



Real-Time Bird's Eye Surround View System: An Embedded Perspective

Mo'taz Al-Hami, Raul Casas, Subhieh El Salhi, Sari Awwad & Fairouz Hussein

To cite this article: Mo'taz Al-Hami, Raul Casas, Subhieh El Salhi, Sari Awwad & Fairouz Hussein (2021) Real-Time Bird's Eye Surround View System: An Embedded Perspective, Applied Artificial Intelligence, 35:10, 765-781, DOI: [10.1080/08839514.2021.1935587](https://doi.org/10.1080/08839514.2021.1935587)

To link to this article: <https://doi.org/10.1080/08839514.2021.1935587>



Published online: 25 Jun 2021.



Submit your article to this journal [↗](#)



Article views: 2041




View related articles [↗](#)



View Crossmark data [↗](#)



Real-Time Bird's Eye Surround View System: An Embedded Perspective

Mo'taz Al-Hami^a, Raul Casas^b, Subhieh El Salhi ^a, Sari Awwad^c,
and Fairouz Hussein^a

^aDepartment of Computer Information Systems, The Hashemite University, Zarqa, Jordan; ^bIP Group, Cadence Design Systems, San Jose, California, USA; ^cDepartment of Computer Science & Applications, The Hashemite University, Zarqa, Jordan

ABSTRACT

Bird's-eye surround view is a new type of Advanced Driver Assistance System (ADASs) that provides drivers with a real-time 360° top-down view of their vehicle. This paper presents an architecture for a surround view system based on a Tensilica Vision P5 embedded DSP. Fish-eye cameras mounted on each side of a vehicle are employed in the proposed model. The cameras are able to cover the entire view around the vehicle with overlapping areas between neighboring cameras. We use a morphing approach for joint lens and perspective correction of each view. For stitching frames from different cameras, we assume a fixed geometry which permits off-line calculation of frame transformations, thereby reducing the real-time processing load. To further reduce computational complexity, we employ lookup-tables in the view morphing and blending phases. The Vision P5 core combines the four frames with XGA resolution into a single XGA surround view at 30FPS.

ARTICLE HISTORY

Received 16 September 2020
Accepted 1 February 2021

Introduction

Advanced Driver Assistance Systems (ADASs) are one of the fastest-growing segments of the automotive market described in several studies like Butakov and Ioannou (2014); Ziebinski et al. (2017); Ziebinski et al. (2016); Mosalikanti, Bandi, and Kim (2019). ADASs improve the driver's control of the vehicle by providing real-time enhanced visualization and interaction with the surrounding environment. In ADASs project, the goal is to develop technologies that are able to enhance the awareness of the vehicle's driver. Such technologies concern the driver and vehicle safety and work together toward autonomous driving. Several services are offered in ADASs proposed technologies including lane departure monitoring, adaptive cruise control, speed limit monitoring, emergency brake assistance, surround view monitoring, and many other services mentioned in Figure 1.

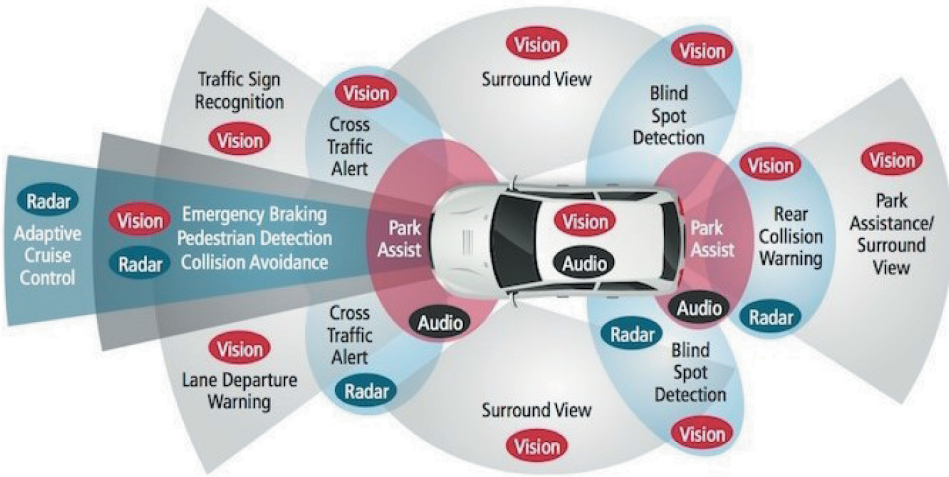


Figure 1. List of the available ADASs services. Adopted from Cadence Design Systems (2020).

Bird's-eye Surround View (SV) is a type of ADASs which integrates multiple lateral cameras to produce a virtual bird's-eye (i.e. top down) view in real time. While having a rearview camera, currently standard in many vehicles, can help a driver when moving in reverse or parking, adding cameras on all sides of the vehicle can dramatically improve awareness of surrounding objects, vehicles and pedestrians. Greater visibility of the area surrounding a vehicle improves the driving experience and overall safety.

SV is a sensor fusion problem whereby multiple sensors, in this case, cameras are combined and processed jointly to provide a complete picture of the environment around a vehicle and to help take fast decisions, either by a machine or human driver. Processing multiple sensors in real time can be a challenging computational problem, especially for high bandwidth applications involving video or radar signals according to Ananthanarayanan et al. (2017). Some of the challenges include interfacing with multiple high-speed signals, efficiently managing memories to buffer only the minimum amount of data required, and executing complex algorithmic operations at speed, all while maintaining low power consumption and cost. With this in mind, our work addresses the problem of efficient implementation of SV in an embedded DSP.

Our system combines fish-eye cameras located on each side of a vehicle, with overlapping views into a single top-down view using image stitching techniques. Prior to SV systems, image stitching was a well-defined problem with several approaches in the literature described in Kwatra et al. (2003); Szeliski (2006). In a simple stitching problem, we have two images where there is an overlap area between them. Since the overlap can provide two sources of pixels in the overlapped area, the question is how to find

a path or cut that determines the boundary between the two original images and which minimizes the surrounding differences. In the SV system, the problem has more dimensions than the typical stitching process. First, there are more than two frames and these frames are produced at real-time. Second, the generated frames have to be projected onto a planar surface to achieve a top-down view. Our algorithm first corrects the lens distortion in each camera and modifies the perspective from a downwards, diagonal view away from the car to a top-down view in a single step. Next, the corrected frames from each camera are stitched together into one frame. We architect a system that partitions system memory and local memory on the processor so as to minimize cost and to avoid processing cycles wasted in the movement of data. The system is based on a Tensilica Vision P5 core that has an instruction set specialized for image processing and computer vision applications. Furthermore, the firmware running on the core leverages the Tensilica XI Library, a set of image processing and computer vision software functions that are optimized for the Vision P5 core.

The paper is organized as follows: First, the related work is provided. Next, Lens correction and top down view projection are presented. Subsequently, the stitching process for the captured frames using Graph-Cut approach is explained, followed by the frame registration and blending approach. Next, Vision P5 DSP is reviewed. Then, results of experiments with a radio controlled (RC) car are presented and discussed. The concluding remarks reflect what this means for the discussion about the surround view system.

Related Work

Image Stitching

Several works like Brown and Lowe (2007); Kwatra et al. (2003); Levin et al. (2004); Szeliski (2006) propose different approaches to image stitching with varying complexity. The method in Kwatra et al. (2003) uses a graph-cut algorithm to stitch two images together. The cost of this approach is relatively high since preparing the adjacency matrix (i.e. the matrix that measures the pixels intensity differences cost between the images, so it could be used later for finding the optimal cut) is expensive. An alternative, simpler and less computationally complex approach uses blending. In alpha-blending approach, a pixel in the overlap region between adjacent cameras can be stitched and recovered seamlessly using a linear combination of weighted intensities for the same pixel location between these cameras. Other algorithms use the gradient domain like Levin et al. (2004). The approach focuses on minimizing the dissimilarity between the gradient of the input image, and the gradient of the stitched image.

Surround View System

SV has been studied and discussed through many works like Palazzi et al. (2017); Zhang et al. (2014); Kim et al. (2018). The work of Zhang et al. (2014) focuses on embedded implementation of the surround view system based on the Texas Instruments TDAx SoC DSP. Using four calibrated fish-eye lenses, the system is able to generate a top-down view of 360° surround view with resolution 880 × 1080 at a rate of 30 frames per second. By assuming fixed cameras that are used in the model, the system applies geometric alignment to correct the fish-eye lenses, followed by synthesis phase to generate one composite view from all cameras. Finally, photometric alignment is applied to assure a seamless composite view.

The work of Liu, Lin, and Chen (2008) focuses on the algorithmic aspects of the problem without any discussion on an embedded implementation. The work uses six fish-eye cameras mounted on the sides of the vehicle and generates a single composite view. Using a dynamic programming approach termed “dynamic image warping”, the model decides on the regions of the seam in the overlapped area. The work of Yu and Ma (2014) shows a typical approach to generate a composite SV for a parking guidance feature. Starting with fish-eye correction, the system uses the corrected and projected frames to stitch them using an alpha-blending approach.

Lens Correction and Top-Down View Projection

Our SV system employs fish-eye cameras to increase the field of view (i.e. 180°) allowing the system to capture the entire perimeter of the vehicle. The wide field of view yields areas of overlap between neighboring cameras. Fish-eye cameras suffer from lens distortion and require a correction process to convert the distorted frame into a corrected and projected one. To produce a top-down projected view from a fish-eye distorted frame, the typical method is to apply a traditional lens distortion algorithm like Gribbon, Johnston, and Bailey (2003) followed by frame direction adjustment on the planar surface. In this work, we use a morphing or view warping approach which removes lens distortion and applies view direction adjustment at the same time, as will be described next.

Morphing

The SV morphing process completes two tasks at once: (1) correct the distorted frame and (2) apply view direction adjustment. This is accomplished via a calibration phase for each camera using a square reference mesh as illustrated by Figure 2. Pictures of the mesh are taken with the fish-eye cameras as inputs (see Figure 2(a)) and with a top-down view, Nikon 5100 DSLR

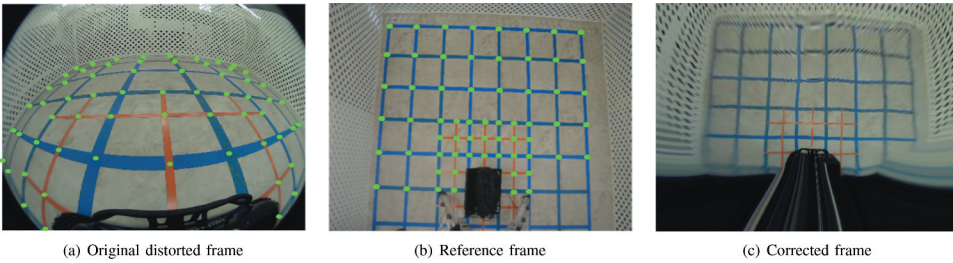


Figure 2. Lens distortion correction and new view direction estimation approach. We remove fish-eye distortion and adjust the camera view using a morphing approach.

projected camera as reference outputs (see [Figure 2\(b\)](#)). Control points on the mesh are manually selected to define an inverse map from the target output frames back to the distorted input frames. Alternatively, control points could be selected automatically by machine detection of the calibration grid. Since the model assumes a fixed geometry (i.e. the cameras are mounted on the vehicle sides and never change viewing angles), the morphing function is estimated off-line, only once during the calibration phase. The rest of the video frames during operation of the vehicle are morphed in the same way.

The morphing method we have used finds an inverse mapping $(u, v) = f(x, y)$ from the reference frame control points (x, y) back to the original distorted frame control points (u, v) using Thin-plate Splines (TPS) to parametrize the mapping f . TPS minimizes the energy required to bend the original frame into the corrected one as mentioned in [Bookstein \(1989\)](#)

$$\min_f \iint_{R^2} \left(\left(\frac{\partial^2 f}{\partial x^2} \right)^2 + 2 \left(\frac{\partial^2 f}{\partial x \partial y} \right)^2 + \left(\frac{\partial^2 f}{\partial y^2} \right)^2 \right) dx dy \quad (1)$$

For more details about the approach see [Bookstein \(1989\)](#).

Embedded Implementation

Once the parameters of the TPS are calculated from the control points, the TPS map is used to generate a table of input frame pixels corresponding to each output frame pixel. Each entry in the table is an integer vertical and horizontal input frame pixel coordinates plus fractional vertical and horizontal offsets from the integer positions. This representation facilitates bilinear interpolation to generate the output. The creation of the lookup tables is done off-line and the resulting tables are stored in the system memory. Each camera has its own morphing lookup-table.

The embedded implementation treats each input frame and corresponding lookup table as a set of dynamically sized tiles (see [Figure 3\(a\)](#)). The input frames and lookup tables are stored in the system memory. The DSP generates output frames one tile at a time according to the regular grid in [Figure 3\(b\)](#).

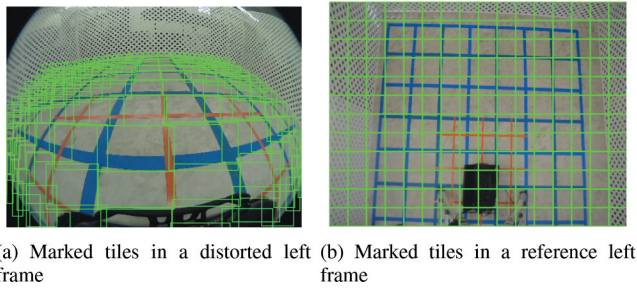


Figure 3. Map of tiles processed by embedded DSP.

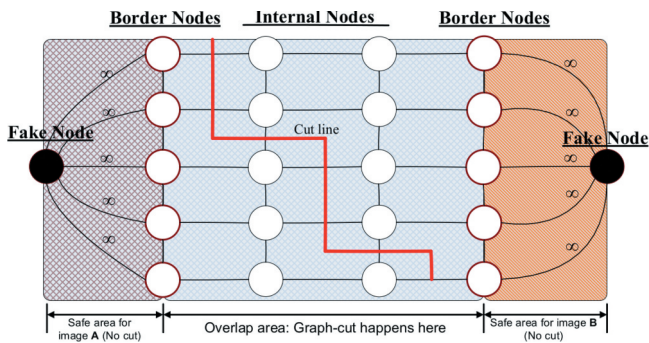


Figure 4. Stitching process requires finding the optimal seam in the overlapping area. Graph-Cut is used to map the original images into a graph, the finds the optimal seam in the overlapping area which is marked by the red line.

Each tile in the distorted frame is loaded from the system memory into the DSPs local memory, along with the lookup table for the mapping, and is morphed into an output tile. For this reason, an additional tiling table that contains the address and dimensions of each input tile is used to load them precisely into the local memory. The size of the tile depends on its location in the distorted frame, as can be seen in [Figure 4](#).

Images Stitching with Graph-Cut Approach

Graph-Cut approach is an optimization methodology that deals with graph problems. In the context of images stitching, the Graph-Cut algorithm is intended to find the optimal seam that combines two images smoothly. Precisely, the combination process focuses on the overlapping region between both images. In these overlapping areas, the Graph-Cut algorithm finds the optimal seam. Such seam minimizes the color differences between both images as much as possible.

First, let us introduce the basic terminology of the images stitching problem. Assume there is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where \mathcal{V} is a set of the graph nodes, and \mathcal{E}

is a set of edges in the graph. Each node v in the graph \mathcal{G} represents an image pixel, and each edge e corresponds to a neighboring pixel in an image. The nodes in the graph \mathcal{G} are classified into two main categories: (1) Border nodes, which represent the safe area that entirely belongs to one of the images. (2) Internal nodes, which represent the candidate area for the cut process (i.e. overlapped area). For border nodes, fake nodes are added to connect all border nodes with costs on edges equal to ∞ . This high cost prevents the Graph-Cut algorithm to exceed the border nodes in the seam selection process.

Frames Registration and Blending

Frames Registration

After applying the morphing process, the view corrected frames for each camera are combined to generate a single composite frame. Our benefits are obtained from the overlapped areas between neighboring cameras to produce a new registered frame. The approach uses affine transformations to register frames together. We apply the registration process using affine transformations. In the affine transformation, we assume that there is an overlap area between moving and fixed frames which needs to be registered together. Since the used cameras are assumed to be fixed and never changed, a set of control points are selected manually and precisely (i.e. other approaches might be used to select them automatically like Sift algorithm; however, manual selection makes it more accurate). These control points represent the corresponding locations between moving and fixed frames. Based on these control points, the geometrical transformation (i.e., estimated transformation) fits the control points in the moving frame to the control points in the fixed frame. Such geometrical transformation minimizes the error between the corresponding control points as much as possible. The next step, is to apply the geometrical transformation on the moving frame to align it with the fixed frame.

The registration process is applied at three different levels. First, we register the front frame with the left frame (i.e. we call it level 1, see [Figure 5\(a, d\)](#)). Second, the result from level 1 is registered with the back frame (i.e. we call it level 2, see [Figure 5\(b, e\)](#)). Finally, the result from level 2 is registered with the right frame (i.e. we call it level 3, see [Figure 5\(c, f\)](#)). These levels of registrations produce the surrounding view around the vehicle; however, the overlapping areas have not yet been processed to determine the source of pixels in these areas. For efficient processing, lookup-tables were used to maintain the affine transformation for each level. The look-up tables allow the frames to be registered directly without selecting control points and finding the geometrical transformation for each upcoming frame.

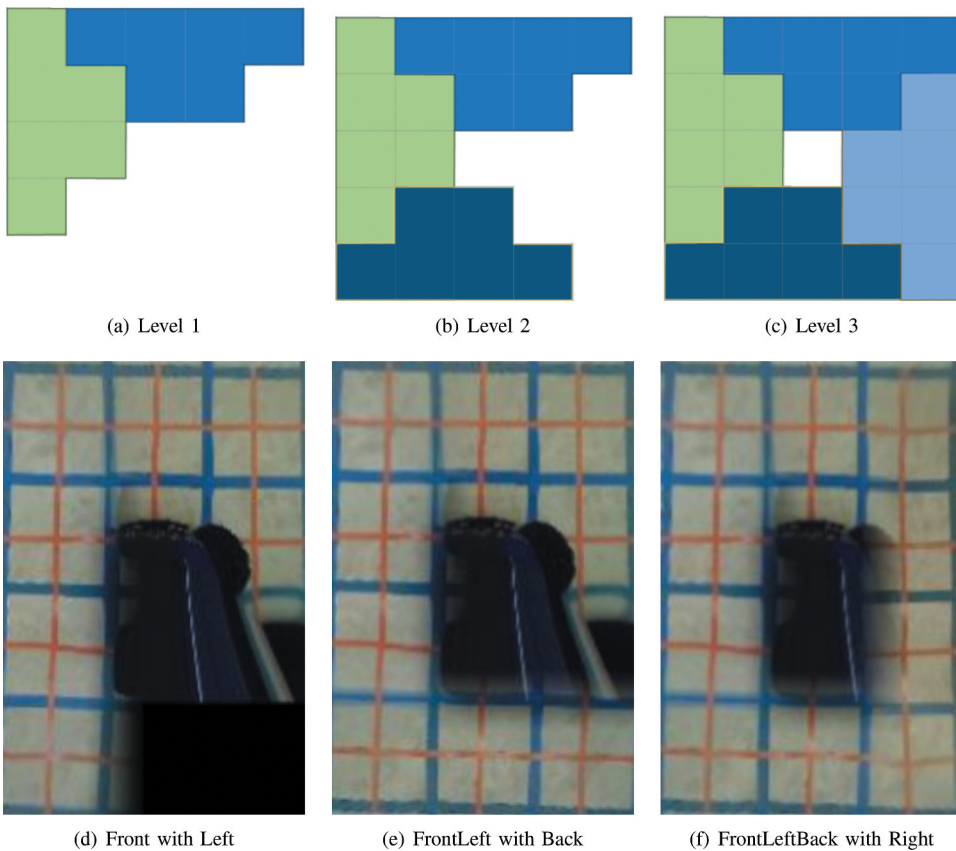


Figure 5. The bird's eye view algorithm implementation process. The upper row shows the abstract explanation of the algorithm, while the lower row shows an actual case (i.e., how the actual view is constructed).

Frames Blending

The last stage in the frame stitching process is to manage the overlapping areas properly. The overlapping areas are a special consideration since the pixels in these areas are related to two sources of frames, and there should be some procedure which takes care of pixels in these areas. In blending, we try to reorganize the pixels in the overlap area such that the resulted frame should look consistent and seamless. To blend the two frames together, assume that there are a left frame \mathcal{L} and right frame \mathcal{R} . These frames were captured such that there is an overlap area between them. These frames are also registered to the same coordinate system using the affine transform, and both have the same size in the registered frames. The pixels in the non-overlapping area have no problem since they are related to one source of data. For pixels in the overlap area, we use a geometrical seam selection such that it minimizes the geometrical differences between frame \mathcal{L} and frame \mathcal{R} (i.e. the spatial differences in

the constructed seam after applying the affine transform). The seam selection is constructed using the Graph-Cut algorithm as mentioned earlier.

The next step is to smooth the transition from the left side of the constructed seam to the right side of the constructed seam. To apply the transition, we build weighted masks, mask \mathcal{M}_l and mask \mathcal{M}_r which control the weighted transition around the seam (mask \mathcal{M}_r is $1 - \mathcal{M}_l$). The masks initially started as a binary masks where \mathcal{M}_l equals to ones in the region of non-overlap area and zeros otherwise. To make the transition smooth we reconstruct a Gaussian kernel, which applies low pass filter and apply this filter on the whole mask \mathcal{M}_l . At this point the mask \mathcal{M}_l is kept as a lookup table and applied as a smooth transition for the frame \mathcal{L} . The other mask \mathcal{M}_r is simply $1 - \mathcal{L}$. We use alpha blending approach to smooth this transition such that the blended frame \mathcal{B} equals to:

$$\mathcal{B}(x, y) = \begin{cases} \mathcal{L}(x, y), & \text{if } (x, y) \in \mathcal{L} \text{ } (x, y) \notin \mathcal{R} \\ \mathcal{R}(x, y), & \text{if } (x, y) \in \mathcal{R} \text{ } (x, y) \notin \mathcal{L} \\ [\mathcal{L}(x, y) * \mathcal{M}(x, y) +, & \text{if } (x, y) \in \mathcal{L} \text{ } (x, y) \notin \mathcal{R} \\ \mathcal{R}(x, y) * \mathcal{M}(x, y)] & \end{cases} \quad (2)$$

Figure 6 shows the blending process, where the frames are linked with their related masks. For efficient processing, the blending masks were kept as a lookup-table for each level of the stitching process (i.e. level 1, level 2, and level 3).

Algorithm 1 Summarizes the main stages in the bird’s eye surround view system. At the beginning, all required lookup tables and the transformation parameters are estimated and kept in order to use them for the incoming frames. Since the geometry is fixed, these lookup tables and transformation parameters are fixed and never changed.

Algorithm 1 Bird’s Eye Surround View Algorithm

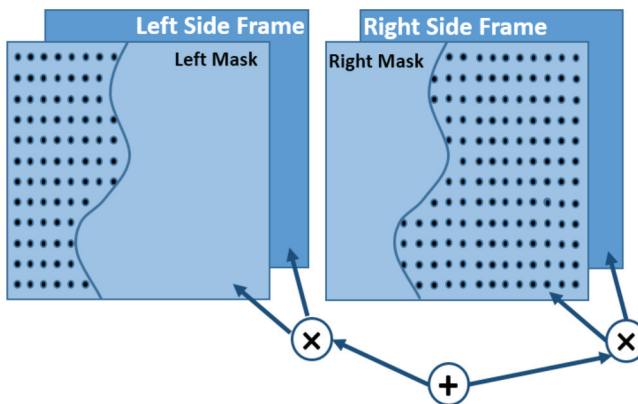


Figure 6. Blending process. The curved line in the left and right masks represents the selected seam that minimizes the geometrical differences along the cut.

- 1: **Input:** Fishy eye distorted frames.
Lookup tables.
Fixed geometry parameters.
- 2: **Output:** Top-down view full frame.
- 3: corrected_frames = **tps**(distorted_frames)
- 4: registered_frames = **affine_trans**(corrected_frames).
- 5: seam_selection = **graph_cut**(registered_frames).
- 6: level_one = **blend** (corrected_front + front_lookup, corrected_left + left_lookup).
- 7: level_two = **blend** (corrected_back + back_lookup, level_one + level_one_lookup).
- 8: level_three = **blend** (corrected_right + right_lookup, level_two + level_two_lookup).

Figure 7 shows how the proposed approach works. We performed an experiment where we used a toy car and we added four fish-eye cameras on the sides. The algorithm was run on a laptop mounted on top of a used toy car. In the experiment area, we drew several colored lines as landmarks, so the car will drive based on these lines. The upper row of Figure 7(a-c) shows the original car at different positions in the experimental area. The lower row of

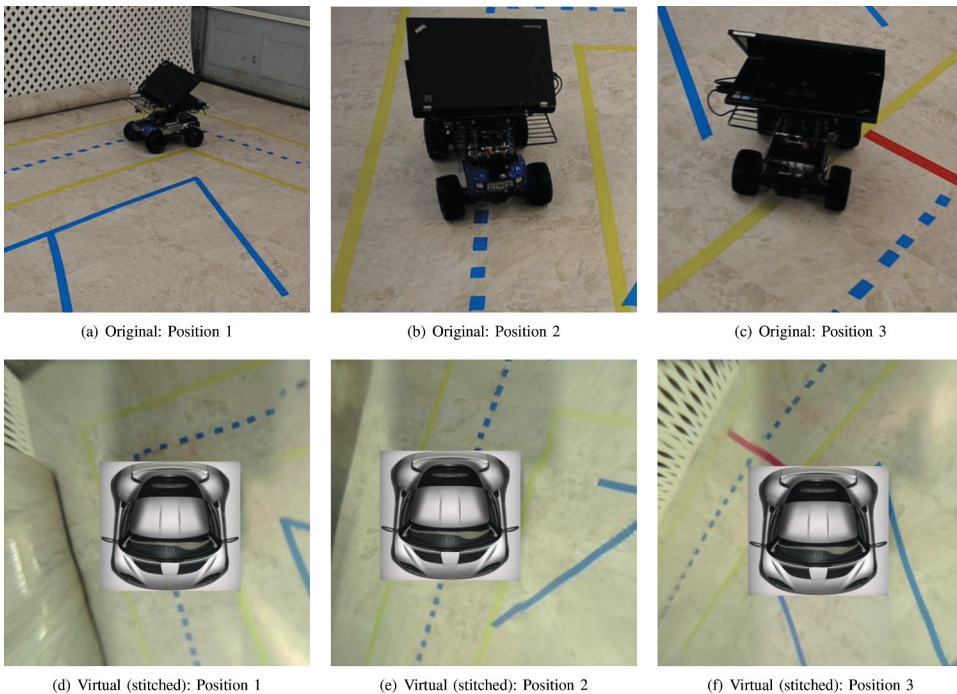


Figure 7. Several shots at different locations for the bird's eye view system. The upper photos are related to the original settings where there are four side cameras mounted on the car, and the used algorithm is running on the used laptop. The lower photos are related the virtual generated top-down view.

Figure 7(d–f) shows the generated stitched bird's-eye view for each image in the upper row.

Embedded Extension Using Vision P5 DSP

The Tensilica Vision P5 (VP5) is a high performance embedded DSP platform which provides N-way SIMD vector engine as described in Vision P5 DSP (2019). The use of instruction set with the supported memory system in the platform allows parallel processing which makes it efficient for many real-time applications like Vision P5 memory details. (2019). The used vector SIMD provides three levels of operations including 64-way 8 bits, 32-way 16 bits, and 16-way 32 bits. In addition to the standard arithmetic and logical operations, VP5 supports rearranging data processing across data lanes by using data selection strategy. For more efficient vision and imaging-based operations, VP5 provides shuffling, shifting, interleaving patterns and data normalization.

The used architecture in VP5 is shown in Figure 8. Such architecture allows processing hundreds of pixel operations per cycle. The optimization used at the algorithmic level makes it efficient at energy consumption level, which makes it support both a high performance computing with efficient energy consumption as described in Vision P5 DSP (2019). Tensilica instruction set (TIE) has been integrated with the VP5 DSP, so a more customizable instructions and operations for different kinds of applications as mentioned in Acevedo et al. (2018).

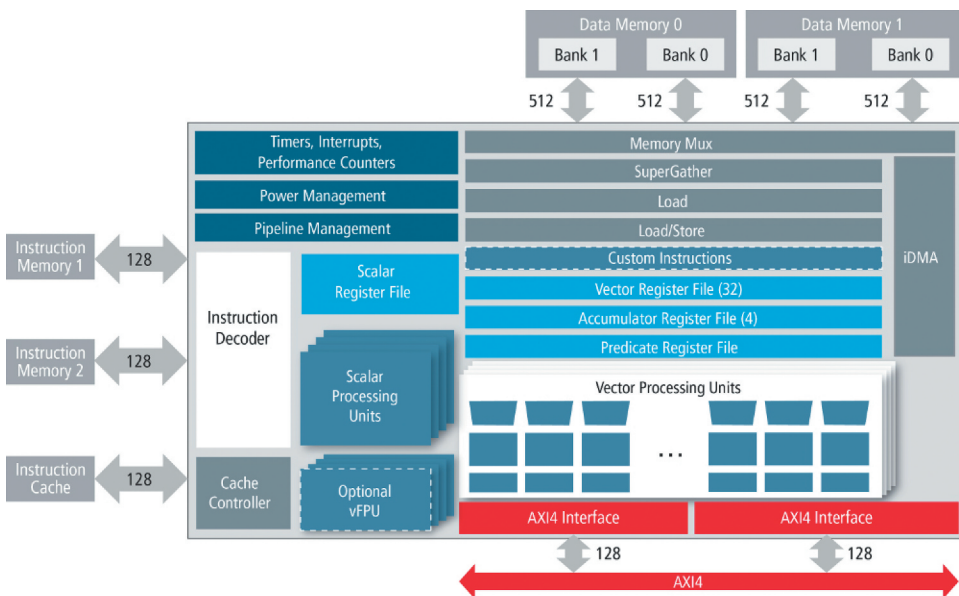


Figure 8. Vision P5 subsystem and core architecture. Adopted from Vision P5 DSP (2019).

Complexity Analysis

Experiment

The generated bird's-eye view passes through several stages. At the beginning, the original fishy-eye distorted frames are downsampled and interpolated into a smaller size frames. Figure 9 explains the process characteristics.

The downsampling process requires $(num_of_frames \times frame_height \times frame_width) / down_sampling_ratio \times frame_channels \times frame_rate / 10^6$ operations [Mops/sec]. The total number of operations in the downsampling process is $((4 \times 768 \times 1024) / 2 \times 3 \times 30 / 10^6)$.

The morphing (interpolation) process is required to correct the frames into projected ones. The distorted frames are transformed into corrected ones (i.e., projected), so the fishy-eye frames are converted with the help of interpolation process into corrected and projected frames. Figure 9 shows the interpolation process settings. During this process, scattered data (i.e., load/store operations) are read and then interpolation is applied. For reading data, the required number of operations is operations [Mops/sec]. Applying the used parameters will result in $(4 \times 348 \times 512 \times 3 \times 2 \times 2 \times 30 / 10^6)$ [Mops/sec]. The same amount of operations is required to apply the interpolation process (i.e., multiply accumulate).

At this point, the generated frames are corrected. However, each frame has its own coordinate system. The next step is to align all frames to one global coordinate system. In the affine transform phase (Figure 10), there will be a mapping for each input frame to a one unified global coordinate system, which makes each frame aware of its location according to the global

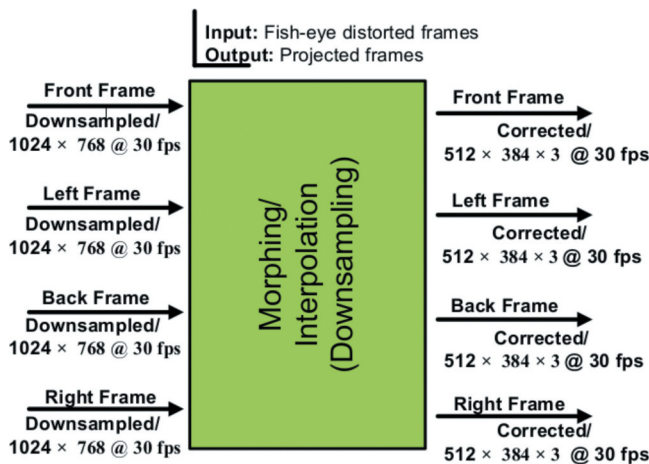


Figure 9. Frames correction process. Fishy-eye frames are interpolated into projected ones.

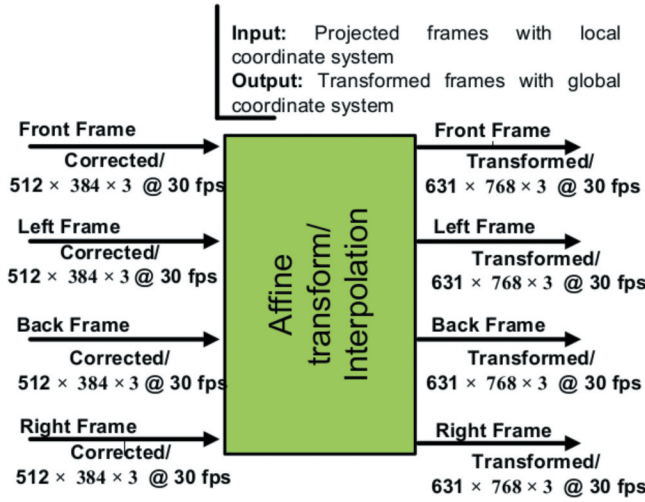


Figure 10. Applying affine transform, to make one global coordinate system for all frames.

coordinate system. Based on this affine transform, each pixel in the input images with local coordinate system has a specific and determined location in the new global coordinate system. One of the frames is considered as a reference frame, and the rest frames use affine transformation (i.e., each one alone) to align them to the new global coordinate system.

The required number of operations for the affine transformation phase is $(num_frames \times morphed_frame_height \times morphed_frame_width \times affine_transform_row \times affine_transform_col \times frame_rate / 10^6)$ operations [Mops/sec]. Applying the used parameters will produce $(3 \times 348 \times 512 \times 3 \times 3 \times 30 / 10^6)$ [Mops/Sec]. Next, interpolation is required to be performed on the new aligned frames. In the interpolation process, there are [Mops/sec] for reading scattered data (i.e., load/store operations) which are equal to $(4 \times 631 \times 768 \times 3 \times 30 \times 3 / 10^6)$ [Mops/sec]. The same amount of operations is required for applying the actual interpolation (i.e., multiply accumulation).

The blending phase (see Figure 11) uses a predetermined masks (i.e., estimated using the Graph-Cut algorithm and processed with some low pass filter), and these masks are fixed for all incoming frames. These masks determine the origin of each pixel in the overlapping areas between neighboring frames. The result of the blending phase is one full frame describing the whole surrounding view. To estimate the required operations during the blending process, the blending process is executed at three different levels. First, the blending is applied between front and left frame and consumes $(2 \times blended_frame_height \times blended_frame_width \times frame_channels \times B_num_mask_filters \times frame_rate / 10^6)$ [Mops/sec] which

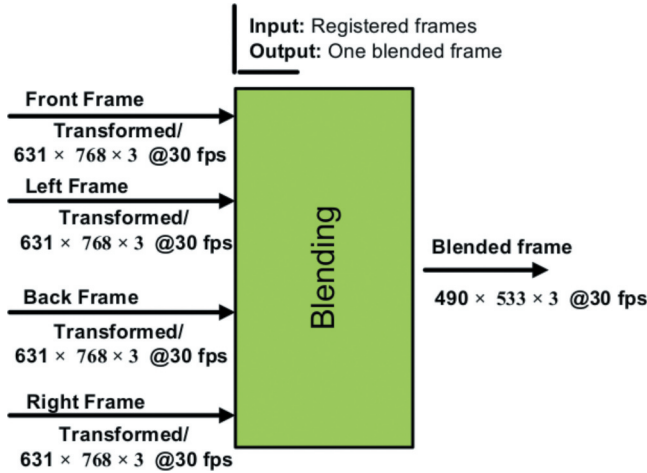


Figure 11. Blending phase, where input frames are transformed into one output frame.

equal to $(2 \times 533 \times 490 \times 3 \times 2 \times 30/10^6)$ [Mops/sec]. Second, the resulted blended (frontLeft) frame is processed again in the same manner with the back frame. The number of required operations is estimated in the same manner as (frontLeft); however, the $(blended_frame_height \times blended_frame_width)$ should be updated accordingly. Finally, the last level in the blending process is applied between the blended (frontLeftBack) frame and the right frame in same way as mentioned earlier. It is important to mention that the dimensions of the predetermined masks are fixed and never change since we are dealing with a fixed geometry.

The last phase of the approach is resizing (i.e., applying interpolation to upsample the blended frame). In this phase, we try to resize the blended frame to the target size (Figure 12). The resizing executes a rescaled locations and bilinear interpolation processes. The rescaling process requires

$$(blended_frame_height \times blended_frame_width \times affine_transform_row \times affine_transform_col/10^6)$$

which equals to $(533 \times 490 \times 2 \times 2/10^6)$ [Mops/sec]. The interpolation process consumes which equals to $(2 \times 1920 \times 1080 \times 3 \times 30 \times 2/10^6)$ [Mops/sec].

Table 1 summarizes the main operations required for building the surround view system, and the required cycles to perform each operation.

Conclusion

Throughout this work, we present an approach to for building a virtual bird's eye surround view system, which is a core unit in the future of car driving. The

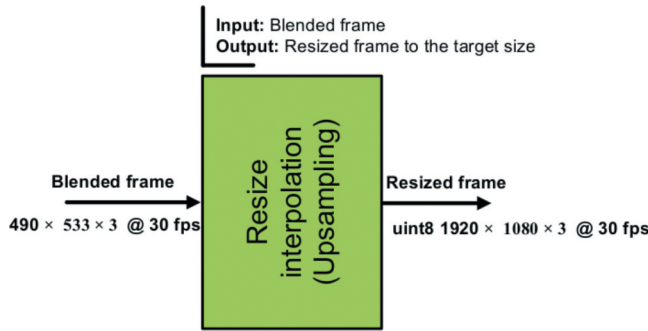


Figure 12. The blended frame is resized and upsampled into a target size through using interpolation process.

Table 1. Cycles analysis for the SV system. For the morphing phase, and each level in the stitching process cycles shows the memory used cycles, and the core’s ones.

		Input			
	Details	Library Function	Performance ops/ cycle	Cycles core Mops/ sec	Performance MHz
Downsampling Morphing	Downsampling	load/store	64	141.55776	2.21184
	Read scattered data	Gather	64	283.11552	4.42368
Affine Transform	Interpolation transform	mula	64	283.11552	4.42368
	Read scattered data	mula	64	1610.3318	25.161435
		Gather	64	715.70304	11.18286
Blending	Interpolation	mula	64	715.70304	11.18286
	Level 1	mula	64	94.0212	1.46908125
	Level 2	mula	64	94.0212	1.46908125
	Level 3	mula	64	94.0212	1.46908125
Resize/ Interpolation	Rescaled		64	17.892576	0.2795715
	Interpolation locations	mula	64	715.70304	11.18286

surround view system is part of ADAS project. To make the generated view real-time view, we used the Tensilica vision P5 DSP. The used DSP allows us to generate the virtual view at rate 30 frames per second. In the surrounding view system, the mounted cameras are fixed and never move, so the used geometry is fixed for all incoming frames. The benefit of fixed geometry appears clearly in the blending process, where the new location of each pixel in the input frame is estimated early and never changes. The Graph-Cut algorithm helps in the seam selection in the overlapping areas between neighboring frames, and this makes the cut lines seamless and smooth because the color differences are optimized to be minimal.

ORCID

Subhieh El Salhi  <http://orcid.org/0000-0002-9700-4862>

References

- Acevedo, J., R. Scheffel, S. Wunderlich, M. Hasler, S. Pandi, J. Cabrera, F. Fitzek, G. Fettweis, and M. Reisslein. 2018. Hardware acceleration for RLNC: A case study based on the xtensa processor with the tensilica instruction-set extension. *Electronics* 7:180. Multidisciplinary Digital Publishing Institute. doi:10.3390/electronics7090180.
- Ananthanarayanan, G., P. Bahl, P. Bodk, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. 2017. Real-time video analytics: The killer app for edge computing. *computer* 50:58–67. IEEE. doi:10.1109/MC.2017.3641638.
- Bookstein, F. L. 1989. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 6:567–85. doi:10.1109/34.24792.
- Brown, M., and D. G. Lowe. 2007. Automatic panoramic image stitching using invariant features. *International Journal of computer Vision* 74:59–73. Springer. doi:10.1007/s11263-006-0002-3.
- Butakov, V. A., and P. Ioannou. 2014. Personalized driver/vehicle lane change models for ADAS. *IEEE Transactions on Vehicular Technology* 64:4422–31. doi:10.1109/TVT.2014.2369522.
- Cadence Design Systems. 2020. ADASs available services. Accessed May 28, 2020. <https://www.allaboutcircuits.com/industry-articles/developing-smarter-safer-cars-adas-automotive-advanced-driver-assistance-ip/>
- Gibbon, K. T., C. T. Johnston, and D. G. Bailey. 2003. A real-time FPGA implementation of a barrel distortion correction algorithm with bilinear interpolation. *Image and Vision Computing New Zealand* 408–13.
- Kim, J. H., S. K. Kim, T. M. Lee, Y. J. Lim, and J. Lim. 2018. Hemispherical 3D around view monitoring algorithm using image synthesis of multi-channel cameras. *15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 1466–71, Singapore: IEEE.
- Kwatra, V., A. Schödl, I. Essa, G. Turk, and A. Bobick. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics (Tog)* 7:277–86. doi:10.1145/882262.882264.
- Levin, A., A. Zomet, S. Peleg, and Y. Weiss. 2004. Seamless image stitching in the gradient domain. *Computer Vision-ECCV Vol 3024*, 377–89. Springer.
- Liu, Y. C., K. Y. Lin, and Y. S. Chen. 2008. Birds-eye view vision system for vehicle surrounding monitoring. *Robot Vision Lecture Notes in Computer Science*, vol 4931, 207–18. Springer.
- Mosalikanti, A., P. Bandi, and S. H. Kim. 2019. Evaluation of different ADAS features in vehicle displays. *SAE Technical Paper*.
- Palazzi, A., G. Borghi, D. Abati, S. Calderara, and R. Cucchiara. 2017. Learning to map vehicles into birds eye view. *International Conference on Image Analysis and Processing*, 233–43, Catania, Italy: Springer.
- Szeliski, R. 2006. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision* 2:1–104. Now Publishers Inc. doi:10.1561/0600000009.
- Vision P5 DSP. 2019. Vision P5 DSP architecture and features. San Francisco, CA. Accessed May 18, 2020. <https://www.prnewswire.com/news-releases/new-cadence-tensilica-vision-p5-dsp-enables-4k-mobile-imaging-with-13x-performance-boost-and-5x-lower-energy-300154540.html>
- Yu, M., and G. Ma. 2014. 360 surround view system with parking guidance. *SAE International Journal of Commercial Vehicles* 7:19–24. doi:10.4271/2014-01-0157.

- Zhang, B., V. Appia, I. Pekkucuksen, A. U. Batur, P. Shastry, S. Liu, S. Sivasankaran, K. Chitnis, and Y. Liu. 2014. A surround view camera solution for embedded systems. *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Columbus, OH, USA, 676–81.
- Ziebinski, A., R. Cupek, D. Grzechca, and L. Chruszczyk. 2017. Review of advanced driver assistance systems (ADAS). *AIP Conference Proceedings* Vol. 1906, No. 1, p. 120002. AIP Publishing.
- Ziebinski, A., R. Cupek, H. Erdogan, and S. Waechter. 2016. A survey of ADAS technologies for the future perspective of sensor fusion. *International Conference on Computational Collective Intelligence*, 135–46, Halkidiki, Greece: Springer.