Single-view obstacle detection for smart back-up camera systems

Jeffrey Lalonde, Robert Laganière
University of Ottawa
Ottawa, ON, Canada
laganier@eecs.uottawa.ca

Luc Martel
Cognivue Corp.
Gatineau, QC, Canada
lmartel@cognivue.com

Abstract

This paper presents an implementation of a 3D reconstruction algorithm for the detection of static obstacles from a single rear view parking camera. To guarantee efficiency and accuracy of the solution, we adopted a feature-based approach in which interest points are tracked to estimate the vehicle's motion and multiview triangulation is performed to reconstruct the scene. A full implementation of the algorithm has been achieved on a parallel SIMD array processor unit embedded in a smart automotive camera system. Current in-vehicle beta trials suggest that system's performance meets industrial requirements for real-world use in back-up camera systems.

1. Introduction

Traditional parking assistance systems use radar or ultrasonic range-finders to detect obstacles. These have proven reliability for detecting large metal objects, even in poor visual conditions. However, these sensors tend to miss small objects because they lack sufficient spatial resolution [23]. Backup cameras are an attractive safety feature because, in addition to being cheaper than range-finders, they display the rear visual field to the driver. The advantage to a vision-based obstacle detector, then, is a combined camera and range sensor in a single, low-cost, piece of hardware.

This paper presents a driver-assistance backup system that uses rear-view camera in order to perform obstacle detection during reverse drive manoeuvres. Our objective was to design a camera-based approach that can get comparable results to approaches that use more costly active sensors. The scene directly behind the vehicle is captured by a rear-mounted video camera, as shown in Figure 1. The camera is generally mounted behind just above the rear license plate, tilted toward the ground to provide a good view of the ground directly behind the vehicle. The system has to detect static obstacles using only the visual input the rear-mounted camera and to alert the driver of any obstacles in the collision path of the vehicle. Because our solution would be ported to a dedicated image processing chip, it had to be as simple as possible, run in real-time and perform sufficiently to be released as a consumer product.

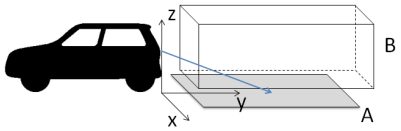

Figure 1. A rear-mounted parking camera setup. (A) is the region in which ground features are used for motion estimation. Any feature located in the collision volume (B) is determined to be an obstacle. The blue arrow indicates the camera's optical axis.

The paper is organized as follows. The next section reviews some of the relevant works. The next section presents the designed algorithm for obstacle detection. Section 4 describes the on-chip implementation. Section 5 presents the experimental results. Section 6 is a conclusion.

2. Related works

Our single-view parking system is composed of two main steps: i) robust estimation of vehicle motion from visual data; ii) 3D reconstruction of the detected obstacle.

2.1. Visual Odometry

Measuring vehicle motion is critical to any monocular 3D structure-based obstacle detection method. Triangulating image features accurately requires not only good feature tracking, but also good motion estimation. Motion estimation of the ground is particularly challenging because of the scarcity of strong features on its surface. As a result, many works have been published on the specific problem of estimating motion from vehicle-mounted cameras; a topic commonly referred to as *visual odometry*.

Ground-based methods combine a motion model of the ground plane with a projective model to obtain a parametric model of the ground's optical flow. The optical flow parameters are fit by observing motion on the image, and these are related to the parameters of the motion model. Some

methods track ground features and use outlier-robust methods to estimate the ground flow parameters [16]. Others use the optical flow over the ground surface [19]. Others use whole-image matching in the so-called *direct methods* [18, 5, 2, 13]. These ground-observation-only approaches have the advantage of being relatively simple to compute. However, they are only suited for scenes in which the roadway is large and unobstructed.

Some approaches use all objects in the scene to estimate motion by 3D scene reconstruction [7, 15]. The advantage to such methods is that they are able to cope with low ground texture and scenes cluttered with static objects. The disadvantage is that these methods require costly bundle adjustment processes for sufficient robustness.

2.2. Obstacle Detection

Stereo methods make use of the disparity field to infer the depth of image features. This allows for detection of the ground plane (if it is unknown) and the detection of obstacles as any feature off the ground plane. Stereo-vision systems have the advantage that no motion is required between the vehicle and the scene in order to recover 3D structure. Also, they do not require strict constraints on the ground surface and so are preferred for autonomous navigation on undulated terrain [1] [20] [3]. Well calibrated stereo systems can achieve impressive results, but at a severe cost, both financially and economically, which restricts the adoption of these systems to market products [19].

Monocular systems operate at a reduced cost and require little maintenance, but lack the depth perception of stereo systems. Monocular systems form three broad classes; appearance-based, motion-based and reconstruction-based methods. Appearance-based methods [21, 11, 6] use colour and shape cues to differentiate image regions belonging to the ground from regions belonging to obstacles. The advantage of appearance-based methods is speed, simplicity and the ability to detect very small obstacles. However, they suffer from underlying assumptions. First, obstacle detection can only occur if obstacles differ in appearance from the ground. Second, obstacles cannot occupy the area near the vehicle assumed to be ground, otherwise their appearance gets incorporated into the ground model. Third, obstacle distance can only be estimated if the obstacle is detected at its base. Of course, all of these assumptions are often violated in a parking scenario.

Motion-based methods are complimentary to appearance-based methods in that they largely ignore colour and shape but rely heavily on the motion of image features and optical flow. Of the motion-based methods, some rely on external sensors to measure vehicle motion [4, 22] while others estimate vehicle motion directly from the images [24, 12, 8, 10, 23]. The ground, assumed to be planar, is expected to move a certain way on the image. A parametric model is constructed to capture the essence of this motion. Then, the ground motion is observed, with or without the help of external sensors, and the parameters of the motion model are estimated. Regions of the image that agree with this model are considered ground, and the rest are considered obstacles. The strength of motion-based methods is the ability to detect obstacles based on scene structure without an explicit 3D recovery of the scene. However, these methods do not provide a direct estimate of the obstacle distance.

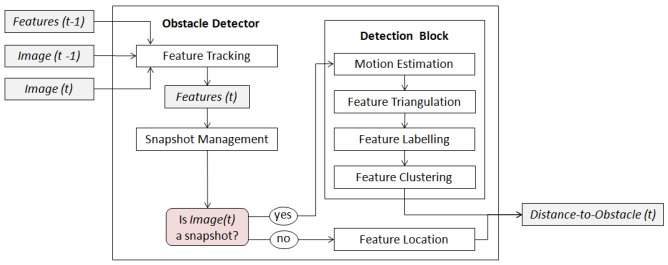

Figure 2. Overview of the obstacle detection algorithm.

Structure-based methods [23] detect obstacles by an explicit 3D reconstruction of the scene. For this to be possible with a single camera, the camera must be in motion, so that images of the scene may be taken from different poses. Obstacles are detected from image features that lie above the ground plane in the 3D model. Reconstruction-based methods are more complex and computationally expensive, but offer a direct estimate of the obstacle's location in space.

Our challenge was to develop and test a monocular obstacle detection algorithm and implement it on a dedicated hardware architecture. In order to meet this challenge, we have opted for the simpler approach of tracking features. Tracking features provides a single mechanism with which we can do both motion estimation and obstacle detection. This algorithm is described in the next section.

3. The OD Algorithm

Our algorithm achieves obstacle detection through 3D reconstruction of the scene using a single camera. To do this we detect and track image features across multiple frames. From the tracked features we estimate the motion of the camera and triangulate the features to create a 3D model of the scene. Features located within a collision corridor behind the vehicle are labelled as obstacles and the system reports the nearest obstacle distance to the driver.

An overview of the algorithm is shown in Figure 2. The current and previous images, and the previous list of features are fed to the *Feature Tracking* module to obtain the current list of features. The *Snapshot Management* module analyzes the motion of the features to determine what images (or snapshots) to use for 3D reconstruction. The *Motion Estimation* module estimates the planar motion of

the camera between the current snapshot and all snapshots in the past. The *Feature Triangulation* module finds the 3D location of features, provided they obey the epipolar constraint. The *Feature Labelling* module assigns interpretive labels (such as "ground" or "obstacle") to each feature based on their location. The *Feature Clustering* module spatially clusters detected features to reduce false detections. The *Feature Location* module updates the location of triangulated points assuming a constant height. Finally, the distance to the nearest obstacle is reported to the driver.

3.1. Feature Tracking

The *Feature Tracking* module has three main responsibilities; (1) track existing features, (2) delete invalid features and (3) detect new features. It takes as input the current image, the previous image, the previous list of features, and it outputs the current list of features.

The *Feature Filter* module deletes features based on two criteria; (1) features with erratic motion and (2) features in close proximity to other features. Erratically moving features are deleted because, since frame rate of the camera is high and the vehicle moves smoothly, we expect the image of the scene to move smoothly. So, any erratic motion in the image is due to either tracking error or some object moving about in the scene. In either case, the feature is unwanted for both ground motion estimation and 3D reconstruction. We measure the non-smoothness of a feature's trajectory as a deviation from a linear trajectory over a recent window of time. Specifically, we least-square-fit the last ω_δ pixel positions of a feature to a line $\mathbf{y}(t) = \mathbf{a}t + \mathbf{b}$. Then we compute the mean reprojection error of the trajectory

$$\bar{\delta} = \frac{1}{m} \sum_i \|\mathbf{y}(i) - \mathbf{x}_i\|. \tag{1}$$

We define a *smoothness of trajectory threshold* ϵ_δ and remove any feature with $\bar{\delta} > \epsilon_\delta$. We use the last $\omega_\delta = 5$ frames and set $\epsilon_\delta = 10$ pixels. The smoothness window of 5 frames corresponds to a very short-term trajectory (0.16s at 30fps). We expect the vehicle motion to be smooth w.r.t. this time interval, so vehicle rotations do not have a significant impact on trajectory smoothness.

The remaining features are then filtered based on their proximity to other features in order to limit the computational load of the algorithm. For each pair of features, if their distance is less than the threshold r_{min}, we eliminate the feature with the greater $\bar{\delta}$ value. We set r_{min} to half-width of the tracking template size (7 pixels). We then generate a *vacancy mask*; a binary image indicating the regions of the image that are currently unoccupied by any features. It is only in vacant regions that we search for features to track, since we do not want to create trackers where some already exist. Next, we use the Good Features to Track algorithm [17] to detect and prioritize strong corner features in the vacant regions.

Of course, our ability to detect and track features reliably depend on video quality and external conditions. Poor lighting, motion blur, small image resolution and small video framerate all adversely effect feature tracking. External conditions such as poor lighting, dirty or out-of-focus lenses, shiny obstacles and road surfaces and precipitation also decrease tracking ability.

3.1.1 Selecting Good Ground Features

For a feature to possibly belong to the ground, it must be within the image region onto which the ground is projected. To assert this, we define a *region of interest* (ground ROI) in vehicle coordinates (cf. Figure 1). Only features that project to the ground ROI are used for motion estimation. To be a *good* feature for ground motion estimation between frames t_1 and t_2, it must meet the following criteria:

1. The feature must be present at both frames t_1 and t_2.
2. The feature must currently be within the ground ROI.
3. The feature must have sufficient visual disparity. Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be the feature's ideal image coordinates at frames t_1 and t_2, respectively. Then we assert $\|\mathbf{x}_2 - \mathbf{x}_1\| > d_{\text{min}}$ where $d_{\text{min}} = 20$ pixels is the minimum disparity. This is to avoid using stationary features since they may correspond to shadows or objects moving with the vehicle, or points at infinity.

3.2. Planar Motion Estimation

The motion of the vehicle is estimated using the identified good ground features. A planar motion is most conveniently expressed in a coordinate system in which the normal is aligned with one of the axes. By choosing to align the normal with the z axis, we may represent planar motion as $\{\mathrm{R}_p, \mathbf{T}_p\}$, where

$$\mathrm{R}_p = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_p = \begin{bmatrix} t_x \\ t_y \\ 0 \end{bmatrix}. \tag{2}$$

Here θ is the angle of rotation and $[t_x, t_y]^{\mathrm{T}}$ is the translation within the plane. So the camera motion $\{\mathrm{R}, \mathbf{T}\}$, in terms of the vehicle's planar motion $\{\mathrm{R}_p, \mathbf{T}_p\}$ and the relative orientation of the camera w.r.t. the vehicle $\{\mathrm{R}_c, \mathbf{T}_c\}$ is:

$$\mathrm{R} = \mathrm{R}_c^{\mathrm{T}} \mathrm{R}_p \mathrm{R}_c \qquad \text{and} \qquad \mathbf{T} = \mathrm{R}_c^{\mathrm{T}} \left((\mathrm{R}_p - \mathrm{I}) \, \mathbf{T}_c + \mathbf{T}_p \right) \tag{3}$$

The following algorithm is used to estimate camera motion:

1. For each image pair $(\mathbf{x}_1^i, \mathbf{x}_2^i)$ get the 2D world point pair $(\mathbf{g}_1^i, \mathbf{g}_2^i)$, where $\mathbf{g}_j^i = w(\mathbf{x}_j^i, Z_i)$ where $w(\cdot)$ is a an image to world transformation. Here the world z-components are assumed to be zero (on the ground).

2. Estimate the planar motion by running RANSAC for the point pairs $(\mathbf{g}_1^i, \mathbf{g}_2^i)$, $i = 1...N > 2$ to get the motion parameters $\{\theta, t_x, t_y\}$ of the model $\{\mathrm{R}_p, \mathbf{T}_p\}$.

3. Transform the motion to camera coordinates by computing $\{\mathrm{R}, \mathbf{T}\}$ according to (3).

3.3. Snapshot Management

Triangulation of correspondence points requires a non-zero translation of the camera. In fact, there must be sufficient translation to overcome the discretization, noise, and tracking error. To this end, we select keyframes for which the estimated camera displacement between them is greater than some appropriate threshold. We call these keyframes *snapshots*. The snapshots are triggered by vehicle displacement, rather than elapsed time. The faster the vehicle moves, the more frequently snapshots are taken, and vice versa. A snapshot is simply indicated by its frame number. At any given time, we have a *snapshot list* $S = \{s_1,\ s_2,\ \ldots\ s_m\}$ consisting of the numbers of all the frames that were chosen as snapshots. If the number of features within the ground ROI is less than N_{min}, we clear the snapshot list. To avoid long-term tracking drifts and make use of recently detected features, we also set a maximum snapshot interval threshold $\Delta_s = 300$. At 30 fps, this amounts to a maximum snapshot interval of 10 seconds.

If there are sufficient features in the ground ROI, we estimate the planar motion. If the motion was successfully computed (*i.e.* there are enough "good" ground features), then we check if there is sufficient displacement between the frames. Then the current frame is added to the snapshot list if $\|\mathbf{T}\| > d_s$. We have found a practical value for d_s to be $0.2h$, where h is the camera height.

If the current image is selected as a snapshot, then the *Detection Block* is called. Otherwise, triangulated feature locations are updated in the Feature Location Module.

3.3.1 Inter-Snapshot Motion Estimation

We now have a list of snapshots $S = \{s_1 \ldots s_m\}$ where s_i is the frame number of the i^{th} snapshot and s_m is the current frame. We want to estimate the motion between each snapshot in the list and the last one. These motion parameters could be obtained from previous motion estimates with little computational effort. However, to avoid the aggregation of tracking error and motion estimation error, we estimate these parameters directly each time a snapshot is taken.

The planar motion $\{\mathrm{R}_1, \mathbf{T}_1\}$ between the previous snapshot s_{m-1} and the current one s_m is estimated in the *Snapshot Management* module, so it need not be computed again. We therefore begin by estimating the planar motion $\{\mathrm{R}_2, \mathbf{T}_2\}$ between snapshots s_{m-2} and s_m as described in Section 3.2. If $\{\mathrm{R}_2, \mathbf{T}_2\}$ is successfully computed, then we estimate the motion between s_{m-3} and s_m. We continue until all the snapshot interval motions are estimated, or until one of the interval motions cannot be computed. Once we are finished, we have a list of k successfully computed planar motion parameters $\{\{\mathrm{R}_1, \mathbf{T}_1\}, \{\mathrm{R}_2, \mathbf{T}_2\}, \ldots \{\mathrm{R}_k, \mathbf{T}_k\}\}$, with $1 \leq k \leq m$.

3.4. Feature Triangulation

Triangulation is the process of depth recovery of scene points from known camera motion. For each tracked feature f, we must determine how many views (*i.e.* snapshots) can be used for triangulation. If the position of feature f in snapshot s_{m-i} is designated by $\mathbf{x}_{s_{m-i}}(f)$ or $\mathbf{x}_i$ for simplicity, then the following expression applies:

$$Z_0(\mathbf{x}_i \times \mathrm{R}_i\mathbf{x}_0) = \mathbf{T}_i \times \mathbf{x}_i \qquad i = 1 \ldots k \tag{4}$$

If we denote $\mathbf{a}_i \equiv \mathbf{x}_i \times \mathrm{R}_i\mathbf{x}_0$ and $\mathbf{b}_i \equiv \mathbf{T}_i \times \mathbf{x}_i$, then we have the system of k equations $Z_0\mathbf{a}_i = \mathbf{b}_i$. If we stack all the $\mathbf{a}$ and $\mathbf{b}$ vectors to form the vectors $\mathbf{A}$ and $\mathbf{B}$, respectively, we get an equivalent single vector equation; $Z_0\mathbf{A} = \mathbf{B}$ which is equivalent to:

$$Z_0 = \frac{\sum_{i=1}^{k} (\mathbf{x}_i \times \mathrm{R}_i\mathbf{x}_0) \cdot (\mathbf{T}_i \times \mathbf{x}_i)}{\sum_{i=1}^{k} \|\mathbf{x}_i \times \mathrm{R}_i\mathbf{x}_0\|} \tag{5}$$

Obviously, if the feature f has not been tracked in snapshot s_{m-i}, then $\mathbf{x}_i$ does not exist and is therefore not included in the least-square solution. Also, due to noise and tracking error, point pairs are not guaranteed to obey the epipolar constraint; point pairs too far from the disparity will then be rejected. We also reject point pairs with insufficient disparity or points located too close to the epipole (which is always located in the image in the case of backward vehicle motion) because they lead to unreliable estimates. More details are given in [9]. If all point pairs of a feature are rejected, then the depth remains undefined.

3.5. Feature Labelling

With all the features robustly triangulated from multiple views, the next step is to label the features as obstacles or not, based on their 3D location. We define an obstacle as any feature occupying the collision volume behind the vehicle (cf Figure 1). The depth of the volume (along the y-axis) can be made small if the user wants to remove long-range detections. The height of the volume must be set to reflect the sensitivity of the scene reconstruction; The volume height must be high enough such that there are not too many false detections of noisy ground features. We have found a practical collision volume height to be $0.2h$, where h is the camera height; point below this height are considered to be on the ground plane (or sufficiently close to it).

If triangulation of a given feature is successful, then we can label that feature based on its 3D location. Any feature inside the collision volume is labelled an *obstacle*. For

features outside the collision volume, we have two possible labels. If they are below the collision volume height, we label them *ground*. Otherwise, we label them *above-ground*. This ground/above-ground distinction is not critical to the detection aspect of the algorithm, but is useful in verifying that the scene is correctly interpreted. Features that were not successfully triangulated remain labelled as *undefined*.

We also provide an extra label for detecting moving obstacles. If a feature has enough disparity on the image, but does not move along its epipolar line, this could indicate a moving object in the scene. When this occurs, we do not triangulate the feature, but we label it *moving*. Now, because we do multiview triangulation we must check these conditions for each pair of views. If all point pairs fail, and the majority reason for failure is that the point does not move along its epipolar line, then we label this point as *moving*.

3.6. Feature Clustering

Unfortunately, due to smooth surfaces, edges and glare, there remain a significant number of features that are mistracked for which the epipolar constraint is still obeyed. These mistracks cannot be disambiguated from good ones. We have observed that these false detections tend to be spatially sparse, both on the image and in the reconstructed space. Then, to filter out these false detections, we spatially cluster the detected features and set a minimum cluster size N_c for detection. The disadvantage to this, of course, is that true detections from small, or sparsely featured obstacles are also filtered.

We cluster detected features in the following way. First, we randomly select a feature and create a cluster for that feature. Then, all other features that have a similar distance to the vehicle are added to the cluster. For all the features that were not accepted into the cluster, the process is repeated to form a second cluster. This continues until every feature is assigned to a cluster. Then we remove all clusters with fewer than the minimum cluster size. The intention here is to cluster large surfaces perpendicular to the ground, so features are clustered based on their distance to the vehicle only. Specifically, we use the relative difference in the feature's world y-coordinate. If feature 1 and feature 2 have respective world y-coordinates Y_{w1} and Y_{w2}, then feature 1 is clustered with feature 2 if

$$\frac{|Y_{w1} - Y_{w2}|}{|Y_{w1}|} < w_c \tag{6}$$

where w_c is the *cluster width threshold*. Of course, the final cluster configuration depends on the randomly chosen seed features. We would like a configuration that clusters the most features in the least amount of clusters. To obtain this we wrap the clustering process in a RANSAC process and search for the configuration that maximizes n_f/n_c, where n_f is the number of features assigned to a cluster and n_c is the number of clusters.

4. On-chip implementation

4.1. The APEX architecture

Parallel processing technology is extremely well suited for the first stages of computer vision applications. The Cognivue CV220X series of programmable ICPs enables real-time embedded image and video analytics powered by massively parallel low power array processor technology called the APEX. The APEX includes: i) a massively parallel SIMD Array Processor Unit (APU) made up of 96 Computational Units (CUs) each with dedicated memory; ii) General purpose dedicated RISC processor (ARM926) to handle control and non-parallel algorithms iii) Multi channel DMA engines devised for efficient data movement into and out of APEX device memory; iv) Hardware acceleration blocks for control, entropy encoding/decoding and bit-level processing.

Typically, the APEX performs all the heavy parallel processing that would typically require an FPGA, while the ARM processor analyzes the extracted feature results, in parallel. As importantly, this parallel operation is non-blocking (unlike a traditional multi-core approach) because the CUs are working on their own local memory leaving the main external SDRAM memory free for the ARM's use. By locating the computational units (CUs) close to the device memory without other extra level of cache, APEX can offer speed per area and power advantages over traditional Harvard architecture, VLIW based DSPs or even GPGPU. Also, since there are many more CUs operating in parallel than the largest number of functional units or ALUs in a DSP, the operating frequency can be kept lower for the same number of operations per cycles, thereby reducing power.

4.2. Implementation of the algorithm on APEX

Feature tracking is by far the most time consuming process of the algorithm. Our initial implementation used the KLT tracker [14]. However, to decrease the execution time, we finally adopted an alternative approach in which motion estimation is performed on a top-view transformed image. Running the algorithm on a top-view has two strong advantages. Ground features on the top-view are not perspective skewed, but undergo only translation and (slight) rotation. As such, we can track features using a simple block-matching (BM) scheme, which is much faster than the KLT tracker. Moreover, the APEX's CUs have optimized instructions to perform SADs which makes the BM-Tracker implementation more efficient than the KLT-Tracker. In addition, using a top-view effectively narrows the detector's focus to the region directly behind the vehicle, the periphery of the input image not being processed.

The implementation is described by Figure 3. We will not describe here the portion of the application controlling the image sensor and the display but rather focus on the ac-

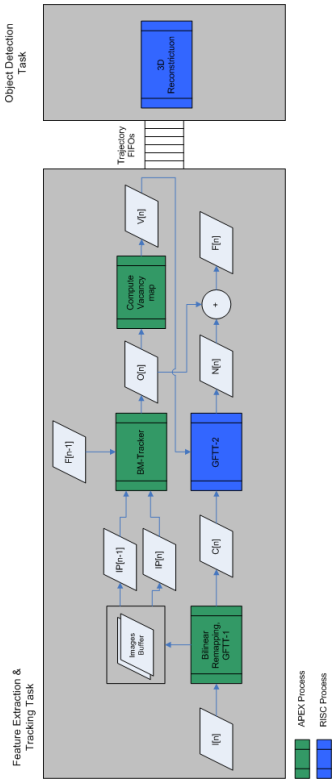

Figure 3. Algorithm implementation on APEX.

tual object detection algorithm. CogniVue's CV220x series ICP processors have 2 ARM926 RISC core, one of them running Nucleus OS being used as the host (ARM-1) and from which the APEX control code (APEX Core Framework) executes. The algorithm is implemented by 2 tasks running on the host processor. The Feature Extraction and Tracking (FDT) task is a combination of APEX and RISC processing. For simplicity we will call APEX any processing not executing on the host ARM core. The Obstacle Detection (OD) task is a pure RISC task. The two tasks run completely asynchronously so that the Object Detection task can fill up the processing cycles unused by the FDT tasks. Those cycles are available when the FDT task is waiting on the APEX to complete its processing.

APEX is programmed by creating process graphs. A process graph is a representation of a specific sequence of operations. APEX process graphs are shown in green in Figure 3. They implement one processing pass during which the data is moved from external DDR to the device memory for the graph operations to execute and the data is moved back to external DDR. The first processing graph take a YVU 422 image input I[n], does a bilinear remapping on the Y component and apply Good Feature To Track on the result. The output is a perspective corrected image IP[n] and a feature point response image C[n]. The perspective corrected image IP[n] is then fed along with the previous perspective corrected image IP[n-1] and the previous list of feature points to the BM-Tracker module. The BM-Tracker output is a list of tracked old features O[n]. We use the list O[n] to compute a vacancy map V[n]. The last stage is done by the RISC processor and uses the vacancy map V[n] and the corner response C[n] to compute a list of new features N[n]. When processing the next frame IP[n+1] the list of features F[n] is the sum of the two lists O[n] and N[n]. The remapping and GFTT detection takes on average $9.2ms$ to run on one frame. BM tracking requires an average of $22.1ms$

The OD task was completely implemented in fixed point on the host ARM processor. It was implemented asynchronously to the FDT task since its processing time varies greatly. It executes the bulk of the processing (the reconstruction) only when a snapshot is detected and its processing time per snapshot was greatly dependant on the number of current snapshots in the list and on the number of obstacle features. It takes about $1.7ms$ to update the vacancy map while the reconstruction process takes on average $35.5ms$ per snapshot.

As a result, the system is capable of processing up to 30 fps VGA when tracking between 500 and 600 features per frame.

5. Experimental results

For testing the algorithm, we have installed a Boyo Vision CMOS rear view colour camera (VTB170) at the rear of a minivan and calibrated it using OpenCV's calibration routine, which estimates the camera matrix and distortion parameters from a series of checkerboard images. The camera's orientation, w.r.t. the ground was manually estimated based on the corresponding homographic top-view transformation. We captured 14 videos of the vehicle backing into various stationary obstacles. The raw video resolution was 640×480 at 30 fps and the resolution after dewarping and cropping was 576×370. The obstacles include large objects (parked vehicles, shrubs, dumpster), medium sized objects (garbage can, fire hydrant, leaf bag), and narrow objects (bike parking rack, wooden 2x4 used as construction site barricade). For performance evaluation, we have established a ground truth of obstacle position by manually marking the base of the objects on the image, for every frame. A series of tests were performed in order to determine the optimal set of parameter values.

To evaluate the obstacle detection range we calculate the detection rate for a given true obstacle distance. Specifically, we divide a $5m$ detection range into $20cm$ bins. Let n_i be the number of snapshots for which an obstacle is located in bin i. Let m_i be the number of snapshots for which an obstacle in bin i was detected. Both n_i and m_i are counted over all fourteen videos. Then, the overall detection rate for the i^{th} bin is given by m_i/n_i. This can be

thought of as the likelihood of detecting an obstacle, given the obstacles distance from the vehicle.

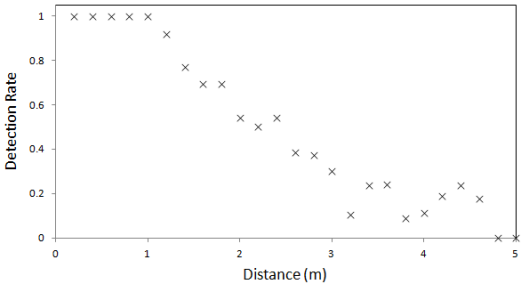

Figure 4. Detection rate as a function of obstacle distance.

As we see in Figure 4, the detection rate is roughly inversely proportional to the obstacle's distance. The detection rate for obstacles $1m$ is 1. That is, in all fourteen videos, obstacles within $1m$ are detected 100% of the time. Obstacles at $2m$ are detected half of the time, and obstacles at 3m are detected one third of the time. The rapid drop in the detection rate is due to the unfortunate fact that obstacles are always in the vehicle's (and so, the camera's) direction of travel. This results in obstacle features having the least amount of disparity for a given camera displacement (compared to the most favorable situation of sideways motion). It is this "minimal disparity" situation that is the greatest technical challenge in scene reconstruction in the context of a parking camera.

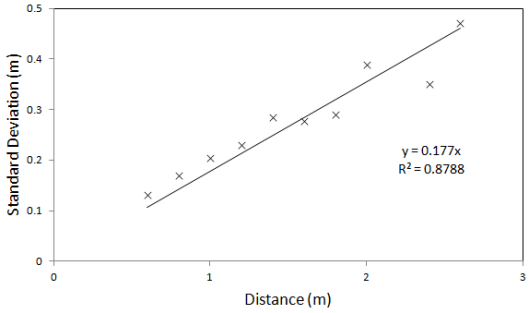

Figure 5. Standard deviation of distance-to-obstacle estimate as a function of distance. The data is fit to a straight line passing through the origin.

To measure the precision of the distance-to-obstacle estimate provided by the detector, we compared the computed distances with the true ones. Again, we divide the detection range into $0.2m$ bins. We then compute the standard deviation of $d_{\text{true}} - d_{\text{OD}}$ (the difference between the true distance and the reported distance) for all occurrences within a given bin. From Figure 5, we see that the standard deviation of $d_{\text{true}} - d_{\text{OD}}$ is proportional to d_{true}. If we define the uncertainty of the detector as being one standard deviation, then the reported distance has a relative uncertainty of 18%.

Finally, Figure 6 shows the output of the OD algorithm for three different obstacles; (a) a large dumpster, (b) a garbage can and (c) a bicycle parking rack. For each obstacle, we show a snapshot of the output when the obstacle is at an approximate distance of $2m$ (top), $1m$ (center) and $0.5m$ (bottom). The points drawn on the images are the features tracked by the Feature Tracking Module. These points are coloured to indicate the labeling provided by the OD module: green points are on the ground, blue points are below the ground, yellow points are above the ground (but not inside the collision volume), red points are obstacles and white points have not yet been labeled. The distance to the nearest obstacle feature is displayed at the top-left of the image, and the projection of that distance on the ground is the horizontal red line. Note that the images shown are the dewarped images.

6. Conclusion

This paper presented a successful implementation of an obstacle detection algorithm for a driver-assistance backup system. The designed solution uses a single rear-view camera and a real-time implementation has been achieved on a parallel SIMD array processor unit embedded in a smart automotive system.

The proposed system successfully meets the objectives of reliability and efficiency as demonstrated by experimentation. Namely, we have shown a very high detection rate for obstacles within $1m$ of the vehicle. The algorithm runs at 25fps on an image cognition processor.

References

[1] P. Batavia and S. Singh. Obstacle detection in smooth high curvature terrain. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA '02)*, May 2002. 2

[2] J. R. Bergen, P. Anandan, T. J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. pages 237–252. Springer-Verlag, 1992. 2

[3] T. Chang, S. Legowik, and M. N. Abrams. Concealment and obstacle detection for autonomous driving. In *Proceedings of the Robotics & Applications*, pages 28–30, 1999. 2

[4] W. Enkelmann. Obstacle detection by evaluation of optical flow fields from image sequences. *Image Vision Comput.*, 9(3):160–168, 1991. 2

[5] B. K. P. Horn and E. J. Weldon. Direct methods for recovering motion. *International Journal of Computer Vision*, 2(1):51–76, 1988. 2

[6] I. Horswill. Visual collision avoidance by segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 902–909. IEEE Press, 1994. 2

[7] B. Kitt, J. Rehder, A. Chambers, M. Schonbein, H. Lategahn, and S. Singh. Monocular visual odometry using a planar road model to solve scale ambiguity. In *Proc. European Conference on Mobile Robots*, September 2011. 2

(a) (b) (c)

Figure 6. Output of the OD algorithm for three objects: (a) a dumpster, (b) a garbage can and (c) a bicycle parking rack. The points drawn on the image are tracked features and are colour-coded as follows: Green = ground, blue = below ground, yellow = above ground (but not in the collision volume), red = obstacles and white = unknown.

[8] W. Kruger. Robust real-time ground plane motion compensation from a moving vehicle. *Mach. Vision Appl.*, 11:203–212, December 1999. 2

[9] J. R. Lalonde. Monocular Obstacle Detection for Moving Vehicles. Master's thesis, University of Ottawa, Ontario, Canada, 2011. 4

[10] G. Lefaix, ric Marchand, and P. Bouthemy. Motion-based obstacle detection and tracking for car driving assistance. In *ICPR (4)'02*, pages 74–77, 2002. 2

[11] L. M. Lorigo, R. A. Brooks, and W. E. L. Grimson. Visually-guided obstacle avoidance in unstructured environments. In *IEEE Conf. on Intelligent Robots and Systems*, pages 373–379, 1997. 2

[12] M. I. A. Lourakis and S. C. Orphanoudakis. Visual detection of obstacles assuming a locally planar ground. In *Proceedings of the Third Asian Conference on Computer Vision-Volume II*, ACCV '98, pages 527–534, London, UK, 1997. Springer-Verlag. 2

[13] S. Lovegrove, A. J. Davison, and J. Ibanez-Guzman. Accurate visual odometry from a rear parking camera. In *Intelligent Vehicles Symposium*, pages 788 – 793, 2011. 2

[14] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision (ijcai). In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 674–679, April 1981. 5

[15] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proc. of Computer Vision and Pattern Recognition*, volume 1, pages I–652 – I–659 Vol.1, june-2 july 2004. 2

[16] D. Scaramuzza, F. Fraundorfer, and R. Siegwart. Real-time monocular visual odometry for on-road vehicles with 1-point ransac. In *Proc. of The IEEE International Conference on Robotics and Automation (ICRA)*, May 2009. 2

[17] J. Shi and C. Tomasi. Good features to track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593 – 600, 1994. 3

[18] G. P. Stein, O. Mano, and A. Shashua. A robust method for computing vehicle ego-motion. In *In IEEE Intelligent Vehicles Symposium (IV2000)*, 2000. 2

[19] T. Suzuki and T. Kanade. Measurement of vehicle motion and orientation using optical flow. In *1999 IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems.*, pages 25 – 30, 1999. 2

[20] A. Talukder, R. Manduchi, A. Rankin, and L. Matthies. Fast and reliable obstacle detection and segmentation for cross-country navigation. In *In IEEE Intelligent Vehicles Symposium*, pages 610–618, 2002. 2

[21] I. Ulrich and I. Nourbakhsh. Appearance-based obstacle detection with monocular color vision. In *Proceedings of AAAI*, 2000. 2

[22] D. Willersinn and W. Enkelmann. Robust obstacle detection and tracking by motion analysis. pages 717 – 722, 1997. 2

[23] K. Yamaguchi, T. Kato, and Y. Ninomiya. Moving obstacle detection using monocular vision. *2006 IEEE Intelligent Vehicles Symposium*, pages 288–293, 2006. 1, 2

[24] Z. Zhang, R. Weiss, and A. R. Hanson. Obstacle detection based on qualitative and quantitative 3d reconstruction. *IEEE Trans. on PAMI*, 19:15–26, 1997. 2