Volume 57, Number 3, 2016

ACTA TECHNICA NAPOCENSIS
Electronics and Telecommunications

AUTOMATIC PARKING ACCESS USING OPENALPR ON RASPBERRY PI3

Elena Roxana BUHUS, Daniel TIMIS, Anca APATEAN
Technical University of Cluj-Napoca, Cluj, Romania

Abstract: Automatic parking access using a real time embedded system for Automatic License Plate Recognition (ALPR) and access control is common these days in urban commercial spaces, office buildings and other similar public spaces. This paper provides another method for implementing an ALPR system using an open source C/C++ library called Open ALPR, based on OpenCV and Tesseract-OCR. The fundamental goal of the system is to use image processing to identify every single vehicle that goes in and out through the barrier gate parking system. The hardware is implemented with the latest version of Raspberry Pi, and a Pi Noir camera.

Keywords: *OCR, Vehicle license plate, OpenALPR, Application Programming Interface (API), Raspberry Pi.*

I. INTRODUCTION

The today's science and IT companies are developing research in intelligent automatic systems, with a great impact on people lives, especially in urban spaces. Many intelligent systems today use computer vision as a way to acquire, process, analyze, and understand images and videos from the real world.

OpenCV (Open Source ComputerVision) is a library of programming functions, initially developed at Intel Research Center initiative, to analyze images by computer vision. Starting few years ago, the support for OpenCV is assured by a nonprofit foundation, with the same name [1], being free for both academic and commercial use.

OpenCV can be used on a variety of operating systems (O.S.) including Windows, Android and Unix-like ones and having interfaces for more programming languages, like C/C++, Python and Java.

Image operations like convolution, filtering, detecting foreground/ background regions and depth, extracting features, classifying, tracking by Haar cascades for example are possible by OpenCV [2].

ALPR (Automatic License Plate Recognition) or simply LPR is an image-processing technology used to identify vehicles by their license plates (LP), being popular in various security and traffic applications: it is often implemented in different parking or access control systems to automatically enter pre-paid or authorized members, to calculate parking fee or to provide a proof of parking in case of a lost ticket or even to help in preventing car hijacking (by using an additional driver face image recognition operation). Other applications include border control - in the entry or exits of different countries, traffic control - the vehicles can be directed to different lanes according to their entry permits, to reduce traffic congestions and the number of attendants, or to produce a violation fine on speed or red-light semaphores and others.

The ALPR task can have complex characteristics given the diverse effects in matters of light and speed. Depending on the use of the application, different systems can be deployed, such as smart cameras for traffic surveillance that has the advantage of allowing direct on site image processing tasks [3].

Most of the existing ALPR systems commonly use cores in Matlab for Optical Character Recognition (OCR). This paper provides a method for implementing an ALPR system using an open source C/C++ library called OpenALPR, which is based on OpenCV and Tesseract-OCR. *Tesseract* is one of the most accurate open source OCR engines, being able to work under various O.S.; it is a free software, initially developed at HP labs and being released under the Apache License [4]. The ultimate version V3.04 brings the total count of support languages to over 100 [5]. Still, to deliver accurate results, images given to Tesseract should be properly preprocessed: images must be scaled up, any rotation or skew must be corrected, low-frequency changes in brightness must be high-pass filtered, and dark borders must be removed [6].

Generally, there are two types of applications employing an embedded system (ES): the ones dedicated for Internet of Things and Big Data, where a server located at distance may collect and process some data registered by the ES and the ones employing a powerful ES, with PC-like capabilities, which locally may perform some processing with the collected data.

The second type of applications generally use a SBC (single-board computer) like development platform that use a SoC to perform several functions locally. In our case, the information is extracted from the input data and sent in a compressed form to a central node implemented with a Raspberry Pi platform, hence decreasing demands on both communication and computation infrastructure. Furthermore, ESs are cheaper than general purpose computers and suitable for deployment in harsh environments due to physical robustness [3].

In this sense, focusing upon the automatic parking access ES, one can say that such video surveillance is important not only for access control but also for tracking of stolen cars, even identification of dangerous drivers. The system runs fully autonomous using a Raspberry Pi board equipped with a Pi Noir camera.

In section II, we provide an overview of the embedded

Manuscript received July 19, 2016; revised September 21, 2016

platform and of the software system, while in section III we present some details and results of the implementation and developed tests. Finally, the conclusion and possible improvements are highlighted in section IV.

II. SYSTEM DESCRIPTION

Our system mainly consists of a hardware part and a software part. The hardware is using a SBC platform, i.e. the 3-rd version appeared recently on the market, Raspberry Pi 3 and as a capability, a Pi Noir camera for image deploying. The software part is mainly based on an open source packet-library [7]. Even the application is designed for a complete system, thus including to lift up or down the barrier, in this paper we mainly focus on the part concerning the Raspberry Pi board.

II.A Embedded Platform Description

The ES we used, consists of a Pinoir Camera for image acquisition and a Raspberry PI3 platform for image processing; besides, a barrier control algorithm is also used, but is not detailed in this paper. In Figure 1, a part of the setup used for tests is presented.

On the Raspberry Pi 3 platform, a Broadcom SoC, which includes an ARM compatible CPU and an on chip Graphics Processing Unit GPU (a VideoCore IV) is located. The CPU speed is ranked at 1.2 GHz, with an on board memory of 1GB RAM and some consistent cache for the quad-core 64-bits CPU. Table I compares the most important capabilities of some Raspberry Pi models. As one could notice, the CPU speed ranges from 700 MHz to 1.2 GHz, the on board memory range from 256 MB to 1 GB RAM, and most boards have between one and four USB ports. HDMI, composite video output and a 3.5 mm jack for audio are also present on the board. There is also a number of 26 or 40 GPIO pins which support common protocols like I2C. Some models have an Ethernet port and the Raspberry Pi 3 has also on board WiFi 802.11n and Bluetooth.

Using a Secure Digital SD card, the Raspberry Pi platform stores the O.S. and program memory in either SDHC or MicroSDHC size [8]. The Raspberry Pi foundation provides Debian and Arch Linux for download [9]. For this project, a Debian distribution was used meaning Raspbian, dedicated for Raspberry Pi platforms.

This type of platform has a wide area of possible applications [10], [11], [12].

Three years ago, with a relatively low-cost, the Raspberry Pi Noir Camera board was first launched on the market. It has no Infrared IR filters (thus the name) and provides great capabilities also for low light conditions. A 5MP Omnivision 5647 Camera Module provide still pictures with resolution of 2592x1944 pixels, and the later version launched in May 2016 has 8 MP [9]. Video stream are also supported: 1080p at 30fps (frames per second), 720p at 60fps and 640x480p 60/90 [9]. The advantage for the chosen application is that the camera is fixed and frontal to the barrier. It is used for close distance object detection, in this context vehicle license plates.

The camera board is equipped with a flexible flat cable that plugs into the CSI connector of the board, which is located between the Ethernet and HDMI ports, as shown in Figure 1.

Table I. The main characteristics of Raspberry Pi boards

Gen.	V.1		V.2	V.3
Model	A, A+	B, B+	Model 2	Model 3
USB Hub	No	Yes, with 2 (resp.4) USB ports	Yes	Yes
Ether-net	No	Yes	Yes	Yes
SoC: BCM 283x with CPU	2835		2836	2837
	32b,700MHz ARM1176 JZF-S		32b,900MHz 4-cores ARM Cortex-A7	64b,1.2GHz 4-cores ARM Cortex-A53
CPU cache	L1: 16KB, L2: 128KB (L2 – mostly by GPU)		L1: 16KB, L2: 256KB (shared)	L1: 16KB, L2: 512KB (shared)
RAM on board	256 MB	512 MB	1 GB	1 GB
GPIO pins	40	26, resp.40	40	40

Considering the hardware implementation of the system, this technology does not need any installation per car, the prototype being only mounted on the barrier or near it.

The system may use illumination (such as infrared light source if there is no source of light) and the camera will take the image of the front or rear of the vehicle. In this way, a monochrome image result, since the infrared spectrum is above the normal color spectrum. The image-processing software analyzes the image and extracts the LP information. If it recognizes the vehicle number, it opens the barrier.

The LP recognition task is also called in different references as Automatic Vehicle Identification, Car Plate Recognition, Automatic Number Plate Recognition, Car Plate Reader or even Optical Character Recognition for Cars and possibly others [13].

Besides, the system must consider that different country standards may exists as concern the vehicle license plates: they may differ in form, shape and material. Therefore, the LPR systems are country specific and are adapted to the country where they are installed and used. Also, the process may be inappropriate for some plates if they present additional information (written inside the LP area or attached to the plate).

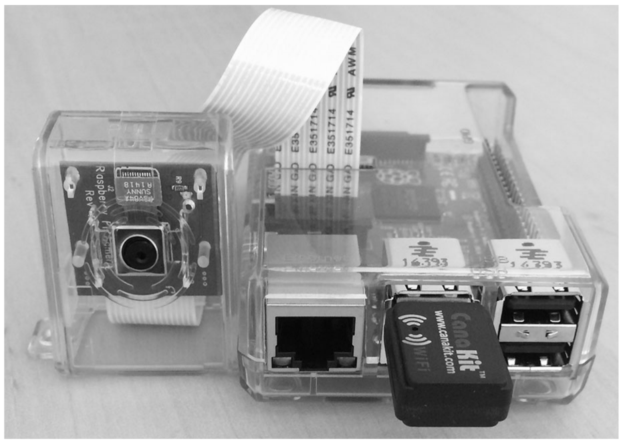


Figure 1. Raspberry PI3 with Pi Noir camera setup

II.B. Software System Overview

The proposed system functions in the following manner:

while the vehicle approaches the barrier, the *LP recognition* unit *automatically* reads the LP registration number, compares it to a predefined list and opens the barrier if there is a match. Thus, the software framework consists of two modules: one to accomplish the ALPR task and another one to provide the barrier control (the hardware corresponding part not being presented in the current paper).

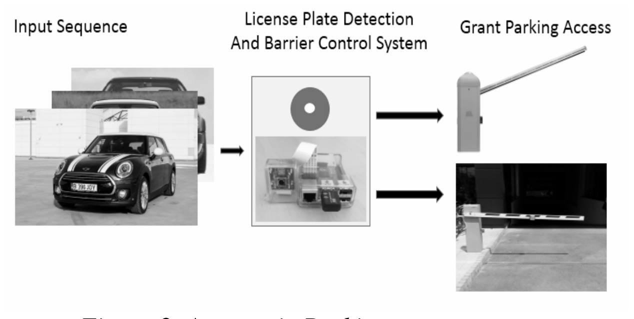
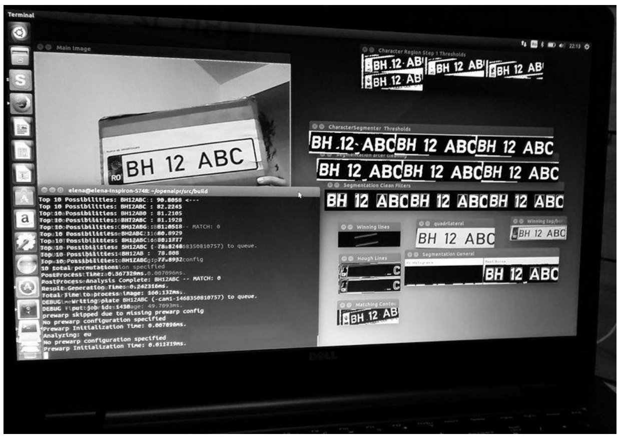


Figure 2. Automatic Parking access concept

The ALPR module provides input for the control algorithm that will take the decision if the barrier will authorize the access for the current vehicle (present in front of it) or not. The focus in this paper is upon the ALPR module instead of the barrier control module.

The OpenALPR library, available in C/C++, has bindings in C#, Java and Phyton. Furthermore, the software can be used as standalone black box to process video streams and make the data available to another subsequent system. Pre-compiled binaries are available for 32/64-bit Windows and Ubuntu Linux. The Application Programming Interface (API) operates as a pipeline in matter of processing stages, and the output of each stage can be viewed in the debug mode, as illustrated in Figure 3.

There are 8 pipeline stages, as follows: 1. LP image detection, 2. image binarization, 3. char analysis, 4. compute LP image edges, 5. LP image deskew, 6. character segmentation, 7. OCR, 8. post processing. To easily explain in text, we grouped the 8 pipeline stages in 3 main modules, as shown in Figure 4.


BH 12 ABC
Main Image
Character Region Step 1 Thresholds
CharacterSegmenter Thresholds
Segmentation Clean Filters

```
elena@elena-Inspiron-5748: ~/openalpr/src/build
Top 10 Possibilities: BH12ABC : 90.8058 <---
Top 10 Possibilities: BHI2ABC : 82.2245
Top 10 Possibilities: BH12AB0 : 81.2105
Top 10 Possibilities: BHI2ABC : 81.1928
Top 10 Possibilities: BH12ABG : 81.0518
Top 10 Possibilities: BH2ABC : 80.8929
Top 10 Possibilities: 8H12ABC : 80.1777
Top 10 Possibilities: 5H12ABC : 78.8248
Top 10 Possibilities: BH12AB : 78.808
Top 10 Possibilities: BH1ZABG : 77.8932
10 total permutations
PostProcess Time: 0.367329ms.
PostProcess Analysis Complete: BH12ABC -- MATCH: 0
Result Generation Time: 0.242316ms.
Total Time to process image: 166.132ms.
DEBUG Writing plate BH12ABC (-cam1-1468350810757) to queue.
DEBUG put job id: 1430
prewarp skipped due to missing prewarp config
No prewarp configuration specified
Prewarp Initialization Time: 0.007096ms.
Analyzing: eu
No prewarp configuration specified
Prewarp Initialization Time: 0.011719ms.
```

Figure 3. Processing pipeline stages in debug mode

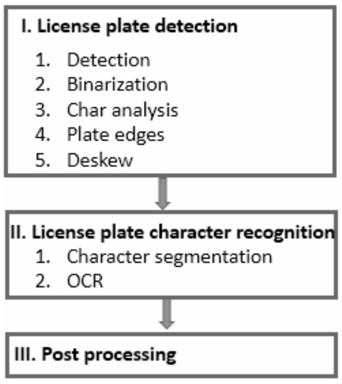


Figure 4. Block schematic of the ALPR algorithm

As presented in Figure 4, the API to accomplish the ALPR task can be seen as divided in three main modules: license plate detection, license plate character recognition and post processing, according to the pipeline stages.

The ALPR algorithm starts with the detection stage, where potential LP regions are identified. Next, in the binarization stage, the LP region image is converted into an image with only black and white pixels, while in the char analyses stage, the character-sized “blobs” in the LP region are identified. In the following step, the edges of the LP are identified and provided as input to the deskew stage, where an image transformation is applied to improve readability, view based on the ideal LP size.

After the LP area was detected and the data was prepared as shown in the first module from Figure 4, the next module performs the recognition of the characters inside the LP image. Thus, the character segmentation stage isolates and cleans up the characters, setting proper input for the OCR stage, where it analyses each character image and provides multiple possible letters. In the last stage or module, the one of post processing, the algorithm determines the best possible LP letter combinations, providing the information to the Beanstalk queue as JSON data. These are detailed in the next sections.

Module 1: LP detection

LP area detection

For each input image the detection phase is first applied. The Local Binary Patterns (LBP) algorithm is used for classification, to find possible LP regions and the dimension values (x, y, width, height).

The basic idea of LBP computation is to summarize the local structure in an image by comparing each pixel with its neighborhood. Thus, the LBP algorithm considers a pixel as center and threshold its neighbors against. If the intensity of the center pixel is greater or equal to that of its neighbor, then denote it with 1 and 0 if not. In this sense, each pixel will be represented in binary, after 8 compare computations, for example like 11001111.

With 8 surrounding pixels, there are 256 possible combinations, called LBP features or sometimes referred to as LBP codes [17]. The detection phase is usually the most processing-intensive one.

To improve the system performance, this operation can be GPU accelerated [9].

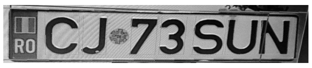


Figure 5. An example of LP input used for ALPR testing

LP image binarization

This phase occurs once for each possible LP region detected in the first step. Within this phase, multiple binary images are created for each LP region, these helping to provide the best possible chance of finding all the characters inside the LP region. For example, if the image is too dark or too light, a single binarized image may miss some characters. The method used to accomplish this operation is inspired by [15], [18] but with various parameters. Each of the binary images are processed in subsequent phases, as presented in the following.

Character Analysis

Within this phase, the goal is to find character-sized regions in the LP region. This is done by first finding all connected blobs in the LP region. Second, it searches for blobs that are roughly the width and height of a LP character and have tops/bottoms that are in a straight line with other blobs of similar width/height. In each region, this analysis is performed multiple times: it starts by looking for small characters, while gradually searches for larger ones. If there are no findings in the region, then the region is thrown out and no further processing takes place. If it finds some potential characters, then the character region is saved and further processing takes place.

Compute LP image edges

The next phase is to find the edges within the LP image. If in the detection phase, a region is provided where a possible LP exists, this being larger or smaller than the actual plate, in this stage the goal is to find the precise top/bottom/left and right edges of the LP. The first step is to find all of the Hough lines for the LP region, by processing the LP image and by computing a list of horizontal and vertical lines.

In the second step, using this list as well as the character height (computed in the third stage of Character Analysis) the likeliest LP line edges are found. The algorithm used in this stage employs a number of configurable weights to determine which edge would be a good fit. Based on the default edge, meaning the ideal width/height of the LP, it identifies a good match. The theory used is as presented in [19], already implemented in OpenCV by functions HoughLines and HoughLinesP.

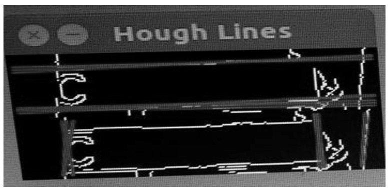


Figure 6. An example of LP edges in the output stage

Perform LP image deskew

At this point, using the LP edges information, the deskew stage remaps the LP region to a standard size and orientation. In an ideal approach, this should provide a correctly oriented LP image, no rotation or skew being present in this stage.

Module 2: LP characters recognition

Character Segmentation

The character segmentation phase, has the goal to isolate all the characters that make up the LP Image. It uses a vertical histogram to find gaps in the LP characters. In this phase, also the character boxes is cleaned up by removing small, disconnected speckles and disqualifying character regions that are not tall enough. It removes "edge" regions so that the edge of the LP to not be inappropriately classified as a '1' or an 'I'. This stage also uses filters to clean the image after segmentation, so that the provided output will be more accurate for the next processing stage.

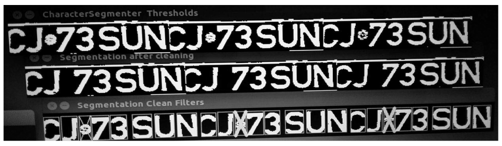

CharacterSegmenter Thresholds
Segmentation after cleaning
Segmentation Clean Filters

Figure 7. An example of segmentation output before cleaning filters applied, and afterwards

OCR (Optical Character Recognition)

The OCR phase, based on the Tesseract OCR engine, analyzes each character independently. For each character image, it computes all possible characters and their confidences.

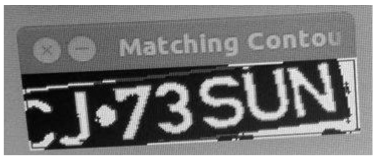


Figure 8. An example of LP matching number as provided by the OCR stage

Module 3: Post processing

The OCR stage will provide a list of all possible OCR characters and the confidences associated to those, and the post processing stage will identify the best plate letter combinations.

In this stage, all characters below a particular threshold will be disqualified. At this stage, the region validation is handled if requested. For example, if in the OpenALPR is specified that it is a "Missouri" plate, then it will try and match the results against a template that matches the Missouri format (e.g. [char][char][number]-[char][number][char]) [7].

III. Software architecture overview and configuration

The *ALPRD module* of the OpenALPR application enables the use of the ALPR library on a mjpg video stream in order to achieve real-time LP numbers recognition in real world situations.

The module works by first analyzing the configuration files and providing runtime parameters, checking if present and then connecting to a valid video-camera/mjpg video stream. The configuration files required to accomplish this operation are *openalpr.conf* and *alprd.conf* which are

responsible for the behavior of the alpr *library*, respectively of the alprd *module*.
The configurable options of the ALPRD module as mentioned in *alprd.conf* are used to:
- set the LP format the users wishes to identify, the available possibilities for the LP format are the US or Europe format;
- define a site name for the provided stream which helps to sort or label saved results from multiple devices to a server;
- define the interested stream to be processed, providing the IP address if it is a network or IP camera stream, or using the web cam keyword to enable the application to manage the video-stream from a connected video-camera via using the OpenCV library;
- set the number of results provided by the OCR library Tesseract to be considered;
- configure if images containing identified license plates should be stored or uploaded via POST to a remote server and also configuring the destinations for both options.

After the runtime dependencies and parameters have been analysed and set, the daemon enters within a *for loop* that will keep it running for as long as the provided stream is active. Inside the *for cycle,* a new process is forked, that will create two new threads for the image processing and recognition and for uploading/storing the results to a location set in the *alprd.config* file.

The stream information is retrieved and the first thread is created to run a *streamRecognitionThread* function which in turn will run the ALPRD library *recognition function* and process its results by converting the information to JSON format, as illustrated in Figure 9.

After this thread is finished, the upload data thread takes the results and sends them via POST to a server if enabled and uploads the results to a Beanstalk message queue where they can be consumed by the end user.

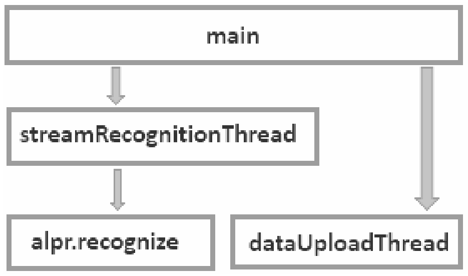


Figure 9. Software implementation for ALPR on Raspbian

The maximum resolution used for image captures is 1280x720 pixels, while the threshold detection for contrast detection was set to a value of 0.3. The minimum distance of the car in front of the camera was calibrated to 1m. The maximum range distance from the camera was observed while testing between 2-3m, however this depends on the lighting conditions. If these ranges are not meet, experiments showed that the LP recognition operation can be inaccurate, even from the beginning, e.g. no detected plate. If the light condition affect too much the contrast of the captured image, no plate is detected.

The level of confidence for the OCR stage was set to 10, meaning that from 10 possible proposed identified LP numbers. The candidate with the highest level of confidence will be considered as the correct result.

An example of the performance of the application at each pipeline stage, is provided in Table II. Processing speed depends much upon the contrast level identified in the input capture, meaning that a better contrast will provide a lower processing time. The extra added delays, given by intermediate processing between stages are not listed in the table. The overall effort was summarized and listed, as presented in the last line of Table II.

Table II. Example of processing performance for ALPR

Processing stage	***Time***
Capture contrast	81.5 ms
Character Analysis Time	168.1 ms
High Contrast Detection Time	1.0 ms
Plate Lines Time	55.8 ms
Plate Corners Time	22.6 ms
Deskew Time	9.6 ms.
Character Segmentation Time	124.2 ms
OCR Time	129.8 ms
Post processing time	4.4 ms
Total Processing time	**800.2 ms**

III. EXPERIMENTAL RESULTS

Considering some constraints of the application, due to the fixed camera used for close distance LP recognition, it was assured that distance range would be most of the time meet by the driver, as mentioned in section II.B.

Some basic tests were run on random test images, five at number, as shown in Table III, to check the processing speed and detection success over the contrast parameter of the images. The contrast parameter for each image was measured using ImageJ 1.51d, an open platform for scientific image analysis [16]. ImageJ is an open source image processing program designed for scientific multidimensional images. It is highly extensible, with thousands of plugins and macros for performing a wide variety of tasks, and a strong, established user base. The results are represented in Table III and Figure 10 illustrate the prototype mounted on a real barrier.

Table III. Testing more images- the obtained results

Image	***Contrast Value***	***Total Processing time [ms]***
Image1	95.8	1003.5
Image2	81.5	800.2
Image3	152.3	2740.1
Image4	122.3	3019.2 – no LP found
Image5	142.6	826.4

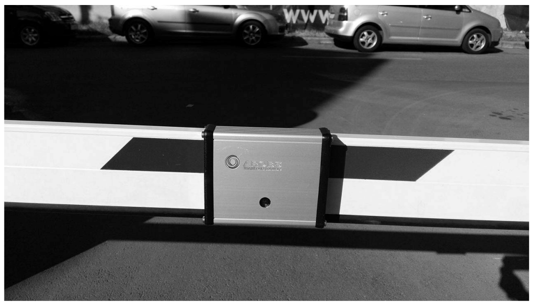


Figure 10. The prototype mounted on a real barrier

IV. CONCLUSION

In this paper, a real-time embedded system used for automatic parking access using openALPR API over Raspberry PI 3 platform was presented. The system is based on a Raspberry Pi platform, a hardware which has evolved through several versions and feature variations in CPU capabilities, memory capacity and peripheral-device support. The system operates on image frames acquired with a Pi Noir camera board, without any additional sensor input. The camera has no Infrared IR filters, providing thus great capabilities also for low light conditions. The performances of the system are more than enough for the application itself required in matter of response time and compensation of low image resolution by considering the classification results of subsequent frames. Considering the complete integration on an embedded device, the system operates autonomously, reporting only the finals classification results to the control module for the barrier control.

The openALPR library proved to be quite stable and ready to use for this type of applications. It can be customized using the configuration option together with training capabilities for new data sets, for a certain country, increasing thus the accuracy of the system. Even so, the obtained results are quite good, meeting the expectations. For further development, it would be interesting to extend the application so that, the vehicles that are new for the parking area should be added to the list in an automated manner. One solution would be via a mobile application, that sends the new license plate number as SMS. This would imply that the user will have to install the application on a smart phone, and according with some defined steps, to be able to automatically add the license plate number of his vehicle to the list. After receiving a confirmation SMS (where other possible verifications to be performed), he will be able to grant the access to the parking area.

Much of such systems are in numerous installations and the number of systems are growing exponentially, efficiently automating more and more tasks in different market segments. The LPR unit checks if the vehicle appears on a predefined list of authorized cars, and if found - it signals to open the gate or lift the barrier by activating its relay.

REFERENCES

[1] www.opencv.org [Online, accessed 19 July 2016]

[2] J. Howse, OpenCV Computer Vision with Python, Packt Publishing, 2013, ISBN: 978-1-78216-392-3

[3] Pratiksha J.,Neha C., Vaishali G. “Automatic License Plate Recognition using OpenCV”, International Journal of Computer Applications Technology and Research, vol.3,pp.756-761,2014

[4] Smith, Ray. "An overview of the Tesseract OCR engine.", 2007

[5] wikipedia, http://www.sk-spell.sk.cx/update-of-language-files-for-tesseract-ocr-304, [Online, accessed 19 July 2016]

[6] https://github.com/tesseract-ocr, [Online, accessed 19 July 2016]

[7] https://github.com/openalpr/openalpr, [Online, accessed 19 July 2016]

[8] https://en.wikipedia.org/wiki/Raspberry_Pi, [Online, accessed 19 July 2016]

[9] http://www.modmypi.com/raspberry-pi/camera/raspberry-pi-noir-infrared-camera-board-5mp-1080p-v1.3, [Online, accessed 19 July 2016]

[10] Apatean A., Dunca F., “An Intelligent Eye-Detection Based, Vocal E-Book Reader for the Internet of Things”, Acta Tehnica Napocensis. Electronics and Telecommunications, No. 2/2016, Vol. 57, ISSN: 1221-6542

[11] Sai Yamanoor, Srihari Yamanoor, Raspberry Pi Mechatronics Projects HOTSHOT, 2015, pp. 606, ISBN 978-1-84969-622-7

[12] A.K. Dennis, Raspberry Pi Home Automation with Arduino, 2015, pp. 148, ISBN 978-1-78439-920-7

[13] http://www.licenseplaterecognition.com/, [Online, accessed 19 July 2016]

[14] https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/, [Online, accessed 19 July 2016]

[15] J. Sauvola and M. Pietikainen. Adaptive document image binarization. Pattern Recognition, 33(2):225– 236, 2000.

[16] http://imagej.net/Welcome [Online, accessed 19 July 2016]

[17] Ahonen, T., Hadid, A., and Pietikainen, M. *Face Recognition with Local Binary Patterns.* Computer Vision - ECCV 2004 (2004), 469–481

[18] C. Wolf and J. Jolion. Extraction de texte dans des videos: ´le cas de la binarisation. In 13eme congres francophone de reconnaissance des formes et intelligence artificielle, volume 1, pages 145–152, Jan. 2002

[19]http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html