Embedded Computing Framework for Vision-based Real-time Surround Threat Analysis and Driver Assistance

Frankie Lu Sean Lee Ravi Kumar Satzoda Mohan Trivedi
University of California, San Diego
[frlu,yhl014,rsatzoda,mtrivedi]@eng.ucsd.edu

Abstract

In this paper, we present a distributed embedded vision system that enables surround scene analysis and vehicle threat estimation. The proposed system analyzes the surroundings of the ego-vehicle using four cameras, each connected to a separate embedded processor. Each processor runs a set of optimized vision-based techniques to detect surrounding vehicles, so that the entire system operates at real-time speeds. This setup has been demonstrated on multiple vehicle testbeds with high levels of robustness under real-world driving conditions and is scalable to additional cameras. Finally, we present a detailed evaluation which shows over 95% accuracy and operation at nearly 15 frames per second.

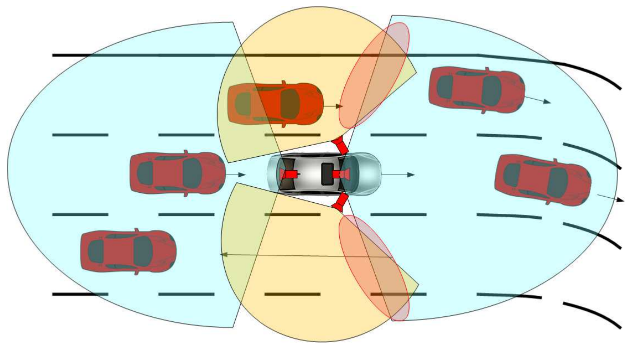
Figure 1: Surround scene assessment using cameras and embedded processors proposed.

1. Introduction

Scene perception and understanding are critical components required in the design and development of autonomous vehicles and advanced driver assistance systems (ADAS) [1]. One way to accomplish scene understanding is through vision; cameras are becoming an integral part of vehicles for sensing obstacles such as vehicles [2] and pedestrians [3]. Developing an independent computing environment in an automobile has constraints including limited hardware resources, running with little energy consumption, and using software economically [4,5].

Vision based methods for driver assistance have become very accurate, but are still based on computationally expensive algorithms. Current embedded and system-on-a-chip based platforms that can be put in automobiles today will be resource constrained and have to find a balance between compute speed and power consumption [5]. The goal is then to develop more efficient algorithms that cater to the computational constraints posed by an embedded platform, but at the same time resist compromising on the accuracy and robustness that is necessary for tasks like collision avoidance.

In this paper, we present a vision-based embedded computing framework that is able to analyze the surroundings of the vehicle and provide critical safety related alerts to the driver in real-time. The proposed system uses camera sensors as the sole sensing modality for analyzing the surroundings of the host or ego vehicle.

There are three main contributions in this paper. First, we present a technique for detecting 'high-threat' vehicles in front and rear of the ego-vehicle as seen by forward and rear facing cameras respectively from the ego-vehicle. The techniques are particularly designed to operate at real-time frame rates on embedded processors. Next, we present an optimized technique to detect the presence of vehicles in the blind spot areas. Finally, we present a distributed processing framework that employs the optimized detection modules in order to analyze the threat posed by surrounding vehicles to the ego-vehicle. Fig. 1 illustrates the proposed multi-perspective system. The entire system is implemented using Snapdragon embedded processors, and it has been tested under real-world driving conditions with high levels of accuracy.

2. Related Recent Studies

There is a rich body of literature which addresses various aspects of vision-based driver assistance systems. Lane detection [6], vehicle detection [7] and vehicle activity detection [8] in particular have been studied extensively. These operations have been further used to estimate the threat posed by surrounding vehicles on the ego-vehicle [9–11]. Although robust vision algorithms have been proposed for these algorithms and systems, there are fewer works in the context of embedded electronics that can be deployed in vehicles using resource constrained embedded CPUs [5, 12–14].

Computationally efficient lane detection is implemented in works such as [14–16]. A complete lane detection and tracking system is implemented on FPGAs in [15]. A lane departure and drift warning system is implemented on a Snapdragon processor in [14]. Lane feature extraction using steerable filters is presented in [16]. Similarly vehicle detection has also been explored and is an integral component of commercial ADASs such as MobilEye [17].

Multiple perspectives and modalities are used to understand the surroundings of the vehicles in works such as [18, 19]. In [18], multi-perspective camera based ADASs are discussed for implementation on Texas Instruments processors. Another similar work is presented in [19], wherein multi-modal and multi-camera based ADASs are surveyed for embedded implementation. Stereo-vision systems have also been implemented on embedded platforms in works such as [20, 21]. It should be noted that consumer vehicle manufacturers have deployed some of these algorithms as part of their vehicles using electronic control units (ECUs) for applications such as lane departure warning (LDW). Additionally, there are plug-and-play vision-based electronic assistance systems such as MobilEye [17] which can be installed in vehicles to assist the driver. Dedicated automotive grade embedded vision processors such as Texas Instruments' EVE have also been developed for deployment in vehicles [22].

The above list provides a brief review of related works. A more detailed study can be found in [13, 19]. A study of these works reiterates the point that the challenges of translating computer vision algorithms into robust and real-time embedded electronics are yet to be fully addressed.

3. Surround Scene Analysis System: Overview

An overview of our proposed embedded surround scene analysis system is presented. The main goals when designing our proposed system were the following:

1. The system should detect and evaluate the threat posed by the surrounding vehicles that are classified as 'high-threat'.
2. The system should operate in real-time under real-world driving conditions.

We want to elaborate on the threat levels posed by surrounding vehicles. A low to intermediate threat vehicle is anything that is not covered in the definition of a 'high-threat' vehicle and is not analyzed. A 'high-threat' vehicle is one which can pose an immediate or high threat to the ego-vehicle. These include the vehicles: (a) in the host lane (both in front and rear of the ego-vehicle), (b) in the adjacent lanes on the right and left of the ego-vehicle (both in the front and rear views of the ego-vehicle), (c) in the blind spots on the adjacent lanes of the ego-vehicle. This definition identifies the primary regions of interest that will be analyzed by the proposed system. If there are multiple vehicles in any lane, only the vehicle that is closer to the ego-vehicle will be considered as a high-threat vehicle. Each 'high-threat' vehicle will be assigned a threat value by our system.

Fig. 1 illustrates the desired fields of view for the system to analyze the surroundings of ego-vehicle. It comprises of four cameras that capture the visual information from the forward view, rear view and the left and right blindspots of the ego-vehicle. In the current setup, the cameras capture nearly 300 degrees around the vehicle. The red ovals in Fig. 1 indicate the regions that are currently not covered by the cameras described in this paper. The system is currently designed to address traffic flows on highway driving conditions.

4. High-threat Vehicle Detection for Embedded Implementation

We will present two different vehicle detection techniques to analyze the threat to the ego-vehicle from the regions of interest (RoIs) described in Section 3. The objective is to design vehicle detection methods that are optimized sufficiently for implementation on an embedded processor such that they operate at real-time frame rates. The first vehicle detection technique caters to the forward and rear views seen by the ego-vehicle, while the second is used to detect the presence of a vehicle in its blind spots.

We have two datasets that we will use for evaluation of our implementation. Each dataset is a collection of real-world drives, under variable driving conditions (sunny, cloudy, lots of shadows, traffic flow, etc.). The first dataset is a two-camera view of the front and rear of the ego-vehicle. It contains over 10,000 annotations each of front and rear vehicles, and will be used for training our front and rear-view detectors. The second dataset is the four camera surround view that we have presented for this paper (front, rear, and two blind spots). There are over 100,000 frames in total for each view. A small portion of this dataset will be used for training the blind-spot detectors. For the rest of

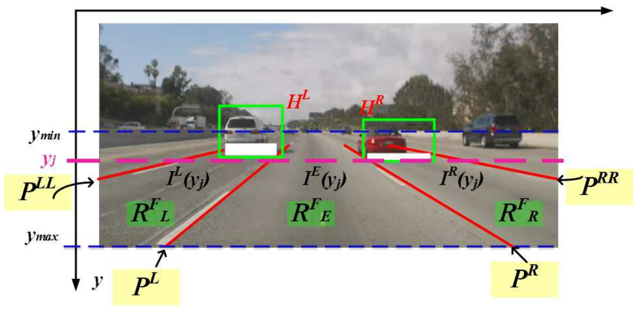

Figure 2: Illustration of the RoIs and the different parameters that are used for vehicle detection in the forward view.

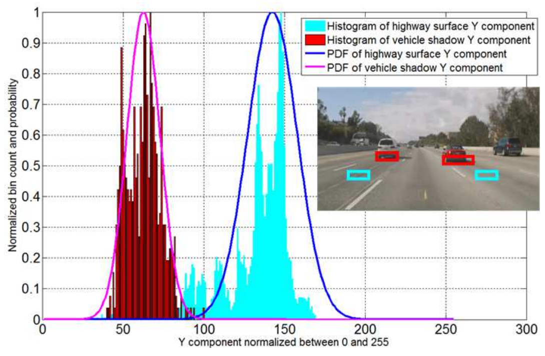

Figure 3: Probability density functions for road surface and under vehicle region pixels. Sample annotations are shown for the road surface and under vehicle region pixels.

this dataset, we will select a variety of sequences of around 1000 frames each to use for testing, that demonstrate varied road conditions.

4.1. Vehicle Detection in Forward and Rear Views

We present an embedded platform for detecting vehicles in the surround of an ego-vehicle. From our definition of 'high-threat' vehicles, there are three RoIs where they can be found - the host lane, the adjacent left lane and the adjacent right lane. For the rest of this section, we discuss the detection of vehicles in these three RoIs in front of the ego-vehicle.

Before continuing the description of our algorithm, we would like to further emphasize the differences of using a traditional desktop processor versus an embedded platform. In order to understand the computational bottlenecks, we profiled a Haar-Adaboost based vehicle classifier on a Snapdragon 810 processor. The Haar-Adaboost classifier was applied to an forward-view input image in a conventional multi-scale and multi-window approach. The computation, which runs on real time (30 fps) on a desktop processor, took 1 second to process each frame (1 fps) on the Snapdragon processor. In order to overcome this limitation on timing, a two-step context-aware approach is employed for detecting vehicles in the three RoIs.

Now we describe the algorithm for front-view detection, which is also applied to detect vehicles in the rear-view. First, the host and adjacent lanes are detected using a simplified variant of the steerable filter based approach in [16, 23]. In the modified approach, we do not apply RANSAC (which is used in [23]) for eliminating outliers. Instead, we use the road models to estimate possible lane markings of the host lane. The adjacent lanes are also required for generating RoIs in the proposed system, but extracting adjacent lane features of the adjacent lanes will increase computational complexity. Given that the system is primarily designed for highway driving and that lanes in the U.S. have standard lane widths, an inverse perspective mapping [24] is used to estimate the lane positions from the host lane markings. The detected lane markings would result in three RoIs: adjacent left lane, ego-lane and adjacent right lane. The different RoIs (R^F_L, R^F_E and R^F_R) and the lanes are shown in Fig. 2). A more in-depth analysis and evaluation of the lane detection method is found in [16, 23].

We now want to look for 'high-threat' vehicles in each RoI (three vehicles, one from each RoI, which are nearest to the ego-vehicle). Instead of applying the vehicle classifier directly in each of the regions, hypothesis candidates are first generated by capitalizing the presence of under-vehicle dark regions. The lower parts of vehicles (bumper, shadow) seen in front of the ego-vehicle have a lower light intensity than the other parts of the scene. The darker regions are caused by the reflected light from other surfaces being greater than the reflected light underneath the vehicle [25]. Each RoI will be analyzed for the presence of under-vehicle dark regions (shown in Fig. 3 inset annotations). We first annotate the under-vehicle regions of the first dataset. Using a YUV color format from each image frame, we analyze the mean Y component of the under-vehicle regions is collected from the training data, resulting in the probability density functions $P_{UV}(\mu_{UV}, \sigma_{UV})$ and $P_{RS}(\mu_{RS}, \sigma_{RS})$ for the under-vehicle region and the road surface respectively, as shown in Fig 3.

Given a test image frame I with the boundaries of the RoIs computed as described previously, I is scanned along the y-direction starting from the bottom of the image, i.e., the point that is nearest to the ego-vehicle (refer to Fig. 2). For a particular y-position y_j, the image pixels in that scan line are divided into the three segments $I^L(y_j)$, $I^E(y_j)$, and $I^R(y_j)$ corresponding to the three RoIs (denoted in Fig. 2). The image pixels in each of three scan line segments $I^L(y_j)$, $I^E(y_j)$, and $I^R(y_j)$ are then thresholded using the PDFs for under-vehicle region and road surface - $P_{UV}(\mu_{UV}, \sigma_{UV})$ and

$P_{RS}(\mu_{RS}, \sigma_{RS})$ using the following equation:

$$I(x_k, y_j) = \begin{cases} 1 & \text{if } P_{UV}(I(x_k, y_j)) \geq P_{RS}(I(x_k, y_j)) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

After thresholding the scan line segments, the number of pixels that are classified as under-vehicle region is determined for each segment. If the number of under-vehicle region pixels form $\beta\%$ of the scan line segment, then that scan line segment is marked as an under-vehicle shadow segment. This process is repeated with the next scan line. In each RoI, the number of *consecutive* under-vehicle shadow segments are counted which are denoted by n^L, n^E and n^R corresponding to the three RoIs. If n^L is greater than κ, then the n^L scan line segments are considered as possible constituents of an under-vehicle region in the adjacent left lane RoI. The same is repeated with the ego-lane and the adjacent right lane RoIs. If an RoI is found to have possible under-vehicle region, then a hypothesis window H is drawn in which vehicle classifiers are applied.

The hypothesis window H is scaled depending on how far the detected shadow is from the ego-vehicle. The width-to-height ratio is calculated by averaging the width-to-height ratios of all the samples in our first dataset (around 10,000 vehicles) and is found to be 1.325. β is learned from the training data and is set to 40% so that it can handle vehicles that are changing lanes. It was found that setting κ to 3 gives the most optimal results with over 95% accuracy and real-time frame rates. All the different parameters are illustrated in Fig. 2.

We note that instead of applying the vehicle detection classifiers on the entire image, they are applied in limited patches of H which is much more computationally efficient. We train two Adaboost classifiers (one for front, one for rear) with Haar-like features using our first described dataset, using the active learning methodology described in [26] is used. If a vehicle is detected within H, the remaining scan lines in the RoI above y_{bottom} are not processed anymore. The above approach is repeated for the three different RoIs resulting in the detection of a maximum of three vehicles corresponding to the three nearest vehicles in the three RoIs.

4.2. Vehicle Detection in Blind Spots

In the forward and rear views, it was important to determine the position of the vehicles with respect to the ego-vehicle in the lanes. In the blind spots however, the proposed system simply determines the presence or absence of a vehicle. This is because the proposed system continuously monitors the movement of the vehicle in the rear view until it disappears from the rear camera view and enters into the blind spot RoI. Given that the blind spot RoI is less than 5 meters, we can just determine a binary decision value (i.e. presence or absence of a vehicle in the blind spot) without losing much in vehicle resolution.

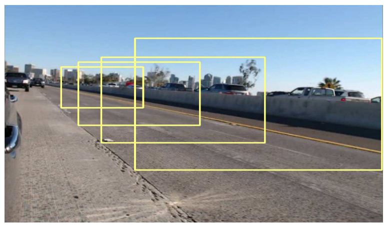
Figure 4: Four detection windows where the vehicle is detected in the blind spot.

We use a simplified clustering based method. The following details are explained using the camera view from the right blind spot. The same technique is applied to the left blind spot.

Training: We create a training dataset comprising of 500 positive and negative samples each. Each positive sample includes the side view of a vehicle from the right blind spot. Histogram of oriented gradients (HOG) [27] feature vectors are generated for the samples resulting the two sets of features corresponding to positives and negatives respectively: $\mathbf{H}^+ = \{\mathbf{h_0}^+, \mathbf{h_1}^+, \ldots, \mathbf{h}^+_{\mathbf{N}_+}\}$ and $\mathbf{H}^- = \{\mathbf{h_0}^-, \mathbf{h_1}^-, \ldots, \mathbf{h}^-_{\mathbf{N}_-}\}$. All the features are normalized as follows:

$$\mathbf{h_i'}^+ = \frac{\mathbf{h_i}^+ - \mu}{\sigma} \quad \text{and} \quad \mathbf{h_i'}^- = \frac{\mathbf{h_i}^- - \mu}{\sigma} \quad (2)$$

where μ and σ are the mean and standard deviation of all HOG vectors in $\mathbf{H}^+$ and $\mathbf{H}^-$. The mean normalized HOG vectors are then used to generate two centroids, which are the means of the two clusters, i.e.,

$$\mathbf{h}_\mu^+ = \frac{1}{N_+}\sum_{i=0}^{N_+} \mathbf{h_i'}^+ \qquad \mathbf{h}_\mu^- = \frac{1}{N_-}\sum_{i=0}^{N_-} \mathbf{h_i'}^- \quad (3)$$

Testing: Given a test image I, we consider 4 detection windows W_1, W_2, W_3 and W_4 along the length of the car which are spaced at every 1 meter with respect to the ground plane (refer to Fig. 4). HOG features are generated for the four windows (appropriate scaling to produce HoG feature vectors of similar length). After normalizing the feature vectors, the Euclidean norm is computed between each feature vector $\mathbf{t_j}$ and the two centroids $\mathbf{h}_\mu^+$ and $\mathbf{h}_\mu^-$. Therefore, each window W_j gives a distance $d_j^+ = ||\mathbf{h}_\mu^+ - \mathbf{t_j}||$ and $d_j^- = ||\mathbf{h}_\mu^- - \mathbf{t_j}^+||$. The window W_j is assigned as vehicle if $d_j^+ > d_j^-$. In a given frame, if either of the four windows detects a vehicle, a vehicle is considered to be present in the blind spot of the vehicle.

5. Embedded System Setup

Our embedded system implementation is described.

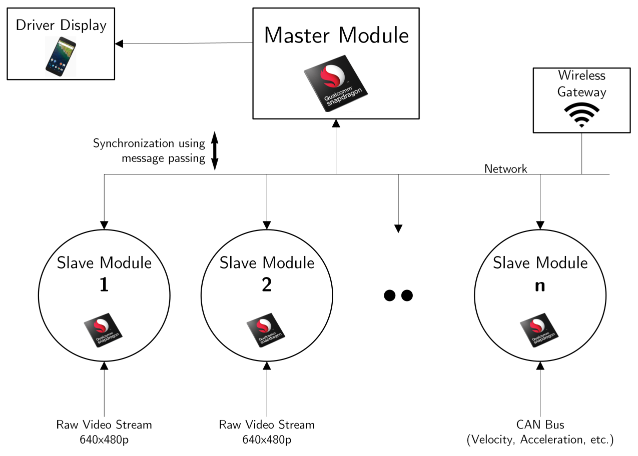
Driver Display
Master Module
Wireless Gateway
Synchronization using message passing
Network
Slave Module 1
Slave Module 2
Slave Module n
Raw Video Stream 640x480p
Raw Video Stream 640x480p
CAN Bus (Velocity, Acceleration, etc.)

Figure 5: Scalable system architecture for surround scene analysis.

5.1. Computing Modules

We choose to use Android as the operating system because its platform can be extended for use on other embedded systems [28]. It demonstrates a resource constrained environment that is associated with trying to implement driver assistance algorithms in vehicles. For our setup, we will use a development board with an embedded Snapdragon 810 SoC. The Snapdragon 810 is a consumer-grade processor found in many smartphone devices as of 2015 and is representative of the state of modern embedded platforms.

Early on, we found that attempting to do all the risk analysis computation on a single Snapdragon system had two problems. The first was that all the computations done in Section IV could not be done in real-time on a single processor. The second was that the required bandwidth of accepting multiple video streams (front, rear, and blind spot cameras) was too high. Therefore we separate the computations to multiple Snapdragon boards, which we refer to as computing modules (we will use the terminology 'modules' and 'processors' interchangably for the rest of this paper).

Each computing module performs a task related to risk assessment. Our current system uses four computing modules. The first and second are used for lane and vehicle information of the front and rear views, respectively. The third and fourth look for vehicles in the left and right blind spots. Modules 1-4 use a single USB webcam connected to the system, receiving and processing 640×480p video streams at around 15 fps. Lane and vehicle detection are performed using methods described in Section IV.

5.2. Synchronization of Modules

We designate one of the computing modules as the master driver, and all the other modules as slave modules. In our implementation, the medium of communication used is a TCP/IP wireless connection. The master driver hosts a server-client connection with each slave module.

To make communication among modules fast, synchronization is done using a message-passing scheme. At every frame, each slave module performs its given task at and computes a risk value, which is passed to a buffer at the master driver. Since the master driver may receive risk values from each slave module at varying frame rates, it will select only the most current value from each slave module and use it to compute an overall risk assessment of the scene. Thus the total frame rate of the system is dependent on the frame rate of the master driver. We more formally define the messages that are sent from the slave modules in Section 5.4. The system is able to run at 10-15 cycles per second, which translates to a real-time frame rate required for automotive applications [29].

The TCP/IP communication method we have discussed can be replaced with other forms of low-latency communication. This, coupled with the small footprint of the computing modules, gives us freedom to design the system in many spaces inside the vehicle.

5.3. Scalability of Modules

We note that we can also scale up the number of modules that are used in the system. Examples of additional modules include two more cameras to provide for a 360 degree surround view, or CAN bus access. By doing so, we are able to distribute computationally demanding algorithms into separate modules without slowing down the system. The complete proposed architecture is shown in Figure 5.

5.4. Threat Estimation

Now we will consider the threat posed by the surrounding vehicles using the detection information from each module.

Threat posed by vehicles in forward and rear views: After detecting the vehicles, one set of detection windows is generated by each module resulting in $\Psi^F = \{W_L^F, W_E^F, W_R^F\}$ and $\Psi^R = \{W_L^R, W_E^R, W_R^R\}$, where the superscripts ^{F} and ^{R} refer to forward and rear RoIs respectively, and the subscripts $_L$, $_E$ and $_R$ refer to the adjacent left, ego and adjacent right lanes respectively. A detection window W is a set of four parameters, i.e. $W = [x, y, w, h]$, where (x, y) denotes the coordinates of the top left corner of the window and w and h denote the width and height of the detection window. If the vehicle detector does not detect a vehicle in a lane, then that window is set to a null value.

Given the detection windows in Ψ^F and Ψ^R, the relative distances of the vehicles are estimated using the inverse perspective mapping (IPM) transformation matrix $\mathscr{H}$, which is computed using a one-time calibration method described in [24]. $\mathscr{H}$ transforms a coordinate in the image domain to real-world ground plane. Given $\mathscr{H}$, the mid point of the bottom edge of the detection window is used to determine the position of the vehicle along the road from the ego-vehicle resulting in two vectors $\mathbf{d}_\mathbf{V}^\mathbf{F} \in \mathbb{R}^3$ and $\mathbf{d}_\mathbf{V}^\mathbf{R} \in \mathbb{R}^3$, corresponding to the distances of the vehicles in front and rear of the ego-vehicle. If any window is null (no vehicle in the RoI), then that particular distance d is set to d_{max} which is the maximum distance from the ego-vehicle which is being processed for threat estimation. The rear module now acts as a client and sends $\mathbf{d}_\mathbf{V}^\mathbf{R}$ to the host module, i.e. the forward Snapdragon module. The host combines $\mathbf{d}_\mathbf{V}^\mathbf{F}$ and $\mathbf{d}_\mathbf{V}^\mathbf{R}$ to generate a 6-valued vector $\mathbf{d}_\mathbf{V} = \left[d_L^F, d_E^F, d_R^F, d_L^R, d_E^R, d_R^R\right]^T$. The first three values in $\mathbf{d}_\mathbf{V}$ denote the distances from the forward view and the last three values denote the distances of the vehicles from the rear-view.

The host module uses the distance vector $\mathbf{d}_\mathbf{V}$ to compute a risk vector Γ using a risk function $f_r(\cdot)$. The risk function can be defined in multiple ways depending on the factors that are considered for risk estimation. In this work, we consider a distance based risk function which computes the risk vector Γ using the following equation:

$$\Gamma = f_r(\mathbf{d}_\mathbf{V}, d_{max}) = \left(1 - \frac{\mathbf{d}_\mathbf{V}}{d_{max}}\right) \tag{4}$$

where $d_{max} = 50$. Γ is a six-valued vector with six risk values γ_i corresponding to the six RoIs, where $\gamma_i \in [0,1]$. γ_i tends to 0 if the vehicle is far away from the ego-vehicle. If a vehicle is close to the ego-vehicle, γ_i approaches 1. The threat values in Γ are used to alert the driver if there is any vehicle that poses high risk to the ego-vehicle. This is done by thresholding the risk values in Γ, where the threshold is set using the time to collision (TTC) metrics defined in literature [30]. This risk metric is based on the relative distance only and we would like to highlight that this is one of the possible definitions for the risk function $f(\cdot)$.

Threat posed by vehicles in blind spots: In the case of blind spots, if a vehicle is detected, a threat value of 1 is assigned to the vehicle. This is based on the naturalistic driving studies (NDS) which consider vehicles in blind spots as high risk vehicles to the ego-vehicle [30]. Therefore, the two slave modules that monitor the blind spots of the ego-vehicle send either a 1 or 0 depending on whether they detect a vehicle in the blind spot. The host module uses this information to alert the driver of the presence of vehicles in the blind spots.

6. Performance Evaluation & Discussion

In this section, a detailed evaluation of the proposed techniques and the system is presented. An evaluation of the detectors is presented first, followed by the details of the system's performance under real-world driving conditions.

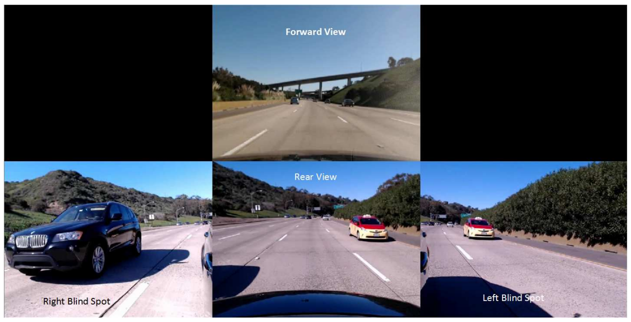

Figure 6: Sample images from the surround scene analysis dataset.

6.1. Accuracy Analysis

In order to evaluate the accuracy of the proposed techniques for vehicle detection in the different perspectives, we use the surround view dataset that we have previously discussed in Section 4. Sample images of the dataset from the four perspectives are shown in Fig. 6.

We first evaluate the vehicle detection technique that was proposed in Section 4.1 for the forward and rear perspectives using Haar-Adaboost classifiers. In order to do this, 10 test sequences are chosen, each containing over surround-view 4000 frames. In each frame, three nearest vehicles to the ego-vehicle are annotated within a distance of 50 meters from the ego-vehicle. We calculate the number of true positives (TP), false positives (FP), and false negatives (FN) for each sequence to determine true positive rate (TPR) and false detection rate (FDR) which are given by $TPR = TP/(TP+FN)$ and $FDR = FP/(FP+TP)$. Table 1 shows the results of our testing. We see that the total TPR over all our test sequences is around 95% with an FDR of less than 6%. Overall, the accuracy is high and has not been compromised by the optimizations we have detailed for running the Haar-Adaboost cascades on embedded CPUs. Sample detection results from the two perspectives are shown in Fig. 7.

Table 1: Accuracy results of the vehicle detection algorithm applied over 10 driving sequences from forward and rear views

	TP	FP	FN	TPR	FDR
Front View	1442	74	79	0.948	0.049
Rear View	2340	150	118	0.952	0.060
Total	3782	224	197	0.950	0.056

In order to evaluate the detection accuracy of the proposed method for vehicle detection in the blind spots, the following metric is used. As described in Section 4.2, if a vehicle is detected in any of the four detection windows, it is considered as a true positive. We consider a set of 10 overtaking sequences, where a vehicle passes the ego-vehicle. There are nearly 40 frames in each sequence resulting in over 400 frames where there is a vehicle in the blind spot. An additional 600 frames without any vehicle in the blind spot are considered as negative test images. The proposed detection technique for the blind spots showed 100% detection rate with no false positives. Fig. 8 shows sample detection results of the vehicles in the blind spots.

Table 2: Computation times for different perspectives

	Processor	Timing (fps)
Front view	Snapdragon 810	14.5
Rear view	Snapdragon 810	15.3
Left blind spot view	Snapdragon 600	13.2
Right blind spot view	Snapdragon 600	13.2

6.2. Computational Speed Analysis

The main objective of proposing our optimized vehicle detection techniques is to ensure that the system operates in real-time without compromising on accuracy. The techniques were profiled for timing on Snapdragon embedded processors.

We note that with the synchronization scheme we have described, the most accurate FPS will be set by the lowest FPS module among all the modules (13.2 fps). However, updates for other faster modules will update as fast as the speed of the master module.

The computation times are listed in Table 2. We note the usage of Snapdragon 600 (lower grade) processors for the blind spot detections because their algorithm is not as computationally expensive as that for front and rear view detection, which are implemented with Snapdragon 810 processors. Our computational time includes image acquisition by the camera, lane detection (for forward and rear views), and vehicle detection. The timing of the master (front view) module also includes the time taken to compute the threat or risk values. Overall, we achieve a frame rate of around 13 fps, which is considered as real-time operation for automotive safety applications [29].

7. Risk Assessment & Threat Visualization

In this section, we present the threat/risk estimation results that are computed using one of the sequences. This is particularly illustrated for the forward and rear perspectives because the risk is binary in the case of blind spots. Fig. 9 plots the risk vector Γ for a 500-frame sequence. It can be seen that the risk values change in different ways during the drive sequence. The risk values are approaching 1 in certain segments of the drive in Fig. 9, which implies that there are multiple instances when there is high level of threat to the ego-vehicle from the surrounding vehicles.

An analysis of this plot can reveal a variety of semantics that are related to the dynamics of the surrounding vehicles with respect to the ego-vehicle. For example, there is possibility of tailing vehicle behind the ego-vehicle in ego-lane for nearly two-thirds of the time in this sequence. This is indicated by the magenta plot in Fig. 9. Similar high values of threat are also seen from the vehicles approaching the ego-vehicle from the rear right lane (cyan plot in Fig. 9). Additionally, the vehicle in front of the ego-vehicle in the adjacent left lane also posed high threat for about 150

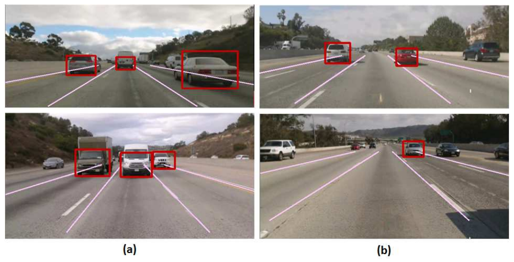

Figure 7: Sample detection results of the vehicles in the forward and rear views. Detection is performed on the vehicles that are present in the ego-lane, adjacent left and adjacent right lanes only.

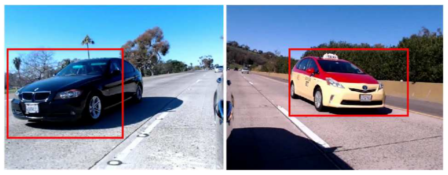

Figure 8: Sample detection results of the vehicles in the right and left blind spots.

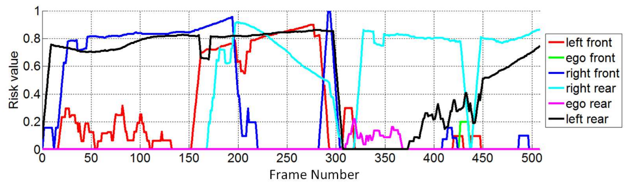

Figure 9: Threat vector Γ that is generated for the vehicles that are detected in the forward and rear views of a sequence with 500 frames.

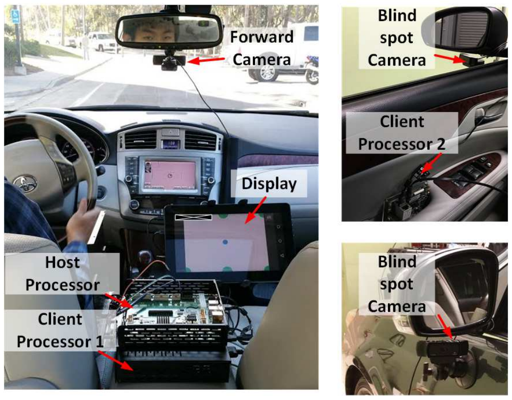

Figure 10: Hardware setup inside the testbed showing the different processors and cameras. The masked block (black rectangle with a cross) is made to preserve anonymity for the review process.

frames or 10 seconds. The threat values in Fig. 9 are verified by visually going through the segments.

7.1. System Demonstration

The entire embedded system is developed and deployed in our automobile testbed. We connect the master Snapdragon processor to an Android tablet as a driver display. Fig. 10 shows the hardware setup of the demonstration in our testbed. The same testbed was used to acquire the datasets we have used for training and testing.

A 3D visualization developed on Unity3d is shown in Fig. 11(a), which shows the relative positions of the surrounding vehicles. Each vehicle is color coded with a light source to indicate the threat it poses to the ego-vehicle. A variant of this system was demonstrated at the 2016 Consumer Electronics Show (CES) as a collaboration with Qualcomm. A snapshot of that demonstration is also shown in Fig. 11(b).

8. Conclusions

In this paper, we have presented a detailed analysis, evaluation and discussion of a real-time embedded driver assistance system. This system includes a four-camera setup to assess threat posed by surrounding vehicles to the ego-vehicle and is able to run at real-time speeds. We have introduced the use of a variety of optimization techniques that ultimately help to reduce the search space of potential vehicle and make detections run much faster.

Evaluation of the system has shown that the high rates of accuracy for vehicle detection are maintained. The proposed embedded framework allows for a scalable system, in which additional modules can be added in the future to account for more sources of threat estimation.

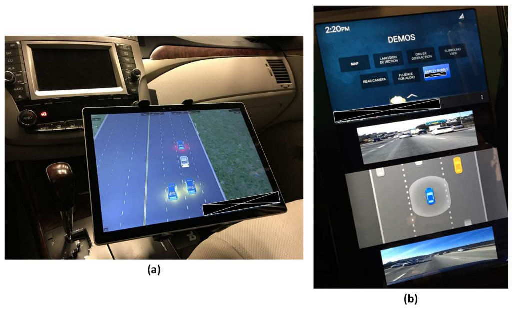

Figure 11: (a) 3D visualization on a portable monitor showing the threat assessment on the vehicles detected by the proposed system. Each vehicle has a light blob indicating the threat associated with the vehicle, (b) The visualization presented in the demonstration of our system at CES 2016.

References

[1] M. Menze and A. Geiger, "Object Scene Flow for Autonomous Vehicles," in *Computer Vision and Pattern Recognition*, 2015, pp. 3061–3070. 1

[2] M. Rezaei and M. Terauchi, "Vehicle Detection Based on Multi-feature Clues and Dempster-Shafer Fusion Theory," in *Pacific-Rim Symposium on Image and Video Technology*, 2014, pp. 60–72. 1

[3] S. Zhang and R. Benenson, "Filtered Channel Features for Pedestrian Detection," *2015 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1751–1760, 2015. 1

[4] S. Chakraborty, M. Lukasiewycz, C. Buckl, S. Fahmy, P. Leteinturier, and H. Adlkofer, "Embedded systems and software challenges in electric vehicles," *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 424–429, Mar. 2012. 1

[5] F. Stein, "The challenge of putting vision algorithms into a car," *2012 IEEE CVPR Workshops*, pp. 89–94, June 2012. 1, 2

[6] A. Borkar, M. Hayes, and M. T. Smith, "A Novel Lane Detection System With Efficient Ground Truth Generation," *IEEE Trans. on Intelli. Transp. Systems*, vol. 13, no. 1, pp. 365–374, Mar. 2012. 2

[7] B.-f. Wu, C.-c. Kao, S. Member, and C.-l. Jen, "A Relative-Discriminative-Histogram-of-Oriented- Gradients-Based Particle Filter Approach to Vehicle Occlusion Handling and Tracking," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 8, pp. 4228–4237, 2014. 2

[8] A. Almagambetov, S. Velipasalar, and M. Casares, "Robust and Computationally Lightweight Autonomous Tracking of Vehicle Taillights and Signal Detection by Embedded Smart Cameras," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3732–3741, 2015. 2

[9] H. Takahashi, D. Ukishima, K. Kawamoto, and K. Hirota, "A Study on Predicting Hazard Factors for Safe Driving,"

IEEE Transactions on Industrial Electronics, vol. 54, no. 2, pp. 781–789, 2007. 2

[10] G. Ogawa, K. Kise, T. Torii, and T. Nagao, "Onboard Evolutionary Risk Recognition System for Automobiles Toward the Risk Map System," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 2, pp. 878–886, 2007. 2

[11] S. Sivaraman and M. M. Trivedi, "Dynamic Probabilistic Drivability Maps for Lane Change and Merge Driver Assistance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 5, pp. 2063–2073, 2014. 2

[12] P. Jeong and S. Nedevschi, "Efficient and robust classification method using combined feature vector for lane detection," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 15, no. 4, pp. 528–537, Apr. 2005. 2

[13] R. K. Satzoda and M. M. Trivedi, "On Enhancing Lane Estimation using Contextual Cues," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 11, pp. 1870 – 1881, 2015. 2

[14] R. K. Satzoda, S. Lee, F. Lu, and M. M. Trivedi, "Snap-DAS : A Vision-based Driver Assistance System on a Snapdragon TM Embedded Platform," in *IEEE Intelligent Vehicles Symposium*, no. Iv, 2015, pp. 660–665. 2

[15] R. Marzotto, P. Zoratti, D. Bagni, A. Colombari, and V. Murino, "A real-time versatile roadway path extraction and tracking on an FPGA platform," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1164–1179, Nov. 2010. 2

[16] X. An, E. Shang, J. Song, J. Li, and H. He, "Real-time lane departure warning system based on a single FPGA," *EURASIP Journal on Image and Video Processing*, vol. 2013, no. 1, p. 38, 2013. 2, 3

[17] Mobileye, "No Title," 2015. [Online]. Available: mobileye.com 2

[18] S. Dabral, "Trends in Camera Based Automotive Driver Assistance Systems (ADAS)," in *IEEE MWCAS*, 2014, pp. 1110–1115. 2

[19] J. Horgan, C. Hughes, J. McDonald, and S. Yogamani, "Vision-Based Driver Assistance Systems: Survey, Taxonomy and Advances," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. Ieee, Sept. 2015, pp. 2032–2039. 2

[20] G. P. Stein, Y. Gdalyahu, and A. Shashua, "Stereo-Assist : Top-down Stereo for Driver Assistance Systems," in *IEEE Intelligent Vehicles Symposium*, 2010, pp. 723–730. 2

[21] A. D. Costea, A. V. Vesa, and S. Nedevschi, "Fast Pedestrian Detection for Mobile Devices," in *IEEE ITSC*, 2015, pp. 1–6. 2

[22] S. Kamath and B. Valentine, "Implementation Details of Mid-Level Vision on the Embedded Vision Engine (EVE)," in *IEEE ISCAS*, 2014, pp. 1283–1287. 2

[23] J. McCall and M. Trivedi, "Video-Based Lane Estimation and Tracking for Driver Assistance: Survey, System, and Evaluation," *IEEE Trans. on Intelli. Transp. Systems*, vol. 7, no. 1, pp. 20–37, Mar. 2006. 3

[24] M. Bertozzi and A. Broggi, "GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection." *IEEE transactions on image processing*, vol. 7, no. 1, pp. 62–81, Jan. 1998. 3, 6

[25] H. Mori and N. M. Charkari, "Shadow and rhythm as sign patterns of obstacle detection," in *Industrial Electronics, 1993. Conference Proceedings, ISIE'93-Budapest., IEEE International Symposium on*. IEEE, 1993, pp. 271–277. 3

[26] S. Sivaraman and M. M. Trivedi, "A General Active-Learning Framework for On-Road Vehicle Recognition and Tracking," *IEEE Trans. on Intelli. Transp. Systems*, vol. 11, no. 2, pp. 267–276, June 2010. 4

[27] N. Dalal and W. Triggs, "Histograms of Oriented Gradients for Human Detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR05*, vol. 1, no. 3, pp. 886–893, 2004. 4

[28] C. Maia, L. Nogueira, and L. Pinho, "Evaluating android os for embedded real-time systems," *International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 63–70, 2010. 5

[29] C. Caraffi, T. Vojir, J. Trefny, J. Sochman, and J. Matas, "A system for real-time detection and tracking of vehicles from a single car-mounted camera," *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pp. 975–982, Sept. 2012. 5, 7

[30] "Researcher Dictionary for Video Reduction Data," VTTI, Tech. Rep., 2012. 6