An Efficient Obstacle Awareness Application for Android Mobile Devices

Razvan Itu, Radu Danescu
Computer Science Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
itu.razvan@gmail.com, radu.danescu@cs.utcluj.ro

***Abstract*— Recent developments in the mobile devices have made image processing on the go much more feasible. In this paper, we propose a monocular approach that can be deployed on smart phones and tablets and we evaluate its performance and the potential for further development of driving assistance system using mobile devices. This monocular approach relies on the main camera of the mobile device to observe the environment. Using the camera calibration parameters, an inverse perspective mapping of the scene is created, and then segmented into obstacle and free areas. The obstacles are then identified by polar and radial histogram processing. From real world experiments, we found that through modern mobile devices we can accurately detect obstacles in real time using a single camera. Developing driving assistance software is possible with the aid of such devices in urban roads, but also on highways.**

Keywords*— *obstacle detection, advanced driving assistance system, android, smartphone

I. INTRODUCTION

Multiple techniques for obstacle detection in video sequences (or in real time) have been described in literature, and demonstrated by experiment in multiple driving scenarios, including the highway and the urban traffic.

Obstacle detection methods based on computer vision and image processing follow two main directions: monocular based detection and binocular (stereo) camera detection. Obstacle detection using a single camera has always been a challenge, because not only must the obstacle's presence be deduced from the image data, but an estimate of the obstacle's position with respect to the host vehicle must be produced. This estimate can be extracted from monocular images using assumptions, such as the assumption of a flat road.

Monocular methods can further be separated in two main categories: appearance based or motion based. Appearance based techniques rely on color and shape information to detect regions belonging to obstacles. In [1] Ulrich and Nourbakhsh propose a color segmentation technique that is based on the fact that the ground plane has a constant color distribution. A texture based detection method is presented in [2]. Motion based methods often rely on the assumption of a flat road surface and most methods depend on image motion and optical flow. Many optical flow algorithms have been proposed, such as [3] or [4] and there are obstacle detection solutions based on optical flow [1], that compute the displacement between consecutive frames from a sequence of images.

Developing an affordable and portable advanced driver assistance system has been a challenge in the past few years. A simple, single camera-based solution is described in [5], where the system uses a webcam and a computer for image processing. However, the constant evolution of both hardware and software platforms that are used in mobile devices has led to endless possibilities of new solutions regarding the mobility, portability and the cost factor of a driving assistant system. The Android platform has proved to be an affordable and highly available alternative compared to an integrated, embedded solution.

There are very few mobile and tablet applications that perform locally (on the device) the entire processing: starting from image acquisition, then the image processing and ending with displaying the results. Current existing approaches mostly offer lane detection features and very basic, limited obstacle detection that works mostly in highway scenarios. A centralized advanced driver assistance system (ADAS) using mobile devices is described in [6].

This paper presents an obstacle detection method for driving assistance, which has been implemented as a mobile application, and thus can be integrated easily in all vehicles without additional costs.

The method we propose relies on removal of the perspective effect from the monocular images captured by the smart mobile device, a transformation useful for both detection and ranging. Then, using the assumption that the immediate road patch in front of our vehicle is free of obstacles, the bird-eye view image is classified at pixel level, into obstacle areas and free areas. The classified obstacle areas are then processed by radial and polar histograms, using the method introduced by Bertozzi and Broggi in [8], a method robust enough to work reliably even if some areas are incorrectly classified. As the IPM transformation is highly sensitive to the pitching of the host vehicle, the built in accelerometer of the mobile device is used for automatic adaptation to pitching conditions.

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

The remainder of the paper is organized as follows: in section 2, an overview of the entire monocular system is presented. Section 3 describes in detail the image processing algorithm, including the application of inverse perspective mapping, segmentation, and polar and radial histogram processing. Experimental results are provided in section 4. Finally, a discussion of the proposed system's feasibility and limitations is presented in section 5.

II. Algorithm Overview

The data flow of the presented solution is shown in Figure 1. When the application is first started, the preliminary computations required to build the Look-Up Tables are performed. Then, each frame acquired by the mobile device's camera is subjected to the processing steps described in the figure.

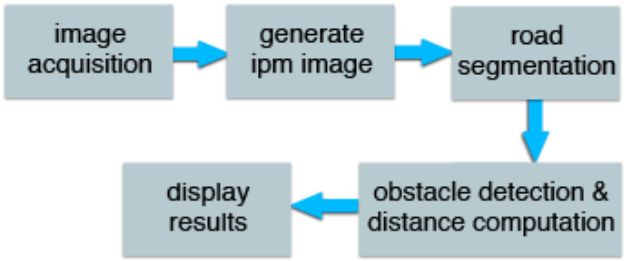

Fig. 1. Algorithm overview

Before the system can be operational, the camera setup has to be calibrated, in order to obtain the intrinsic and the extrinsic parameters. These parameters will be used later for removal of image distortions, and for perspective effect removal through Inverse Perspective Mapping (IPM). The calibration is achieved using a set of known patterns and the OpenCV calibration software.

Due to the real-time requirements of the application, the computation time has to be reduced as much as possible, and one technique to achieve this goal is the use of Look Up Tables (LUTs). The removal of the perspective effect algorithm uses complex calculations for each pixel in the image, but the end result is a remapping of its position, and therefore we create two LUT's, one for each of the two coordinate axes. The x LUT will contain the correspondent pixel in the original image for the remapped image's x axis, whereas the y LUT will contain the correspondent pixels for the remapped image's y axis. Thus, the step "generate IPM image" of figure 1, which will be executed for each frame, becomes a simple step of consulting the LUT's to get, for each pixel of the remapped image, the coordinates of the original pixel in the perspective image.

An operation of distortion removal is also applied on each of the input images, in order to remove any distortions that may be introduced by the mobile's camera lens. The computation of the necessary parameters used by the distortion removal operation is performed in the initialization step, and the results are included in the IPM LUTs, therefore no additional overhead is added to the processing of the frames.

Due to the fact that the pitching motion of the host vehicle alters the calibrated relation between the road surface and the camera, the IPM transformation may sometimes become incorrect. For this reason, the built in accelerometer of the mobile device is used to detect the pitching motion, and the transformation LUTs are updated with the appropriate values (multiple LUTs are computed, and the most suitable one is selected based on the accelerometer reading).

After the IPM image is obtained, the pixels of this image are classified into obstacle areas and road (free) areas, using their color. An area immediately in front of the host vehicle is assumed to be free of obstacles, and from this area the average color of the road surface is computed. The remaining pixels of the image are compared with this color, and the similar ones are assumed to be part of the road, and the dissimilar ones are assumed to be part of obstacles. This classification process is simple, fast, but far from perfect.

For a robust identification of the obstacles, one should take into account the following facts: 1. the segmentation process is imperfect, and 2. due to the removal of the perspective effect under the assumption that all pixels belong to the road, the obstacles will be extended in a radial manner towards the far end of the IPM image. Thus, the true obstacle areas will be significantly larger than falsely labeled road areas, and they will have a radial layout in the image. Inspired by [8], we use a polar histogram to identify the angle of the obstacle with respect to our traveling distance, and a radial histogram to identify the distance between the obstacle and the host vehicle.

Once the obstacle is detected, it is displayed, and if the obstacle is too close to us, and on our trajectory, a warning is displayed.

III. Obstacle Detection Steps

A. *Inverse Perspective Mapping*

The images acquired from a camera exhibit the perspective effect. If the image is of a traffic scene, the obstacles on the road will appear smaller when they are at a higher distance from the host vehicle, and larger when they are closer. The road boundaries appear not parallel, but intersecting in a point at the horizon, the vanishing point (Figure 2, left). In the perspective image, the variable level of detail creates difficulties for computer vision tasks such as obstacle detection, and the 3D information is more difficult to infer.

Under certain assumptions, such as the assumption that the road is flat and aligned with our coordinate axes such as its height is always zero, we can transform the perspective image into a bird eye view of the traffic scene (Figure 2, right).

In the bird eye view image, each pixel coordinate is in a simple 1 to 1 relation with a coordinate in the road plane *XOY* (Figure 3), assuming that the height coordinate Z is zero. Thus, obstacles detected in the IPM image can be immediately positioned in the 3D coordinate system.

Fig. 2. Normal image (left), and IPM image (right)

In order to describe the IPM transformation process, we'll assume the coordinate systems $W=\{(X,Y,Z)\}$, the world coordinate system, and $I=\{(u,v)\}$, the coordinate system of the perspective image, as seen in figure 3.

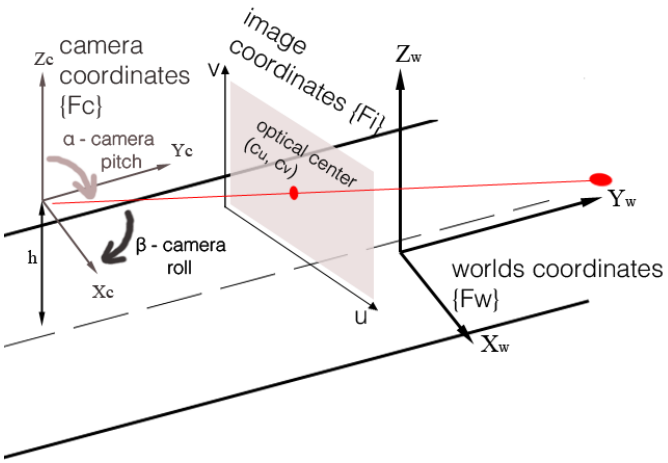

Fig. 3. Coordinate systems and the projection of a point

The algorithm for IPM image computation is the following:

Algorithm IPM Transformation
Input: Source image I
Output: IPM image I_T
For each pixel of coordinates (u_T, v_T) of I_T
$X_W = k\ u_T + X_0$
$Y_W = j\ v_T + Y_0$
$Z_W = 0$
(u, v)=Projection(X_W, Y_W, Z_W, **P**)
$I_T(u_T, v_T) = I(u, v)$
End For

The constants k, j, X_0 and Y_0 are chosen in such a way that the most relevant portion of the road plane is displayed in the remapped image. The Projection function multiplies the 3D coordinate vector $(X_W, Y_W, Z_W, 1)^T$ with the projection matrix **P** in order to obtain the original coordinates in the perspective image for any 3D point. **P** can be computed using the following equation:

$$\mathbf{P} = \mathbf{A}[\mathbf{R} \mid \mathbf{T}] \quad (1)$$

Where **A** is the matrix of the intrinsic parameters of the camera, containing the focal distance and the principal point:

$$\mathbf{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

The translation vector **T** and the rotation matrix **R** are the extrinsic parameters of the camera, and they depict the required rotation and translation from a point in the world coordinate system to be expressed in the camera's own 3D coordinate system. All these parameters are estimated by camera calibration.

Due to the fact that computing the projection for each IPM image point takes time, this computation is only done once, at initialization, and the results stored in a LUT. At running time, the IPM is performed by consulting the LUT for each (u_T, v_T) to get the original (u, v).

B. Automatic Compensation of the Vehicle Pitching

The movement of the vehicle may change the angle between the camera of the mobile device and the road, such that the rotation matrix obtained with the camera and car in a parked position becomes unfit to the real world conditions. The rotation matrix affects the projection matrix, leading to a false IPM transformation, as seen in Figure 4.b.

Thus, dynamic adjustment is needed for the camera's projection matrix. Fortunately, the smart mobile device is aware of the pitching motion, due to the fact that most such devices have a built in 3-axis accelerometer that provides real time acceleration values to the developers through Android API's. In our solution we used these values to correct the projection matrices that are being used for IPM transformation.

(a) (b) (c)

Fig. 4. Input image (a), without auto-correction (b), with auto-correction (c)

To implement the dynamic adjustment, we use pre-computed projection matrices for different values of the acceleration read for the Z axis (the vertical axis). For example, when the mobile device has a pitch value between 1 and 1.5 degrees with respect to the road surface, we use projection matrix #1, when the device is pitched even more and the angle is between 1.5 and 2 we use projection matrix #2, and so on. Lookup Tables are pre-computed for each pitch angle interval, so that the IPM process can work in constant time, without the need for re-initialization.

The raw sensor data returned from the 3-axis accelerometer found on most Android devices is sometimes erroneous,

containing a lot of noise. To increase the robustness and accuracy of the proposed solution we used a low pass filter for the accelerometer data. If the previously estimated accelerations are in the form of a vector $\mathbf{X}=(a_x, a_y, a_z)^T$, and a new sensor readout is in the form of $\mathbf{Z}=(ra_x, ra_y, ra_z)^T$, the newly estimated acceleration vector **X'** is computed as a weighted average between the past estimation and the new measurement:

$$\mathbf{X}' = (1-w)\mathbf{X} + w\mathbf{Z} \quad (3)$$

The parameter *w*, having a value between 0 and 1, is tuned experimentally for the best compromise between filtering and faster response to dynamic pitching.

C. IPM Image Segmentation

The IPM transformed image is a bird-eye view of the scene in front of the camera, including both road pixels and obstacle pixels. As the purpose of the application is to detect obstacles, a classification of all pixels in the image must be performed. This classification is in fact a thresholding process, leading to a binary image containing either black pixels – obstacles, or white pixels – the road surface.

Due to the fact that only one image source is used, the obstacle areas can only be identified by their color with respect to the road surface. The road surface itself may have different colors, depending on the nature of the road, the illumination conditions, etc. However, we'll assume that at least for the limited range we are surveying, in a single frame the road is almost homogeneous in color. The second assumption is that we are not too close to an obstacle, and the nearest patch of road surface in front of the vehicle is free.

At each frame, the nearest patch of the road (corresponding to the bottom part of the IPM image, see figure 5) is used for computing the color characteristics of the road. The mean value and the standard deviation are computed for each color component Red, Green and Blue, of the road pixels belonging to this region.

After the color characteristics of the road are learned, the remainder of the IPM image is scanned, and each pixel's color components are compared to the RGB components of the road. If the difference between the color of the pixel and the road color exceeds the standard deviation, for each color component, the pixel is labeled as obstacle; otherwise it is labeled as road. The results of classification are shown in figure 5.

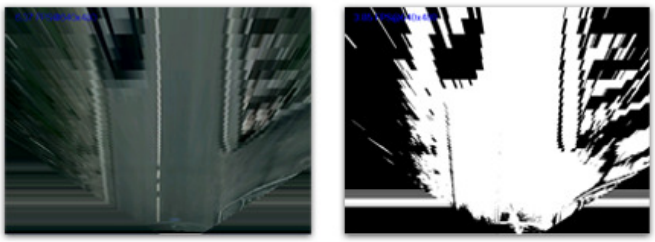

Fig. 5. IPM image (left), and IPM image after pixel classification (right).

The chosen classification method is very fast, but obviously not perfectly accurate. The road may contain features that are not of the same color as the training patch, and these features may be labeled as obstacles. However, the obstacle identification method described in the next section is able to overcome this problem.

D. Obstacle Identification

The obstacle pixels of the IPM image must be grouped into distinct obstacles whose position in the 3D space can be determined. The main challenge is the presence of many false positive pixels, as the result of an imperfect classification method. Thus, in order to recognize the true obstacles, we have to use additional properties about their shape in the IPM image.

In [8] and [9] Bertozzi and Broggi proposed a very efficient method for processing binary IPM images in order to identify the obstacles. Their idea was based on the fact that when the IPM transformation is performed on an image under the assumption that all the points belong to the road, the obstacle points will be mapped differently for different cameras, while the road features will be mapped in the same spot. They used a stereo setup, performed two IPM transformation, and highlighted the differences between them. The obstacle areas were clearly highlighted, but false positives appeared in their solution as well. For this reason, they developed a polar histogram based processing method for determining the azimuth angle of the obstacle with respect to the host vehicle, and a radial histogram method for determining the distance along the azimuth line. The main challenge for their method was that in the difference IPM image, only the obstacle sides were highlighted, while the obstacle middle was sometimes cleared. In the polar histogram, this caused peaks not in the middle of the object, but on its sides, and therefore a heuristic for joining peaks had to be developed.

For our situation, only one image source is used, and therefore no difference between views can be performed. However, it is clear that the radial disposition of the obstacle features in the IPM image is preserved. These features converge into a point corresponding to the position of the camera in the 3D world, the Focal Point, F.

The use of a monocular approach based on color classification of the obstacle features leads to an important difference between this approach and the stereo IPM described in [8] and [9]: the obstacle is fully highlighted, not only its sides. The process of polar histogram computation is described by the following algorithm:

Algorithm Polar Histogram Computation
Input: IPM binary image I_B
Output: Polar histogram H
Set $H[i]=0$, for all i=0..36
For each point (u, v) of I_B
  If $I_B(u, v)$==Obstacle
  α=angle between the line from (u, v) to F and the horizontal axis u
  $H[\alpha/5]$++
  End If
End For

The polar histogram H is then smoothed with a 1D Gaussian kernel, and its local maxima are identified. Only the local maxima that pass a certain threshold are kept, the rest are discarded.

Once the local maxima are found, the polar histogram is searched, around the local maxima, to find the range of values higher than half the value of the maximum. This way, the obstacle's angular limits are established, as seen in figure 6.

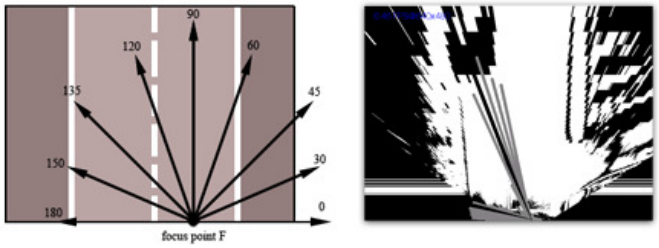

Fig. 6. The focus point and scanning directions (left), and the polar histogram (right)

Now that the angular limits of the obstacle are established, the only thing that remains to be determined is the distance. For each possible distance (radius), a triangle is formed using the two rays starting from the focal point and encompassing the object, and a third line whose distance from the focal point is equal to the selected radius, and who is perpendicular to the center line passing through the object. For each possible radius, and therefore for each possible triangle, the ratio between the number of obstacle points in the triangle and the total area of the triangle is computed. The minimum radius that has a significant value for this ratio (meaning the first triangle that has a significant amount of obstacle points) is kept as the distance to the obstacle.

Building the polar histogram is computationally demanding, due to the need of computing an angle for each obstacle point in the image, which means computing the inverse of the tangent function. To improve the processing time, a LUT for the arc tangent function for each position in the image is computed at initialization time.

IV. Experimental Results

To prove the feasibility of the approach, the application was deployed on various mobile phones, and tested on multiple traffic scenarios. The application was tested on multiple mobile devices, including: HTC Desire, HTC One S, Samsung Galaxy S3. The first device, the HTC Desire features a 5 megapixel camera and a single core 1 GHz CPU paired with 576MB ram, running Android operating system version 2.3 Gingerbread. The One S features a more powerful dual core 1.5 GHz CPU with 1GB ram and a 8 megapixel camera running Android 4.1 Ice Cream Sandwich. The last device used for testing is a quad core 1.4 GHz Samsung Galaxy S3 that features 1GB of ram and runs Android version 4.1 and that has a 8 megapixel camera. To drastically improve the performance of the obstacle detection algorithm, the most part of the code was written in native code using Android NDK.

In each of the above-mentioned mobile devices we used a fixed images size at a resolution of 640x480 pixels. The acquired images are in 24-bit, RGB format.

In order to test the accuracy of the system, we placed known patterns on the road surface in front of the vehicle to test the IPM image – figure 7. The patterns were placed at known distances and also proved to be useful to determine a correlation between the 2d image pixels and real world linear length expressed in meters.

Fig. 7. Calibrating the camera with known patterns to produce the IPM image; the patterns used on the road are highlighted

To test the overall performance we chose three popular devices featuring single core, dual core and quad core processing. We then used the same set of input images on all devices and measured the processing time required for each image.

The chart in figure 8 represents a comparison chart between the single core HTC Desire and the dual core HTC One S. The X axis represents the processing time required for the frame (expressed in seconds) and the Y axis represents the current frame rate.

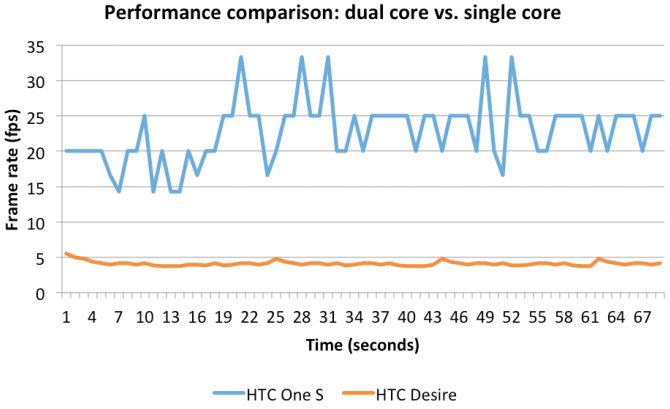

Fig. 8. HTC One S vs HTC Desire performance (dual core vs single core)

The chart in figure 9 represents the performance comparison between the dual core HTC One S and the quad core Samsung Galaxy S3. The representation used is the same as described for figure 8.

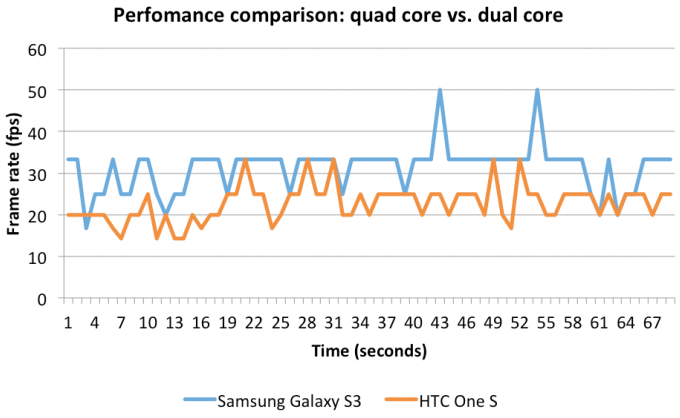

Fig. 9. Samsung Galaxy S3 vs HTC One S performance test (quad core vs dual core)

The effective range of the obstacle detection algorithm is between 4 and 40 meters. This interval has been chosen to provide a good accuracy both in urban environments and highway scenarios. The mobile device camera was calibrated to offer good accuracy on this region interest that has an effective width of 13 meters, the width of approximately two traffic lanes. Figure 10 better illustrates the system's effective range and region of interest.

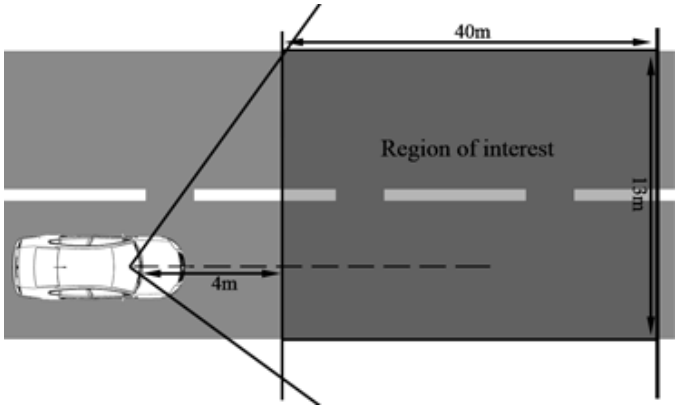

Fig. 10. Region of interest of the proposed system

In order to have a reference of the overall performance of the proposed solution and to validate the obstacle detection and distance determination in real world scenarios, we compared with existing alternatives available. The comparison was made on a two lane street. The two alternatives used are [10] and [11], both of which are available on the Android Play Store.

Figure 11.a represents the case where a vehicle is headed towards the ego-vehicle and it is not detected by the first application. Figure 11.b shows a case when a vehicle is departing from the ego-vehicle and is still not detected.

The second competitor was tested in similar conditions, figure 11.c illustrates the case when a vehicle is approaching the ego-vehicle and it is not detected. Also, in figure 11.d we can observe that the vehicle departing from the ego-vehicle was detected too far, making it irrelevant from the point of view of driving safety.

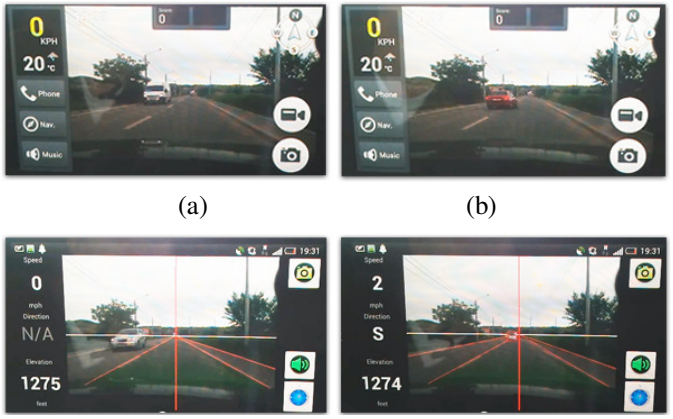

Fig. 11. iOnRoad test results in (a) and (b), drivea results in (c) and (d)

Our proposed system had better results in the two test scenarios and these results can be seen in figure 12.a and 12.b.

Fig. 12. DriveAssist test results (a) and (b)

The accuracy and robustness of the obstacle detection algorithm also refers to correctly identifying the obstacles in various traffic and weather conditions. The algorithm must be immune to various illumination conditions such as shadows that might be on the road surface. Figure 13.a illustrates a successful detection when building shadows are present on the road, whereas figure 13.b represents the detection in low light conditions during night.

Fig. 13. DriveAssist results with variable lighting conditions (a) and during night time (b)

In regards to the speed performance of the system, we managed an average of 8-10 frames per second on a dual core mobile phone. This includes the image acquisition, the processing of the image and also displaying the results. The overall performance increases directly proportional with the number of available cores, this means that on the quad core mobile phone the average frame rate increased to 13-15 frames per second. These results may vary depending on the luminosity of the scene and the number of detected obstacles.

V. CONCLUSIONS

The solution presented in this paper is a robust and fast obstacle detection system for Android based smartphones, which can improve the safety on the road and the overall driving experience for the user.

Further developments for the obstacle detection and distance determination on mobile devices include improving the color segmentation algorithm, and developing an obstacle tracking feature using the currently detected obstacles as input. Also, the calibration of the camera can be improved to help generating the IPM image by finding a way to use the mobile device's accelerometer to generate the camera parameters from the roll, yaw and pitch angle. The proposed software solution will soon be available for public use [12].

REFERENCES

[1] I. Ulrich and I. Nourbakhsh, "Appearance-Based Obstacle Detection with Monocular Color Vision", Proc. of the AAAI National Conference on Artificial Intelligence, Austin, TX, July/August 2000, pp. 866-871

[2] T. Kalinke, C. Tzomakas, and W. von Seelen, "A Texture based Object Detection and an Adaptive Model-based Classification", in Procs. IEEE Intelligent Vehicles Symposium'98, Stuttgart, Germany, October 1998, pp. 341–346.

[3] A. Bruhn, J. Weickert, and C. Schnorr, "Lukas/Kanade meets Horn/Schunck: Combining local and global optic flow methods", International Journal of Computer Vision, vol. 62(3), pp. 249-265, 2005.

[4] F. Stein, "Efficient computation of optical flow using the census transform", DAGM04, pp. 79–86, 2004.

[5] S. Tuohy, "Distance determination for an automobile environment using Inverse Perspective Mapping in OpenCV", Signals and Systems Conference (ISSC 2010), June 2010, pp. 100-105.

[6] A. Corti, V. Manzoni, S. Savaresi, "A centralized real-time Advanced Driver Assistance System based on smartphones", Advanced Microsystems for Automotive Applications 2012 (AMAA 2012), pp. 221-230.

[7] H.H.Little, J.J. Bohrer S. Mallot, H.A. Bulthoff, "Inverse perspective mapping simplifies optical flow computations and obstacle detection", Biological Cybernetics, vol. 64, pp. 177–185, 1991.

[8] M. Bertozzi, A. Broggi, A. Fascioli, "Stereo inverse perspective mapping: theory and applications", Image and Vision Computing, vol. 16, pp. 585-590, 1998.

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

[10] iOnRoad, website: www.ionroad.com

[11] Drivea, website: www.drivea.info

[12] DriveAssist, website: www.driveassistapp.com