International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-1, November 2019

A Student Smart Parking Solution using Raspberry Pi

Răzvan Vîlceanu, Andrei Ternauciuc, Mihai Oniţa

***Abstract*: The concept of a smart city was born out of the need to provide improved quality of life to citizens in various fields, such as environment, governance, education, economy, infrastructure, transportation, traffic, and parking. There are many countries, many players and many projects already finished or underway. The city of Timisoara has already taken the first steps towards a smart city. It has high potential in this regard, but for the moment is still lacking in some key areas. For example, Timisoara has excessive traffic and usually overcrowded parking lots without some efficient monitoring solutions. As a result of this fact, a team made up of a Master student in his second year of studies, and two faculty coordinating members propose a parking management solution based on a very restrictive budget.**

The low-cost prototype we developed for testing purposes contains a mockup parking platform with 8 parking spaces, one Raspberry Pi Zero W micro-computer, and a camera module attached to the Raspberry Pi. The idea is to broadcast live feed towards a computing platform tasked with identifying the available parking spots and consequently transmit this information in real-time, via the Internet, to interested parties. We tested and compared two detection algorithms based on corner detection of the parking spots, and the detection of circles placed inside the parking spot, respectively. These algorithms were implemented using open-source technologies such as Python version 3.6.5, OpenCV 3.4.0 specialized modules and functions, digital processing (NumPy), chart plotting (Matplotlib) and data formatting (JSON). The main functions we used were: cv2.imread, cv2.imshow, cv2.imwrite, cv2.VideoCapture, cv2.line, cv2.circle, cv.puttext, cv2.rectangle, cv2.elipse, cv2.blur, cv2.gaussianblur, cv2.canny, cv2.houghlinesp, cv2.houghcircles, cv2.hough_gradient and cv2.selectROI.

The solution we proposed is focused on improving the way the Politehnica University Timisoara's staff and students approach parking. It can reduce traffic congestion and the level of CO2 or other pollutants around University buildings and surrounding areas. Overall traffic would be greatly diminished because the driving lanes would mainly be used for transportation, instead of low-speed cruising in search of a free parking space.

Revised Manuscript Received on November 30, 2019.
***** Correspondence Author
Răzvan Vîlceanu*, Department of Communication, Politehnica University Timisoara, Timisoara, Romania. Email: razvan.vilceanu@student.upt.ro
Andrei Ternauciuc, Department of Communication, Politehnica University Timisoara, Timisoara, Romania. Email: andrei.ternauciuc@upt.ro
Mihai Oniţa, Department of Communication, Politehnica University Timisoara, Timisoara, Romania. Email: mihai.onita@upt.ro

***Keywords*: Smart Parking, Smart Cities, Computer Vision, Raspberry Pi, Python, Open CV**

I. INTRODUCTION

The smart city concept is aimed at managing cities (urban areas) in a modern way, using the latest technical means offered by advanced technologies (including IT) [1]. The smart city concept originated from various definitions including those of the "intelligent city", "information city", "knowledge city", "digital city" and "ubiquitous city". Other popular terms include: "creative city", "green city" and "clever city" [2], [3].

The concept of smart city was born to provide improved quality of life to citizens [4]. It is generally assumed that cities can be defined as smart if they have the following elements: smart governance and smart education, smart economy, smart healthcare, smart building, smart infrastructure, smart transportation / traffic / parking, smart technology, smart energy, smart mobility, smart environment, smart citizen, smart living etc. [1], [5]. The key idea is to integrate information system services of each domain of the city to provide public services to citizens efficiently and ubiquitously [4].

There are numerous challenges in the realization of a smart city: from the technical infrastructure that needs to be put in place, to the adoption of the resulting system by the citizens, and to various technical and non-technical obstacles that need to be dealt with [4].

There are many countries, many players and many projects ongoing. Here are several examples according to Prakash [6], Kola-Bezka [2] and Garfield [7]:

- Amsterdam: successful incubator programs like Startup Delta and Startup Amsterdam;

- Brasov: Business Intelligence Reporting System brings together all the data from every partner, contractor and subcontractor who reports to the city hall;

- Constanta: street lightning and 50 parking lots that are operated by a single control panel;

- Copenhagen: a healthy startup ecosystem, a large number of Wi-Fi hotspots, low amount of traffic congestion, clean energy with the goal of being 100% carbon-neutral by 2025;

- Geneva: energy-efficient infrastructure in its buildings and public transit;

- Melbourne: 4G connectivity technology

- Singapore: cost-efficient public transport networks;

- Stockholm: city buses and trains run on clean fuels, renewable power sources account for 52% of Swedish energy production;

- Tokyo: over 100 train lines and 14 billion passengers per year;

- Zurich: urban plan that includes a high percentage of green space.

A. Smart Parking

In the context of the smart city, smart parking is a necessity because searching for a parking space in towns has become such a frustrating activity for drivers. There are cities that implement modern technologies in managing parking, but many drivers are not aware of these parking systems. There are cities that lack the budget to implement an efficient solution. To overcome these problems, as well as to save the fuel they waste, the recommended step is to implement a smart parking system [8] and to improve the awareness of such a solution. Smart Parking is one of the most popular and fastest growing smart city elements across the world. Airports, universities, shopping centers and city garages are just a few entities that have begun to realize the significant benefits of automated parking technology [9].

There are different methodologies used for smart parking, Kharde [9] and Fraifer [10] said that the solutions are based on: agent modelling, Fuzzy logic, vehicular to infrastructure communication (V2I), Global Positioning Systems - GPS, Computer Vision, RFID technology, wireless sensor networks systems or hybrid solutions, M2M, IoT Systems.

According to Ahad [8] and Saric [11] the potential advantages of implementing a Smart Parking System are numerous, and they include:

- Efficient use of vehicle travel;

- Reducing traffic congestion;

- Optimizing parking space usage;

- Guiding residents and visitors to available parking;

- Accurately predicting and sensing spot/vehicle occupancy in real-time spots;

- Reducing the level of CO2 and other pollutants;

- Parking pricing strategies manipulation considering (real-time) demand;

- Fewer vehicles parked illegally by the roadside;

- Enabling intelligent decisions using data, real-time status applications and historical analytics reports;

- Improvements in general mobility and future development of smart city mobility solutions.

The applications and possible revenue/benefits of the use of smart parking were discussed in detail over the last few years. There are several approaches that can be found in literature.

Anusooya [12] proposes a solution that includes IR Sensors, Raspberry Pi and RFID Sensors which communicate with the Google App Engine and is rendered on various Applications such as the Website, Twitter and the Mobile Application. In the same direction Vimal [13] presents a prototype with Raspberry Pi 3 Model B, USB Camera, RFID Reader RC522, LED display 16*2, Servo Motor, Bread Board, jumper wires.

Gupta [14] and Sabbea [15] systems have Ultrasonic sensor, Arduino Uno, ESP8266-01 Wi-Fi Module, Cloud server and User-End Applications on Smartphone.

A prototype system composed of the MSP430F2274 microcontroller, an AAH002 analog magnetic sensor, an LDR NORPS-12 optical sensor and a Zigbee transceiver (ETRX2) is the work of Dhaou [16].

The paper written by Kamble [17] aims to propose micro-controller based parking lots and GSM used for monitoring the available spaces through which the reservation is made with the help of an Android application for the users. The interface between all the components is covered by the PIC16F87XA. The actuation of the physical process is done through the motors. The user interaction comprises of the Android app and the Keypad. The communication of the whole system is managed by the ESP8266EX module.

Saric [11] describes parts of smart parking solutions developed for Dubrovnik and it contains software for Web and mobile application and hardware sensor nodes and routers on the hardware side.

Marcu [18] from "Lucian Blaga" University of Sibiu (LBUS) Romania describes a hardware/software embedded system for managing the institution's parking lots.

II. NECESSITY AND PURPOSE

The city of Timisoara has already taken the first steps towards a smart city. It has high potential in this regard, but at the moment suffers from excessive traffic, insufficient and usually overcrowded parking lots, as well as „chocked" streets and junctions, leading to cars parked on sidewalks and green spaces. The same applies to the Politehnica University Timisoara's staff and students' parking needs.

Private parking systems are few and require either high maintenance costs, or prohibitive infrastructure modifications, and therefore have had limited success in mitigating this problem. Public parking are overcrowded and do not usually offer monitoring solutions. The few exceptions employ access barriers or sensors mounted beneath each parking space. The time spent searching for a parking spot can differ according to time of day and neighborhood, but the main issue of this process stems from the extra traffic being generated. For instance, drivers enter a parking lot hoping to find a free parking space; when they can't find one, they leave and head towards a different parking lot, and when they cannot find one there either, they return to the first one, hoping that in the meantime, a space might have been vacated. It this scenario, the vehicle contributes to the existing traffic and the mounting degree of pollution (not to mention the driver's wasted time). Overall traffic would be greatly diminished if few to no such cases existed, since the driving lanes would only be used for getting from point A to point B.

Research on this subject revealed an overview for this project which aims to develop a low-cost prototype containing: a video-camera aimed at a parking lot, broadcasting its live feed towards a computer tasked with identifying the available parking spots, and consequently transmit this information in real time, via the Internet, to interested parties. This prototype ultimately consisted of a miniature platform containing 8 parking spaces, surveyed by a Raspberry Pi Single-Board Computer (SBC), resulting an estimated cost of 50 euros.

III. THE INTELLIGENT PARKING SYSTEM PROPOSAL

A. The main hardware and software components of the system

The proposed solution employs a reduced number of low-cost devices (a Raspberry Pi and an attached video camera), as well as a detection algorithm based on open-source technologies such as Python version 3.6.5, OpenCV 3.4.0 specialized modules and functions, digital processing (NumPy), chart plotting (Matplotlib) and data formatting (JSON).

The Raspberry Pi Zero W micro-computer

The Raspberry Pi is a Linux-based single-board computer, which we used as a lightweight low-power server system. The Zero W version of the Raspberry Pi was launched in February 2017, and features the following hardware specifications [19]: 802.11 b/g/n wireless LAN, Bluetooth 4.1, Bluetooth Low Energy (BLE), 1GHz single-core CPU, 512MB RAM, Mini HDMI, 2X MicroUSB (power + data) and Raspbian as the main Linux-based operating system [20]. It was tasked with the live feed video processing (via the Motion software library) of the signal coming from the camera attached to its USB port. The feed was also broadcast through the Pi's TCP 8081 Port.

The camera module attached to the Raspberry Pi

The official camera module was launched by the Raspberry Pi Foundation in 2013 under the name Camera Module v1, alongside the NoiR infrared version. This module offers an alternative to the webcams attached to a PC, as well as providing the opportunity to turn the Raspberry Pi into a full-fledged surveillance camera, with motion detection, live streaming and cloud storage [21]. The hardware specifications include [22]: $25 net price, 25 × 24 × 9 mm size around and 3g weight, 5 Megapixels resolution with 1080p30, 720p60 and 640 × 480p60/90 as video modes, OmniVision OV5647 sensor with 2592 × 1944 pixels resolution and 3.76 × 2.74 mm sensor image area and 35 mm full-frame SLR lens equivalent. Also the camera has the V4L2 driver available Linux integration and API Open MAX IL for programming.

The OpenCV library

The main OpenCV (Open Source Computer Vision Library) functions we used for the development of the detection algorithm included: cv2.imread, cv2.imshow, cv2.imwrite, cv2.VideoCapture, cv2.blur, cv2.Canny, cv2.HoughLinesP, cv2.HoughCircles, cv2.hough_gradient and cv2.selectROI [23], [24].

OpenCV functions (description and parameters) are:

- cv2.imread ('imagine.jpg', 0): loads the image data into the code for processing; has two parameters: the name of the image, and the color information (0 - black and white, 1 - color);

- cv2.imshow ('imagine', img): image visualization, with two parameters: the name of the windows where the image needs to be displayed, and the variable which currently stores the image data;

- cv2.imwrite ('imagine.png', img): storing the image in the current folder. The first argument is the name of the file to be written, and the second is the source of the image (variable name);

- cv2.VideoCapture(0): continuous image capture, with the 0 parameter indicating that the camera is embedded;

- cv2.line(), cv2.circle(), cv.puttext(), cv2.rectangle, cv2.elipse(): object placement over the captured images, with the purpose to highlight certain areas (lines, rectangles, circles, ellipses, text, etc). Parameters include the image to be modified, the coordinates of keypoints, the color and with of the resulting lines;

- cv2.blur(img, (5,5)): image pre-processing with the purpose of smoothing the sharp edges. The two parameters are the working image, and the size of the nucleus matrix for which to calculate the average pixel color;

- cv2.gaussianblur(): applies a gaussian blur on the image;

- cv2.canny(img, 100, 200): edge detection for the objects inside the image, based on the color contrast of neighboring pixels. The result of this function is an image with black background and white contours. The first parameter is the working image, while the 2nd and 3rd parameters are the lower and upper limits for pixel color differences;

- cv2.houghlinesp(): line detection capabilities. The result contains the coordinates for starting and stopping points for lines (x1,y1), and (x2,y2), the minimum length of the lines (minLineLength) and the maximum distance between the lines (maxLineGap). The lines which were detected in the original image can be reconstructed in a blank image;

- cv2.houghcircles(): extraction of circled from the working image, with results containing the center coordinates (a, b) and radius (r). Parameters include minRadius and maxRadius, setting lower and upper limits for circle detection;

- cv2.hough_gradient: provides information on the edge gratient for the circles found with cv2.HoughCircles();

- cv2.selectROI(img): this function allows selecting a region of interest in an image, similarly to the Crop tools in graphics editing software. The result is defined by four values: the coordinates x and y, the width and the height of the selected region. This selection can be reconstructed as a new image using the data returned by the function.

B. The developed algorithms and configuration steps

Our research used a three-pronged approach, by focusing on the implementation of different detection algorithms and choosing the most effective one.

For the first approach, the manual detection algorithm for testing the availability of a parking space is set manually, and requires the following steps:

- Loading the image data:
img=cv2.imread('parking.jpg',1); #Reading the image

- Initiating the region of interest selection process:
r=cv2.selectROI (img); #Storing the region of interest into r variable

- Creating the target image by using the portion of the image previously selected:
sub_img=[int(r[1]):int(r[1]+r[3]),int(r[0]):int(r[0]+r[2])] #Generating a separate image out of the r variable

- Edge highlighting for sub_img:
edg= cv2.Canny(sub_img,100,150) #Exposing the edges of the image stocked in r variable and storing it into edg variable

- Count the number of pixels from the edg variable:
pix=cv2.countNonZero(edg) #Counting the white pixels of edg variable and storing them into pix variable

- Validating the spot as either free or occupied:
if pix>1500:
#draw a red rectangle if the number of pixels exceeds 1500
cv2.rectangle(im, (int(r[0]), int(r[1])), (int(r[0] + r[2]), int(r[1] + r[3])), (0, 0, 255), 3)
else: cv2.rectangle(im, (int(r[0]), int(r[1])), (int(r[0] + r[2]), int(r[1] + r[3])), (0, 255 , 0), 3) #draw a green rectangle if the number of pixels is bellow 1500

The image bellow includes the decision for each parking spot: a green rectangle if the spot is free, and a red rectangle if the spot is taken.

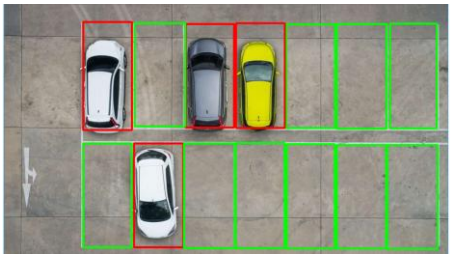

Fig. 1.The result of the manual algorithm

The process of manually selecting each parking spot was considered cumbersome, therefore other automated detection methods were found and tested, and the manual solution henceforth ignored.
We started by detecting the vertices for each parking spot, storing the coordinates and using them to trace red or green rectangles. For this method we implemented the following steps:

- Image loading:
img= cv2.imread('parkinglot.jpg',1) #Reading the image

- Converting the image to black and white:
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#Storing the black and white image into the gray variable

- Edge detection
edg= cv2.Canny(sub_img,100,150) #Exposing the edges of the image stocked into the grey variable

- Line detections:
lines=cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength=0,maxLineGap=3) #Using HoughLinesP function to detect the lines in the image

- Drawing the lines on a black image:
contours=np.zeros((height, width,3),np.uint8) for line in lines:
x1,y1,x2,y2 = line[0]
cv2.line(contours,(x1,y1),(x2,y2),255,2) #Drawing each line on a separate black image

- Vertices detection
corners= cv2.cornerHarris(gray,2,3,0.04) #Detecting the vertices on the new black image
corners= cv2.dilate(corners, None) #Enlarging the found vertices

- Vertices highlighting using the color red:
*img[corners>0.01*corners.max()]= [0, 0, 255]* #Coloring the vertices found and enlarged in red to be easy to spot

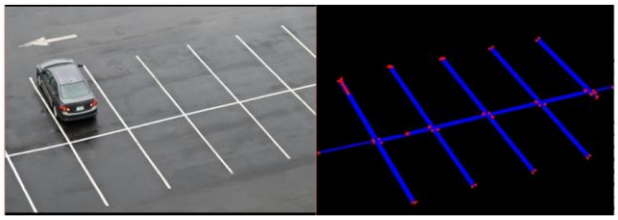

Fig. 2.The result of vertices detection, combined with line detection

Detecting vertices is not always sufficiently precise and can yield false-positive results. The vertices detected at the end of the lines and at the intersections are perfectly suited to make this algorithm work efficiently, however the vertices detected along the lines affect the algorithm by returning the wrong coordinates and therefore skewing the decision of declaring a parking spot free or occupied.

For the final approach, we resorted to detecting circles purposefully placed inside the parking spot's lines. The algorithm works as follows:

- Image loading and processing:
img= cv2.imread('circles.jpg',1) #Reading the image
img= cv2.medianBlur(img, 1) #Applying a blur for
cimg= cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
#Storing the black and white image into the cimg variable
l=0 #Initiating the number of the circles

- Circle detection:
circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT, 0.5, 100, param1=50, param2=40, minRadius=20, maxRadius=30) #Detecting the circles with specific parameters

- Placing the coordinates of the circles found, on a black image:
circles = np.uint16(np.around(circles)) #Creating a black image with circle coordinates on it

- Tracing the circles on the initial image and incrementing the variable which stores the number of free parking spots
for i in circles[0,:]:
x, y, r= i[0], i[1], i[2]
cv2.circle(cimg, (x, y), r, (0,0,255) ,2)

l+=1 #Tracing the circles and incrementing the free parking spots number

- Displaying the result and the number of free spots *cv2.putText(cimg,"freespaces: {}".format(str(l)),(10,950), font, 2,(0,255,0),3,cv2.LINE_AA)* #Displaying the number of available spots

cv2.imshow('Original',img) #Displaying the original image

cv2.imshow('Circles',cimg) #Displaying the resulted image

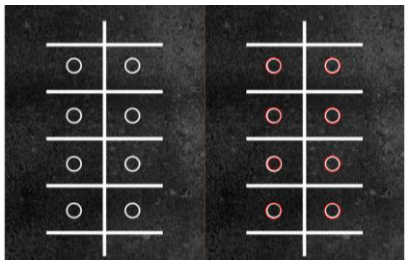

Fig. 3.The result of the algorithm for free parking spaces using circles

All of these algorithms were implemented on still images. Considering this application is based on a real-time broadcast, the algorithms need to be applied to each frame captured by the camera. In order to avoid unnecessary use of resources, we set the number of frames per second to 2, or one frame every 500ms, instead of the standard 30 frame per second.

Configuring the Raspberry Pi

After the installation of the Rapbian operating system, the Raspberry Pi (RPi) is ready to be configured. First, the camera module needs to be installed. The following comands are need to allow the RPi to stream the images captured by the attached camera, on a local IP:

- *Sudo apt-get install motion* – initiating the installation process for the necessary software packages;
- *Sudo nano /etc/motion/motion.conf* – parameters configuration: daemon (ON), framerate (ex: 100), Stream_port (8081 by default), Stream_quality (ex: 100), Stream_localhost (OFF), webcontrol_localhost (OFF), quality (100), width (640), height (480), post_capture (5);
- *Sudo nano /etc/default.motion* – setting the start_motion_daemon" parameter on „yes;
- *Sudo service motion start* – start the live transmission.

The live images from the RPi can be viewed in a web browser, at the IP address 192.168.1.4:8081.

IV. TESTING

The algorithms we researched were tested on a miniature parking platform with 8 available parking spaces. This platform consisted of a cardboard box, inside which we placed each of the images relevant to the tested algorithm. We tested the functionality of the setup in different conditions and on real-time transmissions. Obviously, the performance of the algorithms decreased once we switched from still images to 2 frames per second.

We began by testing the circle detection algorithm. After multiple tests we reached the conclusion that although it is a promising solution for the automatic detection of parking spaces, it is not stable enough in the imposed testing conditions. Adjusting the values of the parameters did little to improve the accuracy, and the circles were always detected as too many or too few. Aside from the erroneous results, this algorithm also has issues depending on the distance between the circle and the camera, in that in only detects the circles closest to the camera. Another problem is the light's reflection, especially if there is a strong light which interferes with the circles' detection.

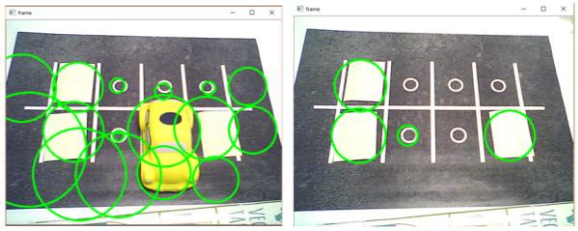

Fig. 4.The result of the circle detection algorithm with low, and high sensitivity settings

By testing the algorithm which offers the coordinates of the parking spaces, we obtained superior results compared to the circle detection algorithm. The proposed scenario included a vehicule as well as a pedestrian present on the parkin spaces, and the algorithm correctly identified as free the spot where the pedestrian was standing

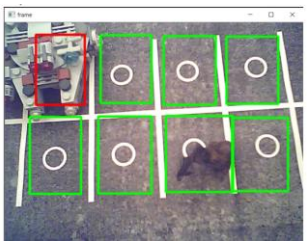

Fig. 5.The result of the coordinate-based detection algorithm.

In the presence of a strong light source, this algorithm generates errors as well, but significantly less than the previously analysed solution.

Table - I: Detection algorithm through coordinates vs. circles

Functions	Detection algorithm through:	
	Coordinates	Circles
Automated detection of the parking spots	No	Yes
Detection in low-light conditions	No	No
Detection in strong-light conditions	Partial	Partial
Detection of extra parking spots	No	Yes
Detection of less parking spots	No	Yes
Offers correct results in optimal conditions	Yes	No
Offers stable results	Yes	No
Requires infrastructure modifications	No	Yes

Pedestrians can influence the results	No	Yes
Requires parameters adjustments prior to deployment	Yes	Yes

If the light is insufficient, the detection algorithms will both fail 100% of the times. This is due to the same reasons that impede the human vision to detect a parking space during the night hours: the lack of visual information.

V. CONCLUSIONS

We consider that for the problem we studied, the best solution among those investigated is the algorithm to detect the availability of parking spaces through the coordinates' detection, instead of deciding this availability based on the detection of circles. The solution we chose has the advantage that it offers a concise result for one or more parking spaces at a time, but also the disadvantage of offering erroneous results if the camera is moved. Obviously, like all solutions based on Computer Vision, it cannot offer results during the night, or if another object obstructs the camera's view.

As future improvements, we intend to adapt the algorithm to make it suitable for existing surveillance cameras, in order to test in real-life parking lots. A significant improvement would be to change the shape of the parking spot selection, and use irregular polygons instead of rectangles. Introducing a night vision camera should compensate the night-vision limitation of the solution.

For further implementation we want to develop a mobile app which can let the user know where the parking spots are available using a Google Maps API and to offer access to the live stream for a better information related to the spot itself.

To implement this at the University level the requirements would be 4xRaspberry Pi + Camera module and a server to broadcast the data and feed the application. The total cost of implementation would be around 1000 Euros.

This type of application is not the most efficient when it comes to checking the availability of parking spaces, but is most likely among the cheapest and approachable solutions. We consider that it does not require prohibitive changes to the infrastructure in order to work, and the financial investment is negligible for an institution or a smart city that would like to implement it.

REFERENCES

1. D. Sikora, "The Concept Of Smart City in the Theory and Practice of Urban Development Management", Romanian Journal of Regional Science, http://rjrs.ase.ro, Vol. 10, No. 1, Summer 2016, ISSN 18423-8520
2. M. Kola-Bezka, M. Czupich, A. Ignasiak-Szulc, "Smart Cities in Central and Eastern Europe: viable future or unfulfilled dream?", Journal of International Studies, 2016, Vol. 9, No 1, pp. 76-87. DOI: 10.14254/2071-8330.2016/9-1/6
3. J-P. Exner, "Smart Cities - Field of application for Planning Support Systems in the 21st Century?", Proceedings of CUPUM, The 14th International Conference on Computers in Urban Planning and Urban Management, 2015, ISBN: 978-0-692-47434-1
4. N. Bawany & S. Jawwad, "Smart City Architecture: Vision and Challenges", International Journal of Advanced Computer Science and Applications, 2015, Vol. 6, No. 11, DOI: 10.14569/IJACSA.2015.061132
5. S.D. Prakash, A. Kumar, S. Shekhar, A. Khant, "Understanding the Concept Of Smart Cities", Scientific Journal of Impact Factor (SJIF): 4.14, 2016, Volume 3, Issue 3, March, e-ISSN (O): 2348-4470, p-ISSN (P): 2348-6406
6. C. Vrabie and A.M. Tirziu, "Top ten smart cities in the world. What do they have in common and how can Eastern European cities use that?" Munich Personal RePEc Archive, Online at https://mpra.ub.uni-muenchen.de/88291/, MPRA Paper No. 88291, posted 8 August 2018, accessed 20.08.2019
7. L. Garfield, "These 10 cities are the most prepared for the future", https://www.businessinsider.com/smart-cities-ranking-easypark-group-2017-11/#10-%20melbourne-australia-received-a-perfect-score-on-its-4g-connectivity-1, accessed 20.07.2019
8. A. Ahad & Z. Khan, "Intelligent Parking System", Scientific Research Publishing Inc., World Journal of Engineering and Technology, 2016, 4, 160-167, http://dx.doi.org/10.4236/wjet.2016.42014
9. P. Kharde, S. Pal, S. Kawle, "Smart Parking System", International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 2018, IJSRCSEIT, Volume 3, Issue 1, ISSN : 2456-3307
10. M. Fraifer and M. Fernström, "Investigation of Smart Parking Systems and their technologies", Thirty Seventh International Conference on Information Systems, Dublin 2016, Volume: "IoT & Smart City Challenges and Applications" - ISCA 2016, http://iot-smartcities.lero.ie/wp-content/uploads/2016/12/Investigation-of-Smart-Parking-Systems-and-their-technologies.pdf, accessed 17.09.2019
11. A. Šarić and B. Mihaljević, "Smart Parking System in the City of Dubrovnik", RIThink Multidisciplinary Online Journal, Volume 6, 2017, http://www.rithink.hr, accessed 07.09.2018
12. G. Anusooya, C. Jackson, K. Sathyarajasekaran and K. Kumar, "RFID Based Smart Car Parking System", Research India Publications, International Journal of Applied Engineering Research, 2017, Volume 12, Number 17, pp. 6559-6563, ISSN 0973-4562
13. A. Vimal, R. Yashvanth, J. Seetha, "Smart Parking Bay Based on Image processing and Internet of Things", Research India Publications, http://www.ripublication.com, International Journal of Applied Engineering Research, Volume 13, Number 6 (2018), pp. 4423-4427, ISSN 0973-4562
14. A. Gupta, S. Kulkarni, V. Jathar, V. Sharma, N. Jain, "Smart Car Parking Management System Using IoT", American Journal of Science, Engineering and Technology, 2017; 2(4): 112-119, doi: 10.11648/j.ajset.20170204.13
15. B. Sabbea, M. Irfan, S. Karama ALtamimi, S. Mabkhot, A. H. M. Almawgani and H. Alghamdi, "Design and Development of a Smart Parking System", Journal of Automation and Control Engineering Vol. 6, No. 2, December 2018, doi: 10.18178/joace.6.2.66-69
16. I. D. Dhaou, A. Kondoro, A. Hamad Alsabhawi, O. Guedhami and H. Tenhunen, "A Smart Parking Management System Using IoT Technology", Proceeding of The 22nd Conference Of Fruct Association, 2018, pp. 309-312, ISSN 2305-7254
17. P. Kamble1, S.Chandgude, K. Deshpande, C. Kumari and K. M. Gaikwad, "Smart parking system", International Journal of Advance Research and Development, 2018, Volume 3, Issue 4, pp.183-186, https://www.ijarnd.com/editions/volume-3-issue-4/, accessed 17.09.2019
18. S. D. Marcu and A. Florea, "Smart parking system - another way of sharing economy provided by private institutions", Proceedings of Thirteenth International Conference on Digital Information Management, 2018, ICDIM, pp. 18-23, 2018, Berlin, Germany, ISBN 978-1-5386-5244-2
19. MagPi Magazine, "Introducing Raspberry Pi Zero W", written in 2017 https://www.raspberrypi.org/magpi/pi-zero-w/, accessed 10.09.2019
20. W. Harrington, "Learning Raspbian", Published by Packt Publishing Ltd., 2015, ISBN 978-1-78439-219-2
21. S. Monk, "Raspberry Pi Cookbook", Published by O'Reilly Media, 2016, ISBN 978-1-491-93910-9, 2016
22. Raspberry Documentation, "Camera Module v1 vs. Camera Module v2", https://www.raspberrypi.org/documentation/hardware/camera/, accessed 05.09.2019
23. OpenCV Documentation, "API Reference" https://docs.opencv.org/2.4/modules/refman.html, accessed in 10.09.2019
24. O. D. Suarez, "OpenCV Essentials", Publisher: Packt Publishing, 2014, ASIN: B00N1X693A, ISBN: 978-1783984244

AUTHORS PROFILE

Răzvan Vîlceanu is a student in the second year of Technologies, Systems and Applications for eActivities Master at Politehnica University of Timisoara (UPT). He is activating in the field of IoT, Computer Vision, Cloud Computing and Programming. Razvan is a graduate of Telecommunications section, Multimedia specialty of Faculty of Electronics, Telecommunications and Informational Technologies at the Politehnica University Timisoara. He was involved in school projects such as:

Developing a SmartHome System using OpenHAB. This project was developed using OpenHAB platform installed on a local server and configuring it's widgets and devices for a proper displaying information about a house, temperature, TV control, automated lights etc.

Developing an online store from scratch using LAMP + WordPress. This project was hosted on Google Cloud Platform on a Debian9 Virtual Machine. Further a LAMP system was installed in order to have a functional web platform where a WordPress plus WooCommerce store was build. After the implementation was done the security and payments part had to be resolved.

Developing a hospital database using MySQL. This project was build using MySQL and the purpose was to offer a management database for a hospital including doctors, rooms, schedules, patients and treatments.

Developing computer vision algorithms for a smart parking solution. This project is the root of this article. I started working on it by desire of having a smart parking system in my city as I don't like to waste time on searching for a parking spot. I knew that some systems we're implemented already but this had to be different. I wanted it to be easy to administrate and cheap to implement, so there the price of implementing to be considered as a bottleneck for institutes or whoever wants to have this system.

Razvan is currently working as a Technical Writer for a local company in the satellite communications area. Also, he is starting to build his academic career and is teaching Object Oriented Programming in the 2nd year at the Communication department. He also held a course about Video Streaming and how the data is processed leaving from the user until it gets in a video streaming platform

Andrei Ternauciuc has finished his bachelor studies at the Politehnica University of Timisoara (UPT), with a degree in Electronics and Telecommunications Engineering, and a specialization in Multimedia Sciences. He was awarded his PhD in 2011, for his research in communication tools and e-Learning Platforms.

Andrei Ternauciuc is currently a member of the Multimedia Research Center, as well as the Center for e-Learning of the Politehnica University of Timisoara. He has over 10 years experience in working with learning management systems, and has helped design and implement various e-learning platforms, including ViCaDiS - Virtual Campus for Digital Students, CVUPT, UniCampus.ro, OpenVirtualMobility Learning Hub and others. He took part in over a dozen national and international research and academia projects, and has over 25 published papers in fields such as e-learning, multimedia sciences and life-long learning.

He is currently a Lecturer with the Communications Department of the same university, as well as one of the technical administrators of the Campus Virtual of UPT, the official e-Learning platform of the University."

Mihai Onița is a lecturer in Communications Department, Politehnica University Timisoara. He activate in field as: Computer Graphics, Digital Media, Audio-Video Techniques, System Television, IoT. He has a PHD diploma and post PHD studies based on distribution of interactive video content in educational platforms with high concurrent access.

He was a member in the implementing team in projects such as:

ATRACTING" - POSDRU/159/1.5/S/137070 - Increasing the Attractiveness and Performance of Doctoral and Postdoctoral Training Programs for Researchers in Engineering Sciences.

eSTART, POSDRU/86/1.2/S/54956 - Multi-regional Master's Degree Program in the Field of eActivities

CONCORD, POSTDRU/ 87/1.3/S/61397 - National Network of Continuous Training of Teachers in Pre-university Vocational and Technical Education.

DidaTec, POSDRU/86/1.3/S/60891 - University School for Initial and Continuous Training of Teaching Staff and Trainers in the Field of Technical and Engineering Specializations.

He has scientific papers in the field of video, mobile, MOOCs and computer generated materials.

Mihai Onița, Sorin Petan & Radu Vasiu, Review of Interactive Video-Romanian Project Proposal, International Education Studies, International Education Studies; Vol. 9, No. 3; 2016, Published by Canadian Center of Science and Education, ISSN 1913-9020, doi:10.5539/ies.v9n3p24

Mihai Onița, Camelia Ciuclea, Radu Vasiu, The Value of Video Lectures in the Classroom – UPT Study Case, ICIST 2015, Lituania, Druskininkai 15-16 Octombrie, ISBN 978-3-319-24769-4, DOI 10.1007/978-3-319-247770-0

Mihai Onița, Mihăescu Vlad, Vasiu Radu, Technical Analysis of MOOCs, TEM Journal, 4(1), 60 -72, ISSN: 2217-8309 (Print), eISSN: 2217-8333 (Online). Mihai Onița, Rafael Leucuța, Creating and using computer generated video materials in educational environment, Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara, Seria Electronică şi Telecomunicaţii Transactions on Electronics and Communications, Tom 59(73), Fascicola 2, 2014, ISSN 1583-3380, pp 15-20, http://www.tc.etc.upt.ro/bulletin/ Ivanc Daniel, Vasiu Radu, Mihai Onița , Usability Evaluation of a LMS Mobile Web Interface, 18th International Conference on Information and Software Technologies, Kaunas, Volume 319, Information and Software Technologies, ISBN 978-3-642-33307-1, 2012, WOS:000312463800029