Automatic Monitoring of vehicles at UPC Parking Campus Nord

J. Maroto
Supervisors: J. Ruiz and L. Torres
Escola Tècnica Superior de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
(Dated: May 25, 2016)

The new progresses in the Camera Vision field enables new approaches to existing applications that could reduce the costs and enhance the quality, the quantity and the reachability of the information that can be extracted from a surveillance system. In the application we are analyzing, which is implementing a security system for the UPC Parking Campus Nord, we aim to replace the current sensor system by a camera surveillance one that offers much more information to the security staff and the common user of the parking with reduced cost.

Keywords: parking, camera vision, multi-camera tracking, surveillance, security

I. INTRODUCTION

The objective of this project is to design a system able to control a parking automatically and provide the security agents a quick and easy way of analysis of its conditions.

Following the logical order of the driver's behavior in parking, the initial task is to identify when a car enters and what is its license plate, and then we follow its position to see where it parks; at that time, the specific place will be marked as occupied and the car license plate stored. If the car decides to park in double row the security agent will be notified. Finally, at the time the car leaves the parking it should be the reverse process.

The whole system will be monitored from a simple and intuitive application in charge of: announcing the available places and the occupied ones, surveilling the parking in real time with the cameras, knowing when a car has parked in double row, has entered or exited the parking and which is its license plate, in addition to describing where is every car parked using their license plate as their identifier. Additionally, we present the user of the parking a mobile app to be able to know in advance if there are free places in the parking.

In the article scope, we will focus in the global system design, how each part of the project interacts, obtains and gives information to the other algorithms and, specifically, we will explain in detail the assignation block of the multi-camera tracking algorithm, as these matters were mostly designed and implemented by the author of the article. The other algorithms, the camera installation and the business perspective of the project are acknowledged to the following people: D. Rodríguez, E. Torres, F. Lluís, M. Górriz, M. Gil, S. Melissa, X. Ferrer and L. Mora.

II. SYSTEM DESIGN

The system is prepared to work in real time in the region of the D5 and D6 building of the Campus Nord UPC parking, but the same design idea could be used for another parking after some modifications in the location, number and resolution of the cameras, the masks used for the parking spots and the variables values used in the algorithms. There are three principal subsystems: hardware, software and app.

The hardware subsystem consists of 6 cameras: 4 inside the parking at the D5 and D6 building balconies, one at the entrance and one at the exit. The images will be taken at 23 fps with resolution 1280x720. The communication will be realized using CAT5 Ethernet cables and all the frame information will be routed to one computer in which the software subsystem is yield. Because we were not able to implement the communication system in the Campus Nord parking due to the lack of Ethernet communication equipment, in the tests we first record the videos, store them using SD memories installed in the cameras and use the videos to simulate this hardware subsystem having a process that extracts frames periodically.

The software subsystem (Fig. 1) is executed in one computer and the internal algorithms are executed in sequential order. The Graphical User Interface or GUI is executed parallel to the main algorithms as it needs to respond to the user actions, like choosing which camera to surveil. We will explain how the main thread makes the processing of the information received by the cameras and obtains the occupancy data, the plates of the parked cars and the notifications (entry, exit and bad parked cars) and show them in the GUI.

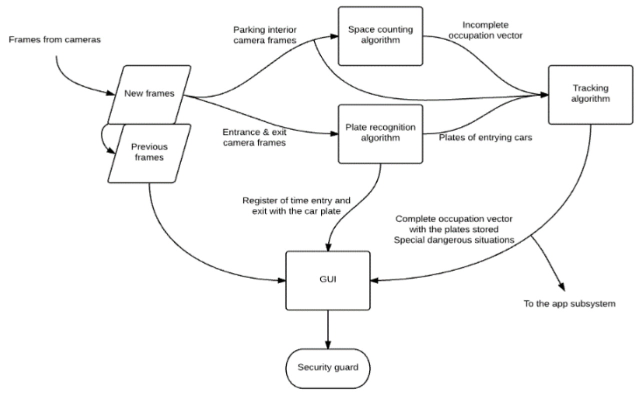
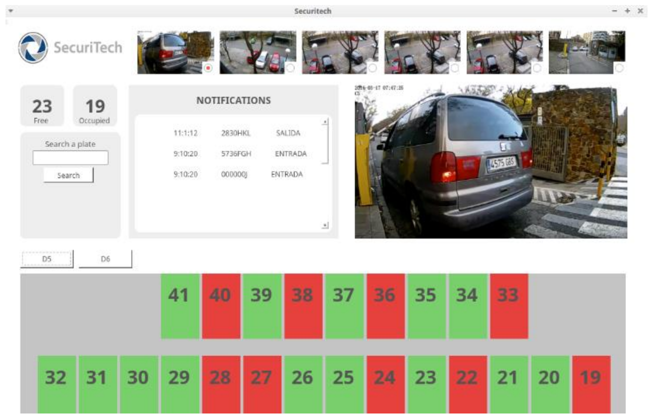

Frames from cameras
New frames
Previous frames
Parking interior camera frames
Space counting algorithm
Incomplete occupation vector
Tracking algorithm
Entrance & exit camera frames
Plate recognition algorithm
Plates of entrying cars
Register of time entry and exit with the car plate
Complete occupation vector with the plates stored Special dangerous situations
GUI
To the app subsystem
Security guard

FIG. 1. Block diagram of the software subsystem



```
11:1:12   2830HKL   SALIDA
9:10:20   5736FGH   ENTRADA
9:10:20   000000J   ENTRADA
```

FIG. 2. Graphical User Interface

First of all, we wait until the 6 frames of the camera are obtained. There could be delay between cameras, but as long as they all send frames at the same rate we will not need to resynchronize the cameras. That is to say, the delay between cameras will remain constant. The delay between cameras is assumed to be of lower magnitude than the delay between frames in each camera. As a working condition we have also that the delay between frames must be lower than the time the algorithms take to process the frames received. The solution we apply is to discard sets of frames until the algorithms are finished, forcing the working condition. This affects the probability that the algorithms fail obtaining information. The tracking algorithm is more dependent on the delay between the frames than the other algorithms, which only require detection of events, so it will be the more affected by this solution. With more time we would be able to reduce the algorithms processing delay via improving the code efficiency or executing the GUI process in another computer.

After the reception of frames, the GUI is refreshed with the information processed previously. If the number of sets of frames discarded is always the same, we ensure periodicity when showing the information. The GUI process purpose is to display the results of our system clearly (Fig. 2). The information that the GUI process requires is:

- The set of frames
- The occupancy of each place and the car's plate if it is occupied.
- Any special event, time and plate of the car included. Special events are the entry or exit of a car in the parking and when a car is stationed in double row.

In further versions of the system, this refreshing process will be executed parallel to the other algorithms to be able to refresh always the camera frames every time we obtain new frames, even if these frames will be not used by the analysis algorithms.

After the GUI process we start the space counting and the plate recognition processes.

The space counting process requires the frames of the cameras inside the parking. What it does with these frames is obtain after a series of transformations a value for each parking spot that determines if that spot is free or occupied. There are some spots that are covered by leaves, branches or even a lamppost where the space counting algorithm cannot be certain (more than a 20% probability of failing) of the occupancy of the spot. These spots are neither free nor occupied, are undetermined. The results are stored in the occupancy vector, which is sent to the tracking algorithm, which will resolve the undetermined situations and complete the occupancy vector.

The license plate recognition process requires the frames of the entrance and exit cameras. What it does with these frames is detect if there is a car entering or exiting the parking and, in that case, obtain and recognize the characters of the license plate after several transformations. This plate will be sent to the tracking algorithm in the case a car is entering. This way the tracking algorithm will know a car is going to enter and its plate. The license plate recognition algorithm also sends the plate with the time of entry or exit to the GUI to show that event to the security guard. This time is obtained using the computer's clock time.

The tracking algorithm must be executed after the license recognition and space counting process have finished. It will use the information obtained from these two processes and its own stored information to track the cars. It also requires the frames of the cameras inside the parking. The principal functions of this algorithm are to obtain the situations where a car is stationed in any zone that is not a parking spot, to complete the occupancy vector received from the space counting algorithm and to add to that vector the plates of the cars tracked from the entrance to the parking spot.

After that, the information obtained from the tracking except the bad stationed cars notifications is converted to a .txt and uploaded to a private server. This upload is done every second approximately, which is equivalent to 23 frames for the cameras we have used.

All the relevant information obtained in this cycle is stored to be used by the GUI in the next iteration of the software subsystem.

The app subsystem consists of a server in which the information about the occupancy of the parking is stored along with the plates of the parked cars and an app, which allows the user of the parking to know if there are any free parking spots and, thanks to the login system that is implemented, using the plate used as login to know where its car is parked inside the parking. This second service has more purpose for a possible future implementation of this product in a larger parking.

When a new .txt from the software subsystem is uploaded, it rewrites the existing one. This offers security in case of leaking because we will only have the present information. The app downloads the .txt from the server and extracts the data when the user requests it. This download is asynchronous and independent of the refreshing rate of the server. The information is showed to the user graphically (Fig. 3). The difference between the app and the GUI is that the user will not be able to surveil the parking and to know when a car enters the parking or is bad stationed. It will also only know the number of free places and the location of its own car.

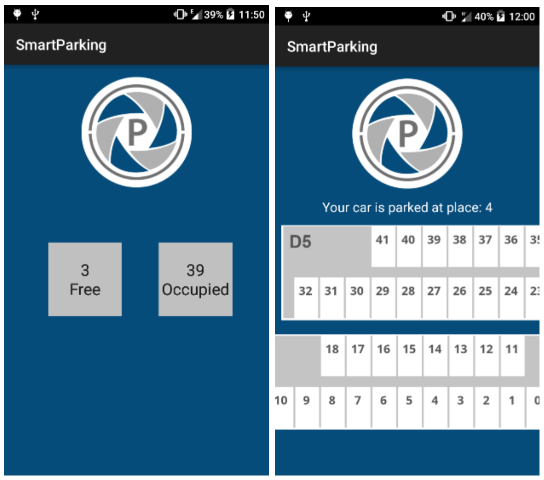


FIG. 3. App Interface

III. MULTI-CAMERA TRACKING

First of all, we need to simplify all the information that is in the frames received by the inside cameras. For the tracking we are only interested in the movement information in the parking lane and in the parking spots. We first divide each frame by its RGB channels, this is done to get the movement image for the red, green and blue components. We ensure this way less errors due to car color and illumination in the following steps. Then, we apply adaptive background subtraction with Gaussian mixtures [1] with adaptive parameter alpha equal to 0.01, which is adequate for the speed at which the cars move in the parking. We smooth the background and the actual images with a Gaussian Blur filter of 21x21 pixels. The threshold used after computing the difference between the background and actual image is 25 (out of 255) for the three components. We obtain the movement images after thresholding. They are binary images where there are a 1 or a 0 in each pixel that defines if there is movement there or not. The movement images of each of the three components are added using logical OR. The resulting movement image is then masked, so we will only take into account the movement inside the parking lane and the parking spots. The masks are different for each camera. The resulting movement image has still some appreciable noise, so we apply closing to erase it. After that, we apply erosion to expand the movement zone so it is less fragmented (Fig. 4).

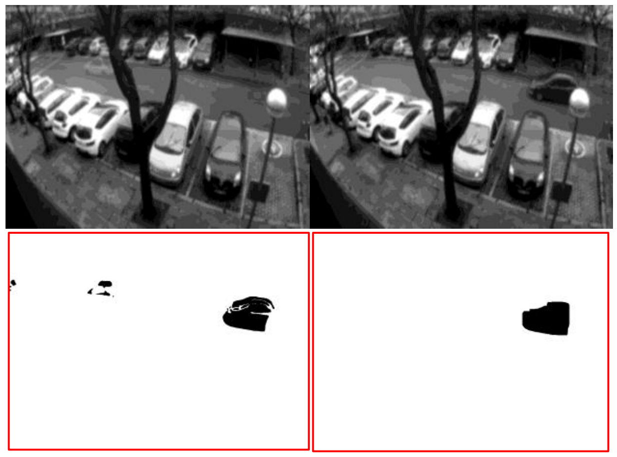


FIG. 4. Up-left is the averaged background image, up-right is the actual image, both in grayscale after being smoothed. Down-left and down-right are the resulting movement images before and after the closing and erosion respectively.

A Blob is a group of connected pixels in an image that share some common property. In our improved movement image each big object moving is associated with a Blob. We use a Blob Detector [2] to obtain the center and the size of the Blobs. We have simplified the image to a set of centers and sizes that we can work with.

Because it is a multi-camera tracking algorithm we need to know that the cars always move from right to left of the frame

- The cars always move from right to left of the frame.
- Each camera inside the parking controls one region of the parking. When we talk about previous camera, we refer to the camera that controls the adjacent region of the parking that is closer to the entrance.

We now start the assignation part of the multi-camera tracking algorithm. If there are two or more Blobs that are very close we assume they form part of a bigger Blob that is fragmented. To calculate this Blob center position we make the weighted mean of the smaller Blob centers using as weight their size. We remove the smaller Blob centers and add the bigger Blob center to the set [3].

After that we need to know which of the center positions correspond to an actual car. We apply the following rules in this order to the position *p*:

- If there were existing cars in the previous frame whose positions are near to *p,* then the closest car has the position *p* this frame.
- If we know that a car is coming from the previous camera and *p* is at the right of the frame, we assign *p* to the car we expect.
- If we know that a parking spot has been freed and *p* is near that parking spot, we assign *p* to the car that was in that parking spot.
- If none of the above is fulfilled, *p* is discarded.

We are advised by the space counting algorithm in the third case and by the license plate algorithm for the second case if it was the camera closer to the entrance. In the rest of the cases the own tracking algorithm is the advisor.

We need to store the car positions to know if in the previous frame there was a near car. We use a dataframe for each camera in which we store the frame, the plate of the car, the position, its previous position, the speed and an indicator of how many times we had to predict that car. This way we can search all the information needed of the cars of the previous frame for the first case. We also have one stack for each camera in which is stored the plate information of the cars that are expected to enter the frame, which solves the problems of the second case.

With this assignation system we avoid assigning a plate to a person or a motorcycle as we need confirmation from more robust algorithms: the license plate and space counting algorithms.

Another problem comes when the car is behind any static object like a tree or a lamppost. When that occurs the Blob disappears or is partially shifted, which cause the car to temporally have no position detected in the frame or a noisy position. The solution we had applied is to correct before storing the center positions of the Blobs or predict them when there are no detection. To correct and predict we have used the alpha-beta filter [4] (1-4). Although is not as good as the Kalman filter, is much less CPU-intensive, which is required for our application, and is easier to implement. As the alpha and beta parameters we have chosen 0.1 and 0.005. The alpha-beta filter requires also the speed of the car, so it is stored in the dataframe.

$$\widetilde{s_n} = s_{pred} + \alpha(s_{meas} - s_{pred}) \tag{1}$$

$$\widetilde{v_n} = v_{pred} + \beta(s_{meas} - s_{pred}) \tag{2}$$

$$s_{pred} = s_{n-1} + \ v_{n-1} \tag{3}$$

$$v_{pred} = v_{n-1} \tag{4}$$

We use correction before storing any position in the dataframe and we use prediction for the cars without any close position detected in the frame. The number of times the car have been predicted is stored in the dataframe so we can know when a car has been predicted too many times. If that is the case we do the following:

- If the car has speed near to zero and is near any free parking place that the space counting algorithm has notified as occupied after being recently free or undetermined, it stores the car plate to that parking spot and discard the car.
- If the car has speed near to zero but it is in the parking lane, we create a notification for the GUI saying that that car is parked in double row. We do not discard this car, but give it speed zero.
- In other case, the prediction has failed as the car continues being predicted when it should have been detected long ago. We discard that car.

Also, if the car is located at the left of the frame, we store its plate in the stack of the next camera and discard the car.

IV. TRACKING TEST RESULTS

We have not been able to test properly the multi-camera tracking algorithm because of the lack of synchronized videos due to errors in the recordings. We have tested the tracking algorithm for one camera and without using checking from other algorithms for 30 cars crossing and parking the D6 region between 7:28:44 to 7:35:17. The results are:

- 76.7% of the cars were correctly tracked from beginning to end.
- 23.3% of the cars were mostly lost due to multiple car movement in the same frame combined with the tree that covers a quarter the frame. There were up to 4 cars moving at the same time.
- 9 false positives due to failing assigning with multiple cars, a motorcycle and people crossing the parking. These errors can be avoided via checking with the other algorithms.

ACKNOWLEDGEMENTS

I wish to acknowledge the supervisors of this project: J. Ruiz and L. Torres for their guidance and motivating us to make such project possible, the professors A. Bonafonte, J. Solé, G. Utrera, J. Fornes and R.Pérez for lending their offices for the camera installation in the D5 and D6 buildings and A. Lázaro, J. Pujal and the Campus Nord UPC staff for giving us the permissions to record.

[**1**] Sutjiadi, Raymond, Endang Setyati, and Resmana Lim. "Adaptive Background Extraction for Video Based Traffic Counter Application Using Gaussian Mixture Models Algorithm." *TELKOMNIKA (Telecommunication Computing Electronics and Control)* 13.3 (2015): 1006-1013.

[**2**] http://www.learnopencv.com/blob-detection-using-opencv-python-c/

[**3**] Gabriel, Pierre F., et al. "The state of the art in multiple object tracking under occlusion in video sequences." *Advanced Concepts for Intelligent Vision Systems*. 2003.

[4] Penoyer, Robert. "The alpha-beta filter." *C User's Journal* 11.7 (1993): 73-86.