Smart City Parking Lot Occupancy Solution

Paula Tătulea[1], Florina Călin[2], Remus Brad[3], Lucian Brâncovean[4], Mircea Greavu[5]
Industrial Software SRL, Sibiu, Romania

***Abstract*—In the context of Smart City projects, the management of parking lots is one of the main concerns of local administrations and of industrial solution providers. In this respect, we have presented an image processing application, which overcomes the issues of classical electro-mechanical solutions and employs the feed of a surveillance camera. The final web-based interface could provide to the clients the real-time availability and position of the parking space. The proposed method uses a series of feature measures in order to speed-up and accurately classifies the occupancy of the space. Using a published benchmark, our method has proved to provide very accurate results and have been extensively tested on two proprietary parking locations.**

Keywords*—Smart city; car parking; occupancy; monitoring system; image features

I. Introduction

A large number of the parking lot systems use counters on entry and exit barriers, but unfortunately, erroneous results are obtained if a vehicle occupies more than one parking space or when the parking lot includes several types of parking spaces. A solution to the problem was to add magnetic sensors on each parking space, however, in practice; this type of system involves very high costs.

Therefore, the scope of our research was the development of a computer vision system for the detection of parking spaces. The motivation behind it resides in the fact that, a video camera installed in the car park for security reasons can also monitor the parking spaces. Implementing this smart parking system has many benefits: it is helpful to customer drivers, low cost, can be used in everyday life, decreases the time needed to search for a parking space as the driver can steer quickly to the available parking spaces, eliminates redundant traffic generated by the search for available spaces, can be installed quickly and easily, and it is easier to maintain than current systems.

However, the main challenges of the camera-based systems are the lighting conditions: low light or temporal and spatial lighting fluctuations, shadows and reflections from the surface of other vehicles.

In this respect, the method proposed in paper [1], employs geometric modelling and parking areas layouts as means to automatically extract the parking lot configuration. The extraction of the parking spaces is based on white or yellow lines, using high-resolution aerial images, indicating that a large number of parking spaces can be accurately. Thus, after defining the geometric and layout models, a method of parking space extraction is proposed, using both parking space and vehicle detection. Once the objects detection has been performed, by comparing against the model, a grouping function is applied to the relative positions, according to the rules of the geometric.

In the paper of Gálvez del Postigo et al. [2], after initialization, a background extraction was performed and a map was created. Using this map, the vehicles are detected and tracked in order to determine their status. The first step of consists of defining the parking areas to be analyzed, from which a binary mask will be created. Thus, a background model is needed in order to detect moving vehicles. For this purpose, the chosen approach involves the implementation of Mixture of Gaussians techniques. The Transition Map is a technique that works well for detecting parked vehicles. In order to determine the status of the parking spaces, two instances are analyzed: parked vehicles and moving vehicles.

In Števanák et al., the issue of the parking space occupancy is addressed using an open source solution called PKSpace [3]. It uses a vision-based approach, employing an automated learning model, in order to categorize the images of the parking spaces as either occupied or vacant. It also allows the user to choose either a default model, that is part of the solution proposed, or to create their own data set for a particular parking lot and to develop the model based on this information. At the same time, this solution offers application programming interfaces (APIs) which allow external systems to process the data collected and store it for later use.

In paper [4], the authors have studied the performance of image processing algorithms when the multithreading approach is applied on different platforms (single core / multi-core). Results shown that multithreading improve processing time on single-core or multi-core platforms. With a single core, the best results are achieved when using a combination of small size images and less complex algorithms, while the combination of a smaller size image and more complex algorithms improves performance when working with multi-core processors. Multithreading programming can improve the performance of the multi-core processor when complex image processing algorithms are applied.

Vítek et al. shown that detecting parking spaces occupancy is constantly on the rise, especially in big cities [5]. The paper uses wireless cameras to manage parking spaces and determines the parking space occupancy based on the camera feeds. The proposed system employs small camera modules based on Raspberry Pi Zero and an efficient algorithm for occupancy detection based on the Histogram of Oriented Gradients (HOG) and Support Vector Machine (SVM) classifier. The basic features include information concerning the vehicle's orientation, where it can be more accurately determined. The solution presented can provide occupancy information at a rate of 10 parking spaces per second with an accuracy of more than 90% in various weather conditions.

The scope of this paper was to propose a solution that can extract the parking spaces and detect the occupancy of the parking lot, using a sequence of images acquired from video cameras. Therefore we have combined techniques that are capturing and detecting the presence of vehicles on parking spaces using image processing, parking spaces detection based on markings, detection of the parking lot occupancy, identification of available and occupied parking spaces, outlining parking spaces, counting the number of available and occupied parking spaces. The proposed method allows sending information about the occupancy of the parking lot to customers using HTTP requests.

II. Proposed System

Most studies in this field have a sequential approach; and therefore, a lot of time required for processing. Thus, we have proposed a method in which the sequencing is done by parallel segmentation, each image is divided into several parts and then one thread controls each one of them. After segmentation, all parts are merged and the desired result is obtained [4].

The processing steps of the method are the following:

a) Parking lot map description;

b) Image frame acquisition and pre-processing;

c) Adaptive background modelling;

d) Computation of features;

e) History calculation;

f) Fusion of results for each parking space;

g) Defining parking space status;

h) Feeding data to Client-Server system;

The general structure of the proposed system is presented in Fig.1.

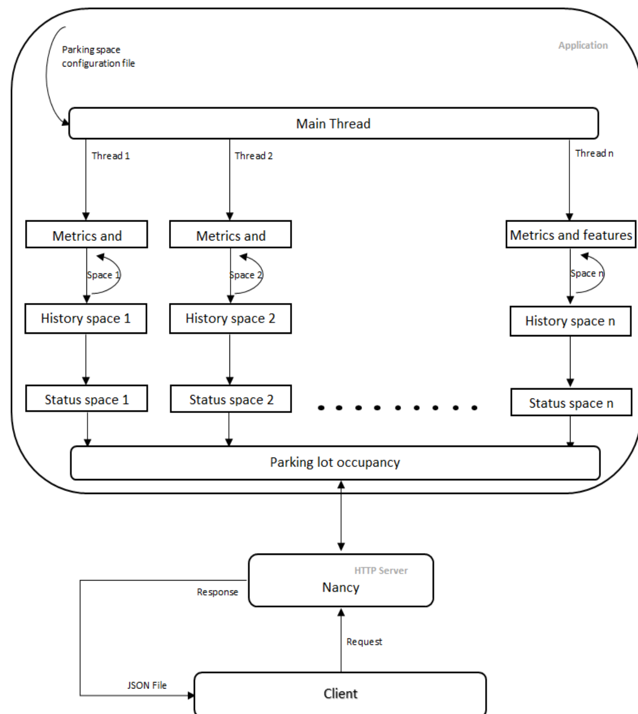


Fig. 1. System Block Diagram.

A. Parking Lot Map

In an offline configuration stage, using a template image taken from each camera, the following data are defined: the coordinates of each parking space, the number of parking spaces, as well as the centroid/midpoint of each parking space. This information is saved in a configuration file, uploaded one time, at system initialization. The coordinates of the parking space, as well as the midpoint, are converted from screen coordinates into Cartesian coordinates.

Sample file:

```
Number:1;Points:-626 -24,-541 4,-496 -92,-594 -116;
Centroid:-563 -58
Number:2;Points:-471 12,-415 32,-367 -44,-428 -66;
Centroid:-420 -17
Number:3;Points:-336 52,-290 65,-240 -6,-292 -17;
Centroid:-289 23
Number:4;Points:-234 84,-177 102,-117 35,-184 19;
Centroid:-177 59
```

The original image is further divided into several regions, aiming to simplify the image representation into a more relevant and easier way to be analyzed. This process is used to locate each parking space and its limits, based on the four coordinates from the configuration file. A perspective transformation is also applied for each parking space and a new region is obtained from the resulting transformation matrix. During this process, a label is assigned to each parking space. In consequence, the template image is divided into several regions and a thread will process each one of them, in order to determine its status.

B. Multithreading Programming

In the multithreading approach, the shared memory in which the threads work is the image pixel matrix. The work load and the part of the matrix that each thread has to handle are determined by the main thread [6]. A multithread process has several simultaneous execution points [7]. Using multiple threads allows an application to allocate long-term tasks, so that they can be performed concurrently. This is also possible due to the significant improvements in the multi-core systems.

Nowadays multi-core processors are widely deployed in both server and desktop systems. The performance of multithreaded applications can be improved when using multi-core systems, since the thread charge can be moved to the core, which works with several threads simultaneously [8]. A good example of applications that benefit from multithreading is Computer Vision ones [9]. The main idea behind parallel processing of images is to divide the problem into simple tasks and solve them simultaneously so that the total processing time is the sum of the finished tasks (best case scenario) [10]. Image processing can be a time consuming task based on the image matrix structure that drives this process towards a multithreading algorithm.

The proposed solution uses execution threads to improve application performance. After identifying the number of parking spaces, one thread is created to process each parking space, using Thread Pooling. These are pre-fabricated threads that can be launched more quickly by the OS. We cannot define a name for a thread in the thread pool. The threads in the thread pool only work in the background. The execution

threads are created using the Task Parallel Library (TPL). TPL offers a basic form of structured parallelism and is based on the concept of task.

For each parking space, the image processing involves the analysis of the features (Histogram of Oriented Gradients-HOG, Scale Invariant Feature Transform–SIFT corner detector, color spaces - YUV, HSV and YCbCr).

HOG is a feature descriptor used to detect objects by counting the occurrences of gradient orientation in areas of interest. SIFT is a descriptor used to detect the number of the points of interest (corners).

The HSV color space has 3 channels: the Hue, the Saturation and the Value, or intensity. The Hue channel represents the "color". The saturation channel is the "amount" of color (this differentiates between a pale green and pure green), and intensity is the brightness of the color (light green or dark green). YUV defines a color space in terms of one luminance (Y) and two chrominance (UV) components. YCbCr color model contains Y, the luminance component and Cb and Cr are the blue-difference and red difference Chroma components.

The standard deviation values for the three channels V (`devSYUV`), S (`devSHSV`), Cb (`devSYCrCb`) of the color spaces YUV, HSV, YcbCr, the number of corners (`noSIFT`), and the HOG descriptor mean (`meanHOG`) were used to create a history, based on predefined thresholds: 0.03 for mean HOG, 7 for number of corners resulting from SIFT, 1.4 for V component from YUV, 9 for S component from HSV and 1.1 for Cb component from YCbCr This history tracks the availability of parking spaces.

Each value from metrics and measurements compares with a default threshold. Based on this comparison, counted the values of 1 (`statusOccupied`) and 0 (`statusAvailable`), and the status is determined by the predominant value. If predominant is 1, the parking space is occupied, otherwise, is free.

```
Function SetStatus (indexParkingLot)
    if meanHOG[indexParkingLot] > 0.03 then
statusOccupied++;
    else statusAvailable++;
    end if
    if noSIFT[indexParkingLot] >= 7 then
statusOccupied++;
    else statusAvailable++;
    end if
    if devSYUV[indexParkingLot] > 1.4 then
statusOccupied++;
    else statusAvailable++;
    end if
    if (devSHSV[indexParkingLot] > 9) then
statusOccupied++;
    else statusAvailable++;
    end if
    if (devSYCrCb[indexParkingLot] > 1.1) then
statusOccupied++;
    else statusAvailable++;
    end if
    if (statusAvailable > statusAvailable) then
status = 0
    else status = 1
    end if
EndStatus
```

For more accurate results, was created a history of 20 frames that contains the status of each parking space, obtained from measured metric and measurement results. With each new frame, it is tested if the number of frames originally set has reached. If yes, then adding the status to the list produces the effect of "Sliding Window", which requires the elimination of the first value of the buffer and the addition of the new value in the list, according to the FIFO principle. Thus, this technique allows for the last changes to each parking space to be retained in order to determine the status. Over time, this process helps stabilize changes in the background, such as the gradual change from day to night, different weather conditions.

```
for parkingSpace=0:totalNumberParkingLot
   if(sizeOfBuffer < 20)
        Status = SetStatus(parkingSpace)
        Add status in buffer
   else
        SlidingWindow
        Status = SetStatus(parkingSpace)
        Add status in buffer
   end if
   if predominat is 1 in buffer StatusParkingSpace=1
   else StatusParkingSpace = 0
   end if
end for
```

The information concerning the parking lot occupancy emerges once each frame is processed, thus identifying the number of available spaces, as well as occupied ones, out of the total number of parking spaces considered.

C. Client Interface

There are many alternatives to creating a web server for the purpose of client access to parking lot occupancy. Within this system we have used HTTP server based on Nancy, which is a framework for building HTTP-based services in .NET. With these HTTP-based services, this framework can handle all standard HTTP methods such as GET, POST, PUT, DELETE, HEAD etc. Everything in Nancy is "HOST's". A host acts as a framework or adapter for a hosting environment, and allows Nancy to run on existing technologies such as ASP.NET, WCF and so on.

The application must be downloaded and installed using the NuGet package manager, as it will download the complete references to the current solution or project. Once Nancy is installed, the first module can be created. The requests are handled by the modules. The Nancy website regards a module as "the place where you define the behavior of your application". Like Controllers in MVC, there are modules in Nancy. A single module must be defined for a Nancy app, which becomes the starting point of an application. We can create as many legacy modules from NancyModule as we need. In the class builder, the routes are defined with Get ["/"]. A route must follow the same pattern as Literal Segments, Capture Segments (`{yourname}`), and regular expressions.

The system listens to one or more addresses in order to send the necessary information to an http request. The system can listen to multiple addresses by creating a URI array and assigning it to the constructor for NancyHost. The result is that you can listen on multiple network interfaces. This is useful for example, in situations where you have a server that has to listen on two different interfaces, and respond differently to both.

With Nancy HTTP, you can tell which request came from which IP, allowing you to selectively say in program code which connection is allowed access to which functionality.

The life cycle of each Nancy application starts with receiving the HTTP request and ends when it sends the HTTP response back to the client. Any good web framework allows you to send data to it. Nancy is very flexible in terms of answering. To prepare more complex answers with headers for the information to be sent, a new Response object is built. The response is a JSON and it looks as follows:


```
{
        "Parking_lot_1":
        {
          "TotalParkingSpaces":20,
          "FreeParkingSpaces":12,
          "OccupiedParkingSpaces":8
         },
        "Parking_lot_2":
        {
          "TotalParkingSpaces":38,
          "FreeParkingSpaces":15,
          "OccupiedParkingSpaces":23
        }
}
```


The data sent in JSON format includes the names of the installed cameras, and for each one the following information is provided: the total number of parking spaces, the number of vacant spaces and the number of occupied spaces.

Example of HTTP request: http://localhost:5000/data

III. Results

In the purpose of evaluation, three metrics were computed: accuracy, sensitivity and specificity defined in equations 1, 2 and 3. In these equations, TP (True Positive) is the number of occupied spaces classified as occupied, TN (True Negative) is the number of vacant spaces classified as vacant, FP (False Positive) is the number of vacant spaces classified as occupied and FN (False Negative) is the number of occupied spaces classified as vacant [11].

$$Accuracy = (TP + TN)/(FP + FN + TP + TN) \quad (1)$$

$$Sensitivity = TP/(TP + FN) \quad (2)$$

$$Specificity = TN/(TN + FP) \quad (3)$$

where TP (True Positive) is the number of occupied spaces classified as occupied, TN (True Negative) is the number of vacant spaces classified as vacant, FP (False Positive) is the number of vacant spaces classified as occupied and FN (False Negative) is the number of occupied spaces classified as vacant [11].

Table 1 shows the accuracy resulted from testing our system under various weather conditions: overcast, rainy and sunny days in comparison to the CNRPark+EXT results [12]. We have employed the same parking lot map as in the benchmark provided in CNRPark. For the first benchmark, 8 parking spaces were selected, and for the second benchmark, 26. It can be seen that the results were encouraging, with a minimum accuracy rate of over 90%.

The accuracy, sensitivity, and specificity are calculated in various weather conditions, with results ranging between 91%-99%. The best accuracy is in overcast conditions for both benchmarks, as seen in Fig. 3 and 5. In Fig. 2 and 4, the incorrect and correct detections can be observed on a number of frames from the two benchmarks.

A. Specific Benchmarks Results Examples

1) Overcast weather: The results on an overcast day for the two benchmarks are over 98% accurate, which translates into very good detection accuracy. Figure 6 shows screenshots of the two examples used.

*2) Rainy weather:*The screenshots in Figure 7 were taken on a rainy day for various benchmarks. The accuracy under these weather conditions is over 90%. It shows that the problems persist in the case of camouflage.

*3) Sunny weather:*The screenshots in Figure 8 were taken on a sunny day for various benchmarks. The accuracy under these weather conditions is over 91%. It shows that the problems persist in the case of camouflage and shade.

TABLE I. RESULTS COMPARISON ON DIFFERENT BENCHMARKS

Results	Accuracy		
	Overcast%	**Rainy%**	**Sunny%**
Benchmark 1 with proposed method	99.708	90.088	91.566
Benchmark 1 from [12]	100	100	99.900
Benchmark 2 with proposed method	98.313	95.070	91.564
Benchmark 2 from [12]	100	100	100

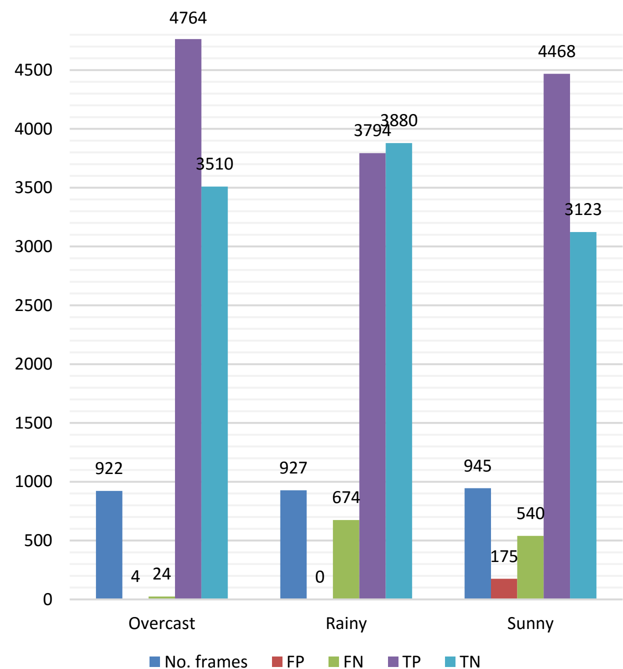


Fig. 2. The Benchmark 1 Classification Results under different Weather Conditions.

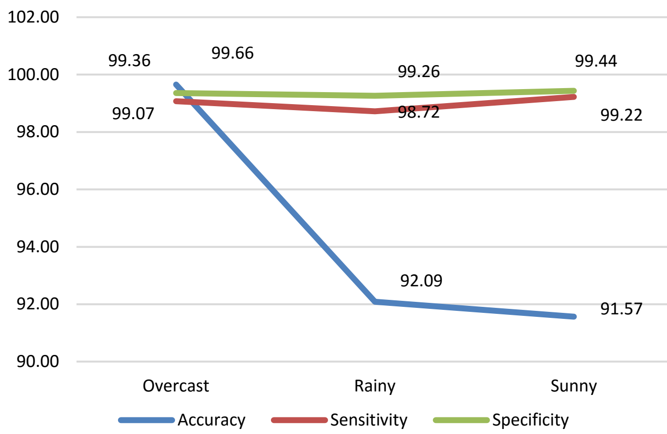


Fig. 3. The Benchmark 1 Results of Three Measures on different Weather Conditions.

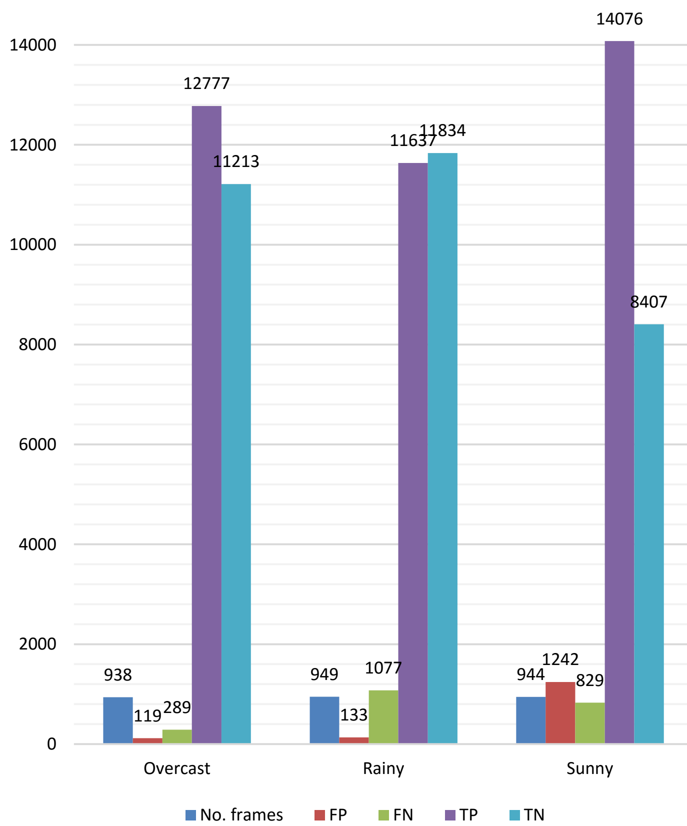


Fig. 4. The Benchmark 2 Classification Results under different Weather Conditions.

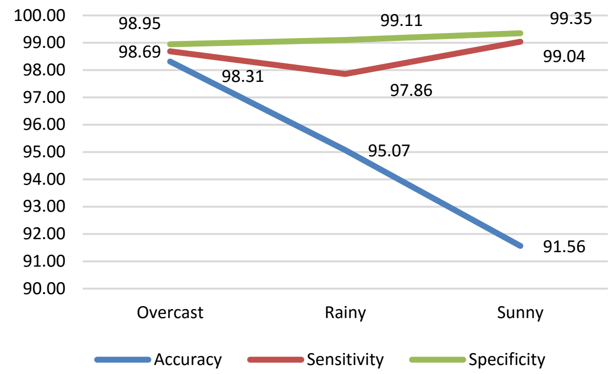
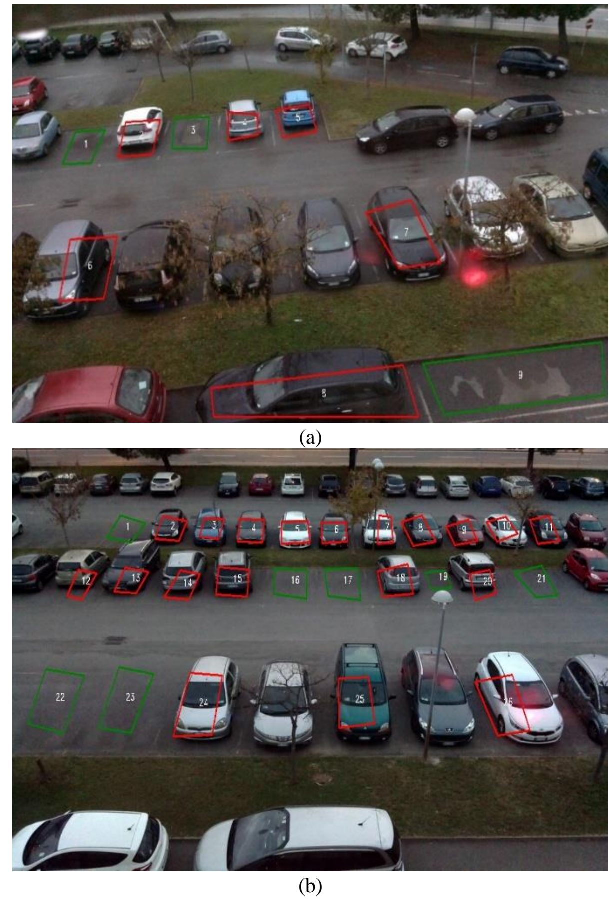


Fig. 5. The Benchmark 2 Results of Three Measures on different Weather Conditions.



(a)

(b)

Fig. 6. Results on an Overcast Day for (a) Benchmark 1; (b) Benchmark 2.

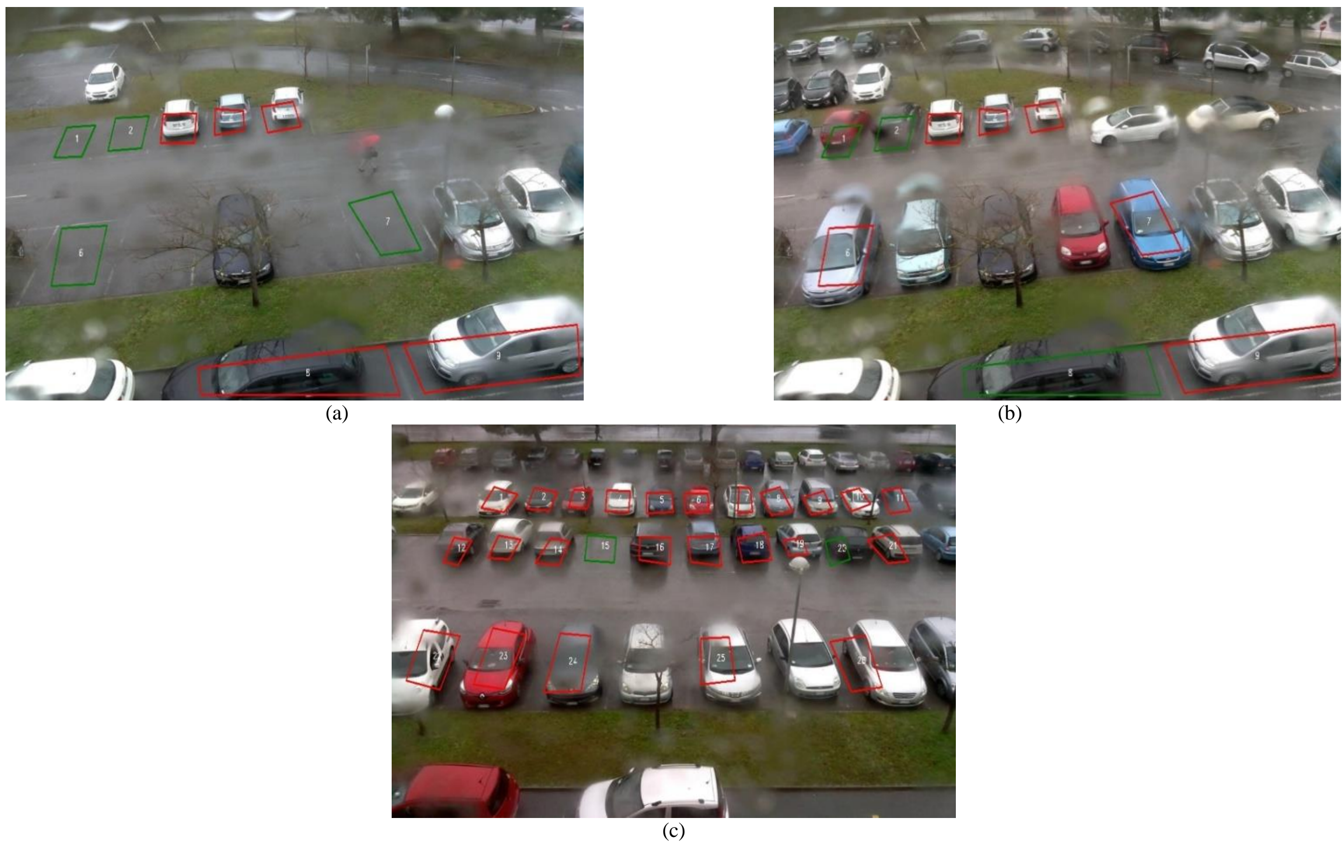


Fig. 7. Results on a Rainy Day for (a) and (b) Benchmark 1 (c) Benchmark 2.

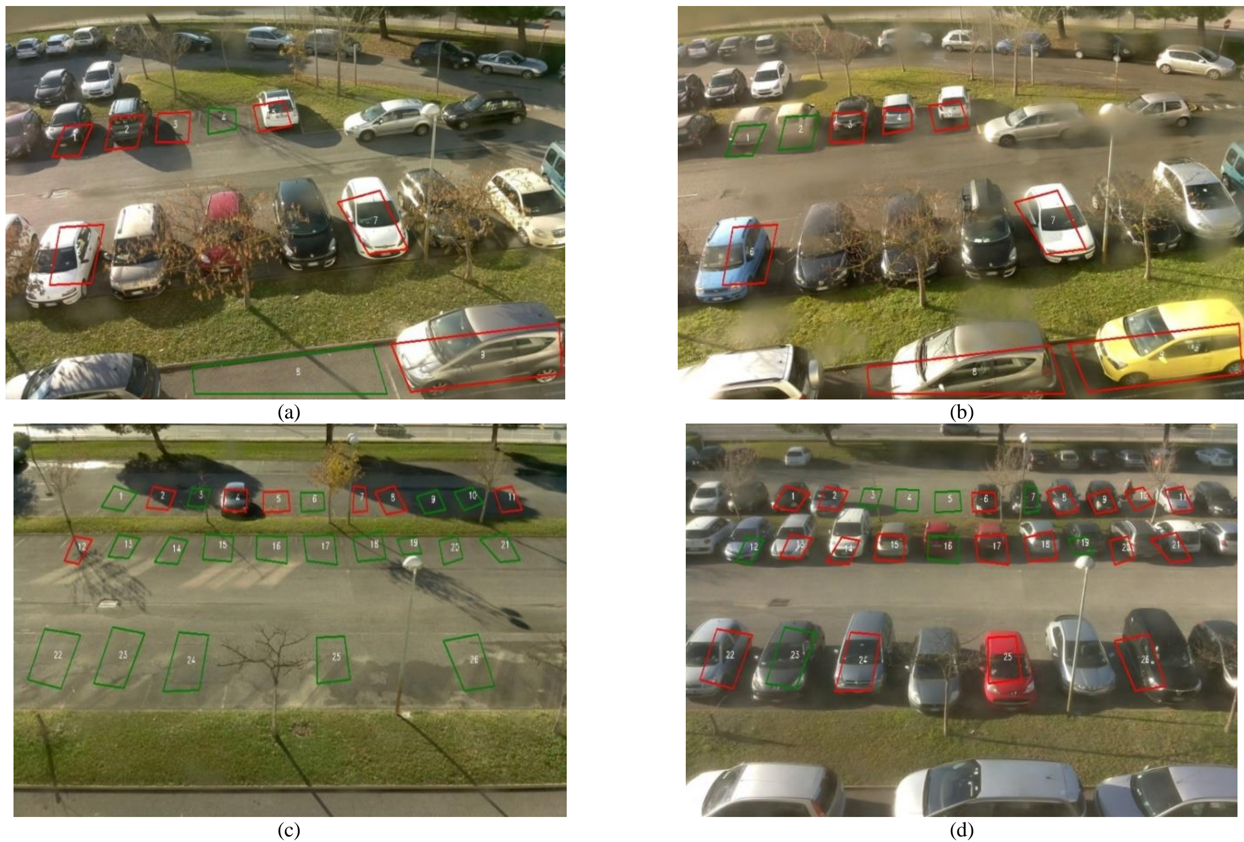


Fig. 8. Results on a Sunny Day for (a) and (b) Benchmark 1; (c) and (d) Benchmark 2.

IV. CONCLUSIONS

In this paper, we have presented a solution for the detection of parking spaces using image sequences acquired from video cameras. A technique based on computer vision algorithms have been investigated, together with a parallel implementation and the facility to provide to clients the information about the availability of a parking lot using HTTP. Based on two available benchmarks, the results obtained achieved minimum accuracy rate of over 90%. The unsolved system problems remain the presence of shadows and camouflage. Shadows are recognized as objects, in this case a vehicles, and thus, a partially shaded vacant space is detected as occupied.

ACKNOWLEDGEMENTS

The research was financed by the project SMART CITY PARKING–INTELLIGENT SYSTEM FOR URBAN PARKING MANAGEMENT, SMIS 115649, Grant number 31/28.06.2017. The project is co-financed by the European Union through the European Regional Development Fund, under the Operational Competitiveness Program 2014-2020.

REFERENCES

[1] G. Koutaki, T. Minamoto and K. Uchimura, "Extraction of parking lot structure from aerial image in urban areas," International Journal of Innovative Computing, Information and Control, vol. 12, no. 2, pp. 371-382, April 2016.

[2] C. G. del Postigo, H. Torres, J-M. Menéndez, "Vacant parking area estimation through background subtraction and transience map analysis", IET Intelligent Transport Systems, vol. 9, no. 9, pp. 835-841, 2015.

[3] R. Števanák, A. Matejov, O. Jariabka, M. Šuppa, "PKSpace: An Open-Source Solution for Parking Space Occupancy Detection", Proceedings of the 21st Central European Seminar on Computer Graphics, CESCG 2017.

[4] S. Bose, A. Mukherjee, Madhulika, S. Chakraborty, S. Samanta, N. Dey, "Parallel image segmentation using multi-threading and k-means algorithm", IEEE International Conference on Computational Intelligence and Computing Research, 2013.

[5] S. Vítek, P. Melničuk, "A Distributed Wireless Camera System for the Management of Parking Spaces", Sensors, vol. 18, no. 1: 69, 2018.

[6] A. Kika, S. Greca, "Multithreading Image Processing in Single-core and Multi-core CPU using Java", International Journal of Advanced Computer Science and Applications, vol. 4, no. 9, 2013.

[7] M.Shanthi, A.Anthony Irudhayaraj. Multithreading, "An Efficient Technique for Enhancing Application Performance", International Journal of Recent Trends in Engineering, vol 2, no. 4, pp. 165-167, November 2009.

[8] Kuo-Yi Chen, Fuh-Gwo Chen, "The Smart Energy Ma nagement of Multithreaded Java Applications on Multi-Core Processors", International Journal of Networked and Distributed Computing, vol. 1, no. 1, pp.53-60, January 2013.

[9] G. A. Baxes, Digital Image Processing: Principles and Applications. John Wiley and Sons, Inc, New York, NY, 1994.

[10] S. Saxena, N. Sharma, Sh. Sharma, "Image Processing Tasks using Parallel Computing in Multi core Architecture and its Applications in Medical Imaging", International Journal of Advanced Research in Computer and Communication Engineering, vol. 2, issue 4, April 2013.

[11] D. Acharya, W. Yan and K. Khoshelham, "Real-time image-based parking occupancy detection using deep learning", Proceedings of the 5th Annual Con-ference of Research@Locate, vol. 2087, pp. 33–40, 2018.

[12] http://cnrpark.it/, A Dataset for Visual Occupancy Detection Parking Lots.