Parking Assistance For Embedded Platforms

Waziri Ajibola Lawal

Abstract— ADAS 'Rear View Compact' is a system that aims to assist the driver in parking and reverse maneuvers, providing visual information space, objects around the vehicle, visual and audible warnings of possible collisions. The module that will be developed in this project is a subsystem that reliably and accurately detect a free parking space and displays the image of this parking space on the vehicle screen. This project will develop an algorithm from which the available parking space will be detected. The development of the system will be carried out with an embedded development platform based on Nvidia, a 'fisheye' lens camera and automotive hardware.Since the lens of the camera used in this project causes a great distortion in the image, different calibration methods will also be carried out to correct the distortion.

Index Terms— Calibration, embedded system, fisheye, guidance, images, intelligent, MXCAM, OpenCV, vehicle, rear view camera.

Resum— ADAS 'Rear View Compact' és un sistema que té com a objectiu ajudar al conductor a les maniobres d'estacionament i de marxa enrere, proporcionant un espai d'informació visual, objectes al voltant del vehicle, avisos visuals i audibles de possibles col·lisions. El mòdul que es desenvoluparà en aquest projecte és un subsistema que de forma fiable i precisa detecta una plaça d'estacionament lliure i mostra la imatge d'aquest espai d'estacionament a la pantalla del vehicle. Aquest projecte desenvoluparà un algoritme a partir del qual es detecta la plaça d'aparcament disponible. El desenvolupament del sistema es durà a terme amb una plataforma de desenvolupament integrat basat en Nvidia, una càmera de lent 'fisheye' i amb maquinari d'automoció. Atès que la lent de la càmera utilitzada en aquest projecte causa una gran distorsió en la imatge, es duràn a terme diferents mètodes de calibratge per corregir la distorsió.

Paraules Claus— Calibratge, fisheye, guidance, imatges, intel·ligent, MXCAM, OpenCV, vehicle, rear view camera, sistema integrat.

1 INTRODUCCIÓ

CONNECTIVITY, digitization and autonomous cars are the trends that will mark the future of the automotive industry [1]. The Advanced Driver Assistance System, or better known as ADAS System, is a mechanism that allows to improve the security of the driver at the moment of the driving, and thus, to be able to minimize the risk of suffering a road accident or collision with other vehicles [2].
There are different types of ADAS system: on the one hand, there are active systems such as 'Automatic brake', 'Automotive night vision', 'Autonomous cruise control system', 'Parking Assist'. On the other hand, there are passive systems such as 'Blind spot detection', 'Traffic Signal Recognition', 'Advanced Parking Assistance', etc [3].

In this project we will focus on the parking assistance systems. Specifically, in a driver support system known as 'Rear View Compact'.
This paper will be divided into the following sections: introduction, state of art, objectives, parking space and use cases, methodology and environment, parking assistance for embedded platforms, experiments and results, and finally conclusions.

2 STATE OF ART

Parking systems use different technologies such as ultrasonic parking sensors and vision systems, all these technologies aims to help drivers at the time of parking [4].

- *E-mail de contacte: waziriajibola.lawal@e-campus.uab.cat*
- *Menció realitzada: Computació.*
- *Treball tutoritzat per: Gemma Sanchez Albaladejo (Ciències de la computació)*
- *Curs 2016/17*

Currently there are three systems that support the driver at the parking maneuvers:

2.1 Parking distance sensor (Passive)

Parking sensors use ultrasonic wave technology to detect parking spaces and distance from other cars or objects.
To detect a parking space the sensor measures the longitudinal or transverse space to park and if this is adequate, it warns the driver by acoustic signals and by visual alerts on the screen [5]. Fig. 1 shows a parking lot detection using ultrasonic sensor.

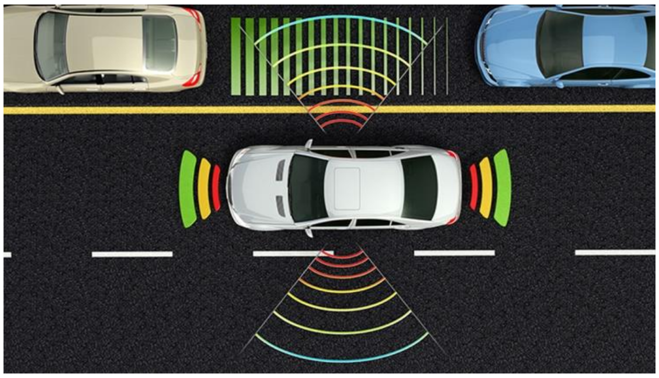

Fig. 1: Parking lot detection using ultrasonic sensor

2.2 Rear view camera (Passive)

This system will be activated when the reverse gear is activated and a color image of the surroundings of the car rear will be displayed on the instrument panel, showing the proximity to any obstacle. It will also be shown the planned route to the parking space through a dynamic guidance lines with the aim of facilitating the parking (Fig. 2). The system that is implemented in this project corresponds to this driver assistance system [4].

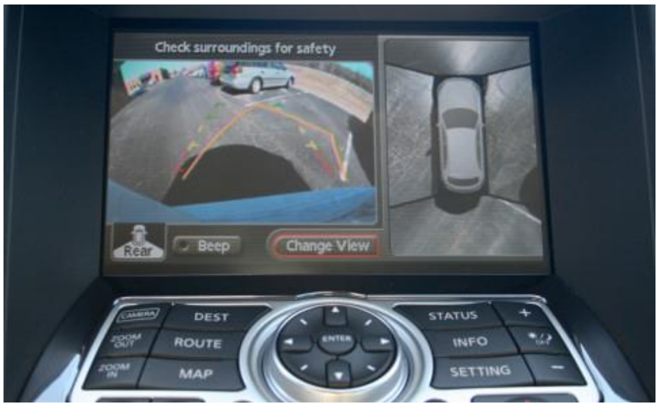

Fig. 2: Car rear view with guiding lines.

2.3 Intelligent parking assistance (Active)

Intelligent Parking Assistance system helps to easily park autonomously in a parking space.

Parking space is detected by ultrasonic sensors, the rear camera displays in a panel the free parking space and the intelligent parking system moves the steering wheel until the car is parked at the desired location [15]. In the following figure you can see the steps followed by the assistance system (Fig. 3).

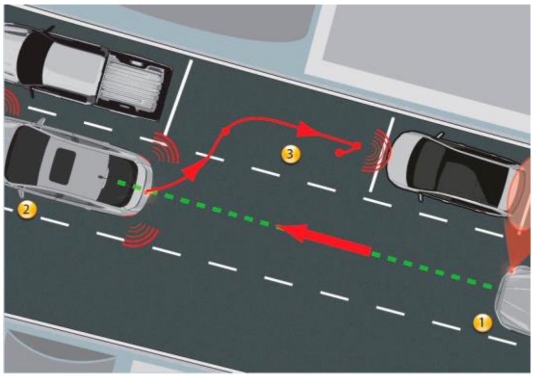

Fig. 3: Intelligent parking steps

3 OBJECTIVE

As indicated above the main objective of this project is the development of a system that is able to reliably and accurately detect a free parking space. In addition to creating a calibration method for the correction of the distortion caused by the camera used in this project.
To achieve these objectives, the requirements have been divided into sections: image acquisition, parking area detection and camera calibration.

3.1 Image acquisition

For image acquisition, FICOSA's 'MXCAM' camera is used, connecting it to an embedded system, particularly, the Nvidia Jetson TK1 Pro.

This camera has a resolution of 1280x806 pixels in YUV4:2:2, with a pixel size of 4.2 μm x 4.2 μm. The minimum field of view is 135º in vertical direction and 190º in horizontal direction. The images produced by this type of camera can be projected by different software, although this produces a certain loss of detail in the image.

The images acquired by the camera will be used as input data to the algorithm created for the detection of the parking space

3.2 Camera calibration and distortion correction

The 'Fisheye' lens of the camera causes distortion due to the shape of the lens or the fact that in the assembly process, the lens was not precisely aligned with the camera. Two types of distortion (radial and tangential) can occur in the 'Fisheye' lens.

For the distortion correction it has been followed previous work done in the company. A calibration algorithm was created following the OpenCV website tutorial for 'pinhole' cameras calibration, adapting and complementing it with new functionalities for our camera [6].

3.3 Parking area detection

Once the images have been acquired with the 'MXCAM', we will proceed to analyze these images and detect free parking space. For the parking space detection an algorithm has been created using several functions of the OpenCV library.

If our system finds a free space to park it will be displayed on the vehicle screen a translucent quadrangle overlay on the corrected image that will serve as reference to the driver.

4 PARKING SPACE AND USES CASES

For the project it has been specified the type of parking space to be detected, several use cases to define different parking situation and the results to be obtained from our algorithm for each use cases.

4.1 Parking space

This work is focused for parking spaces detection of battery parking delimited by lines that form a rectangle. In Fig.4 we can see an image of several parking spaces delimited by white rectangles.

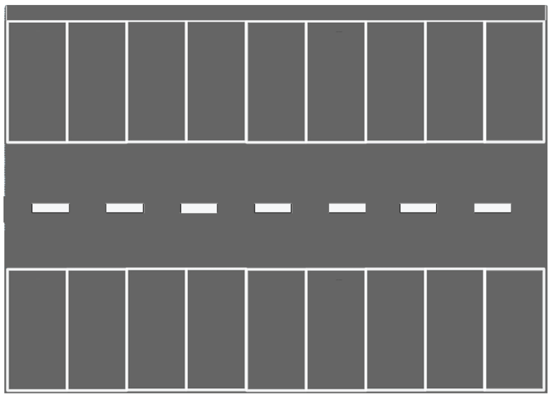
Fig. 4: Parking lots represented by white rectangles

4.2 Use cases

We have defined several use cases of different parking situations that may occur and for each of them we also defined the result that we must obtain from the system to be implemented. Below, each use case is explained in more detail and an image of each of them will be shown where the field of view that will cover our camera will be indicated.

4.2.1 Use case 1

In this use case we defined a parking situation when all the parking spaces are occupied. For this case our system must not detect any available parking space. Fig.5 shows an image of a parking area with all parking lots occupied.

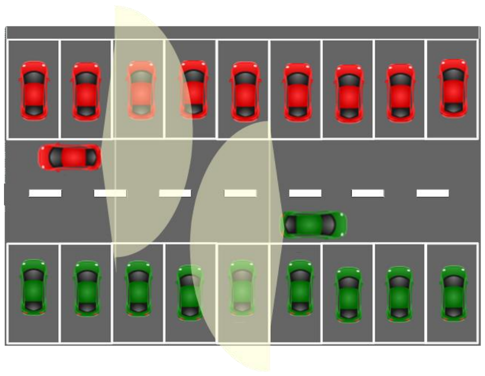
Fig. 5: Image of a parking area where all parking spaces are occupied. The yellow light that is seen behind the cars indicates the field of view that covers our camera.

4.2.2 Use case 2

In this case there is one free parking lot and our system must be able to detect it. In Fig.6 we can see the parking space (indicated with a green dot) that detects the system from the position where our car is located.

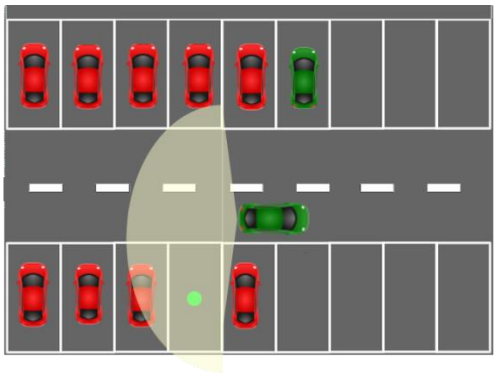
Fig. 6: Parking detection for one parking space. Green dot indicate the parking lot to be detected.

4.2.3 Use case 3

Unlike the previous use case, we have severals free parking spaces (Fig. 7) and therefore our system must be able to detect each parking lot as we pass near the parking region.

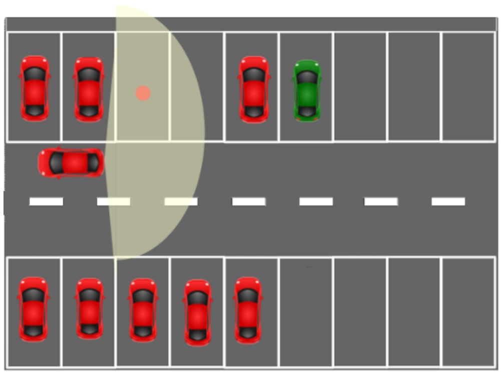
Fig. 7: Several free parking spaces. Red dot indicate the parking that will be detected in the car position.

4.2.4 Use case 4

In this use case one of the parked cars occupies a part of another parking space (Fig. 8) and therefore our system should treat this parking lot as occupied.

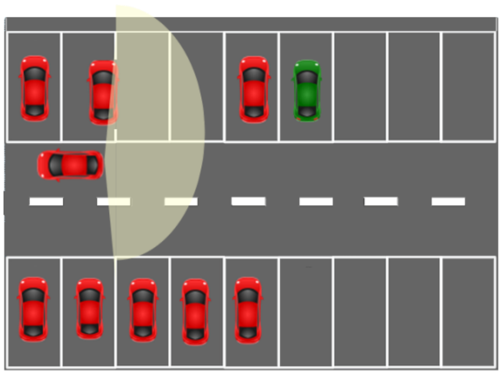

Fig. 8: Car parked between two parking lots.

4.2.5 Use case 5

As last use case we have a situation when all parking lots are free and therefore our algorithm must detect each parking space as the car circulates through the parking area.

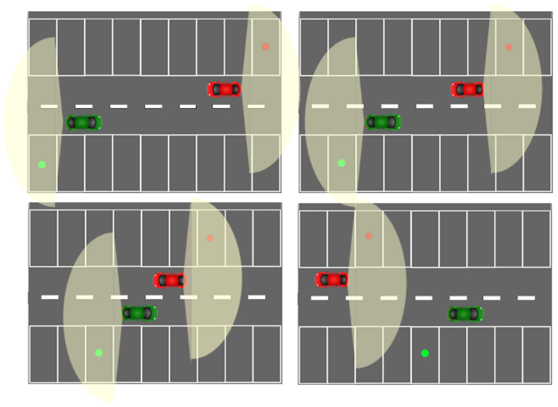

Fig. 9: Parking area with all parking spaces free. Red and green dots indicate the parking lots to be detected by the red and green cars respectively depending on their position.

5 METHODOLOGY AND ENVIRONMENT

It is very important to use an adequate methodology in addition to implement a system that fulfills all the project objectives in a clear and structured way. Below, it will be explained the planning carried out in this project and the environment used in the project development.

5.1 Planning

For the good development of the project it has been created an initial planning to have a control of the tasks performed and to be able to observe if it is being followed the initial line marked by the project or there have been some changes. The composition of this project has been divided into different phases:

Planning phase

- Project description: explanation and specification of the work to be carried out in the project and the goals to be achieved. 2 days
- Project requirements: approach and analysis of the project requirements. 15 days.
- Development environment preparation: we decide the programming language and define the development environment. 10 days
- State of art study: previous study was made about differents parking assist system ant their funtionalities. 10 days.
- Image Processing: initial study on different image processing methods. 7 days

Solution development phase

- Algorithm development: we develop the algorithm for our project. 45 days
- Testing: we create an image dataset for testing. The tests serve to study where improvements or changes must be made to improve the result obtained. 45 days.

Documentation phase

Throughout the project different documents have been drafted to have a control of the tasks that have been done. 18 days

5.2 Environment

The environment in which the project has been developed has been in an Ubuntu 14.04, the implementation has been done with Eclipse in C ++ and the OpenCV 2.7 library has been used. The hardware of the system used is an embedded system of NVIDIA Jetson Tegra K1 Pro, and a 'Fisheye' MXCAM camera property of FICOSA.

6 PARKING ASSISTANCE FOR EMBEDDED PLATFORMS

For our project two algorithms have been created: the first algorithm is for the camera calibration and the second algorithm for the parking space detection.

For the camera calibration we have followed the tutorial from OpenCV website for 'pinhole' camera calibration adapting and complementing it with new functionalities for our camera.

For the parking space detection we created an algorithm using OpenCV library. For the estimation of a parking space, a set of functions is defined based on image features to estimate which pixels or areas of the image are valid to be chosen as a parking space. We use the image contours features and the appearance to detect a parking space [12].

6.1 Camera calibration and distortion correction

Camera Calibration allows to obtain the intrinsic and extrinsic parameters of the camera. The intrinsic parameters are those that define the internal geometry and the optics of the camera. They are constants, therefore the relative characteristics and the positions between the optical and the sensor do not vary. The extrinsic parameters relate to the real world reference system and the camera. It also describes the position and orientation of the camera in the real world coordinate system. To obtain the intrinsic and extrinsic parameters of the camera we use OpenCV functions in addition to a flat structure with a chessboard pattern [6].

For the distortion correction we use the intrinsic and extrinsic parameters obtained as a result of the calibration. A new camera matrix for a distortion rectification is estimated and a geometric transformation is applied to the image. Below in Fig.4 we can see the block diagram of the calibration algorithm.

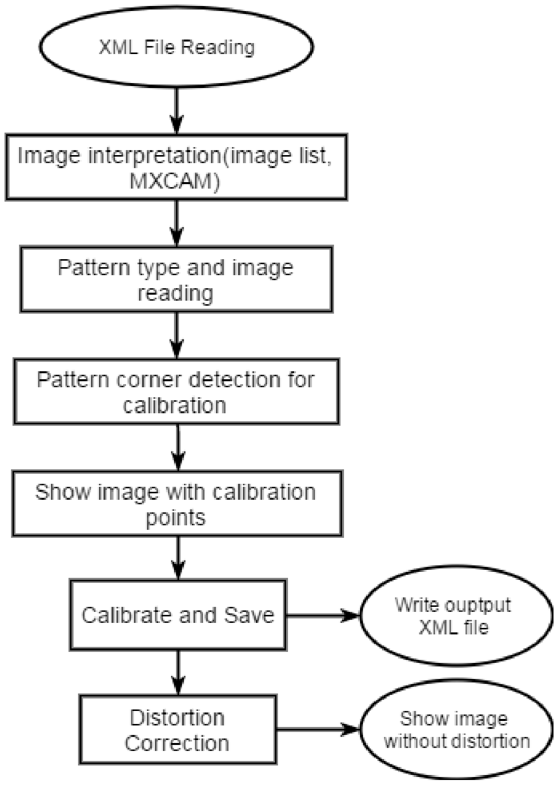
XML File Reading
Image interpretation(image list, MXCAM)
Pattern type and image reading
Pattern corner detection for calibration
Show image with calibration points
Calibrate and Save
Write ouptput XML file
Distortion Correction
Show image without distortion

Fig. 10: Camera calibration block diagram

6.1.1 Calibration

Once we read the input XML file, interpret the input type and the pattern, the corners of the pattern is found using OpenCV function, **findChessboardCorners**. This method returns a point vector of the corner found in the pattern. If "Chessboard" calibration method is used to calibrate, we improve accuracy when saving the corners of the pattern with **cornerSubPrix** [6]. In Fig. 11 we can see the corners detected by the findChessBoard on the pattern.

Calibration and saving are the two main functions of the program. Once the calibration is completed, the program returns an XML file with the parameters used. The most important data are the calibration matrix, the distortion, the rotation matrix and the image translation. The image rotation and translation are returned since they are the view that relates the coordinates of the world to the coordinates of the vehicle. The pattern should be placed on the floor as shown in the following Fig. 11.

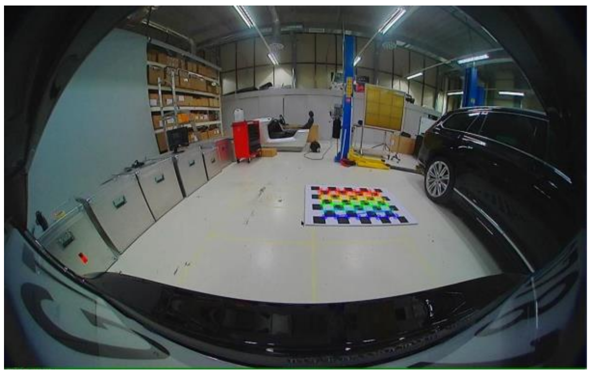

Fig. 11: Image with detected corners

To calibrate, we need the initialization parameters, calculate the position of the corners of the calibration pattern and calculate the re-projection error [9].

Once the calibration is finished and the results obtained are correct, all the data used is saved in an XML. The output parameters are the time, the used image and pattern data, the fundamental matrix, the distortion, image rotation and translation, the re-projection error and the image points.

6.1.2 Distortion Correction

The distortion presented by the camera will be removed depending on the model used. For the correction of the distortion a new camera matrix for a rectification of the distortion is estimated and a geometric transformation is applied to the image [9].

To estimate the camera matrix we use OpenCV fuction, **estimateNewCameraMAtrixForUndistortRectify ().** The input parameters for this method are: the fundamental matrix, distortion coefficients, image size, rectification transformation initialization in the object space, the new focal length, the new focal length divisor. The function returns a new camera matrix to rectify the image.

After obtaining the new camera matrix to undistort the image we use **initUndistortRectifyMap ()** to computes undistortion and rectification maps for image. And finally, we use **remap ()** to apply a geometrical transformation to an image [9].

6.2 Parking space detection

The parking space detection system consists of 3 clearly differentiated parts, the first of which is to perform an image preprocessing, the second part is the parking space detection, and the third and last one is the parking space visualization through a translucent quadrangle overlay in the image where we the system detect a parking lot.

In Fig.12 show a block diagram of the implemented algorithm.

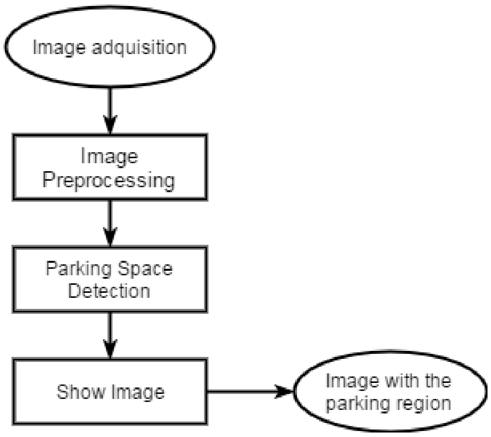

Fig. 12: Parking space detection algortihm block diagram.

6.2.1 Image Preprocessing

Performing an image preprocessing can help provide better results in the image to be analyzed. The first aspect to be treated in terms of image preprocessing is the unnecessary noise reduction within the image [11]. Another aspect to be treated is the color conversion to later apply an egde detection to detect edges (canny, sobel operator, laplacian).

Another possibility studied has been to use morphological transformations to close small holes inside the foreground objects, or small black points on the object [14].

For noise reduction and edge detection different algorithms have been used analyzing the result obtained in each one to decide which would be better for our solution.

Smoothing

- **Blur:** for each pixel it calculates the average of its kernel neighbors.
- **Median filter:** run through each element of the image and replace each pixel with the median of its neighboring pixels.
- **Gaussian filter:** convolving each point in the input array with a *Gaussian kernel* and thensumming them to produce the output array.

Edge detector

Edge detection involves detecting sharp edges in images and producing a binary image as the output. Edge detection is a fundamental tool in image processing, machine vision and computer vision, particularly in the areas of feature detection and feature extraction [12].

- **Canny edge detector:** canny edge detector is an edge detection operator that uses a multisatge algorithm to detect a wide range of edges in images.
- **Sobel operator:** sobel operator is a discrete differentiation operator. It computes the gradient approximation of an image intensity function.
- **Laplacian operator:** laplacian operator calculates the laplacian of a source image by adding up the second derivatives calculated using the sobel operator.

6.2.2 Parking space detection

Once the image has been preprocessed, we use the resulting image to find contours using OpenCV “findContours” function.

Contour is a curve joining all the continuous points (along the boundary), having same color intensity. Contours are useful for shape analysis and object detection and recognition [13] [14].

Once we get contours regions using 'findContours', we approximate these polygonal regions using ‘approxPolyDP ()' algorithm to find shapes in the image. This algorithm approximates a curve or a polygon with another curve/polygon with fewer vertexes so that the distance between them is less or equal to the specified precision [13][14].

Depending on the points that represent the shape that we detected through this algorithm we decided if it is a parking space.

6.2.3 Parking space visualization

Once we detect a parking space we draw a translucent quadrangle overlay to indicate the region of parking detected. The appendix section shows the results of the complete pipeline of the implemented algorithm. In the use case shown we can see that it does not always obtain the same result using the different edge detectors

7 EXPERIMENTS AND RESULTS

The experiments carried out in this project aim to demonstrate the performance of the implemented algorithm for the parking space detection.

This requires a sequence of images that show how this algorithm works and make clear its functioning in the final result image. The tests performed next will always be a sequence of images, where it will appear a green translucent quadrangle overlay in the region of the parking space detected.

7.1 Testing

To test the system we have recorded severals videos for each use cases defined earlier in more than five diferents situations from the car circulating in one of the parking areas of Ficosa. With the recorded videos we create an image dataset to perform test and demonstrate the performance of the algorithm implemented. The tests that have been carried out have been based on the use cases defined earlier in this document in section 4.2.

The objective is to show that the algorithm as we circulate and capture images of the surroundings of our car, it will analyze these images and when the system detects a free parking space it will draw a translucent quadrangle overlay over the detected parking region.

7.2 Results

After the test, we analyze the results obtained by the algorithm. In the algorithm created image preprocessing plays an important role since the results obtained from the preprocessed images are those used to analyze the image features and detect a possible parking space. As mentioned in section 6.2.1, during preprocessing different egde detectors have been tested, combining them with different image smoothing techniques. Here are several results tables of our algorithm output applying these initial preprocessing methods.

LO	Da-taset	FPS	BF	MF	GF	FP		
UC 1	2421	0	0	0	0	9	3	4
UC 2	3452	240	18	33	45	8	15	12
UC 3	3372	355	220	174	183	22	31	19
UC 4	3443	118	118	118	118	0	0	0
UC 5	3274	520	68	127	95	0	0	0
Total	**15962**	**1233**	**424**	**452**	**441**	**39**	**49**	**35**
			34,5 %	**36,7 %**	**35,8 %**			

Table 1: Parking space detection algorithm result applying laplacian edge detector to the 3 diferents smooting techniques defined. (LO: Laplacian Operator; UC: Use Case; FPS: Free parking space; BF: Blur Filter; MF: median filter; GF: gaussian filter; FP: false positive detection).

LP	Da-taset	FPS	BF	MF	GF	FP		
UC 1	2421	0	0	0	0	0	5	2
UC 2	3452	240	102	130	173	74	58	62
UC 3	3372	355	230	280	276	16	23	13
UC 4	3443	118	118	113	118	8	5	0
UC 5	3274	520	200	234	261	0	0	0
Total	**15962**	**1233**	**650**	**757**	**828**	**98**	**86**	**75**
			52,7 %	**61,4 %**	**67,2 %**			

Table 2: Parking space detection algorithm result applying canny edge detector to 3 diferents smooting techniques defined. (CN: Canny; UC: Use Case; FPS: Free parking space; BF: Blur Filter; MF: median filter; GF: gaussian filter; FP: false positive detection).

SO	Da-taset	FPS	BF	MF	GF	FP		
UC 1	2421	0	0	0	0	8	8	10
UC 2	3452	240	150	186	204	30	34	25
UC 3	3372	355	281	273	264	28	24	18
UC 4	3443	118	118	118	118	1	5	3
UC 5	3274	520	239	285	273	0	0	0
Total	**15962**	**1233**	**788**	**862**	**859**	**67**	**72**	**56**
			63,9 %	**69,9 %**	**69,7 %**			

Table 3: Parking space detection algorithm result applying canny edge detector to 3 diferents smooting techniques defined. (SO: Sobel Operator; UC: Use Case; FPS: Free parking space; BF: Blur Filter; MF: median filter; GF: gaussian filter; FP: false positive detection).

In table 1 to 3 we can see the percentage of parking lots detected by the algorithm using *laplacian, canny* and *sobel* edges detectors, combined with smoothing methods *blur filter, median filter and gaussian filter*. In addition we also count the number of false positive obtained.

As we can see using laplacian operator the percentage of parking space detected is very low compared to canny and sobel operator. The best result is obtained using sobel operator applying a smoothing with median filter or gaussian filter with almost 70% of correct parking space detected by our algortihm.

Finally several tables of performance measurement are shown computing the recall, presision and accuracy.

Recall: proportion of correct positive classifications (true positives) from case that are actually positives.

Precision: proportion of correct positive classifications from cases that are predicted positives.

Accuracy: proportion of correct classifications from over-all number of cases.

Laplace	Recall	Precision	Accuracy
Blur	34,5%	91,6%	94,7%
Median	35,8%	90,2%	94,8%
Gaussian	35,8%	92,6%	94,6%

Table 1: Performance measurement for *laplacain operator* combined with diferent smoothing technique.

Canny	Recall	Precision	Accuracy
Blur	52,7%	86,9%	95,7%
Median	61,4%	89'8%	96'8%
Gaussian	67,2%	91,7%	96'9%

Table 2: Performance measurement for *Canny* combined with diferent smoothing technique.

Sobel	Recall	Precision	Accuracy
Blur	63,9%	92,2%	96,7%
Median	69,9%	92,3%	97,2%
Gaussian	69,7%	93,8%	97'3%

Table 3: Performance measurement for *sobel operator* combined with diferent smoothing technique.

7.3 Problems found

After analyzing the results obtained, it has been seen that the algorithm is robust to shadows in the images produced by some objects (vehicles, street lamps, fences, etc.) that have not been able to remove correctly and that cover the parking lot lines (cause that some parking lots are not correctly detected) or produce false positives. We have also been able to observe that in some videos recorded for testing, some parking spaces lines are in poor condition and therefore the percentage of parking lots detected has decreased in these cases.

8 Conclusions and Future Lines

8.1 Conclusions

The objectives proposed at the beginning of the project have been largely achieved. A study of a method for parking spaces detection has been presented combining several preprocessing methods with other features detection algorithm. Comparing the results of the different preprocessing methods using the final output of our algorithm we can oberve that we obtain better results with sobel edges detector. As a result of this project it has been possible to realize an algorithm for parking space detection for the type of parking defined at the beginning of this project.

At learning level I have improved my knowledge about image processing and also known the possible uses in different areas.

8.2 Future lines

As future lines it could be made a more extensive validation of the algorithm, evaluating its operation in different conditions and in more parking areas outside Ficosa as it has only been tested in our facilities. It could also be possible to extend the functionality of the algorithm to detect other types of parking.

Aknoledgements

First I would like to thank my two tutors at Ficosa, Miguel Humberto Rib and Ramon Llorca and my supervisor David Moreno for guiding me in the development of the project and for being available at the times when I needed their help.

I also thank my tutor from the university Gemma Sanchez Albaladejo, for her dedication and the information provided for the realization of the project as well as the resolution of all the doubts that have been arising.

And finally I thank my colleague Albert Sole for his support and help throughout the project. Also to all my classmates throughout these years, especially Elisabet Maruenda Joya who has been the person who has helped me the most during these 4 years without forgetting my great friend John Freddy, his motivation and dedication have been a great help during these 4 years.

Bibliografia

[1] Automotive rellevants trends. https://home.kpmg.com/es/es/home/sala-de-prensa/notas-de-prensa/2016/01/conectividad-digitalizacion-tendencias-relevantes-sector-automocion-2025.html

[2] https://en.wikipedia.org/wiki/Advanced_driver-assistance_systems

[3] Difrerents Advance Driver Assistance System https://www.lifewire.com/advanced-driver-assistance-systems-534859

[4] Parking assist system. https://www.carwow.co.uk/guides/glossary/parking-systems-explained

[5] Volkswagen parking assist. Ultrasonic parking sensor. http://www.volkswagen.co.uk/technology/parking-and-manoeuvring/park-assist

[6] Camera calibration With OpenCV: http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

[7] Kannala, J. Generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. IEEE transactions on patterns analysis and machine intelligence (pp. 1335-1340), 2006.

[8] Fisheye Lens. Retrieved from En.wikipedia.org: https://en.wikipedia.org/wiki/Fisheye_lens.

[9] Geometric Image Transformations — OpenCV 2.4.8.0 documentation. Retrieved from docs.opencv.org: http://docs.opencv.org/2.4.8/modules/imgproc/doc/geometric_transformations.html

[10] Scaramuzza, D. << OCamCalib: Omnidirectional Camera Calibration Toolbox for Matlab>>. Retrieved from Sites.google.com: https://sites.google.com/site/scarabotix/ocamcalib-toolbox

[11] Smoothing Images. OpenCV 2.4.13.2 documentation. Retrieved from Docs.opencv.org: http://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html

[12] Feature Detection. OpenCV 2.4.13.2 documentation. Retrieved from Docs.opencv.org:, http://www.docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=canny
[13] Contours. OpenCV: Contours. Retrieved from Docs.opencv.org: http://docs.opencv.org/trunk/d4/d73/tutorial_py_contour_begin.html
[14] Structural Analysis and Shape Descriptors. OpenCV 2.4.13.2 documentation. Retrieved from Docs.opencv.org: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours}
[15] Intelligent parking assist System http://kereta.info/intelligent-parking-assist-system/
[16] Driving assistance system. 10 sistemas de asistencia a la conducción imprescindibles: http://www.autopista.es/reportajes/articulo/diez-10-sistemas-de-asistencia-conduccion-imprescindibles-100839
[17] Driving assist. Estos son los cinco sistemas de asistencia a la conducción con los que tienes un coche casi autónomo: http://www.circulaseguro.com/estos-son-los-cinco-sistemas-de-asistencia-la-conduccion-con-los-que-tienes-un-coche-casi-autonomo

APÈNDIX

A1. ALGORITHM PIPELINE USING CANNY

This section will show the results obtained by the algorithm using canny edge detection.

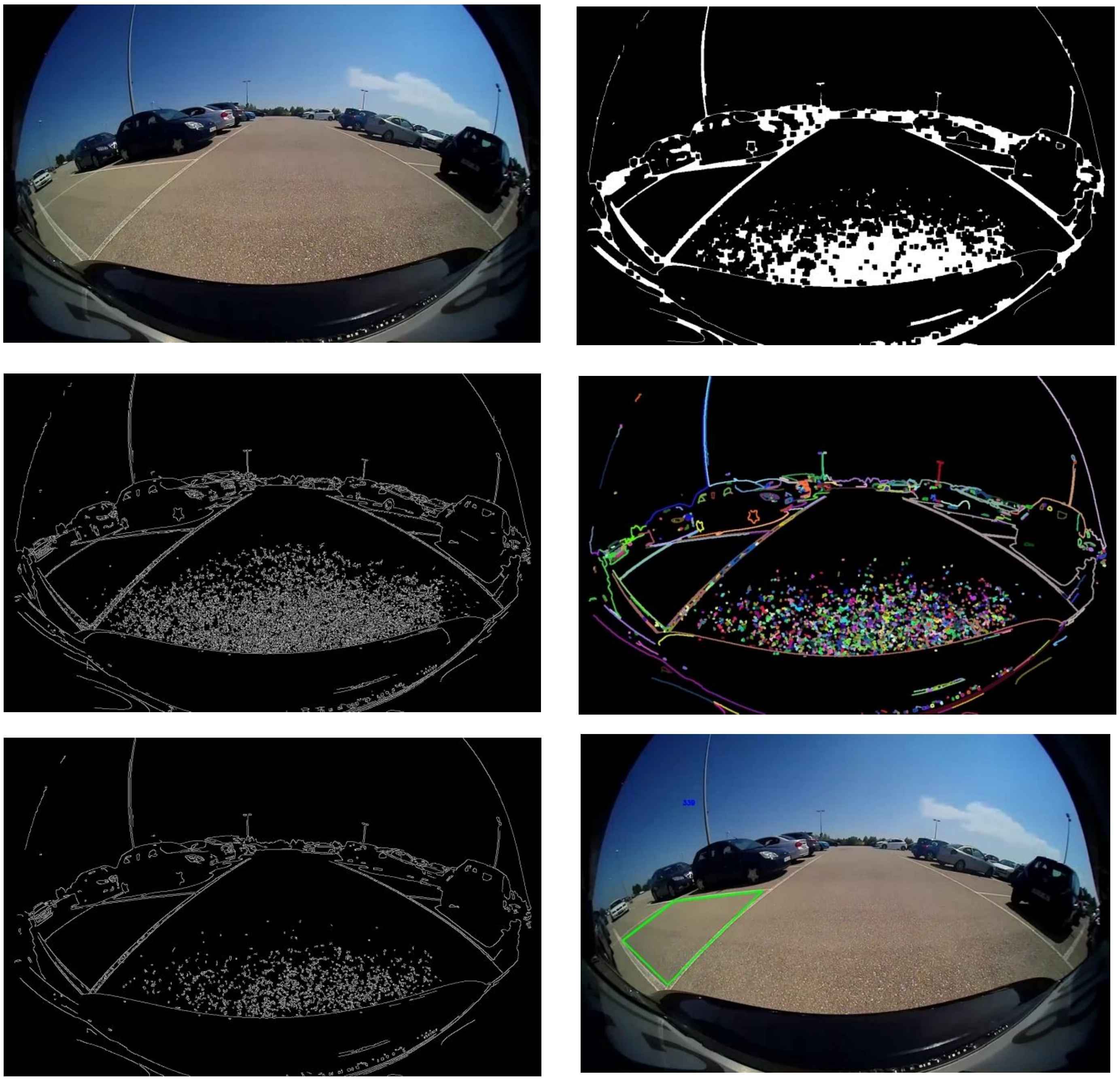

Ilustración 4: Different methods applied output in our algorithm is shown. Image 1: original image; Image 2: image applying canny without image smoothing; Image 3: canny with image smoothing; Image 4: applying morphological to canny output image; Image 5: detecting contour from morphical transform output; Image 6: Final result.

A2. ALGORITHM PIPELINE USING LAPLACIAN

This section will show the results obtained by the algorithm using laplacian edge detection.

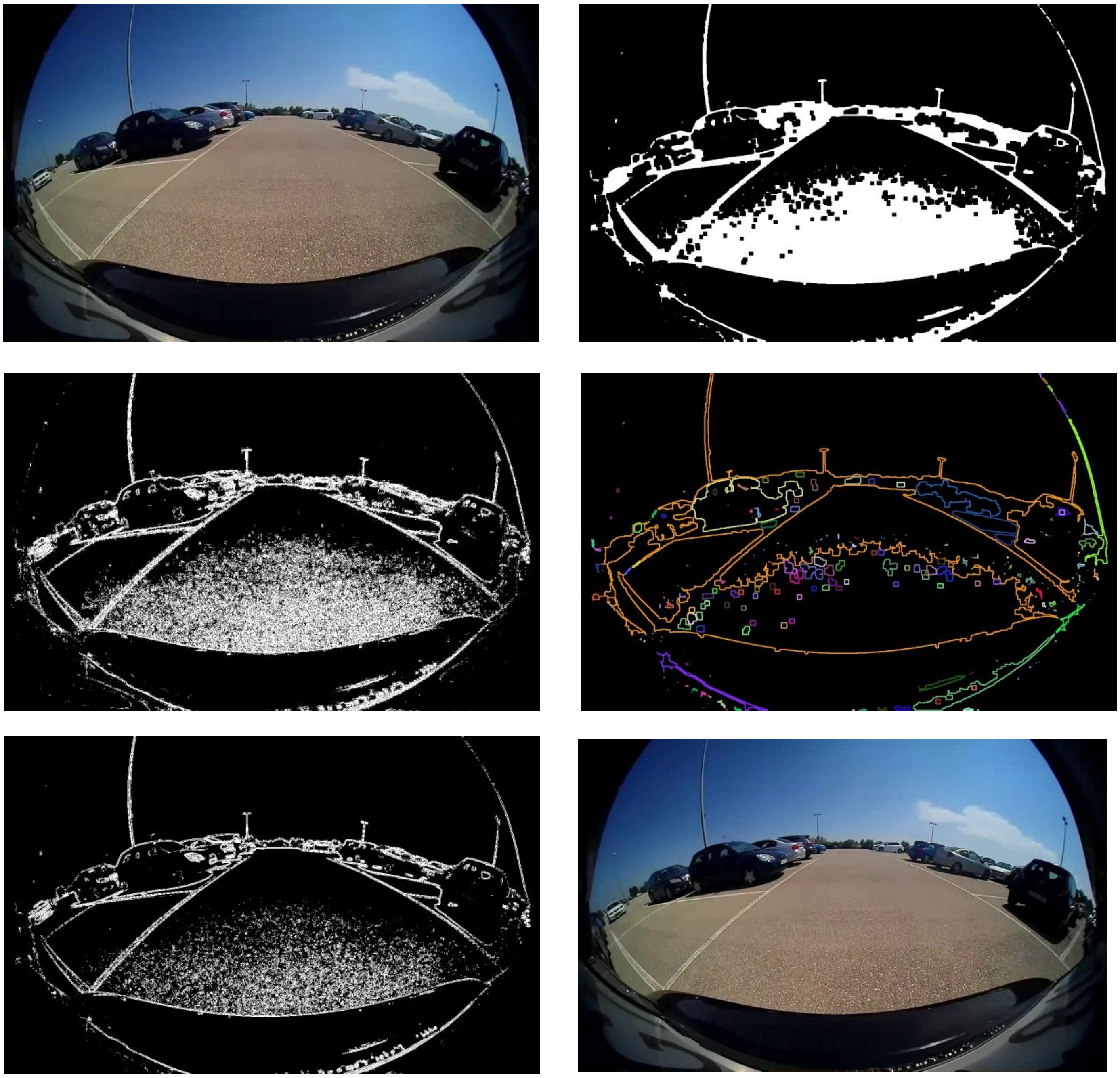

Fig. 14: The output of the different methods applied in our algorithm is shown. Image 1: original image; Image 2: image applying canny without image smoothing; Image 3: canny with image smoothing image; Image 4: applying morphological transform to canny output; Image 5: detecting contours from the morphological transform output; Image 6: our algorithm final result;

A3. Algorithm pipeline using sobel

This section will show the results obtained by the algorithm using sobel edge detection.

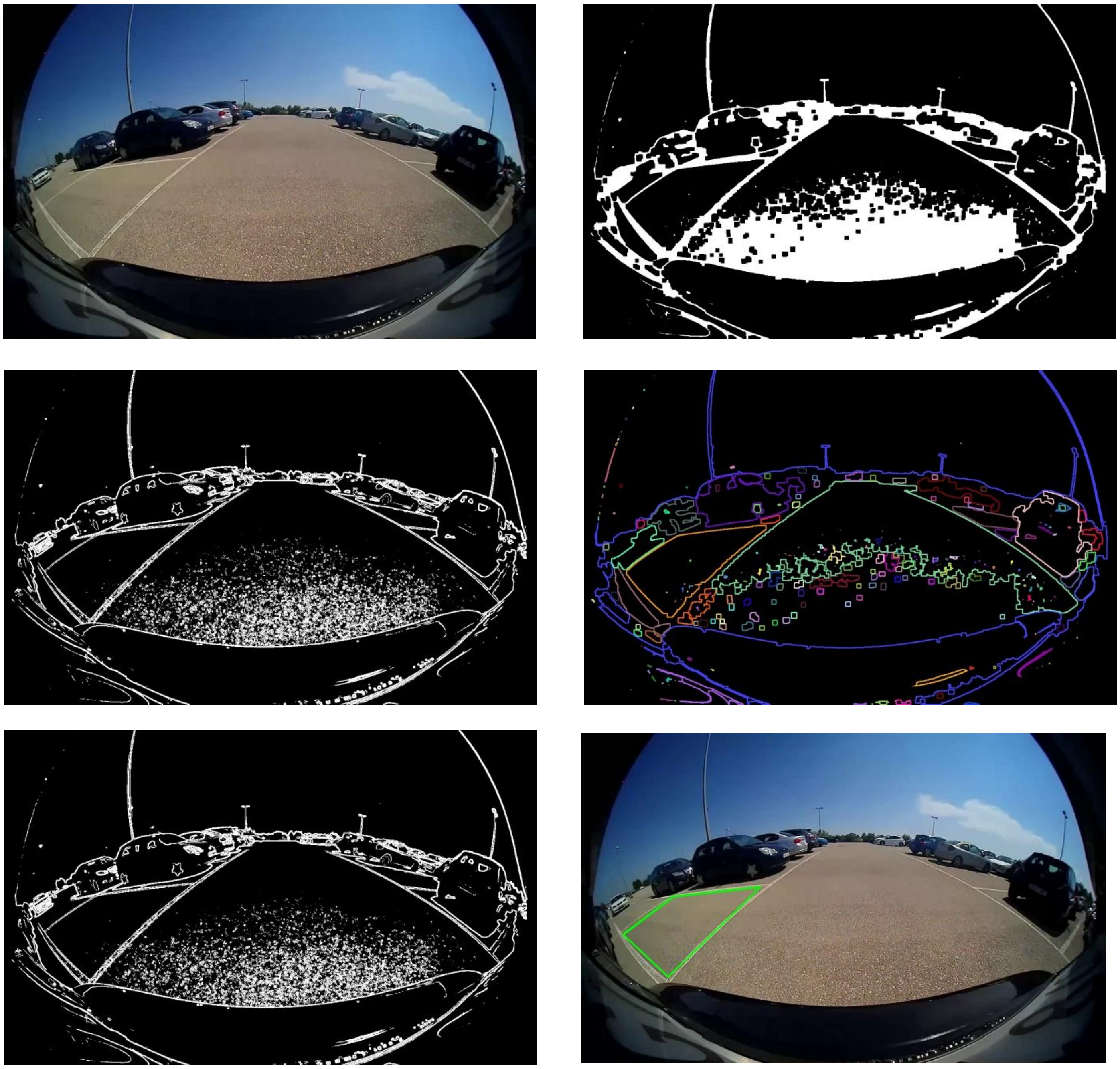

Fig. 15: Different methods applied output in our algorithm is shown. Image 1: original image; Image 2: image applying sobel without image smoothing; Image 3: sobel with image smoothing; Image 4: applying morphological to sobel output image; Image 5: detecting contours from the morphological transform output; Image 6: our algorithm final result;