to the parking slots. Illumination changes, shadows and occlusions have to be taken into account to perform an accurate detection.

2 Related work

In this section, we briefly describe some of the prior approaches to the classification of vacant parking spaces problem.

2.1 Vacant Ground Color Model using PCA

In the system proposed by [WuX06], a Gaussian model of the ground (vacant space) color is employed. For each analyzed region, the probability of every pixel of belonging to the model is computed. Principal component analysis (PCA) is then employed to reduce the dimensionality of the obtained vector. In order to provide the system with robustness against occlusions, shadows, and illumination changes, each parking row is projected onto a ground plane and regions of 3 contiguous parking spaces (patches) are analyzed at the same time. SVM classification is used to classify each vector into 8 possible occupation states. A Markov random field framework is described to solve classification discrepancies.

2.2 Color Histogram Method using Corner Detection

In [Tru07], two different kinds of features are extracted. In an initial step, parking spaces are manually labeled, and the color space is converted to $L * a * b$, as it is invariant to chances in lighting conditions. Color histograms are extracted in each labeled region, and classified using k-NN or SVM algorithms. Additionally, the author proposes detection by extracting car features inside the analyzed parking space. For this purpose, Harris corner detection is employed to extract feature points. A feature dictionary is created from images centered at each feature points. For new images under analysis, extracted features are compared against the training set using Normalized Cross-correlation. As future work, the author proposes a combination of the two approaches to further reduce the misclassification rates.

3 Proposed approach

3.1 Camera setup and training data acquisition

For our project, we will collect image snapshots from a PTZ (pan-tilt-zoom) camera pointed at Commonwealth Avenue. There are PTZ cameras mounted on the roof of PHO; these cameras are operated by the ISS lab. There are many cameras available ($ptz1, ptz2, vsn1, vsn2, vsn3, vsn4, vsn5$, and $vsn6$) to be used to obtain the training set and the query images to be classified. After investigating the quality of images acquired, it was determined that $ptz1$ and $ptz2$ were the only two useful cameras for the acquisition of images to be analyzed. At times (depending on the angle and outdoor lighting conditions), both cameras appear overexposed, as a result of the automatic exposure control. Cameras are also subject to vibration noise, caused by the HVAC on the roof. This is particularly noticeable at close zoom levels.

In an initial step, still images are acquired from the camera pointing at different locations. In order to cover a wider array of situations, different angles, distances and zoom levels will be covered which result in different amount of occlusions, at different times of day to account for changes in lighting conditions. The parking spaces will be manually identified for each camera preset, and the spaces will be labeled as vacant or occupied in each snapshot.

To gather snapshot information, we query the current video frame from the cameras at a specified rate. The camera server provides a URL to obtain the most recent frame in compressed form. The following bash script is used to gather the desired data:

```
#!/bin/bash

URL=http://ptz1-iss.bu.edu/axis-cgi/jpg/image.cgi #image URL
CAM=1 #camera number
LOC=4 #camera preset (location)
```

```
i=1 #start number

while sleep 60; do

printf -v FNAME "cam%02d_loc%02d_%03d.jpg" $CAM $LOC $i
curl $URL -o $FNAME
let "i+=1"
echo $FNAME

done
```

The same operation could be performed by using a Matlab central function to handle video streams(*mmread.m*), but it proved inefficient in terms of CPU resources.

This operation provides the capability to acquire all the images required for training and testing. Acquiring images at different time of the day (over different days) and at different viewing angles at a rate of a frame every minute provides diversity in the training set for feature extraction and classification.

Figure 1: Camera 1 Aimed at Location 1 on Commonwealth Avenue

3.2 Feature extraction

In [TPM06], the use of covariance matrices as region descriptors is proposed. The covariance matrix provides a way of fusing multiple features that might be correlated. For each pixel inside a rectangular image region R, the k^{th} sample feature of the j^{th} training set is given by the length M vector $\xi_{j,k}$. We can represent the region R with the $M \times M$ covariance matrix of feature points for the j^{th} training set as $\mathbf{C_j}$, with its associated mean vector $\mu_{\xi_\mathbf{j}}$:

$$\mathbf{C_j} = \frac{1}{n-1}\sum_{k=1}^{n}(\xi_{j,k} - \mu_{\xi_\mathbf{j}})(\xi_{j,k} - \mu_{\xi_\mathbf{j}})^T$$

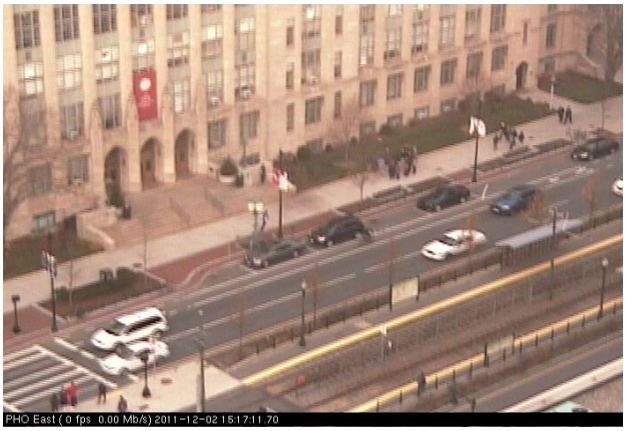

Figure 2: Camera 2 Aimed at Location 2 on Commonwealth Avenue

Figure 3: Camera 2 Aimed at Location 3 on Commonwealth Avenue

$$\mu_{\mathbf{j}} = \frac{1}{n}\sum_{k=1}^{n} \xi_{j,k}$$

The covariance matrix of feature points can incorporate many features of images, such as various color representations, position, luminance, etc. The chief idea is that given the features of interest over a certain region of the image can be compactly represented and the covariance of the region of interest can be compared to another region's covariance. The similarity of the two, in loose terms, can be heuristically thought of as a *match* between the two sub-images. Therefore, if each sub-image has similar covariance matrices, one may conclude that a parking spot is vacant or occupied. One of the main issues with this approach, however, is the fact that covariance matrices are, by definition, positive semi-definite quantities; this property hinders the ability to define a *metric* between any two covariance matrices. This metric is required if we hope to determine the *similarity* between two region covariance matrices.

A metric d is valid if and only if it meets the 3 (three) properties of all distance

metrics:

- $d(\mathbf{C_i}, \mathbf{C_j}) \geq 0 \ \forall \ i, j \in R$, and $d(\mathbf{C_i}, \mathbf{C_j}) = 0$ only if $(\mathbf{C_i} = \mathbf{C_j})$
- $d(\mathbf{C_i}, \mathbf{C_j}) = d(\mathbf{C_j}, \mathbf{C_i}) \ \forall \ i, j \in R$
- $d(\mathbf{C_i}, \mathbf{C_j}) + d(\mathbf{C_i}, \mathbf{C_k}) \geq d(\mathbf{C_j}, \mathbf{C_k}) \ \forall \ i, j, k \in R$

In [FoM99], the following metric, which exploits the concept of generalized eigenvalues and eigenvectors, for covariance matrices is proposed:

$$d(\mathbf{C}_i, \mathbf{C}_j) = \sqrt{\sum_{m=1}^{n} \ln^2 \lambda_m(\mathbf{C}_i, \mathbf{C}_j)}$$

where $\mathbf{C}_i$ and $\mathbf{C}_j$ are two covariance matrices to be compared, in our context from the i^{th} acquired image and j^{th} trained image. From an information theoretic point of view, the information of a Gaussian random variable scales with the natural logarithm of the variance $\ln \sigma^2$. In [FoM99], their assumption is that for non-Gaussian sources, $d^2 = \sum_m \ln^2 \lambda_m$. The $\lambda_m(\mathbf{C}_i, \mathbf{C}_j)$ are the m^{th} sets' generalized eigenvalues computed between the covariance matrices, which are found by solving the following eigenvalue problem, with eigenvectors $\mathbf{x}_i$:

$$\lambda_m \mathbf{C}_i \mathbf{x}_m = \mathbf{C}_j \mathbf{x}_m$$

Lastly, to obtain d, the $\sqrt{(.)}$ operator is applied to the squared metric. One idea is to define the feature vector as follows:

$$\xi = \left(x, y, Cb(x,y), Cr(x,y), \frac{\partial L(x,y)}{\partial x}, \frac{\partial L(x,y)}{\partial y} \right)^T$$

The feature variables x,y are the coordinates of the pixel within the region, R; $C_b(x,y)$ and $C_r(x,y)$ are the color values (chromas) in the YC_bC_r color space, which are independent of illumination level; and , $\frac{\partial Y}{\partial x}$ and $\frac{\partial Y}{\partial y}$ are the first order derivatives of the image luminance intensities with respect to x and y. The motivation behind this attribute selection is to obtain a covariance matrix that can efficiently discriminate a parking space between vacant and occupied. We assume that the color of most cars differs from that of the pavement, and the YC_bC_r color space is used to make the system more robust against illumination changes. Additionally, the first order derivatives provide information about the edges in the image. When a car is parked, we expect car edges to be present, as opposed to smoother, or even the absence of, edges when the space is vacant.

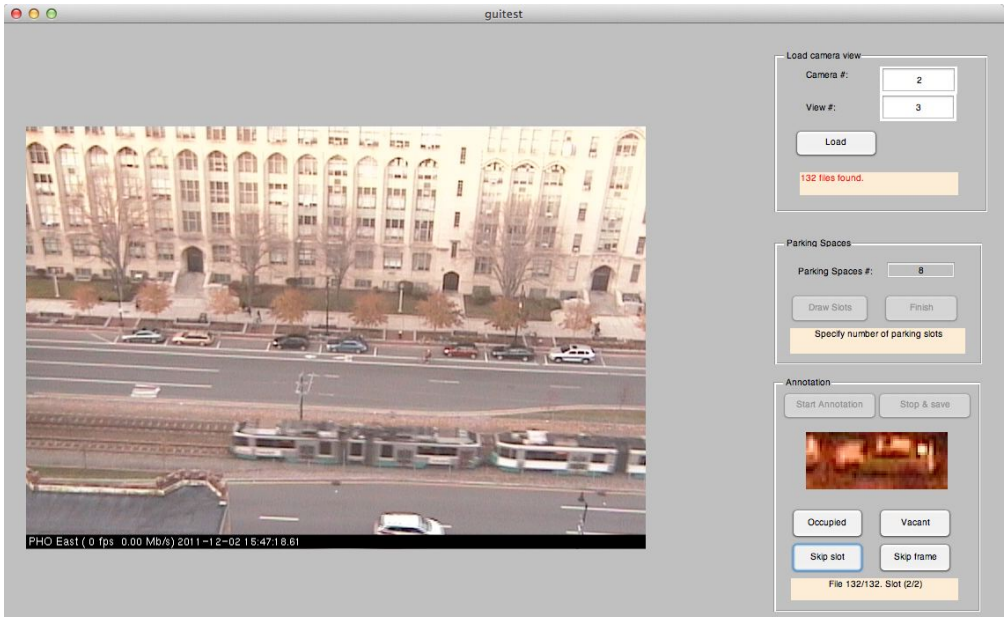

Figure 4: System GUI which Facilitates Training and Feature Extraction

3.3 Training

Having implemented the defined feature vector ξ, distance metric based on generalized eigenvalues, $d(C_i, C_j)$ and modifying a k-nearest neighbor $(k - NN)$ classifier, the next task is to efficiently gather training and query data to provide the classifier with enough general knowledge to make it effective.

Our training approach involved acquiring images (at 1 minute intervals) of parking spaces along Commonwealth Avenue at 7 different PTZ configurations; mainly within the 3 pm-5 pm time of day (over multiple days), where the lighting variations are maximum. Furthermore, acquiring images at different angles and zoom levels provides the classifier with many different perspectives of what a *car* looks like in its feature space. It should be noted that many acquired images were discarded due to the compound effect of full parking spaces being occluded at more extreme angles and camera vibration levels were too extreme to efficiently extract useful parking space features.

In total, 436 snapshots were retrieved from both cameras, with the total number of parking spaces in each view varying from 4 (close zoom) to 10 (distant zoom), resulting in 2198 labeled parking spaces (a pair of covariance matrix and label). For this purpose, we developed graphical user interface (GUI) to facilitate this process. This interface allows one to load a particular camera/view combination. Upon successful loading all the snapshots acquired from a given camera/view combination the user has the choice to dictate the number of parking spaces of interest, n. The user can then outline by creating rectangular boxes overlaid on the loaded acquired image. Once the user has identified and drawn out all n parking spaces, the GUI proceeds to the training phase. In this phase all mn parking spaces are identified and the user labels

each space as occupied or vacant. For each of these mn parking space images, its covariance matrix is computed and stored along with its label. The training dataset to be used by the classifier consists of over two thousand 2198 computed covariance matrices with associated binary labels.

Herein some of the concerns regarding the information used to train the classifier are documented, along with the rationale to proceed with including them.

- Firstly, for most camera views, the parking situation does not exhibit significant changes in many different frames in the dataset. The consequence being that we may be adding the same car or vacant space to the dataset and accumulating some sort of bias in the classifier. It was chosen to include all of these repetitions in the dataset to account for common frame-to-frame variations: camera sensor noise, motion blur (induced by vibration), and lighting.

- Secondly, approximately a quarter of all acquired images are, in fact, of vacant spaces. In most cases, the parking spaces are occupied. Again, the dataset was chosen to train the classifier due to the variations in the acquisition process, which allow the two sets to be sufficiently coherent but simultaneously different from each other.

3.4 Classification

Once the training phase is complete, the algorithm proceeds to attempt to classify new features against the training set. The task of classification involves comparing a collection of training datasets (with stored *features* and *labels*) to a newly encountered dataset (with its own *features*) and assign the new dataset a classification label. In this context, as mentioned previously, the labels are binary (e.g. occupied or vacant). The algorithm chosen to perform the classification in this investigation is the k Nearest Neighbors (k-NN) algorithm,. k is a user specified constant which determines how many training samples (k) are required to be closest, in terms of a defined metric, to the new feature for querying. Often times metrics such as *Hamming* and *Euclidean* are applied to feature vectors, but for this investigation the aforementioned covariance metric is employed to determine the k nearest neighbors. One problem that often occurs is when the training phase is biased towards one decision, a simple fix to this issue is to apply weights the metric obtained. Furthermore it is common practice to have k be an odd integer, as this avoids ties, simplifying the problem to a simple majority vote among the$k$ closest samples. In our experiments, we set $k = 3$.

A simple example of a k-NN classifier output is depicted in Figure 5, where $k = 10$. This example is trying to minimize the metric (Euclidean in this case) between the point of interest(the origin) with the observations within the space. As one may see, the 10 closest points to the origin are returned (interior of red curve) and the remaining data points are discarded (exterior of red curve).

This classification task is performed on each new dataset in the covariance matrix feature space with the appropriate modifications implemented. Nonetheless, the

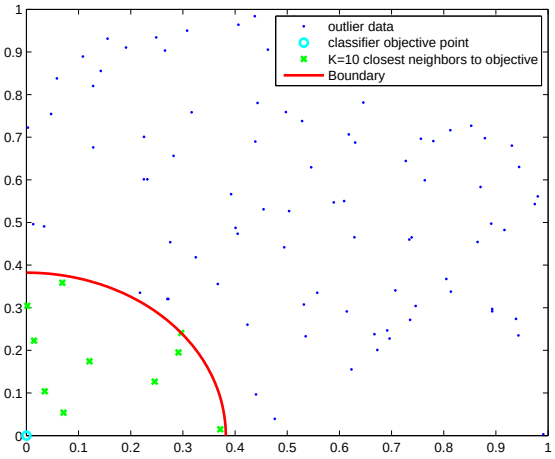

Figure 5: Simple Illustration of the k-NN Classifier Concept

simple idea of selecting the k closest neighbors, in terms of the defined metric, to the dataset of interest still holds.

Figure 6 illustrates the summary flow of the algorithm in both the training and classification phases.

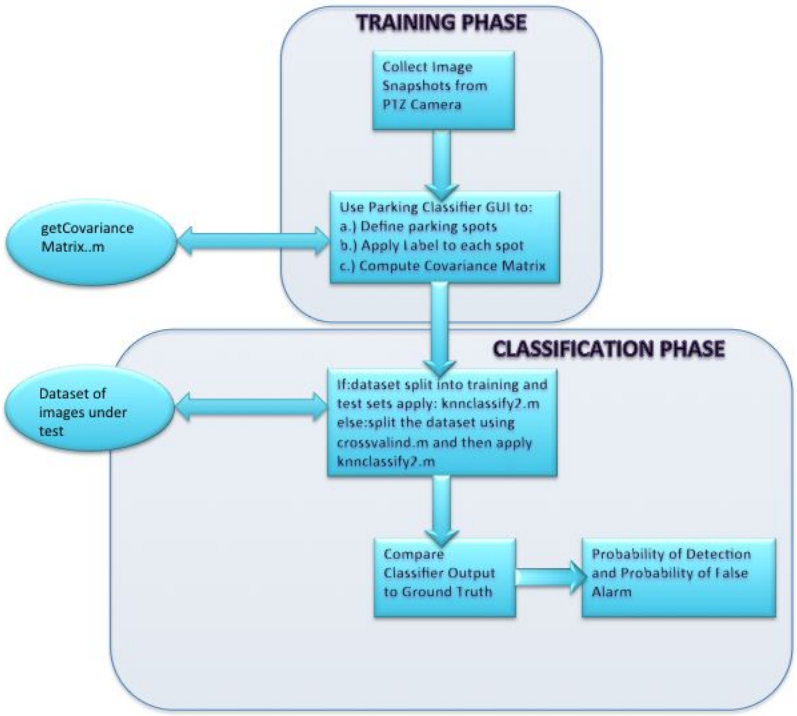

Figure 6: Smart Parking System Algorithm

3.5 Validation

To validate the proposed method, the classification will be tested on new (unseen) snapshots of the same parking spaces used for the training phase, and the performance of the classifier will be assessed in terms of predictive accuracy (percentage of correctly classified samples). There are two (2) validation approaches performed in this investigation, one in which cross validation is performed using leave-part-out-cross-validation (LPOCV) and one involving real-time querying with entirely new (unseen) data.

For cross-validation, LPOCV is repeated exactly 10 times as recommended in [RTL09] using the entire training dataset since it is sufficiently large. Each time, the dataset is broken into two groups: 90% for training, and 10% for querying. This is done randomly (native Matlab function *crossvalind* provides the random partitions), thereby preserving the proportions of occupied/vacant spaces in both of the new datasets. This provides mean measure of the accuracy. Note that each sample from the original dataset is used both for training and querying, but never at the same time. Most of these measures fell between 90 and 97% classification success, with a mean accurace of 94.3%. A set of Matlab commands is provided to illustrate this LPOCV operation.

```
[trainset, queryset] = crossvalind('Resubstitution', N, [P,Q])
% split dataset into two groups, Train and Query
% can also use, depending on the type of validation,
% 'HoldOut', 'LeaveMOut','Kfold', etc. Method types
[outputclass] = knnclassify2(trainset, queryset, k)
 %provides the output classification between the two datasets
groundtruth = cell2mat(queryset(:,2));
% create ground truth matrix from the queryset
cm = confusionmat(obtainedclasses, groundtruth)
% inspect the results compute the confusion matrix
acc = sum(diag(cm)) / sum(cm(:));
%accuracy is the sum of the diagonal against the total
%number of query elements.
```

For real time validation it will be more difficult to compute accuracy figures unless we are able to produce 'ground-truth' for the processed frames. In a real-time scenario where several parking spaces have to be taken into account, it may be impracticable to perform detection in every frame, especially if the k-NN training set is too big. A more realistic assumption would be to query each location every minute, which, depending on the number of locations and the size of the training set, may be realizable. In this scenario, performance metrics are only possible of the ground-truth if each analyzed region is manually input.

Figure 8 illustrates the *live* or *real time* version of the GUI. The real time GUI performs similarly to the training GUI in so far as it fetches image snapshots (only this

time in real time, not from a file server) from the specified camera (e.g. *ptz1* or *ptz2*. Again, the user can draw out what are thought to be valid parking spaces (regions) up to the amount specified. Once satisfied with the defined regions, the GUI (upon pressing start) will classify parking spots as occupied (red outlined region) or vacant (green outlined region). Moreover, the system can continue to fetch new snapshots from the camera(s) and classify accordingly.

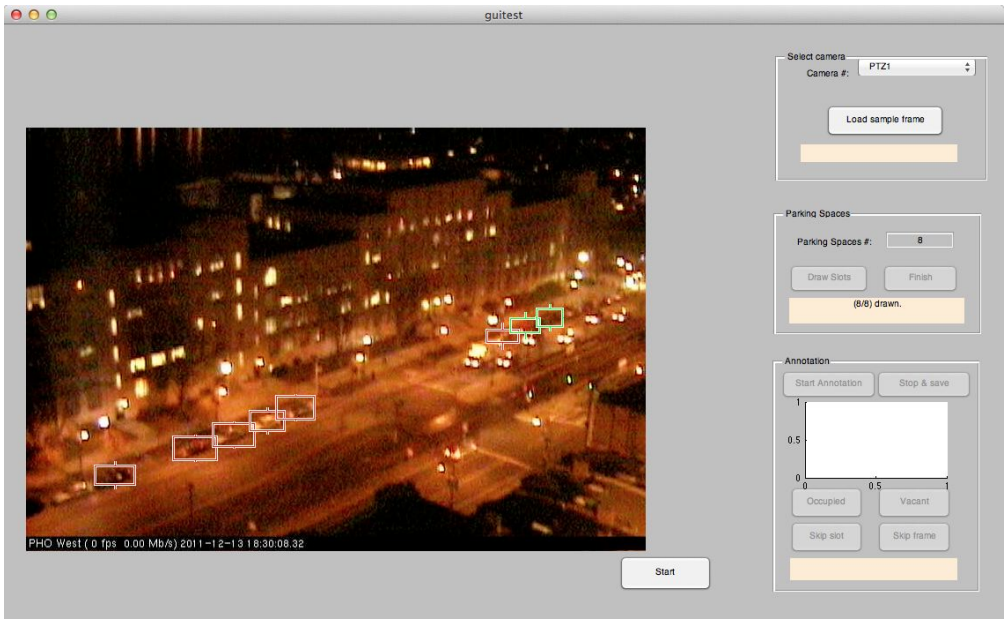

Figure 7: System GUI for Real Time Validation

4 Results and conclusions

The k-NN classification algorithm operating on region covariance matrices yields very good performance under the environments tested. As discussed in the validation section, the performance was gauged using both a LPOCV method and a real-time method. The real-time system performance was tested in both high illumination (HI) levels (mid morning) and low illumination (LI) levels (early morning) at different camera angles.

The LPOCV results yielded repeatable performance in the range of 90-97% correct classification. This may be due to the heterogeneity of the data acquired. Sometimes, especially for zoomed-out scenes, the acquired resolution is too low. Furthermore, if vibration on the roof is active , the images are blurred and the lighting between *ptz1* and *ptz2* is different; *Ptz1* appears overexposed, while *Ptz2* provides crisper images. Lastly, in some image acquisitions, there are double parked cars or trucks in front of the regions of interest. All of these observations were included in the training dataset, which may have skewed the LPOCV results.

The real time performance involves human intervention to observe what the classifier is deeming as occupied or vacant and comparing those results with the ground-truth that the human observes in the picture. The results collected for the real time performance are averaged over a sequence of fifty (50) image snapshots for four (4) different scenarios : *ptz1* at HI, *ptz1* at LI, *ptz2* at HI, *ptz2* at LI. Table 1 summarizes the performance of these experiments. Each scenario measures its average and standard deviation with respect to the number of parking spaces defined (8 spaces in this run).

Table 1: Classification Results

Measure	PTZ#1 LI	PTZ#1 HI	PTZ#2 LI	PTZ#2 HI
% Success Average	85.50%	88.00%	85.75%	89.50%
% Standard Deviation	10.42%	10.09%	13.45%	9.23%

One may observe that the performance is best in HI environments with *ptz2* performing slightly better possibly due to the orientation of camera allowing parking spaces to match the rectangular region mapping better than the angular (parallelogram) view of *ptz1*. The same conclusion follows in the LI cases; however both camera report poorer classification results possibly due to not having as much information in the feature space in terms of chromas and spatial change of luminance during this time of day..

Overall, the classification system performed very well, but there are many avenues one may explore in attempt to improve its performance. The application of a more robust classification algorithm e.g. Support Vector Machine (SVM) or Neural Network (NN), could improve the success rate of accurately classifying vacant parking spaces. Moreover, modifying the feature vector to include more descriptive features relevant to rigid objects or even implementing a corner detection algorithm could improve upon the results obtained. Moreover, the GUI can be modified to allow for various polygons (e.g. parallelograms) depending on the perspective of the camera; A car is not always rectangular from all perspectives.

5 Appendix: MATLAB Code

```
 function varargout = guitest(varargin)
% GUITEST MATLAB code for guitest.fig
%      GUITEST, by itself, creates a new GUITEST or raises the existing
%      singleton*.
%
%      H = GUITEST returns the handle to a new GUITEST or the handle to
%      the existing singleton*.
%
%      GUITEST('CALLBACK',hObject,eventData,handles,...) calls the local
```

```
%      function named CALLBACK in GUITEST.M with the given input arguments.
%
%      GUITEST('Property','Value',...) creates a new GUITEST or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before guitest_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to guitest_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help guitest

% Last Modified by GUIDE v2.5 09-Dec-2011 18:25:59

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @guitest_OpeningFcn, ...
                   'gui_OutputFcn',  @guitest_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before guitest is made visible.
function guitest_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to guitest (see VARARGIN)

% Choose default command line output for guitest
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guitest wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% store window handle on main app
setappdata(0, 'hMainGui', gcf);


% --- Outputs from this function are returned to the command line.
function varargout = guitest_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function cam_Callback(hObject, eventdata, handles)
% hObject    handle to cam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cam as text
%        str2double(get(hObject,'String')) returns contents of
%  cam as a double


% --- Executes during object creation, after setting all properties.
function cam_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
%called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function loc_Callback(hObject, eventdata, handles)
% hObject    handle to loc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of loc as text
%        str2double(get(hObject,'String')) returns contents of loc as a double


% --- Executes during object creation, after setting all properties.
function loc_CreateFcn(hObject, eventdata, handles)
% hObject    handle to loc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
 get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in load.
function load_Callback(hObject, eventdata, handles)
% hObject    handle to load (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%get input numbers
cam=str2double(get(handles.cam,'String'));
loc=str2double(get(handles.loc,'String'));
%filename prefix
```

```
prefix=sprintf('cam%02d_loc%02d_',cam,loc);
%get all files
F=dir([prefix '*.jpg']);
%store this in appdata
h=getappdata(0, 'hMainGui');
setappdata(h, 'Files', F);
setappdata(h, 'currentPic', 1);



numfiles=size(F,1);


set(handles.message, 'String', [num2str(numfiles) ' files found.']);

if (numfiles > 0)
    %load and display first image, save in appdata
    im=imread(F(1).name);
    h=size(im,1);
    w=size(im,2);

    %this will display the image in its size but wont resize the window
    %pos=get(handles.image, 'Position');
    %npos = [pos(1) pos(2) w h];
    %set(handles.image, 'Position', npos);

    h=getappdata(0, 'hMainGui');
    setappdata(h, 'image', im);


    axes(handles.image);
    imshow(im, 'InitialMagnification', 100);

    %change message 2

    set(handles.message2, 'String', 'Specify number of parking slots');
end
```

```
function spacesnr_Callback(hObject, eventdata, handles)
% hObject    handle to spacesnr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of spacesnr as text
%        str2double(get(hObject,'String')) returns contents of
% spacesnr as a double


% --- Executes during object creation, after setting all properties.
function spacesnr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to spacesnr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
 get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in drawButton.
function drawButton_Callback(hObject, eventdata, handles)
% hObject    handle to drawButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
h=getappdata(0, 'hMainGui');
F=getappdata(h, 'Files');
numfiles = size(F,1);

if (numfiles == 0)
    set(handles.message2, 'String', 'No files loaded yet');
    return;
end

%check number of parking spaces
ns=str2double(get(handles.spacesnr,'String'));
if (ns < 1)
    set(handles.message2, 'String', 'Specify number of spaces');
    return;
```

```
end

%at this point, an image was loaded and the number of slots is positive
totalslots = ns;
drawnslots = 0;
slotmap = zeros(totalslots, 4);

setappdata(h, 'totalslots', totalslots);
setappdata(h, 'drawnslots', drawnslots);
setappdata(h, 'slotmap', slotmap);

str = sprintf('(%d/%d)', drawnslots, totalslots);
set(handles.message2, 'String', [str ' drawn.']);

%draw rectangle
hr=imrect;
%hr.setColor([0 255 0]);
hr.setResizable(false);
slotmap(drawnslots+1, :) = hr.getPosition();

%change button behavior
if (ns > 1)
    set(handles.drawButton, 'Enable', 'off');
    set(handles.drawNextButton, 'Enable', 'on');
else
    set(handles.drawButton, 'Enable', 'off');
    set(handles.drawNextButton, 'Enable', 'on');
    set(handles.drawNextButton, 'String', 'Finish');
end

%update text
drawnslots = drawnslots + 1;
str = sprintf('(%d/%d)', drawnslots, totalslots);
set(handles.message2, 'String', [str ' drawn.']);

%set current state
setappdata(h, 'drawnslots', drawnslots);
setappdata(h, 'slotmap', slotmap);
```

```
% --- Executes on button press in drawNextButton.
function drawNextButton_Callback(hObject, eventdata, handles)
% hObject    handle to drawNextButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

h = getappdata(0, 'hMainGui');
totalslots = getappdata(h, 'totalslots');
drawnslots = getappdata(h, 'drawnslots');
slotmap = getappdata(h, 'slotmap');

if (drawnslots == totalslots)
    set(handles.drawNextButton, 'Enable', 'off');
    set(handles.startButton, 'Enable', 'on');
    rmappdata(h, 'drawnslots');
    return;
end

hr = imrect;
hr.setResizable(false);
slotmap(drawnslots+1, :) = hr.getPosition();
drawnslots = drawnslots + 1

slotmap


setappdata(h, 'drawnslots', drawnslots);
setappdata(h, 'slotmap', slotmap);


str = sprintf('(%d/%d)', drawnslots, totalslots);
set(handles.message2, 'String', [str ' drawn.']);

if (drawnslots == totalslots)
    set(handles.drawNextButton, 'String', 'Finish');
end


% --- Executes on button press in startButton.
function startButton_Callback(hObject, eventdata, handles)
% hObject    handle to startButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
h = getappdata(0, 'hMainGui');
slotmap = getappdata(h, 'slotmap');
F = getappdata(h, 'Files');
setappdata(h, 'processedSlots', 0);

%get and redraw current image again
im = getappdata(h, 'image');
axes(handles.image);
imshow(im);

%update displayed message
nfile = getappdata(h, 'currentPic');
nslot = getappdata(h, 'processedSlots');
totalfiles = size(F, 1);
totalslots = size(slotmap, 1);
string = sprintf('File %d/%d. Slot (%d/%d)',nfile, ..
totalfiles, nslot+1, totalslots);
set(handles.message3, 'String', string);

%display a rectangle of the first slot
hr=imrect(handles.image, slotmap(1,:));
setappdata(h, 'lastrect', hr);
hr.setColor('red');
hr.setResizable(false);


size(im)
slotmap(1,:)
% display a thumbnail of the first slot
slot = im(slotmap(1,2):slotmap(1,2)+...
slotmap(1,4),slotmap(1,1):slotmap(1,1)+slotmap(1,3),1:3);

%compute the covariance matrix, store it
X = getCovarianceMatrix(slot);
setappdata(h, 'currCM', X);
%create the cell
dataset = {};
setappdata(h, 'dataset', dataset);


axes(handles.slotView);
```

```
imshow(slot);

%enable the control buttons
set(handles.annotateOcc, 'Enable', 'on');
set(handles.annotateVac, 'Enable', 'on');
set(handles.skipSlot, 'Enable', 'on');
set(handles.skipFrame, 'Enable', 'on');
set(handles.startButton, 'Enable', 'off');




% --- Executes on button press in annotateOcc.
function annotateOcc_Callback(hObject, eventdata, handles)
% hObject    handle to annotateOcc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%to do here
% h = getappdata(0, 'hMainGui');
% X = getappdata(h, 'currCM');
performAnnotation(1, handles, false, false);




%

function performAnnotation(value, handles, skipSlot, skipFrame)
h = getappdata(0, 'hMainGui');
X = getappdata(h, 'currCM');
y = value;

%store the data
if (skipSlot==false || skipFrame==false)
    dataset = getappdata(h, 'dataset');
    dataset = [dataset; {X y}];
    disp('que pasa aqui vamosaverrr');
    {X y}
    setappdata(h, 'dataset', dataset);
end

%slots read
```

```
read = getappdata(h, 'processedSlots');
slotmap = getappdata(h, 'slotmap');
total = size(slotmap,1);

if (skipFrame == true)
    read=total;
else
    read=read+1;
end

if (read < total)
    %if there are still more slots in this image..
    %we need to display the next one, and change the color of the last one

    %updated number of processed slots in this frame
    setappdata(h, 'processedSlots', read);

    %change the color of last rectangle
    hr = getappdata(h, 'lastrect');
    hr.setColor('green');
    hr.setResizable(false);



    %load the (current) image
    im = getappdata(h, 'image');



else
    read=0;
    setappdata(h, 'processedSlots', read);

    %we need to load the new image...
    F = getappdata(h, 'Files');
    i = getappdata(h,'currentPic');

    im = imread(F(i+1).name);
    axes(handles.image);
    imshow(im)

    setappdata(h, 'image', im);
    setappdata(h, 'currentPic', i+1);
```

```
end


%display new rectangle
hr=imrect(handles.image, slotmap(read+1,:));
setappdata(h, 'lastrect', hr);
hr.setColor('red');
hr.setResizable(false);

%update displayed message
F = getappdata(h, 'Files');
nfile = getappdata(h, 'currentPic');
nslot = getappdata(h, 'processedSlots');
totalfiles = size(F, 1);
totalslots = size(slotmap, 1);
string = sprintf('File %d/%d. Slot (%d/%d)',nfile, ...
totalfiles, nslot+1, totalslots);
set(handles.message3, 'String', string);

%extract slot, compute and store covariance matrix
i=read+1;
slot = im(slotmap(i,2):slotmap(i,2)+...
slotmap(i,4),slotmap(i,1):slotmap(i,1)+slotmap(i,3),1:3);
axes(handles.slotView);
imshow(slot);
X = getCovarianceMatrix(slot);
setappdata(h, 'currCM', X);




% --- Executes on button press in annotateVac.
function annotateVac_Callback(hObject, eventdata, handles)
% hObject    handle to annotateVac (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

performAnnotation(0, handles, false, false);
```

```
% --- Executes on button press in skipSlot.
function skipSlot_Callback(hObject, eventdata, handles)
% hObject    handle to skipSlot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

performAnnotation(0, handles, true, false);




% --- Executes on button press in skipFrame.
function skipFrame_Callback(hObject, eventdata, handles)
% hObject    handle to skipFrame (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

performAnnotation(0, handles, false, true);


% --- Executes on button press in saveButton.
function saveButton_Callback(hObject, eventdata, handles)
% hObject    handle to saveButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 function [covMat] = getCovarianceMatrix(img)

w=size(img,2);
h=size(img,1);
n=w*h;


%YCbCr space
Y=img(:,:,1);
Cb=img(:,:,2);
Cr=img(:,:,3);

%first order derivatives
[FX, FY] = gradient(double(Y));

%x, y coordinates variables
[x,y] = meshgrid(1:size(img,2), 1:size(img,1));
```

```
F=zeros(6,n);
k=1;
for i=1:h
    for j=1:w
        F(:,k)=[x(i,j) y(i,j) Cb(i,j) Cr(i,j) FX(i,j) FY(i,j)]';
        k=k+1;
    end
end

%mean vector mu
mu = sum(F,2)/n;


%compute the covariance matrix
p = zeros(6,6);
for i=1:n
    x=(F(:,i)-mu)*(F(:,i)-mu)';
    p=p+x;
end
C=p.*(1/(n-1)); %the covariance matrix

covMat = C;


function [outputclass] = knnclassify2(trainset, queryset, k)

neighborIds = kNN(trainset, queryset, k);

n = size(neighborIds,1);
outputclass = zeros(n, 1);

for i=1:n
    classes = zeros(1,k);
    for j=1:k
        classes(j) = trainset{neighborIds(i,j),2};
    end

    outputclass(i) = mode(classes);
    classes
end
```

```
function [dist] = covMatDistance(A,B)


dist = sqrt(sum(log(eig(A,B)).^2));

function [neighborIds] = kNN(trainset, queryset, k)

%for each element in the query set,
%k columns with the k nearest neighbor IDs from the training set
neighborIds = zeros(size(queryset,1),k);
neighborDistances = neighborIds;



%trainset is a cell, remember that.
numDataVectors = size(trainset,1);
numQueryVectors = size(queryset,1);

for i=1:numQueryVectors,

    vec = queryset{i};
    %dist = distances from the current vector to every vector in the
    %trianing set

    dist = zeros(numDataVectors,1);
    for j=1:numDataVectors
        dist(j) = covMatDistance(vec, trainset{j,1});
    end

    %dist = sum((repmat(queryMatrix(i,:),numDataVectors,1)...
    %-dataMatrix).^2,2);

    [sortval sortpos] = sort(dist,'ascend');
    neighborIds(i,:) = sortpos(1:k);
    neighborDistances(i,:) = sortval(1:k);
end
```

References

[1] [FoM99] Foerstner, W., Moonen, B., *A Metric for Covariance Matrices*, 1999.

[2] [ITR09] Idris, M. Y. I., Tamil, E.M., Razak, Z., Noor, N.M., Kin, L.W, *Smart Parking System using Image Processing Techniques in Wireless Sensor Network Environment*, Information Technology Journal, 2009.

[3] [RTL09] Refaeilzadeh,P. Tang,L., and Liu,L. *Cross Validation*. In Encyclopedia of Database Systems, Editors: M. Tamer zsu and Ling Liu. Springer, 2009.

[4] [Tru07] True, Nicholas, *Vacant Parking Space Detection in Static Images*, University of California, San Diego Technical Report, 2007.

[5] [TPM06] Tuzel, O., Porikli, F., Meer, P., *Region Covariance: A Fast Descriptor for Detection and Classification*, ECCV, 2006.

[6] [WuX06] Wu, Q., Zhang, Y., *Parking Lots Space Detection*,Machine Learning 10-701 Carnegie-Mellon University Project Report, 2006.