Sensing the Driving Environment with Smart Mobile Devices

Radu Gabriel Danescu, Razvan Itu, Andra Petrovai
Computer Science Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
radu.danescu@cs.utcluj.ro

***Abstract*— This paper presents a generic obstacle detection system using single camera image sequence processing, for Android smart mobile devices. The algorithm relies on transforming the images into bird eye views of the scene using Inverse Perspective Mapping, followed by intensity based segmentation. The obstacle area hypotheses are used for updating a particle based occupancy grid. The sensors built in the mobile device are used for obtaining the speed and the yaw rate of the host vehicle, information needed for compensating the motion of the host vehicle in the tracking process. The occupied cells resulted from occupancy grid tracking are grouped as obstacles having position, length and width, and a speed vector. The system's intrinsic parameters are calibrated using the information provided by the Android API, and the extrinsic parameters are calibrated using an easy to use graphical interface. An automatic calibration method for the pitch angle, relying on the cumulated density of horizontal edges on the image rows is also provided. The system was tested on multiple devices, in real urban traffic, and is able to sense most obstacles on the road.**

Keywords—obstacle detection; monocular vision; camera calibration; mobile device

I. INTRODUCTION

The traffic environment is complex and dynamic, and the driver must continuously pay attention, in order to avoid accidents. Due to human limitations, researchers have designed computer based systems, based on multiple sensorial solutions, to help keep an eye on the driving environment.

While the most research and development effort has been dedicated to adding intelligent sensing capabilities to the vehicles themselves, especially to the higher end ones, driving assistance applications on mobile devices are a less costly alternative. Mobile devices are continuously improved with respect to their software and hardware capabilities. A vast array of modern smartphones and tablets feature quad-core processors and dedicated graphic processing units (GPUs). Due to this evolution, computer vision based advanced driver assistance systems (ADAS) are beginning to be developed directly on mobile devices. The algorithms can be optimized such that these systems run in real time, providing accurate obstacle detection and tracking.

The set of driving assistance applications available includes iOnRoad, Drivea, and Movon FCW. iOnRoad [1] is one of the first augmented driving application available for iOS and Android which uses the smartphone's camera and GPS receiver to recognize traffic ahead and warn of potential accidents. A similar application is Drivea [2], which provides obstacle collision warning, lane departure warning, and warning about speeding. A newer application, Movon FCW [3] is one of the few applications that can detect incoming cars, or cars that are not in full view, and can provide actual distance information.

Obstacle detection for driving assistance has been the focus of the research community for many years. Due to the fact that the real world position and speed of the obstacle are important for accident avoidance, the best results in obstacle detection are achieved using sensors that are capable of providing 3D information directly, such as radar, laser, or stereovision [4]. Unfortunately, these sensors must be either built in the vehicle (laser or radar), or they require an elaborate setup (stereovision). While some mobile devices are equipped with stereo cameras, which can be used for short distance accurate detection of obstacles [5], or with time of flight 3D ranging systems, most devices are only equipped with a single camera on a side, and thus they are able to execute only monocular computer vision algorithms.

Lacking direct 3D information, monocular vision algorithms face multiple challenges, as there is no geometrical way of separating the obstacle features from the background or the road. Therefore, obstacle regions in the image are extracted by color based segmentation [6] [7], motion based analysis [8], [9], or appearance analysis, either through heuristics such as symmetry [10],[11], or through the use of machine learning [12], [13]. For extracting the 3D position of the obstacle, monocular vision uses constraints imposed on the structure of the environment, such as the condition that the road is flat, which leads to the measurement of the points on the road by Inverse Perspective Mapping (IPM) [14],[15], [16].

This paper presents a generic obstacle detection system for Android mobile devices, which combines a graylevel based

978-1-4673-8200-7/15/$31.00

segmentation of IPM images similar to the one presented in [16] with a probabilistic model of tracking generic freeform dynamic environments, the particle based occupancy grid [17]. The occupied cells of the grid are grouped as obstacles having position, length and width, and a speed vector. The paper will present an overview of the detection method, a calibration method for accurate but fast setup in any vehicle, and the integration of the algorithm with the capabilities of the Android mobile devices.

II. OBSTACTE DETECTION ALGORITHM OVERVIEW

The process starts with the acquisition of an image from the mobile device's camera. The image is then transformed into a bird eye view of the road by Inverse Perspective Mapping, using the camera calibration parameters. The IPM image is segmented along rays staring from the camera position, and a hypothetic obstacle map is generated. The obstacle map is used to update the particle based occupancy grid. In the update process, the vehicle motion information (speed and yaw rate) is obtained from the available sensors of the mobile device (GPS for speed, gyroscope or accelerometer for yaw rate). The occupancy grid attaches to each cell of the top view world an occupancy probability and a speed probability density.

The estimated grid cells occupancy probabilities, and the speed vectors, are used in the process of obstacle extraction, which are identified as cuboids having length, width, position and speed.

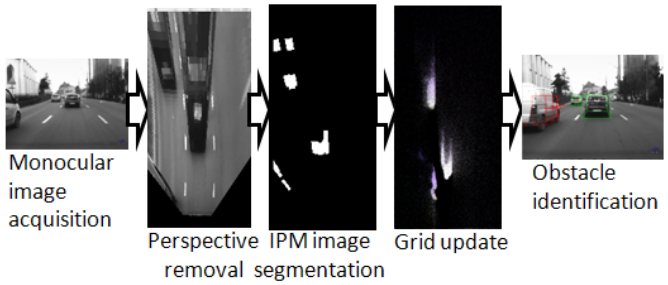

Fig. 1. The steps of the obstacle detection process.

A. Detecting the candidate obstacle areas

Assuming that the road is, to a reasonable degree, flat, the perspective image can be transformed into a bird eye view of the scene. A top view map of cells is defined, each cell being defined by a lateral position *X*, a longitudinal position *Z*, and a null vertical position (in the coordinate system described in section III.B), as it is assumed to belong to the road plane. Using these assumptions, we can use the camera calibration parameters, through the projection matrix, to relate each cell of the top view map with the perspective image, from which the intensity value of the corresponding pixel is extracted.

As the main assumption used in creating the IPM image is that all features of the scene belong to the road surface, only the road features will be shown correctly in the resulted top view image. As figure 1 shows, the obstacles tend to be stretched towards the far end of the IPM image, and have their edges expand from the location of the camera in a radial fashion. This means that in finding the obstacles, we had to take into account two facts: 1. the only usable area that we need to detect is the point of contact between the obstacle and the road, as it will map correctly in the IPM image and provide us the needed distance and lateral position in the 3D driving environment, and 2. the road/obstacle transition point in the IPM image will be searched along rays staring from the camera position.

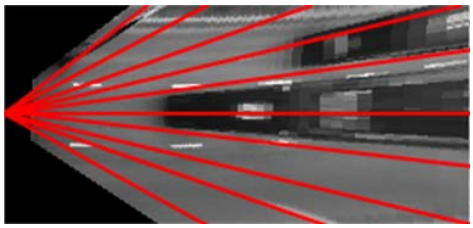

Fig. 2. Scanning the IPM image for roat to obstacle transition points.

The ray based analysis allows us to detect generic obstacles, as shape or size of the obstacle does not influence the aspect of individual rays.

If an obstacle is present along a ray, the intensity profile of the ray should be split in two or three areas: the area near the camera is the intensity of the road surface, the intensity at the far end is the gray level of the obstacle (which can be much more heterogeneous than the road), and, for some obstacles such as cars, the intensity at the point of contact with the road is a dark area due to the obstacle's own shadow. The point on the ray that maximizes the difference between the road area and the obstacle area, or the middle area and both the road and the obstacle areas, is chosen as the road/obstacle separation point. On each ray, at most one such point is chosen.

B. Tracking obstacle areas at cell level

The road/obstacle separation points that are detected on individual rays are then tracked using a probabilistic freeform occupancy model of the environment, the particle based occupancy grid [17]. The occupancy grid has the same number of cells as the IPM image, and for each cell the probability of it being occupied by an obstacle is continuously evaluated.

The occupancy grid tracking is done using a mechanism based on particles. Each cell of the grid can host a number of particles, each one having a speed vector. The particles can move from one cell to another, based on their speed, and can be created or destroyed depending on the measurement data. The movement of the particles is separated from the movement of the host vehicle, which is measured using the on-board sensors of the mobile device (GPS receiver, gyroscope and/or accelerometer).

The measurement data is represented by the road/obstacle separation points detected by analyzing the rays in the IPM image, as described in the previous section. The area in front of such a separation point, along a ray, is assumed to be free, and thus the particle population for the cells corresponding to this area is reduced. The area behind the contact point is considered unobservable (obstructed by the obstacle), and thus the particle population in that area is left unchanged. Around the measured point of contact, using an uncertainty factor which takes into account the error of measurement related to the distance and host vehicle pitching, the particle population is increased, and new particles are created if the cells were previously free.

The effect of particle based grid tracking is that from a set of disparate obstacle points the system can, in time, create consistent blobs of occupied cells, improving the description of the shape of the obstacle. Also, due to the dynamic nature of the particles, a speed vector is estimated for each grid cell.

C. Identification of individual obstacles

The occupied cells in the particle grid (the cells having a number of particles exceeding a threshold) are grouped into individual object clusters, using a labeling algorithm. A cell is added to a group if it neighbors the cells already in the group, and has a speed consistent with the speed of the group.

After labeling, oriented rectangles are fitted to each cluster. If the average speed of the cluster is higher than a threshold, the orientation of the average speed vector is used to extract the orientation of the obstacle. If the obstacle is static, no orientation can be extracted, and the bounding box is aligned with the coordinate axes.

III. Camera Calibration

As the application detects and measures the objects in a 3D coordinate system related to the host vehicle, accurate calibration of the camera intrinsic and extrinsic parameters and has to be performed.

A. Calibration of the intrinsic parameters

The intrinsic parameters of the camera are the focal length (usually expressed in pixels), the position of the principal point (the intersection between the optical axis and the image plane), and the distortion coefficients. All these parameters can be computed using a calibration toolbox [18]. However, using a calibration toolbox requires a certain expertise, and a significant amount of effort from the part of the user, and makes the system hard to set up and use.

An alternative is to make several simplifying assumptions, and use the information provided by the Android API. We can assume that the principal point is in the center of the image (modern camera technology ensures that the principal point is close to the image center, and a precise estimation of this point is not required for monocular applications), and to ignore the distortion coefficients of the lens. The only intrinsic parameter that still needs to be calibrated is the focal distance.

Using the available Android API, developers can access the vertical and the horizontal view angles, by calling the *getVerticalViewAngle*() or the *getHorizontalViewAngle*() methods of the *Camera.Parameters* object. Denoting the horizontal view angle by α, and the image width in pixels by *w*, the focal distance in pixels *f* can be computed as:

$$f = \frac{w}{2\tan(\alpha/2)} \tag{1}$$

The same equation can be applied for the vertical field of view, and the resulted focal length should be the same, if the camera pixel is square. If the camera supports variable zoom, the focal length can be adjusted to take into account the zoom level. Using equation (1), the intrinsic parameters of the camera can be computed dynamically when the system is started.

The intrinsic parameters matrix **A** is thus expressed by equation 2:

$$\mathbf{A} = \begin{pmatrix} f & 0 & w/2 \\ 0 & f & h/2 \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

In order to test the automatic approach for focal distance calibration, we have performed classic calibration, using the Open CV calibration toolbox [18], for several mobile devices. The results are shown in Table I.

Table I. Focal distance calibration comparison.

	Samsung Galaxy S5	**Sony Xperia Z1**	**Motorola Moto G**	**HTC One Mini 2**
f using Android API	697	484	598	530
f using Calibration Toolbox	686	502	617	520

While not perfectly identical, the results are in good agreement, with less than 4% percent error. These values can be used in a monocular vision application.

B. Calibration of the extrinsic parameters

The extrinsic parameters relate the 3D coordinate system of the camera to the 3D coordinate system of the vehicle. The detection is performed in the coordinate system of the host vehicle, which is has the origin on the longitudinal symmetry plane of the car, whose intersection with the road plane is the distance axis, *Z*. The lateral coordinate, *X*, points to the right of the direction of travel, and the *Y* coordinate points towards the road. The position of the origin coincides with the closest point of the road in front of the vehicle, as seen in figure 2.

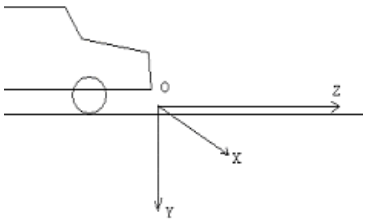

Fig. 2. The coordinate system of the vehicle.

The extrinsic parameters are represented by the rotation matrix **R** and the translation vector **T**. A full calibration of these parameters can be performed using a calibration toolbox, and a reference scene with objects at known positions in the car coordinate system. Unfortunately, this means again that the system cannot be set up easily. Instead, we have made several simplifying assumptions, and designed a simple interface for

allowing the user to easily tune a simplified extrinsic parameter set.

The user has to set up the mobile device in the vehicle, aligning the camera with the longitudinal axis of the car as best as possible. This alignment is not critical, and can be easily achieved by the user. Then, the calibration view of the application allows the user to input the height of the camera above the ground (H), and the distance of the camera from the front end of the car (L). These parameters can be easily measured. A very critical parameter is the pitch angle θ, which must be also input by the user.

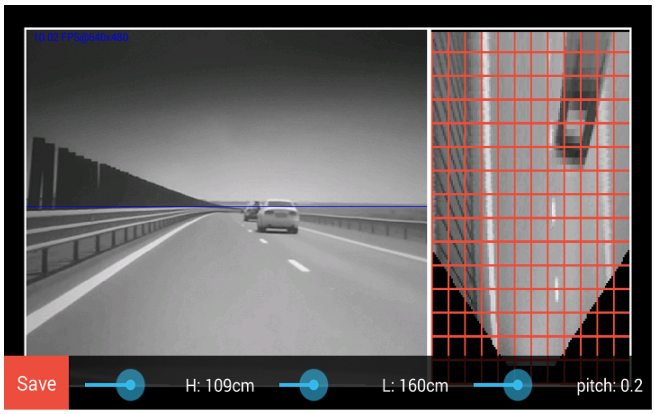

Fig. 3. The calibration user interface.

To assist the user, the system performs live Inverse Perspective Mapping on the capture images, in real time. A grid is superimposed on the IPM result, and the horizon line, resulted from the pitch angle, is projected on the perspective image, as seen in figure 3. This way, the user can adjust the parameters until the horizon line matches the true horizon of the observed scene, and if the observed scene has objects at known distances, these distances can be compared with the metric grid during parameter tuning. If road markings are visible, a good indication for the correctness of the pitch angle is that they must appear parallel in the IPM image.

The parameters H, L and θ input by the user are used to generate the translation vector and the rotation matrix:

$$\mathbf{T} = \begin{pmatrix} 0 & H & L \end{pmatrix}^T \tag{3}$$

$$\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4}$$

Now the intrinsic and the extrinsic parameters can be used to build the projection matrix, which will relate any 3D point in the car coordinate system to a point in the image. This matrix is used for computing the Inverse Perspective Transform image, allowing us to perform all object detection reasoning on a the top view image of the world.

$$\mathbf{P} = \mathbf{A}[\mathbf{R} \mid \mathbf{T}] \tag{5}$$

C. *Automatic pitch and ROI calibration*

The pitch angle is a critical parameter, and, even with an intuitive graphical interface, setting it right is not easy, especially if the system must be set up in a crowded parking lot, when no lane markings are present and no horizon is visible.

The best solution for pitch calibration is for the system to do it automatically. One solution would be to rely on estimating the vanishing point of the road markings, as it will be found on the horizon line. However, computing the vanishing point is expensive in terms of processing power, and the process needs visible lane markings, or at least long road edges.

Our solution for automatic pitch calibration relies on the density of horizontal edges in the image. Due to the fact that we want to capture a generic statistical property of the scene, we apply this technique only when the vehicle is moving, averaging the densities of multiple frames.

A structure $E(v)$ is created, counting the number of horizontal edge points (edges that have a near vertical gradient orientation) for each image row v. This horizontal projection of horizontal edge points is updated when the speed of the host vehicle is higher than 10 km/h. Figure 4 shows the horizontal projection when the system is started, and figure 5 shows this projection after 60 frames.

We can observe that the horizontal projection (or histogram) starts to get a well defined shape after a while. The edge density of the useful scene starts to resemble a Gaussian function, peaking near the horizon line. The hood of the host vehicle and other artifacts that enter the field of view but are not part of the moving scene tend to cause strong, sharp peaks, due to the fact that they are present in all frames, and are not smoothed out by a change in scenery. These properties become the basis for our analysis.

First, the maximum of the projection is located, by summing the values from a 30 pixel wide sliding window. We are interested in the peak of the Gaussian, not in the isolated sharp peaks. After the peak of the Gaussian is found, the standard deviation is computed by analyzing the values of E(v) around the maximum. We have found, experimentally, that the horizon line is found above the Gaussian peak, at a distance of half a standard deviation, and this property seems to be consistent for multiple camera configurations, as seen in Table II. The tests were performed on sequences that were accurately calibrated using reference objects and well established calibration methodologies, but the actual pitch angle of the vehicle in motion may vary from the angle during calibration (which determines the reference horizon line, first column of the table), due to the weight of the driver and of the passengers.

$\alpha + \beta = \chi.$ (1) (1)

Table II. Relation between the horizon line and the peak and standard deviation of the horizontal edge density function.

Horizon line v coordinate	Gaussian maximum v position	Standard deviation of Gaussian	Ratio between the v coordinates difference and the standard deviation
174	186	21.94	0.52
186	197	18.38	0.55
193	198	13.34	0.32
165	173	17.50	0.42
178	187	26.41	0.33

The horizontal projection of edges, $E(v)$, is also used for locating the bottom line of the useful Region of Interest (ROI). We want to exclude, as much as possible, the hood of the host vehicle from the obstacle detection process, as it can create erroneous results. The hood of the host vehicle creates strong spikes, sharp transitions in the edge density function. Starting from the detected horizon line, we search for the first strong peak (having a value of a significant percent of the maximum of E), that is also a sharp transition, being significantly different from the neighbor above it, at a certain small distance.

The horizon line and the bottom line are estimated for each frame, if the vehicle is moving. After 500 frames of successful estimation, the estimated values are locked into position, and the system is considered calibrated.

The automatic method for pitch calibration is optional, and the user has the possibility to select between the automatic and the manual approaches.

Fig. 4. Horizon line and bottom of ROI estimation, initial frame.

Fig. 5. Horizon line and bottom of ROI estimation, after 60 frames (5 seconds). The estimated horizon line coincides with the one from the calibrated pitch, and the bottom of ROI delimits the observed scene from the hood of the host vehicle.

IV. Using the Resources of the Mobile Device

The solution is implemented as an Android application, using Java for the user interface, and native C++ for the algorithm processing workload (Fig. 6).

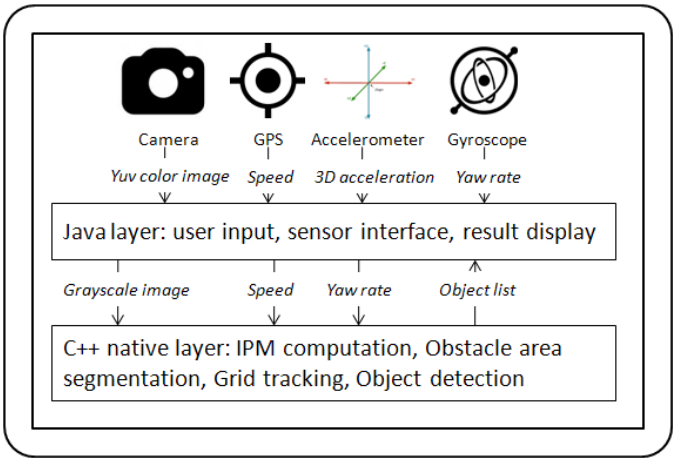

Fig. 6. The architecture of the solution.

The object detection algorithm requires real time acquisition of images from the device camera, and information about the speed and yaw rate of the host vehicle.

The detection and tracking process starts with the image acquisition from the mobile device's camera. Android offers the data frame from the camera as a byte array. The default Android color format used is YUV420. In order to do some processing we need to convert the image to RGB and to grayscale. The RGB system is used for a simpler way of drawing feedback for the user to the screen, whereas the grayscale image is used for processing.

Converting from YUV420 to RGB is usually time consuming, but by switching to Android's RenderScript API we can do a fast conversion using all the available processor

cores and even using the GPU. The RenderScript API features C++ like language in which you can write code that is parallelized automatically at runtime. This is Android's alternative to the OpenCL and other parallelization libraries and solutions.

The GPS sensor from the mobile device is accessed using the available Android APIs, and provides real time information about the ego-vehicle's speed.

The ego-vehicle's yaw rate is computed directly from the gyroscope sensor. The yaw rate provides information about the vehicle's rotation around the Y axis (the axis perpendicular to the road plane) and is expressed in radians/second. If the vehicle turns left, the yaw rate value can be -5 deg/sec, for a right turn an example would be +5 deg/sec, and so on.

If no gyroscope is available on the mobile device, the yaw rate can be computed from the lateral acceleration, present when the vehicle is describing a circular trajectory. Most mobile devices have a built in accelerometer, which measures the acceleration force applied to the device on three physical axes. The acceleration force is expressed in m/s^2 and it includes gravity.

The speed and yaw rate information are used to compensate for the host vehicle motion when tracking the occupancy grid. The yaw rate provided by the gyroscope proved to be precise, and has negligible latency, but unfortunately the speed information from the GPS sensor, and the acceleration information, are both inaccurate and have considerable latency. However, the system can work acceptably even in the presence of these errors.

V. TESTS AND RESULTS

The application was installed on multiple mobile devices and tested in various traffic conditions. The application was tested on three smartphones: Samsung Galaxy S5, Motorola Moto G and Sony Xperia Z1, and on the Samsung Galaxy Tab Pro 8.4 tablet. The Samsung phone features a modern quad-core processor rated at 2.5GHz per core and 2GB of ram memory and runs Android version 4.4 while having a 16 megapixel camera. The Motorola Moto G is an entry level device that also features a quad-core processor at 1.2GHz and paired with 1GB ram with a 5 megapixel camera and runs Android 4.4. The Sony Xperia Z1 is a 2.2GHz quad-core mobile phone that has 2GB RAM, a 21 megapixel camera and runs Android 5.0. The Samsung tablet has a 2.3 GHz quad-core processor, 2 GB of RAM, a 8 megapixel camera, and runs Android 4.4.

The application works on fixed, 640x480 pixels resolution, which can be provided by most cameras.

In order to test the processing time performance, the system deployed on multiple devices was tested on multiple traffic scenes. The processing time depends on the scene load, as more obstacles mean more particles in the occupancy grid. The average frame rate for each device is shown in Table III. The framerates represent the processing speed for the obstacle detection only, excluding automatic calibration related processing, and without any application running in the background (such as a video capture application for recording the results). Also, the devices are powered from the AC adapter, allowing them to run at full power.

Table III. Average frame rate for the tested mobile devices.

Mobile device	Average frame rate
Samsung Galaxy S5	17.6
Motorola Moto G	8.7
Sony Xperia Z1	11.4
Samsung Galaxy Tab Pro	14.5

In order to estimate the detection performance of the system, we have performed multiple test drives, using mostly the Samsung tablet. The drives were performed on urban streets in Cluj-Napoca, in realistic urban traffic, and the system proved capable of detecting most obstacles, of various sizes, speeds, and orientations, starting from a simple scenario such as tracking the leading car (figure 7), to observing crossing cars at an intersection (figure 8) or detecting a complex scene including moving and parked vehicles (figure 9).

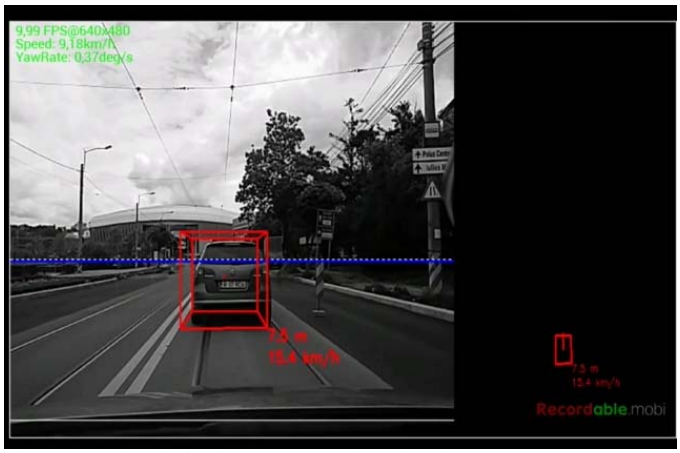

Fig. 7. Detecting the leading car: stop and go situation.

Fig. 8. Detecting multiple obstacles at an intersection.

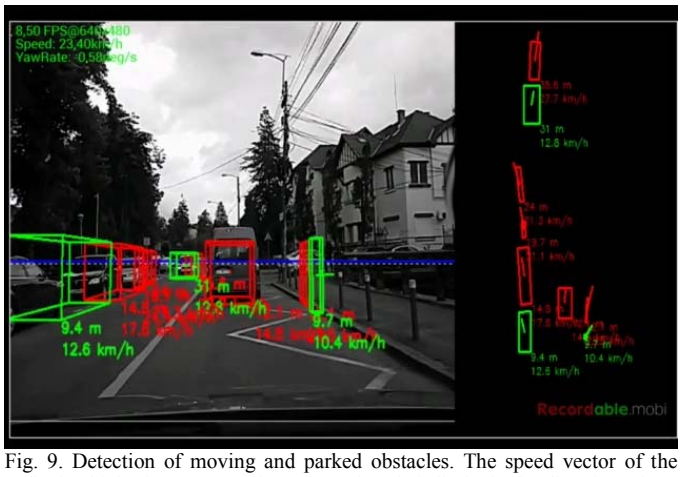

Fig. 9. Detection of moving and parked obstacles. The speed vector of the parked vehicles is not correctly estimated as their limits in the direction of our motion are not clear.

Due to the monocular, grayscale nature of the information used for detection, the results are sometimes affected by false positives, due to shadows on the road (figure 10), or by false negatives when the contrast between the road and the obstacle is not high enough.

Fig. 10. False positives due to shadows on the road.

A video showing the obstacle detection results for a 5 minutes long sequence in urban scenario can be accessed at https://vimeo.com/129640693 .

We have compared our solution to existing mobile applications on the market: iOnRoad[1] and Movon FCW [3]. Due to the fact that we wanted a fair comparison, we tested the applications in the same traffic scenarios. iOnRoad detects only the forward vehicles, located on the same lane with the ego-vehicle. Movon is able to sometimes perceive incoming vehicles or cars viewed from the side, but not most generic obstacles.

We have found that our proposed solution detects all obstacles that the other solutions detect, which are mostly the cars seen from behind, as the host vehicle follows them (as seen in figure 11).

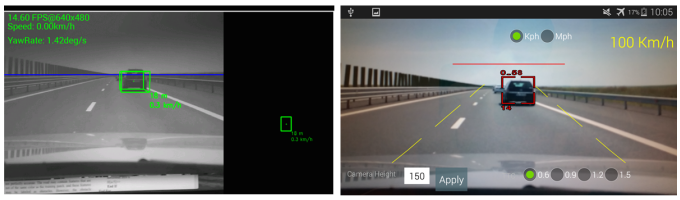

Fig. 11. Leading car, detected by the proposed solution (left) and by the Movon FCW application (right).

Fig. 12. Comparison between DriveAssist (left) and iOnRoad (right).

In order to assess the accuracy of the obstacle measurement process, we have compared it with a stereovision-based obstacle detection and tracking system. The stereo sequence was acquired using a high accuracy, 22 cm baseline stereo rig, able to accurately detect and range obstacles up to 40 meters. The system described in this paper was run offline, on the left images of the stereo pair sequence.

An obstacle in front of the host vehicle was tracked form approximately 800 frames. During this time, the target vehicle increased and decreased its distance from the observer multiple times, and the observing vehicle was following. The range of the target varied from 10 to 35 meters during tracking. After passing beyond 35 meters, the target was lost.

While noisier, the estimations of the monocular system followed closely the stereovision results. For the whole sequence, the distance Root Mean Square Error was 1.33 m, and the distance Mean Average Error was 1.02 m. The main source of errors was the pitching (forward – backward rotation around the X axis), due to acceleration, braking and road irregularities, which affect the generation of the IPM image and therefore the perceive distance to the obstacles. The error due to pitching increases with the distance to the obstacle.

We have also compared the proposed monocular solution with a stereovision-based one, deployed on the LG Optimus V900 Pad tablet, one of the few mobile devices capable of acquiring and processing stereo image pairs. The stereovision solution is a superior obstacle detection method at close range (0 to 10 meters), but the distance errors quickly increase beyond 10 meters, making the obstacle points virtually indistinguishable from the background. The limited performance is due mainly to the low resolution of the stereo images, combined with a very small baseline. The monocular solution works best in the 10 - 30 m interval, and is unable to detect obstacles closer than 5 meters, as the solution requires a patch of the road to be visible in front of the obstacle. The stereovision solution and the monocular solution seem to be complementary, and an ideal system would combine them, for the best coverage of the scene, and the best accuracy. Unfortunately, such a system would be very demanding in

terms of computational power, and limited in terms of compatible devices.

VI. Conclusion and Future work

This paper described a monocular based computer vision solution for detecting and ranging obstacles in driving environments, deployed on mobile devices. While most existing solutions are able to detect clearly visible vehicles, especially those that are followed by the host vehicle, our proposed system is dedicated towards detecting generic obstacles from the environment, independently of their shape, size, orientation or speed. The system was tested in real, heavy urban traffic, and shows promising results, turning a mobile device into a driving assistance sensorial system.

The main problem that has to be solved is related to the false positives caused by strong shadows. The shadow situation has to be detected as a special situation. Color information can help here, but it is yet not clear if the costs of using more image data will not outweigh the benefits.

Another issue that has to be addressed is the accuracy loss due to host vehicle pitching. The pitching effect may be corrected in multiple ways: detecting the vanishing point for each frame (time consuming), tracking the horizon line (less precise in urban environments), or by using the gyroscope provided angular rate (only available on certain devices). All these solutions have to be analyzed, and a solution to the pitching problem will be found.

Acknowledgment

This work was supported by a grant of the Romanian National Authority for Scientific Research, CNCS – UEFISCDI, project number PNII-PCCA 18/2012 (SmartCoDrive).

References

[1] "iOnRoad Augmented Driving Pro", available: http://www.ionroad.com/.

[2] "Drivea - Driving Assistant App", available: http://www.appszoom.com/android_applications/transportation/drivea-driving-assistant-app_bdwmk.html .

[3] Movon Corporation, "Movon FCW", available: https://play.google.com/store/apps/details?id=com.movon.fcw .

[4] S. Sivaraman, M. M. Trivedi, "Looking at Vehicles on the Road: A Survey of Vision-Based Vehicle Detection, Tracking, and Behavior Analysis", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 14, No. 4, December 2013, pp. 1773-1794.

[5] A. Petrovai, A. Costea, F. Oniga, S. Nedevschi, "Obstacle detection using stereovision for Android-based mobile devices", in Proc. IEEE 10th International Conference on Intelligent Computer Communication and Processing, ICCP 2014, pp. 141-147.

[6] V. Haltakov, H. Belzner, S. Ilic, "Scene Understanding From a Moving Camera for Object Detection and Free Space Estimation", in Proc. IEEE Intelligent Vehicles Symposium 2012, Alcala de Henares, Spain, pp. 105-110.

[7] S. Wybo, D. Tsishkou, C. Vestri, F. Abad, S. Bougnoux, R. Bendahan, "Monocular vision obstacles detection for autonomous navigation", in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2008, Nice, pp. 4190-4196.

[8] P. Reisman, O. Mano, S. Avidan, A. Shashua, "Crowd detection in video sequences", in Proc. IEEE Intelligent Vehicles Symposium 2004, Parma, pp. 66-71.

[9] X. Zhang, P. Jiang, F. Wang, "Overtaking Vehicle Detection Using A Spatio-temporal CRF", in Proc. IEEE Intelligent Vehicles Symposium 2014, Dearborn, Michigan, pp. 338-343.

[10] V. Graefe, W. Efenberg, "A Novel Approach for the Detection of Vehicles on Freeways by Real-Time Vision", in Proc. IEEE Intelligent Vehicles Symposium 1996, Tokyo, pp. 363-368.

[11] K. H. Lim, L. M. Ang, K. P. Seng, S. W. Chin, "Lane-Vehicle Detection and Tracking", in Proc. International MultiConference of Engineers and Computer Scientists 2009, (IMECS 2009), March 18 - 20, 2009, Hong Kong.

[12] S. Sivaraman, M. M. Trivedi, "A General Active-Learning Framework for On-Road Vehicle Recognition and Tracking", IEEE Transactions on Intelligent Transportation Systems, Vol. 11, No. 2, June 2010, pp. 267-276.

[13] J. Arrospide, L. Salgado, J. Marinas, "HOG-like Gradient-based Descriptor for Visual Vehicle Detection", in Proc. IEEE Intelligent Vehicles Symposium 2012, Alcala de Henares, Spain, pp. 223-228.

[14] M. Bertozzi and A. Broggi, "GOLD: a Parallel Real-Time Stereo Vision System for Generic Obstacle and Lane Detection", IEEE Transactions on Image Processing, pp. 62-81, 1998.

[15] S. Tuohy, D. O'Cualain, E. Jones, M. Glavin, "Distance Determination for an Automobile Environment using Inverse Perspective Mapping in OpenCV", in Proc. IET Signal and Systems Conference 2010 (ISSC 2010), Cork, June 2010, pp. 100-105.

[16] R. Itu, R. Danescu, "An Efficient Obstacle Awareness Application for Android Mobile Devices", in Proc. IEEE 10th International Conference on Intelligent Computer Communication and Processing, ICCP 2014, pp. 157-163.

[17] R. Danescu, F. Oniga, and S. Nedevschi, "Modeling and Tracking the Driving Environment With a Particle-Based Occupancy Grid," IEEE Transactions on Intelligent Transportation Systems, vol. 12, pp. 1331-1342, 2011.

[18] OpenCV, "Camera calibration With OpenCV", available: http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html .