PredStereo: An Accurate Real-time Stereo Vision System

Diksha Moolchandani, Nivedita Shrivastava, Anshul Kumar, and Smruti R. Sarangi
Indian Institute of Technology Delhi
New Delhi, India
{diksha.moolchandani, anshul, srsarangi}@cse.iitd.ac.in, nivedita.shrivastava@ee.iitd.ac.in

Abstract

Stereo vision algorithms are important building blocks of self-driving applications. The two primary requirements of a self-driving vehicle are real-time operation and nearly 100% accuracy in constructing the 3D scene regardless of the weather conditions and the degree of ambient light. Sadly, most real-time systems as of today provide a level of accuracy that is inadequate and this endangers the life of the passengers; consequently, it is necessary to supplement such systems with expensive LiDAR-based sensors. We observe that for a given scene, different stereo matching algorithms can have vastly different accuracies, and among these algorithms there is no clear winner. This makes the case for a hybrid stereo vision system where the best stereo vision algorithm for a stereo image pair is chosen by a predictor dynamically, in real-time.

We implement such a system called PredStereo *in ASIC*[1] *that combines two diametrically different stereo vision algorithms, CNN-based and traditional, and chooses the best one at runtime. In addition, it associates a confidence with the chosen algorithm, such that the higher-level control system can be switched on in case of a low confidence value. We show that designing a predictor that is explainable and a system that respects soft real-time constraints is non-trivial. Hence, we propose a variety of hardware optimizations that enable our system to work in real-time. Overall,* PredStereo *improves the disparity estimation error over a state-of-the-art CNN-based stereo vision system by up to 18% (on average 6.25%) with a negligible area overhead ($0.003mm^2$) while respecting real-time constraints.*

1. Introduction

Depth estimation from stereo images is one of the most important building blocks of autonomous-driving applications such as object tracking, localization, and navigation. Sadly, such depth estimation techniques that are vital for ensuring the safety of autonomous vehicles, work quite poorly in inclement weather conditions [30]. As a result, it is necessary to supplement the image-based stereo vision with other expensive sensing devices such as lidars, mmWave radars, and ultrasonic sensors [34]. We argue in this paper that the accuracy of inexpensive stereo imaging techniques can be greatly enhanced using our novel ASIC-based technique, *PredStereo*, while respecting the real-time constraints of the autonomous vehicles. *PredStereo* additionally provides a level of confidence – if this value is low, other expensive sensing techniques need to be used.

Let us first appreciate the severity of the problem. The crux of depth estimation is to capture two photographs using two different cameras, identify the similar pixels, and based on the displacement of their relative positions (*disparity*), compute the depth of the object. Even if there is a small 1-pixel error in this process, the error in estimating the depth can be 6.4 m for an object that is actually 50 m away (considering the same camera configuration as used by the KITTI dataset [20]). For a car going at 70 km/hr, the safety buffer time [18] (typically 1.8-2 s) will reduce by 320 ms (see Section 6.5). Hence, there is a need for improving the accuracy of such stereo vision techniques while respecting the real-time requirements of these vehicles.

It would be presumptuous to claim that better stereo vision techniques can completely supplant expensive lidars and mmWave radars; the latter are currently being used by successful self-driving solutions such as Amazon Zoox [32] and Yandex [31]. On similar lines, it would also be impractical to claim that it is possible to train a deep neural network (DNN) that can take care of all possible weather and ambient lighting scenarios. Consider the case of Yandex in Russia [31], where researchers painstakingly collected images from 10 million km of driving data to just take snow into account in LiDAR point clouds – it is hard to find the wherewithal to do this for all kinds of scenarios. Thus, it is likely that the future will by and large continue the current trend: have a large number of stereo cameras as the first-line mechanism, and a smaller array of expensive lidars or radars as the second-line solution. If the first-line

[1] ASIC → Application-Specific Integrated Circuit

solution is accurate, then we can reduce the size and the cost of the second-line solution. At an extreme end of this spectrum is Tesla's [25] proposed solution (based on public statements [34]) to move towards a LiDAR-less stereo vision system. Our proposal is complementary to either of the two scenarios. Our focus is *to design an efficient solution for enhanced accuracy of the stereo vision system subject to real-time performance constraints*. We evaluate our scheme for two hardware configurations: one based on ASICs (synthesized) and the other based on GPUs.

Given our motivation, we began with selecting the most accurate CNN-based and computational geometry-based algorithms and evaluated them on the KITTI-2015 stereo dataset [20]. The conclusions are quite revealing. For roughly half the images, the CNN-based algorithm Highres [40] is better than the most accurate traditional algorithm, SGM (Semi-global matching) [11]; the reverse is true for roughly the other half. The absolute difference of the 3-pixel error (defined in Section 3) of Highres and SGM was quite small (up to 9%). However, the moment we considered augmented images (with snow, rain, and fog), the differences between the two algorithms in terms of the maximum absolute error (3-pixel error) increased to 95%. Clearly, for some images Highres was much better, and for others SGM was markedly better. We need to bear in mind that this was observed in spite of the fact that Highres was trained with several data augmentations such as varying the image color, brightness, and intensity (see supplementary document for more details).

The results motivated us to search for hybrid solutions that combine multiple algorithms and respect real-time constraints. As of today, this is a relatively sparse area. The prominent contributions are by Feng et al. [7] and Spyropoulos et al. [29]. Feng et al. propose to find similarities across consecutive frames using traditional *motion compensation* algorithms (originally used in video compression). However, such approaches require a high frame capture rate for good results. Our empirical studies indicate that with 10 fps (typical frame capture rate for autonomous driving [16, 40]), the resulting accuracy is not at par with the best stereo-matching algorithms. The recent work by Spyropoulos et al. [29] is the most related. Here, the authors propose to choose the most accurate traditional stereo vision algorithm from a pool of algorithms *for each pixel* using a random forest-based predictor. The computational complexity and power dissipation of such pixel-wise decision making is prohibitive for real-time applications. *PredStereo* improves upon this algorithm by making real-time decisions in hardware (ASIC) at the level of full images and also providing a confidence associated with the prediction. We also consider CNNs, whereas [29] does not.

With exhaustive experiments, we show that only two features namely the perceived brightness and the percentage of dark pixels are sufficient to predict which algorithm is better and also yield a number that is indicative of the confidence of the prediction. Efficiently designing a circuit in hardware to rapidly compute these features in real-time was challenging; the issues were solved using a host of arithmetic optimizations and circuit techniques.

1.1. Contributions

❶ We show a detailed characterization of the performance, energy, and accuracy of different stereo vision algorithms on different hardware platforms.

❷ We use the insights from the characterization to design a low-overhead real-time architecture, *PredStereo*, that uses a hardware predictor (area: $0.003mm^2$, accuracy: 82.5%, 96.6% accuracy for scenes with large differences in the 3-pixel error) to decide the most accurate algorithm in real-time (10 ms) using a host of small yet tricky optimizations.

❸ A detailed comparison of our system with competing systems, where we achieve an increase in the disparity estimation accuracy by up to 18% (6.25% on average) over hitherto the most accurate CNN-based stereo vision system [40] and 45% (average) over the latest real-time stereo vision system, ASV [7].

❹ We underscore the importance of the error in disparity estimation by showing its effect on the depth estimation error and the subsequent reduction in the safety buffer of a self-driving vehicle.

2. Background and Related Work

2.1. Depth Estimation

Any 3D point in a real-world scene is projected to a 2D point in an image. Stereo vision involves capturing two images (left and right) simultaneously by two different cameras placed slightly apart. This is similar to human binocular vision where the two cameras are the two eyes. To find the distance of this 3D point from the observer (depth), we need to calculate the distance between the projected points ((x_1, y_1) and (x_2, y_2)) of this 3D point across the two images. This distance between the projected points is called *disparity*. Based on the triangulation principle [27], the depth Z is calculated as $Z = \frac{b \times f}{d}$. Here, b is the distance between the two cameras, f is the focal length of the two cameras, and d is the disparity. If the two camera lenses are parallel, then the disparity is equal to $x_2 - x_1$. As can be observed, b and f are fixed for a camera, hence the problem of calculating the depth becomes equivalent to calculating the disparity. Calculating the disparity for the entire image results in a disparity map.

2.2. Related Work

Feng et al. [7] proposed a hybrid stereo vision system by using a combination of CNN-based stereo algorithms

Model	MACs (10^9)	Dataflow Accelerator (scaled to 16nm [13])						GPU Tesla P100		Error(%)	
		RS		OS		WS					
		Time (s)	Energy (J)	Time (s)	Energy (J)	Time (s)	Energy (J)	Time (s)	Energy (J)	KITTI train	KITTI test
PSMNet	980	2.10	0.70	3.22	1.46	3.22	1.15	0.47	47	0.8	2.32
DeepPruner	775	1.74	0.72	2.68	1.15	2.68	0.92	0.052	5.72	1.1	2.15
Highres	55	**0.09**	**0.048**	0.19	0.089	0.19	0.070	**0.039**	**2.19**	2.6	2.14

Table 1. Execution time per frame, energy consumption, and error for CNN-based stereo algorithms

and motion compensation algorithm to reduce the computational overheads of the CNNs. The idea is to use a CNN algorithm for a frame to calculate the disparity and propagate this disparity to the n consecutive frames as a hint. This hint is used by a motion compensation algorithm to generate the disparity for the n consecutive frames. Partitioning the frames between the CNN accelerator and the motion compensation hardware is an important problem in itself. Feng et al. use static partitioning, i.e., $n = 2$ in their scheme. Another work by Spyropoulos et al. [29] proposes to choose the most accurate traditional stereo vision algorithm for each pixel of the image. They use a random forest (RF) based predictor to choose an algorithm out of a pool of algorithms. Each decision tree (DT) in the random forest provides a prediction and a score; the algorithm chosen by the DT with the highest score is the winner. This method is computationally very expensive and cannot be deployed for latency-sensitive applications. In comparison, *PredStereo* works at the level of full images and considers CNN-based stereo vision algorithms too.

3. Characterization of Stereo Vision Workloads

3.1. CNN-based Workloads

The aim of this characterization is to choose the eligible algorithms from a pool of the most accurate CNN-based stereo algorithms for operation in real-time. We run all the popular open source CNN-based stereo algorithms on an Nvidia Tesla P100 GPU (frequency: 1190-1329 MHz) and choose the top-3 most accurate algorithms. We evaluated the algorithms on the KITTI-2015 stereo dataset.

We then performed a characterization of the top-3 highly accurate CNN-based stereo vision algorithms: PSMNet, DeepPruner, and Highres on the accelerator simulator, *Timeloop*. Timeloop provides an optimal mapping of a CNN model on to the underlying CNN accelerator architecture. It also provides performance, energy, and area numbers. It uses the Accelergy-v0.3 [38] framework with the Aladdin [28] and the Cacti7 [1] plugins to estimate the energy consumption of the various components of the accelerator. We model three popular dataflows [22] for the CNN accelerators: Row Stationary (RS), Output Stationary (OS), and Weight Stationary (WS). All the dataflows are made to occupy the same area (2 mm^2 with 16 nm technology). The PE array is a 24×24 systolic array clocked at 1.2 GHz (similar to prior works [7,41]). The buffer sizes are adjusted to ensure that the total area for all the designs is the same (similar to prior work [22, 23]).

Table 1 (column 2) shows the number of MACs (multiply-accumulate operations) required by the three algorithms. We observe that *Highres* requires the least number of MACs and thus is the fastest on both the GPU and the accelerator. The largest number of MACs is required by PSMNet, and therefore it takes the maximum time to execute. Another observation is that for all the CNN models, the time taken to execute on the RS architecture is less than that taken to execute on the WS and OS architectures. Since the number of MACs of a CNN model remains constant across the dataflows, the difference in the execution time comes primarily from the PE utilization (60-80% for WS and OS and 80-100% for RS), which in turn is dependent on the scheduling of operations, memory bandwidth and NoC throughput. We also observed that RS is the most energy-efficient dataflow for any CNN model mainly due to fewer DRAM accesses. We observe that GPUs are the most energy-inefficient platform as compared to the ASICs.

Note that the execution times of the CNN models on the accelerator are more as compared to that on the GPU. This is because the accelerator just has 576 PEs, whereas the GPU has 3584 CUDA cores and requires 2 orders of magnitude more area and power. This trend was observed by Lin et al. [16] as well. The reason for this is that the designs are made for systems with different form factors. We show two real-time configurations: one that uses ASICs (small form factor) and one with GPUs (large form factor).

The last column of the table shows the 3-pixel error of the CNN models on the KITTI-2015 dataset. 3-pixel error is a standard way of measuring the error for KITTI datasets in the computer vision community [5, 35], where we count all the pixels for which the disparity estimation error is more than 3 pixels and greater than 5% of the ground truth disparity. We choose *Highres* because it is the only algorithm that obeys the timing constraint, execution time per frame<100 ms [40], on both the RS dataflow accelerator and the GPU. Note that we are targeting a sampling rate of 10 frames per second (100 ms per frame).

3.2. Traditional Workloads

We choose the three most popular traditional stereo vision algorithms that are known to be accurate and are highly cited: AD-Census [19], Semi-Global Matching (SGM) [11], and PatchMatch [4]. The CNN-based algorithms have a large variance in performance and accuracy

due to the different number of layers and parameters. Unlike CNN-based algorithms, traditional algorithms are fairly similar in performance on different platforms (based on our experiments). Thus, an in-depth platform specific characterization of these algorithms is not required. In contrast, these algorithms differ significantly in terms of the accuracy of the estimated disparity primarily due to the difference in the basic building blocks. Hence, we characterize the accuracy of these algorithms on the KITTI-2015 dataset.

Table 2 shows the percentage 3-pixel error of the three traditional algorithms on the KITTI-2015 dataset. Out of the characterized algorithms, only SGM achieves an accuracy (error: 2.5%) that is at par with the accuracy achieved by the best CNN-based stereo algorithm *Highres* (error: 2.66%). Thus, we discard the other two algorithms. We then choose a suitable execution platform for SGM. We find that the execution time of SGM on an Intel Xeon CPU is 880 ms and thus does not obey the real-time constraint of 100 ms. Since the algorithm is highly parallelizable, we deploy the algorithm on faster platforms: GPU and ASIC. A multi-threaded version of SGM is found to be 80 times faster (execution time 11 ms) on the Nvidia Tesla P100 GPU as compared to the Xeon CPU. The state-of-the-art implementation of SGM on an ASIC [15] for processing one KITTI-2015 (size 1242×375) frame is 36.4 ms (scaled from 65 nm to 16 nm tech. [13]).

Algo.	Error (%)
AD-Census	10.99
SemiGlobal (SGM)	**2.5**
PatchMatch	15.7

Table 2. Error comparison

Since both the chosen algorithms, *Highres* and SGM, have comparable accuracies, we performed an in-depth accuracy analysis of the KITTI-2015 dataset (see Section 1 and the supplementary doc.).

4. Design of the Predictor

Based on the insights from Section 3, we propose to design a predictor that chooses between Highres and SGM at runtime such that the overall error is limited. We shall show the design of a lightweight *selection predictor* that takes as input the minimum possible number of features from the stereo image frames and then selects an accurate stereo algorithm: Highres or SGM.

The idea of confidence prediction has gained a lot of traction since real-life applications such as self-driving vehicles need nearly 100% accuracy [9] in the estimation of disparity. Hence, we also design a *confidence predictor* that gives a confidence with which the selected algorithm can be used such that collisions (in self-driving scenarios) are completely avoided. Our confidence predictor is able to predict if the chosen stereo vision algorithm will perform satisfactorily on a given frame, be it a CNN-based algorithm (Highres) or a traditional algorithm (SGM).

4.1. Data Augmentation

As discussed in Section 1, a CNN-based algorithm is not expected to generalize to the infinite number of driving scenarios that are possible. The key insight is that we observed that in many of these scenarios where Highres failed, SGM worked fairly satisfactorily and vice versa. In order to make our predictor generalize to different weather scenarios, we created a new version of the KITTI dataset (2200 images) by augmenting it with images capturing several weather conditions (augmented images are shown in the supplementary doc.). We used the *imgaug* [14] APIs to control the degree of different weather augmentations such as clouds, rainfall, snowfall, snow covered land, fog, and frost. We further used dropout augmentation to drop some random regions from the image and provide them a constant pixel intensity. This situation is analogous to many unpredictable situations such as locust attacks or small swarms of black birds (particularly during sunset). This is the standard way to prepare the augmented dataset using photometric transformation [17,33]. More sophisticated methods having higher photorealism can be used to improve the accuracy of the predictor.

4.2. The Selection Predictor

The process of prediction is a binary classification problem: Highres or SGM. The intuition behind the chosen features for the classification problem comes from the results of related work. It is well known that CNNs do not perform well on images that have occluded regions, textureless regions, and reflective surfaces [5, 12]. The current CNN-based stereo algorithms such as PSMNet and Highres have mitigated these issues to some extent by capturing the global context using a pyramid structure. Nevertheless, we use the *percentage of dark regions* [39] in an image as a feature. Similar to the occluded regions, no useful information is available for dark regions in an image and hence finding the disparity for these regions becomes difficult. *Contrast* and *homogeneity* [21] are features that quantify the complexity of the texture in an image. The *perceived brightness* [3] feature captures how an image would appear to humans. This is an amalgam of multiple factors such as the relative contrast, color, reflectance of the surface in an image, and flatness. Similarly, the *SSIM* [36] feature captures the contrast and structural similarity between the two images in a stereo pair.

We conducted exhaustive experiments for all the combinations of features and classifiers using the best configurations of the classifiers (configuration of the classifiers is in the supplementary doc.). Table 3 shows the validation accuracy of the classifiers for different augmentations. We used the *LOGOCV* validation technique to leave out the data points of an augmentation for each experiment (corresponding to a row) in Table 3. We observe that the highest

accuracy is achieved by the decision tree (82.5%). Other predictors also perform between $70-75\%$, however they are difficult to implement in hardware. We thus choose a decision tree as the selection predictor because it additionally provides a great degree of explainability, which may be useful for satisfying many legal requirements associated with self-driving.

Augmentation	MLP	SVM	KNN	LR	AdaBoost	DT
Cloud	83	83.5	79	83.5	81.5	85
Rain	60	59.5	59.5	59.5	65	72
Fog	74.5	74.5	73.5	74.5	74.5	78
Snow	99.5	95	95.5	99	99.5	99.5
Frost	94	90	92	94	94	96
Dropout	51	65.5	56	61.5	65.5	75
Kitti	55	52	56	52	60	72
Mean	**73.8**	**74.28**	**73.1**	**74.8**	**77.1**	**82.5**
LR → Logistic Regression, DT → Decision Tree						

Table 3. Accuracy of the classifiers for different scenarios

Since it is a vital requirement for self-driving vehicles to have explainable models, we also performed an analysis of the decision path (while using all the features). The spread of the violins for the brightness and the percentage of dark regions features is wide as shown in Figure 1. This suggests that they are used in the decision making process for the maximum number of image frames, 100% and 95%, respectively. Additionally, the peaks of the violins for the brightness and the percentage of dark regions features are at 1, 2, and 3, suggesting that they are used multiple times in the decision making process of each data point. The other features have their peaks at 0 and 1, and are hence less important. Sensitivity results of the two features are shown in the supplementary document. We explain the hardware implementation and the functionality of the two chosen features in detail in Section 5.

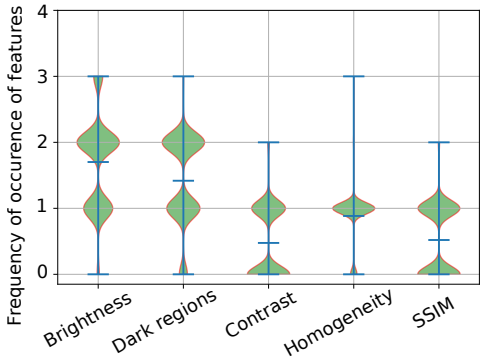

Figure 1. Frequency of features in the decision making process

4.3. The Confidence Predictor

We formulate the confidence prediction as a regression task, where the confidence is a value between 0 and 100%: defined as $100 - \%3 - pixelerror$. The idea is that the predictor should be able to predict the confidence with which the chosen algorithm from the selection predictor can be used. In order to minimize the overheads of the feature collection process, we use the same features that are used by the selection predictor. Thus, the confidence predictor uses the perceived brightness, the percentage of dark pixels, and the selection label as the inputs. The training labels are generated by using the percentage 3-pixel error in the disparity estimation as the metric. For example, if the 3-pixel error is less than 5%, then the confidence is 95%. For other scenarios, the confidence is assigned suitably. We use a decision tree regressor (enables reuse of the selection predictor hardware and has more explainability) and obtain the correct confidence prediction with an accuracy of 97% in all the weather scenarios. We can get a much better error if we search for the best features, however that would add to the feature collection overheads both in terms of execution time and energy consumption.

5. Architecture

We consider two types of SoCs: a GPU-based SoC and an ASIC-based SoC. Figure 2 shows the layout of the ASIC-based SoC in detail. In the GPU-based SoC, the accelerators are replaced with a CPU-GPU system. We now discuss the primary module: the hardware predictor.

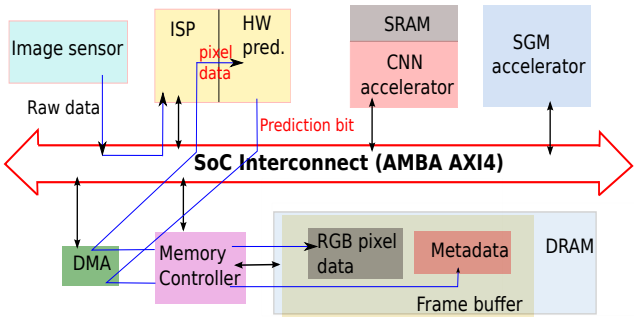

Figure 2. Architecture of the SoC with ASIC

5.1. Hardware Predictor

The hardware predictor consists of two feature calculation modules (for calculating the perceived brightness and the percentage of dark pixels) and a decision tree-based predictor. The entire pipeline, as shown in Figure 3, has four components: the interface to read the data from the SRAM of the ISP (image signal processor) to the hardware predictor, the brightness calculation module, the dark region calculation module, and the decision tree predictor module.

ISP SRAM to Hardware Predictor: We get 128 bytes of data at a time (that correspond to all the three channels of the two images in the stereo pair) from the SRAM of the ISP module (having a cache line size of 64 bytes, dual-ported). Thus, we need to consume 128 bytes of data before the next data chunk is available from the SRAM. In a 64-byte block, we have 21 bytes of data (21 8-bit pixels) for each channel, and the remaining 1 byte is used to indicate

the status (prediction completed or not). We store this data in internal registers and use it for subsequent computations.

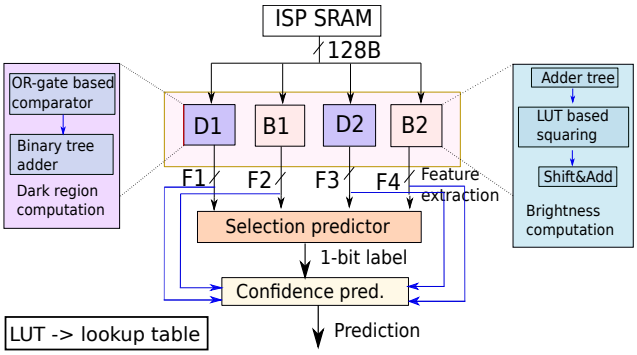

Figure 3. Architecture of the hardware predictor.

Brightness module: The standard equation [3] to calculate the perceived brightness of an image is given by

$brightness = \sqrt[2]{(0.241 \times r^2 + 0.691 \times g^2 + 0.068 \times b^2)}$. Here, r, g, and b are the mean pixel intensities in each of these channels, respectively, for the entire image. The key operations are finding the mean of the pixels, calculating the square, multiplications with floating point numbers, and the square root operation. To compute the mean of the 21 bytes of data (per channel) read from the SRAM, we have two options: compute the sum temporally (iteratively) or spatially (in parallel). An iterative computation will use 1 adder but the operation will be spread over multiple clock cycles, hence we use a binary adder tree (23 adders, tree depth $\lceil log_2 21 \rceil$) that adds 21 bytes in a cycle. The final sum is stored in a 16-bit register. We discard the least significant bits after every iteration to avoid overflows.

A total of 6 instances of the above circuit run in parallel for the 2 images in a pair, each having 3 channels. For calculating the square of a 16-bit number, we use a $128KB$ lookup table. Each 16-bit number is expressed as two 8-bit numbers: $A = A_H A_L$ and $B = B_H B_L$. Thus, $A \times B$ becomes $A_L B_L + A_H B_L \times 2^8 + B_H A_L \times 2^8 + A_H B_H \times 2^{16}$. Considering different combinations of $A_H B_L$ and $B_H A_L$, we get a total of 2^{16} combinations and each combination results in a 16-bit output, which is equivalent to $128KB$ lookup table. One multiplication translates to three lookups, three shifts and three additions. These are very fast operations [24]. Note that dividing the 16-bit number into two 8-bit chunks reduced the size of the lookup table considerably, from $256KB$ to $128KB$. The expensive floating point operations are optimized by expressing the floating point numbers as an approximate sequence of shifts and adds [10]. We omit the square root operation because in a decision tree only the relative values matter.

Dark region calculation module: For calculating the percentage of dark pixels in an image, we need to count all the pixels for which the values of the r, g, and b channels are less than or equal to a threshold. The idea is to use a comparator to compare the 8-bit pixel values of the r, g, and b channels with the threshold. Since the threshold is fixed to 15 in our case (similar to [39]), we logically OR the 4 upper bits of the 8-bit numbers. If the result is zero, the number is less than or equal to the threshold. This is done for all the three channels of a pixel and the outputs are again passed through an OR gate to filter out the pixels for which all of r, g, b values are less than the threshold. The output is then inverted to get a 1 for these cases and a binary adder tree is used to get the final count. We performed two optimizations here. First, we replaced the comparison operations by logical OR operations. Second, we omitted the percentage calculation operation (multiplication by 100 and division by the total number of pixels) because relative values matter in a decision tree. We train our decision tree as per these optimizations.

Selection and Confidence Predictor: The aforementioned modules run in parallel for both the images. After we have all the four features for both the images (two each), we send them to the decision tree. Our decision tree is of depth 5, has a total of 33 nodes, out of which 16 are the leaf nodes. Thus, we have 17 decision making nodes and hence 17 multiplexers. The decision tree is implemented as a tree of multiplexers. The select signal of each multiplexer is the threshold value of that decision node obtained after the training process. This value is stored in a 20-bit register. Each leaf node contains a 1-bit register indicating the final prediction for the stereo image pair under consideration.

Since the confidence predictor is also implemented as a decision tree, we reuse the same architecture. This is possible because the selection predictor and the confidence predictor are executed sequentially since the output of the selection predictor is an input to the confidence predictor. To take care of the different thresholds for comparison at the multiplexers in the selector and the predictor modules, we store the two thresholds for each multiplexer in two separate registers. The leaf nodes contain an additional 8-bit register to indicate the predicted confidence value $(0 - 100)$.

6. Evaluation

6.1. Setup

ASIC-based implementation: We model a 24×24 PE systolic array with 1.5 MB of SRAM, clocked at 1.2 GHz and having an RS dataflow [7] architecture using the Timeloop [23] tool. The performance, energy and area numbers are obtained from Timeloop for the 16 nm TSMC technology node. We get an area of $2mm^2$ for the accelerator. The peak throughput is similar to prior works [7, 41]: 1.152 TOPS. We get a power efficiency of 3 TOPS/W for executing Highres on our accelerator, which is in line with recent CNN accelerators suitable for SoCs [2, 6]. We as-

sume an accelerator for SGM similar to [15]. It is reported to process a 1242×375 image (same as the KITTI dataset) in $36.4ms$. We implemented the hardware predictor in Verilog. We synthesized, placed and routed the hardware using the Cadence Genus Tool (TSMC 16 nm technology) and obtained the power, area and timing numbers. We simulated the memory with the Tejas architectural simulator [26] using memory instruction traces (standard practice in the hardware design community).

GPU-based implementation: We obtained the performance and timing numbers for executing the Pytorch version of Highres, and CUDA versions of SGM and feature calculation modules on the Nvidia Tesla P100 GPU. The timings for the selection and the confidence predictor (implemented in C++) are obtained by executing them on a 48-core Intel Xeon processor clocked at 2.3 GHz. We take into account all the overheads of copying the data from the host CPU to the GPU and vice-versa.

Module	Area (μm^2)	Power (μW)
Brightness module	95.256	10.01
Dark region calculation module	26.17	2.75
Decision tree hardware	47	2.58
Overall hardware (including lookup tables)	2859	317.25
Tool	Cadence RTL compiler, 16 nm	

Table 4. Overheads of the hardware structures

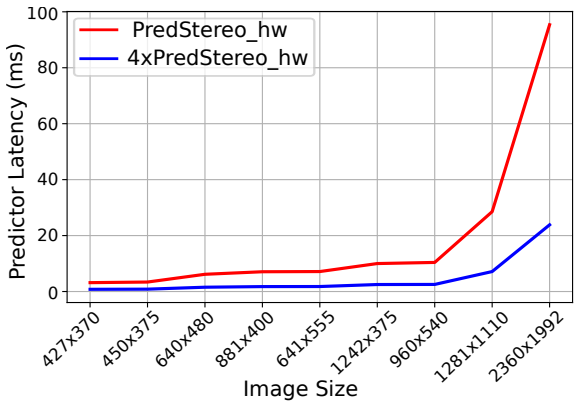

Figure 4. Sensitivity of the total time to the image size

6.2. Synthesis Results

Table 4 shows the area and power overheads of the hardware predictor and its individual modules. The total area is $0.003mm^2$, which is negligible. The total time to calculate the features for the entire image (size: 1242×375 pixels), and provide a selection label and a confidence value is 10 ms. This implementation is generic and its performance does not depend on the underlying classes of stereo algorithms. However, the total processing time of our predictor hardware increases with increasing the image sizes as shown in Figure 4. For very large images, we need to quadruple the predictor hardware and add 3 more cache banks: this will bring the total time for the 2360×1992 image to 24 ms, which is well within our time limit of 100 ms. Hence, the additional *PredStereo* hardware is scalable.

6.3. Performance Comparison

We compare the average execution time per frame (averaged over the entire KITTI dataset) for the four schemes (see Table 5) in Figure 5, where all the schemes are implemented on both GPU-based and ASIC-based SoCs. The execution times of PredictAll and Perfpredict are similar to *PredStereo* and hence not shown in the figure. Note that ASV [7] does not report the absolute performance numbers and hence we do not compare the execution time of *PredStereo* with ASV.

Scheme	Disparity estimation algorithm
Highres [40]	Highres for all frames
Highres+SGM	Highres for first frame, SGM for next two frames
Random	Random selection between Highres and SGM
ASV [7]	Highres for one frame, motion estimation for next two frames
PerfPredict	Perfect predictor to choose between Highres and SGM (100% accuracy)
PredictAll	Predictor that uses all the features to choose between Highres and SGM
PredStereo	Hardware predictor (using brightness and dark regions) to choose between Highres and SGM

Table 5. Schemes and their disparity estimation methods

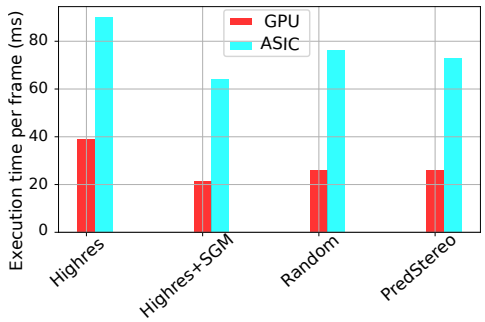

Figure 5. Execution time per frame

Figure 5 shows that the execution times for all the schemes on the GPU ($0.02 - 0.04$ s) are 2-4 $\times$ less as compared to the corresponding execution times on an ASIC ($0.06 - 0.09$ s). Additionally, the *Highres* scheme has the worst execution time as compared to the other schemes. It is also observed that with an increase in the number of frames choosing SGM, the execution time decreases. Hence, Highres+SGM scheme that uses SGM for twice the number of frames as Highres has the lowest execution time. In comparison, *PredStereo*, that uses SGM for only those frames where it is predicted to achieve a better disparity estimate is 6 ms and 5 ms slower than Highres+SGM on the ASIC and GPU, respectively. Nevertheless, as observed in Table 6, *PredStereo* achieves a 2.26% better disparity estimate as compared to Highres+SGM. This improvement is sizeable in the field of stereo vision [40].

Augmentation	ASV [7]	Highres	SGM	Highres+SGM	Random	PredictAll	PerfPredict	PredStereo
Cloud	51.7	10.19	5.09	6.05	6.9	6.56	2.9	3.97
Rain	50.3	8.15	6.24	6.94	7.2	6.23	5.3	5.7
Fog	39.8	6.6	3.27	4.29	4.8	3.33	3	3
Snow	60	26	11	13.44	16.75	7.41	7.4	7.4
Frost	59.6	23.4	12.1	14.3	16.9	10.47	10.2	10.2
Dropout	48.7	3.7	3.06	3.32	3.4	3.05	2.7	2.8
Kitti	45.5	2.66	2.58	2.63	2.62	2.4	2.06	2.12
Mean	**50.8**	**11.47**	**6.19**	**7.28**	**8.36**	**5.63**	**4.79**	**5.02**

Table 6. Comparison of the error in the disparity estimation of different schemes in several weather conditions

6.4. Accuracy Comparison

For the accuracy comparison, we first train the predictor for the entire dataset (with augmentations). We calculate the accuracy of disparity estimation for all the stereo vision schemes and report the accuracy numbers in Table 6. The table shows the error in the estimation of disparity for a sequence of unseen frames taken from the current augmented dataset or the default KITTI dataset (in the x-axis). This experiment essentially quantifies the effect of a bad predictor on the accuracy of the disparity estimation.

We observe that ASV achieves the highest disparity estimation error on all the data augmentations as compared to all the other schemes (same reasons as Section 1). The second highest error is obtained by the *Highres* scheme. Since all the other schemes involve SGM (albeit for different number of frames) for the disparity prediction, they are more accurate especially for the augmented scenarios. Particularly, for the snow and frost data augmentations, our scheme *PredStereo* is 18% and 13% more accurate over *Highres*, respectively. This is because for the images with these augmentations, SGM has a better disparity estimate as compared to *Highres* and our predictor is able to accurately predict the more accurate algorithm out of the two. Moreover, *PredStereo* is on average only 0.23% less accurate in estimating the disparity than *PerfPredict*. This implies that even though the accuracy of our predictor is 82.5% (see Section 4.2), we are predicting the crucial points (the points that have a higher gap in the disparity estimation errors using SGM and *Highres*) accurately (96.6% accuracy), and hence it translates to a lower error in the disparity estimation.

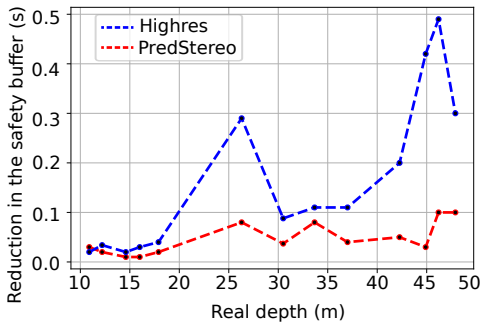

Figure 6. Reduction in the safety buffer time

6.5. Analysis of the Disparity Estimation Error in a Self-Driving Scenario

In this section, we analyze the effect of error in the disparity estimation in a self-driving scenario. Any car has to maintain a half speedometer distance (1.8 s time) [18] from the vehicles in front of it to allow a safety buffer time. Let us assume a car with an average speed of 70 km/hr and following the half speedometer distance rule. We calculate the reduction in the safety buffer time (1.8 s) as a result of error in the prediction of the distance of the vehicle in front. Figure 6 shows the reduction in the safety buffer time (refer the suppl. doc. for the calculation details) for the augmented KITTI images having vehicles at different distances in front of it. We observe that Highres can lead to high reductions (upto 500 ms), while the reductions when using *PredStereo* are limited to 100 ms.

A 500 ms reduction is significant because it leaves 1.3 s to the safety buffer, and it typically takes 1.2-1.5 s [8, 37] to react and apply brakes. Hence, a high accuracy is needed for such safety-critical applications.

7. Conclusion

There are several important takeaways in this work. The first is that CNNs are not a panacea for all computer vision problems; there is a need to recognize the accuracy of traditional algorithms such as SGM, and dynamically choose between the two at run time. A low-overhead yet accurate hardware predictor that uses only two features – the percentage of dark regions and perceived brightness – can be constructed and made to work at real-time. A confidence predictor based on the same features can provide necessary hints to activate a higher-level control system, if the confidence in the disparity estimation is low. Overall, *PredStereo* improves the disparity estimation error over a state-of-the-art CNN-based stereo vision system by up to 18% (on average 6.25%) with a negligible area overhead ($0.003mm^2$) while respecting real-time constraints.

Acknowledgements

The authors thank the anonymous reviewers for their insights. They thank Ashish Kumar of UC Berkeley, Dr. Chetan Arora and Ankita Raj of IIT Delhi for providing suggestions and feedback on various parts of the paper.

References

[1] Rajeev Balasubramonian, Andrew B Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. Cacti 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Transactions on Architecture and Code Optimization (TACO)*, 14(2):1–25, 2017.

[2] Brendan Barry, Cormac Brick, Fergal Connor, David Donohoe, David Moloney, Richard Richmond, Martin O'Riordan, and Vasile Toma. Always-on vision processing unit for mobile applications. *IEEE Micro*, 35(2):56–66, 2015.

[3] Sergey Bezryadin, Pavel Bourov, and Dmitry Ilinih. Brightness calculation in digital image processing. In *International symposium on technologies for digital photo fulfillment*, volume 2007, pages 10–15. Society for Imaging Science and Technology, 2007.

[4] Michael Bleyer, Christoph Rhemann, and Carsten Rother. Patchmatch stereo-stereo matching with slanted support windows. In *Bmvc*, volume 11, pages 1–11, 2011.

[5] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018.

[6] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[7] Yu Feng, Paul Whatmough, and Yuhao Zhu. Asv: Accelerated stereo vision system. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 643–656, 2019.

[8] Nathan A Greenblatt. Self-driving cars and the law. *IEEE spectrum*, 53(2):46–51, 2016.

[9] Abhishek Gupta, Alagan Anpalagan, Ling Guan, and Ahmed Shaharyar Khwaja. Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array*, page 100057, 2021.

[10] Bah-Hwee Gwee, Joseph S Chang, Yiqiong Shi, Chien-Chung Chua, and Kwen-Siong Chong. A low-voltage micropower asynchronous multiplier with shift–add multiplication approach. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(7):1349–1359, 2008.

[11] Heiko Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 807–814. IEEE, 2005.

[12] Yaoyu Hu, Weikun Zhen, and Sebastian Scherer. Deep-learning assisted high-resolution binocular stereo depth reconstruction. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8637–8643. IEEE, 2020.

[13] Wei Huang, Karthick Rajamani, Mircea R Stan, and Kevin Skadron. Scaling with design constraints: Predicting the future of big chips. *IEEE Micro*, 31(4):16–29, 2011.

[14] Alexander B. Jung et al. imgaug. `https://github.com/aleju/imgaug`, 2020. Online; accessed 01-Feb-2020.

[15] Kyuho Jason Lee, Kyeongryeol Bong, Changhyeon Kim, Jaeeun Jang, Kyoung-Rog Lee, Jihee Lee, Gyeonghoon Kim, and Hoi-Jun Yoo. A 502-gops and 0.984-mw dual-mode intelligent adas soc with real-time semiglobal matching and intention prediction for smart automotive black box system. *IEEE Journal of Solid-State Circuits*, 52(1):139–150, 2016.

[16] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 751–766, 2018.

[17] Alexander Mai, Allen Yang, and Dominique E Meyer. Soft expectation and deep maximization for image feature detection. *arXiv preprint arXiv:2104.10291*, 2021.

[18] Markus Maurer, J Christian Gerdes, Barbara Lenz, and Hermann Winner. *Autonomous driving: technical, legal and social aspects*. Springer Nature, 2016.

[19] Xing Mei, Xun Sun, Mingcai Zhou, Shaohui Jiao, Haitao Wang, and Xiaopeng Zhang. On building an accurate stereo matching system on graphics hardware. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 467–474. IEEE, 2011.

[20] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3061–3070, 2015.

[21] P Mohanaiah, P Sathyanarayana, and L GuruKumar. Image texture feature extraction using glcm approach. *International journal of scientific and research publications*, 3(5):1–5, 2013.

[22] Diksha Moolchandani, Anshul Kumar, and Smruti R. Sarangi. Accelerating cnn inference on asics: A survey. *Journal of Systems Architecture*, 113:101887, 2021.

[23] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 304–315. IEEE, 2019.

[24] Eldhose Peter and Smruti R Sarangi. Optimal power efficient photonic swmr buses. In *2015 Workshop on Exploiting Silicon Photonics for Energy-Efficient High Performance Computing*, pages 25–32. IEEE, 2015.

[25] The Robot Report. Researchers back teslas non-lidar approach to self-driving cars. `https://www.therobotreport.com/researchers-back-teslas-non-lidar-approach-to-self-driving-cars/`.

[26] Smruti R Sarangi, Rajshekar Kalayappan, Prathmesh Kallurkar, Seep Goel, and Eldhose Peter. Tejas: A java based versatile micro-architectural simulator. In *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 47–54. IEEE, 2015.

[27] Frank Schumacher and Thomas Greiner. Matching cost computation algorithm and high speed fpga architecture for

high quality real-time semi global matching stereo vision for road scenes. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 3064–3069. IEEE, 2014.

[28] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 97–108. IEEE, 2014.

[29] Aristotle Spyropoulos and Philippos Mordohai. Ensemble classifier for combining stereo matching algorithms. In *2015 International Conference on 3D Vision*, pages 73–81. IEEE, 2015.

[30] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. Misbehaviour prediction for autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 359–371, 2020.

[31] Yandex Self-Driving Team. 10 million kilometers in challenging conditions. `https://medium.com/yandex-self-driving-car/10-million-kilometers-in-challenging-conditions-d1d0045e5df0`.

[32] The Verge. Zoox unveils a self-driving car that could become amazons first robotaxi. `https://www.theverge.com/2020/12/14/22173971/zoox-amazon-robotaxi-self-driving-autonomous-vehicle-ride-hailing`.

[33] Xiang Wang, Kai Wang, and Shiguo Lian. A survey on face data augmentation for the training of deep neural networks. *Neural computing and applications*, pages 1–29, 2020.

[34] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8445–8453, 2019.

[35] Yan Wang, Zihang Lai, Gao Huang, Brian H Wang, Laurens Van Der Maaten, Mark Campbell, and Kilian Q Weinberger. Anytime stereo image depth estimation on mobile devices. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5893–5900. IEEE, 2019.

[36] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[37] Darrell M West. Moving forward: self-driving vehicles in china, europe, japan, korea, and the united states. *Center for Technology Innovation at Brookings: Washington, DC, USA*, 2016.

[38] Yannan Nellie Wu, Joel S Emer, and Vivienne Sze. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[39] Feng Yang, Luciano Sbaiz, Edoardo Charbon, Sabine Süsstrunk, and Martin Vetterli. On pixel detection threshold in the gigavision camera. In *Digital Photography VI*, volume 7537, page 75370G. International Society for Optics and Photonics, 2010.

[40] Gengshan Yang, Joshua Manela, Michael Happold, and Deva Ramanan. Hierarchical deep stereo matching on high-resolution images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5515–5524, 2019.

[41] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. Euphrates: Algorithm-soc co-design for low-power mobile continuous vision. *arXiv preprint arXiv:1803.11232*, 2018.