Journal of Real-Time Image Processing

Real Time Embedded Implementation of Appearance based Terrain Learning for Autonomous Rover Systems

Prabhakar Mishra[1], Anirudh Viswanathan[2], Nidhi A[1], Nithin Eswarappa[1], J K Kishore[3]

[1]*Centre for Intelligent Systems*
PES Institute of Technology
100-ft Ring Road, Banashankari 3rd Stage
Bangalore-560085, India

[2]*Robotics Institute*
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213

[3]*Indian Space Research Organization*
Bangalore, India

The e-mail address, telephone and fax numbers of the communicating author:

Tel: +91 80 26722448 Extn 776

Fax: +91 80 26720886

prabhakar.mishra@pes.edu

Real Time Embedded Implementation of Appearance based Terrain Learning for Autonomous Rover Systems

Abstract: This paper presents the design of novel, embedded computational architectures for real-time implementation of appearance based terrain learning for use in autonomous rovers. Self-supervised appearance based terrain learning algorithms are adapted for real-time implementation. A System-on-Chip based design methodology is adopted to design the computational system in view of constraints on size, weight, on-board power, memory and computational elements on such rovers. Three variants of embedded computational architectures are proposed to implement the optimized algorithm. The first one is a multi-processor system incorporating soft-core processors, the second uses application specific co-processors designed to accelerate the computationally intensive stages, while the third is based on custom design of the entire data path. The proposed architectures are prototyped on state of art Field Programmable Gate Arrays and are compared for latency, resource utilization and power overhead.

Keywords: Appearance Based Terrain Learning, Autonomous Rover, Embedded Computational Architecture, Field Programmable Gate Arrays

1. Introduction

Mission critical applications such as planetary surface exploration, reconnaissance, search and rescue often employ rover systems to meet their objectives. A case in point is the Mars exploration program where rovers have been deployed for planetary surface exploration.

Such missions are characterized by two levels of complexities. The first one arises due to low bandwidth and large latency of communication between deep space missions and the earth stations for command and control. This necessitates endowing high degree of autonomy to such rover systems.

In the paper Autonomy for Mars Rovers: Past, Present, and Future [1], researchers working in the Jet Propulsion Laboratory towards design of Mars rovers opine that "The vehicles used to explore the Martian surface require a high degree of autonomy to navigate challenging and unknown terrain, investigate targets and detect scientific events. Increased autonomy will be critical to the success of future missions".

The key components of autonomous rover navigation in partially known or unknown environments for exploration are: 1) Ability of in-motion mapping and localization 2) Ability of reactive navigation for local collision avoidance 3) Online learning of navigable terrains to adapt to vagaries of terrain inclination and relief profiles and 4) handling dynamism in the environment. Autonomous rovers employ multiple sensors and complex sensor data fusion methods to achieve robust exploration of their environment.

The second complexity in such rover applications arises due to payload considerations. The scientific objectives of planetary missions require dedicated

equipment to be mounted on the rover. This places considerable constraints on the size, weight, on-board power, memory, computational resources available for implementing the different functionalities required for real-time exploration. The nature of constraints and the degree of autonomy in such rover applications can be appreciated by considering the characteristics of the different Mars rovers as an example.

Table 1.

Rover	Power source	Power	Degree of Autonomy	CPU	Memory	Operating System
Sojourner Rover 11.5 kg Payload 7.78 kg	Solar arrays	16 watts	Obstacle detection	2 Hz Intel 80C85	RAM 512 KB Flash 176 KB	Custom cyclic executive
Spirit and Opportunity	Solar arrays	140 watts	Limited autonomy	20 MHz RAD6000	RAM 128 MB Flash 256 MB	VxWorks (multitasking)
Curiosity	Radioisotope thermoelectric generator	110 watts	Limited autonomy	200 MHz RAD750	RAM 256 MB Flash 2 GB	VxWorks (multitasking)

Form NASA website.

The experiences of the Mars rover missions suggest that use of System-on-Chip design paradigm can provide the necessary framework for leveraging advantages offered by multiple processor systems to support parallel and pipelined computations involved in different algorithms. Authors in paper [1max.] emphasize that "The stereo vision and visual odometry algorithms that generally dominate the execution time on the Mars rovers are amenable to custom hardware implementation in an FPGA. Not only would this allow the algorithm to run faster but it off loads the processing from the main CPU. Another method of increasing computations… is using a faster co-processor…".

In line with this, Mars exploratory rover programs have experimented with the use of Field Programmable Gate Arrays (FPGA) for designing the on-board computational architecture. Operations in deep space face extremities of radiation and temperature. These can induce soft-errors in the processing elements and memory. Radiation hardened once programmable FPGAs based on anti-fuse technology provided by Actel has been used extensively on-board different satellites []. However the use of static RAM based FPGAs in deep space missions was first reported in the Mars rover Spirit.

Twenty three Actel FPGAs have been used in the Mars Pathfinder and the Mars Sojourner rovers [Actel 1997 annual report]. Xilinx Virtex families' radiation tolerant XQVR4062 FPGAs have been used in the Mars Exploration Rover's (MER's) "landing craft to control the crucial pyrotechnic operations during a rover's multiphase descent and landing procedure" as reported in [Xcell Journal 2010], and XQVR1000 FPGAs have been used in the MER's Motor Control

Board [Xcell Journal 2010]. Actel FPGAs have been used as part of the camera electronics on the MERs [Actel FPGAs].

Xilinx Virtex-II FPGAs are used in the camera subsystem in the Curiosity Rover. Four Curiosity imagers communicate to image pipelines implemented with Virtex-II FPGAs. As reported in [Xilinx of Mars], "the interface, compression, and timing functions are implemented as logic peripherals of a Microblaze soft-processor core in the Virtex-II FPGA".

Static RAM based FPGAs provide the logic density required for incorporating fault tolerance through triple modular redundancy apart from ease of prototyping and smaller turnaround time. These FPGAs can be used to reconfigure the functionality even after the rover is deployed.

In this paper, we address issues relating to real time implementation of on-line terrain learning for use in resource constrained mission critical autonomous rovers. Self supervised terrain learning algorithm as proposed by [Dhalkamp] has been modified to make it amenable to real-time implementation. Embedded computational architectures using System-on-Chip design methodology targeted to state-of-art SRAM based FPGAs have been designed to implement terrain learning. Three approaches have been investigated. Firstly we reuse existing soft-core processor Intellectual Property (IP) along with existing memory systems and shared bus communication protocols to design a multi-processor System-on-Chip. We characterize latency, power and area overhead incurred in implementing the terrain learning algorithm on this architecture. Computationally intensive tasks in the algorithm are identified and dedicated co-processors are designed to accelerate them in the second approach. A complete custom design for all tasks has been undertaken in the third approach. These approaches provide insights into the trade-off involved in design effort against performance for computationally intensive algorithms for use in autonomous rovers.

The rest of the paper is organised as follows: Section 2 presents related work; Section 3 discusses the adaptations made in the algorithm for real-time implementation and section 4 details computational flow and task graph extraction. The three variants of the architectures are presented in Section 5, which is followed by results and analysis in Section 6. Section 7 concludes the work.

2. Related Work

Autonomous rover navigation in unknown environment requires the ability of some form of traversability analysis to determine whether or not the region in its immediate vicinity is navigable. Assessment of terrain traversability finds plurality of applications such as planetary surface exploration, automated drive systems for urban traffic and off road exploration, search and rescue missions in

mines and sites of disaster etc. Algorithms designed for terrain traversability analysis amalgamate findings from broad areas in robot perception, computer vision, artificial intelligence and machine learning.

Initial attempts at terrain traversability analysis were restricted to higher level classification of navigable and non-navigable terrains. However in critical missions safety considerations necessitate fine grain traversability assessment which includes inclination, texture, slippage, sinkage and moisture content to determine whether or not a particular path chose by the path planner is feasible. A review of different approaches of terrain traversability analysis in unmanned vehicles is provided in [Papadakis, 2013 survey paper]. The taxonomy of terms used for this purpose as observed by the authors include drivability, navigability maneuverability, mobility and traversability.

Two mainstream approaches to assessment of navigable terrains are geometry based and appearance based which employ a combination of proprioceptive and exteroceptive sensory data. Of late researchers are reporting hybrid approaches to terrain classification which use multiple sensors and complex sensor data fusion methods.

One of the earliest approaches employing geometric features was proposed by [Hoffman kraotkov 1989]. This method employs Fourier analysis to obtain a set of parameters which determine terrain roughness. The [Pai and reissell] employed a multi scale space analysis approach using wavelet decomposition to model terrain traversability at multiple resolutions. The environment was modeled as a 2-D occupancy grid and a terrain roughness measure was computed for each cell in the grid. Langer et al [] and Gennery[] used statistic processing for constructing 2-D traversability grid maps based on elevation measure from a set of 3-D points for each cell in the grid. Variance of height and slope were the main factor considered for traversability assessment. Singh et al [] proposed modeling of uncertainty of terrain through the notions of certainty and goodness that were combined to derive a stereo map. The goodness of a cell in the grid was measured based on role, pitch and roughness of the plane of traversal. Thrun et al[] used probabilistic feature analysis in a locally spaced elevation profile where elevation difference in the 3-D point cloud was used to filter regions as obstacle, drivable space or unknown. This method was employed on the vehicle 'Stanely' which won the DARPA grand challenge [] for autonomous driving in desert terrain.

Lalonde et al [] derived shape features such as edges, planar patched and clusters of scattered 3-D points using LIDAR data. Three properties namely scatter-ness and surface-ness and linear-ness were used for terrain analysis. A Gaussian mixture models of these classes were learnt based on expectation maximization on a set of features computed using principle component analysis. Helmick et al [], built upon the work reported by Goldberg et al [] where step roughness, pitch and border hazards were identified. This method has been employed in the early Mars Exploration Rovers. Terrains were classified into definitely traversable, definitely

non-traversable or unknown using goodness map which quantifies traversability by a local planar fit of patches across the map.

Traversability analysis incorporating rover dependent variables by defining a 3-D terrain model in relation to a 3d vehicle model was presented by Simion []. However limited computing capabilities limit the use of complicated terrain and vehicle models and their interaction for real time applications.

Appearance based traversability analysis employs computer vision to obtain features for classification. These approaches have found favor with planetary surface exploration missions because in the early stages of design of such rovers, space grade sensors were limited to cameras. Howard et al [] report the use of regression based terrain traversability analysis for planetary exploration using features such as roughness, slope, discontinuity and hardness. Neural networks were used to classify cliffs and ravines based on discontinuity, wheel slippages based on hardness and horizon based on slope.

Angelova et al [], used a hierarchical approach for terrain traversability analysis based on color statistics and textons. The terrain was classified into soil, sand, gravel, asphalt and woodchips using a top down approach to classification.

An overview of methodologies used in terrain classification using a wide range of sensor suites is given in [Papadakis, 2013] and [Dezoua and Kak, 2002].

Terrain classification using only geometry based features (features extracted from range sensors) is described in [Lalonde et al., 2006]. In this paper, 3D LIDAR data were classified based on scatter-ness, surface-ness and linear-ness for natural terrain analysis. Through the use of ground truth data, Gaussian mixture models of these classes were learnt by employing Expectation Maximization on a set of static features. These features were computed from principal component analysis decomposition of sets of neighboring points across the 3D scene.

Geometry based features including elevation, roughness, shape and size have been used by [Singh et al., 2000] [Vandapel et al., 2004] [Montemerlo and Thrun 2004] for terrain classification.

In MER's (Mars Exploration Rover), Geometric data of the terrain generated using a stereo system is used by (Geometric Estimation of Surface Terrain Traversability Applicable to Local Terrain) GESTALT system [Goldberg et al., 2002] to build a grid-based local traversability map containing, grid cell goodness evaluations and possible paths. Absence of appearance-based terrain analysis capabilities in the MER software restricts hazard detection to geometric obstacles [Biesiadecki,et al., 2006]. Geometric features from range sensors have variable resolution that decays rapidly with distance. The systems which employ only range sensors exhibit myopic behaviour as their sensing is limited to near-field. Thus when generating a three-dimensional (3D) geometry of the terrain, it can be reconstructed only in the near field.

In [Angelova et al., 2007] [Kim et al., 2007] [Vernaza et al., 2008] [khan et al., 2011] [Filitchkin and Byl, 2012], terrain is classified using only appearance based features (features extracted from image – color, texture, etc).

In [Angelova et al., 2007], appearance based terrain classification was performed on feature spaces of varying resolution. A hierarchical terrain classifier is used to efficiently classify terrain into soil, sand, grave, asphalt, grass and woodchips. In [Kim et al., 2007], Superpixel based image classification, are demonstrated to be more accurate than the patch based image classification using only stereo vision sensor. An online self supervised learning algorithm is trained and used to accurately segment an image in to obstacle and ground patches based on supervised input in [Vernaza et al., 2008]. Previous approaches to terrain classification made an implicit assumption of independence in local pixel neighborhoods. In [Vernaza et al., 2008] the terrain classification scheme used, relaxed the independence of pixels to local pixel neighborhoods by modeling correlations between pixels in the sub modular MRF framework.

A comparison of multiple approaches to visual terrain classification for outdoor mobile robots using local features (including texture based descriptors and key point descriptors) is reported in [khan et al., 2011]. In this paper, Random Forest based approach was used for classification and cross validation scheme for verification of results. In [Filitchkin and Byl, 2012], taerrain is classified using a bag of visual words (BOVW) created from speeded up robust features (SURF) with a support vector machine (SVM) classifier. A sliding window technique is also used to classify mixed terrain images with high resolution.

Terrain classification using a fusion of geometry and appearance based features are described in [Dima et al., 2004][Dahlkamp et al., 2006][Happold et al., 2006][Bajracharya et al., 2009][Hadsell, 2009]. Road detection using a combination of sensor information from a laser range finder, a pose estimation

system and a color camera is described in [Dahlkamp et al., 2006]. Using the laser range finder and the pose estimation system, a nearby patch of drivable surface is identified first. Then an appearance model of this patch is constructed and is used to find drivable surface outward into the far range.

Data from multiple sensors (color camera, infrared, (IR) camera and laser range finder) are fused to extend the range of operating conditions of the robotic vehicle in [Dima et al., 2004]. Multiple classifiers are trained using different subsets of features and different classifier fusion schemes are used for robustness. In order to obtain the benefits of fusing the classifier outputs, a higher level of classification described in [Belluta et al., 2000] is employed to reduce manual tuning. Farfield terrain classification according to appearance of nearby navigable terrain is a strategy employed in [Manduchi et al., 2005][Dahlkamp et al., 2006][Happold et al., 2006][Procopio et al, 2008] [Bajracharya et al., 2009][Hadsell, 2009][Mishra et al., 2013].

The algorithms discussed till now are computationally expensive for real-time implementation on resource constrained systems. A computationally inexpensive algorithm for appearance based terrain learning is described in [Mishra and Vishwanathan, 2013].

An implementation of this algorithm with additional modification (to increase its amenability for real-time implementation) is described in this paper. The method uses a similar learning and scoring mechanism to [Dahlkamp et al., 2006], but in the training stage color models are trained by pre-filtering pixels in the perception space. The pre-filtered pixels are assumed to correspond to regions of free-space immediately in front of the rover. The vision algorithm learns dominant colors, as a Mixture of Gaussians. Sub-space clustering is implemented using bit-shifts, to learn color models of maneuverable regions. Additionally, a horizon detection scheme restricts the model update related computation to the ground region. The combined strategies of learning models in scale space, with Look-Up-Tables to establish the pixel-wise mapping between scales, and modifications to suit integer data-path allows for real-time throughput which is orders of magnitude faster as compared to the technique by [Dahlkamp et al, 2006]. The scheme allows for online and in-motion terrain model update. The pipeline architecture used is made amenable to real-time implementation by restricting computations to bit-shifts and accumulation operations. Model initialization and update follows at the coarse scale of the decimated image and is back projected onto the original fine scale.

Three variants of embedded computational architectures have been designed to implement this algorithm. The first architecture is a multi-processor based system comprising of soft-core processors, the second comprises of co-processors used as hardware accelerators and the third comprises of a custom data path. The three architectures are prototyped on a Xilinx Virtex 6 FPGA.

The advantages of FPGAs in space based applications include its reconfigurable fabric, low non-recurring engineering costs and short design cycle [K. Morris, FPGAs in Space, Tech Focus Media, FPGA and Structured ASIC Journal 2004][SRAM FPGA Reliability] [FPGAs on mars]. Actel's RTFXS and RTAXS FPGA families (Radiation Tolerant FX or AX architecture "Space" versions) use metal-to-metal antifuse connections for configuration and include built-in TMR on all registers. The disadvantages of these FPGAs include lower density than high-end SRAM devices and are not reconfigurable.

SRAM FPGAs are susceptible to radiation and for designs incorporating them, cyclic redundancy checking (CRC), triple mode redundancy (TMR) and frequent re-configuration are done to increase reliability. SRAM FPGA vendors such as Xilinx improve the radiation tolerance with techniques such as imbalanced latches for configuration circuitry that make read efficiency better and state stability higher[K. Morris, FPGAs in Space]. Examples of Mars rovers which use FPGAs include the MERs where Xilinx Virtex XQVR1000s were used to control the motors, and Curiosity where a Virtex-II FPGA was used in the camera subsystem [FPGAs on mars][Xilinx of Mars: NASA's Curiosity rover shoots Martian eclipse using Smarter Vision].

A summary of FPGA implementation of computer vision algorithms in planetary rovers is present in [Computer Vision on Mars]. Development and implementation of vision systems for autonomous rover navigation using FPGAs for space applications are described in [SPARTAN project]. The system described in [SPARTAN project] uses visual data for 3D reconstruction and localization.

The terrain learning algorithm detailed in [Mishra et.al] has been implemented on three novel embedded computational architectures in this paper. The different stages in the algorithm include decimation, 3-D histogram building, K-means clustering, horizon detection and scoring. Implementation of K-means clustering algorithm has been described in [Novel K-Means Clustering on FPGAs]. Computational architectures for histogram building and computing covariance matrices on FPGA have been discussed in [bailey]. As far as we know this is the first paper which presents the complete computational architecture for appearance based learning of navigable terrain for autonomous rovers.

3. Algorithm

An overview of the algorithm for computationally inexpensive appearance based terrain learning in unknown environments is shown in Figure 1. The major stages are as follows:

A. Decimation
B. Pre-filtering
C. Color based K-Means pixel clustering
D. Horizon detection

E. Scoring pixels in the perception space

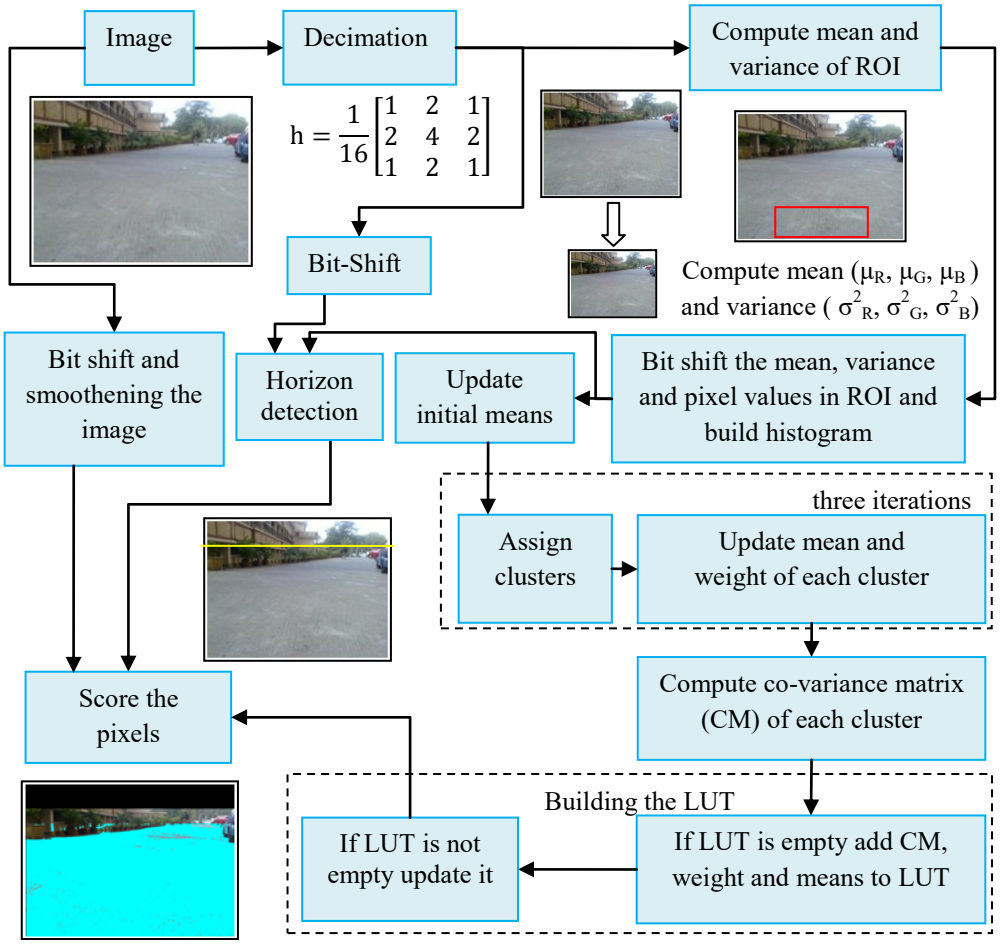

Figure 1. An overview of the algorithm for Learning Navigable Terrain

Each frame 'f', in the incoming video feed is decimated as formulated in [Szeliski, 2010] to reduce the resolution resulting in a smaller RGB color search space for clustering. The incoming frame, 'f', is convolved with a low-pass filter, 'h' as shown in equation 1.

$$g(i,j) = \sum_{k,l} f(k,l)h(i-k,j-l) \qquad (1)$$

In the equation, pixel coordinates are denoted by the ordered-pair (i, j), and the smoothening kernel h(k, l) used in this paper is shown in Figure 1.

The input image is of size 640 x 480 and is 3 channel color image with 8 bits per channel at 15 frames per second. The notation adopted is as follows: each image is represented as $z_{u, v, k.}$ Pixel coordinate is denoted by (u, v) and the corresponding color channel by k. Since the proposed method is applied to each of the color channels k, the subscript k, is dropped in the discussion, and operations are done on $z_{u, v}$.

The description of K-means clustering stage has been reproduced from [Mishra et al., 2013]. In this stage, computations are done on a Region of Interest (ROI) of size 64x128 in the image corresponding to closest navigable region (validated by a proximity sensor). Mean and variance of the ROI are computed and then bit

shifted by 3 bits. The pixel values in the ROI is bit shifted by 3 and a 3D histogram of the bit-shifted ROI is built.

Formally, let {(u1, v1), (u2, v2)…, (um, vm)} ∈ S represent the position of pixel inliers in the ROI. The i^{th} pixel in the color space, is represented by g (u_i; v_i) ∈ R^3. S = {(u_i, v_j) | (i, j) ∈ D}, where D consists of pixel coordinates inside the rectangular ROI. Specifically, the cardinality of D is chosen to be a power of two, so divisions operations for the entire ROI reduce to bit-shifts. The ROI is chosen to be a window of size 64 x 128(2^P X 2^Q). Consequently, pixels in the ROI are represented by X ⊂{S ∩ f} Individual pixels, pre-filtered, in the ROI, are represented by x (u, v) ∈ X.

The following notations are adopted when defining the ROI:
$\mu_X = [\mu_R \quad \mu_G \quad \mu_B]$ - Mean of the ROI.
$\sigma_X^2 = [\sigma_R^2 \quad \sigma_G^2 \quad \sigma_B^2]$ - Variance of the ROI.
$z_{u,v}$ denotes pixel with intensity value z ∈ [0: 255] at pixel position (u, v).

The mean pixel value in the ROI is computed as given in equation 2. This equation reduces to an accumulate step and a bit shift as given in equations 3 and 4 respectively. For the window size (64 X 128), a 13 bit right-shift for division is required. In equation 4, the values of P and Q are 6 and 7 respectively.

$$\mu_X = \sum_{(u,v)\,\epsilon\,s} z_{u,v}/2^{P+Q} \qquad (2)$$

$$\text{Accumulate: } a_X = \sum_{(u,v)\in S} z_{u,v} \qquad (3)$$

$$\text{Shift: } \mu_{ROI} = a_X \gg (P+Q) \qquad (4)$$

The variance of ROI is computed as given by equation 5. Then the mean, variance and the ROI are bit shifted by 3 bits as in shown in equations 6, 7 and 8 respectively.

$$\sigma_X^2 = \sum_{(u,v)\,\epsilon\,x}\left(z_{u,v} - \mu_X\right)^2 / 2^{P+Q} \qquad (5)$$

$$\mu_X = \mu_X \gg 3 \qquad (6)$$

$$\sigma_X^2 = \sigma_X^2 \gg 3 \qquad (7)$$

$$t\,(u,v) \leftarrow x(u,v) \gg 3 \qquad (8)$$

The bit-shifted version of X is T and is defined by $X \xrightarrow{3} T$. This pre-processing before clustering reduces the search space. A 3D histogram H_T, is built from T to reduce the number of computations in the clustering stage.

The operations in pre-filter stage are done on a perception sub-space, T⊂X to reduce computational complexity. The cluster initialization stage in the pre-filter stage uses the mean of the ROI. K-means clustering based on the method described in [Duda and Hart, 2007] is performed on H_T, rather than T itself which

further reduces the number of computations. The m pixel values with non zero weights of the 3D histogram are used as training samples. Specifically, the training data set is

$\{t^{(1)},t^{(2)},\ldots, t^{(m)}\} \in H_T$.

The K clusters (K is chosen to be 3 here), with centroid $\{\mu_1, \mu_2, \mu_3 \in R^3\}$ are initialized such that the centroid of first and last clusters are set to the extremities of the RGB space and the centroid of the second cluster to the center of RGB space. The centroid of the clusters nearest to the ROI mean is replaced by mean μ_x.

In the cluster assignment step, for the i^{th} training example, the index of the cluster is computed using equation 9. Subsequently, the update centroid step of each cluster is performed using equation 10.

$$C^{(i)} \leftarrow k \text{ that minimizes } \|t^{(i)} - \mu_k\| \quad (9)$$

$$\mu_k \leftarrow \frac{1}{|W_k|}\sum_{i\in C_k} t^{(i)} * H_T\left(t^{(i)}\right) \quad (10)$$

The clusters obtained by this process forms the three color models which are used for terrain learning. For each model, a co-variance matrix (CM) is computed as given in equation 11 and 12. In this equation, 'W' denotes the weight (number of pixels in the color model) and R, G, B denotes pixel intensity values in the cluster. A simplified representation of the CM is shown in equation 11. In this equation, CMS denotes Covariance Matrix Sum.

$$CM_k = \frac{CMS_k}{W_k} \quad (11)$$

$$CMS_k = \begin{bmatrix} \sum(Ri-\mu_R)^2 & \sum(Ri-\mu_R)^2(Gi-\mu_G)^2 & \sum(Ri-\mu_R)^2(Bi-\mu_B)^2 \\ \sum(Ri-\mu_R)^2(Gi-\mu_G)^2 & \sum(Gi-\mu_G)^2 & \sum(Bi-\mu_B)^2(Gi-\mu_G)^2 \\ \sum(Ri-\mu_R)^2(Bi-\mu_B)^2 & \sum(Bi-\mu_B)^2(Gi-\mu_G)^2 & \sum(Bi-\mu_B)^2 \end{bmatrix} \quad (12)$$

The K color clusters are characterized by their mean (μ), covariance matrix (CM) and number of pixels in the cluster (or cluster weight W). In addition to the K training models, N learned models are present, which incorporate the past history of dominant terrain color, with N > K. The training models are scored for color similarity with the learned models using Mahalanobis distance. When the condition in equation 13 is satisfied, learned model is updated using equations 14 to 16.

$$(\mu_L - \mu_T)(CM_L + CM_T)^{-1}(\mu_L - \mu_T)^T \leq 1 \quad (13)$$

$$\mu_L \leftarrow \frac{\mu_L W_L + \mu_T W_T}{W_L + W_T} \quad (14)$$

$$CM_L \leftarrow \frac{CM_L W_L + CM_T W_T}{W_L + W_T} \quad (15)$$

$$W_L \leftarrow W_L + W_T \quad (16)$$

If the current training model does not match any of the learned models, as per equation 13, then model update is performed using equation 17.

$$\left.\begin{matrix}\mu_L \leftarrow \mu_T \\ CM_L \leftarrow CM_T \\ W_L \leftarrow W_T\end{matrix}\right\} \text{ that maximizes } W_n \qquad (17)$$

Where W_n , denotes summation of weights in the LUT.

To reduce the number of computations in scoring pixels stage, pixels which comprise of the sky region, are removed using a horizon detection scheme Horizon detection is performed at the coarse scale by thresholding pixel intensity values falling within one standard deviations of the mean in the ROI, denoted by $g_{horizon}$ as given in equation 18.

$$g_{horizon}(u,v) = \begin{cases} 0, & \text{if } z_{u,v} \in (-\sigma_x^2,\ \sigma_x^2) \\ 1, & \text{otherwise} \end{cases} \qquad (18)$$

The horizon in this case is defined as the row of pixels in $g_{horizon}$ which have the largest occurrence of zeros, or minimal sum.

Individual pixels in the input image are scored based on the Mahalanobis distance from the learned models, starting from 2*horizon. If it satisfies the criterion in equation 19, that pixel is labeled as navigable.

$$f_{navigable=} \begin{cases} 1, & \text{if } (\mu_L - f_{u,v})(CM_L)^{-1}(\mu_L - f_{u,v})^T \leq 1 \\ 0, & \text{otherwise} \end{cases} \qquad (19)$$

4. Computational Flow and Task Graph analysis

In this section we briefly summarize the computations involved in all the stages of the algorithms and extract the tasks.

An input image of size 480x640 is decimated to 240x320 in the first step of the algorithm. Near range pixels corresponding to Region of Interest (ROI) of size 64x128 in the decimated image is used to pre-filter the image data. In the pre-filtering stage, mean and variance of the ROI is computed. These results and pixel intensity values of the ROI are bit-shifted and a 3D histogram of the bit-shifted ROI is built.

The mean and variance of the ROI is used to detect the horizon in the image as illustrated in Figure 1. Horizon detection is performed to reduce the number of computations in the scoring stage. Variance of the ROI is used to classify each pixel in the decimated image as belonging to sky region or not. The row containing minimum number of pixels belonging to sky region is considered as horizon.

Pre-filtering stage is followed by K-means clustering. In this stage, after initialization the distance between the means of the cluster and the mean of the

ROI are computed. The mean of one cluster is replaced by the mean of the ROI based on the smallest distance measure. The values of the pixels in the histogram are compared with the means of the three clusters in the R, G and B space. For each pixel, a distance measure is computed by summation of the absolute differences from the cluster means. Based on the minimum distance measure, each pixel is assigned to one of the three clusters.

After all the pixels have been assigned, the mean of the clusters are updated. The clusters obtained by this process forms the three color models which are used for terrain learning. For each model, a co-variance matrix (CM) is computed. The next step is building the Look-Up-Table which comprises of mean, co-variance matrix and weights of the color models. There are four stages here: initialization, updation, addition and replacement. At the initialization stage the LUT is empty. The ROI in the image is initialized based on traversability assessment as validated by the map generated by range sensors. The LUT is initialized by populating it with the models obtained for the first image. When the next image is acquired, the corresponding models are generated. These models are used to build the Look-Up-Table.

If a training model satisfies equation 13 then the learned model is updated as given in equations 14 to 16. When a training model does not satisfy equation 13, and yet there is space in the LUT, these models are also added to the LUT to populate the learned models. This stage is known as addition. Models are 'replaced' if equation 13 is not satisfied by a training model and there is no space in the LUT. Weights of the 'learned models' are then compared to find the minimum weight. Weights of the 'training models' are compared to find the maximum weight. These two weights are compared. If the weight of the 'learned model' is lesser than the weight of the 'training model', the training models replace the learned models in the LUT. If a model fails at all stages of validation, that model is 'discarded'.

Scoring of pixels as navigable or not navigable is the next stage. This stage uses the output of horizon detection as shown in Figure 1. In the scoring stage, values of pixels above the horizon are scored as zero, thus decreasing the number of computations required. Each pixel is compared against the models in the LUT using equation 19. If it satisfies the equation, the pixel is said to correspond to a 'navigable terrain'.

The entire computational flow is divided into 7 tasks as shown in Table 1. Its inter-dependencies are illustrated in Figure 2(a). A task graph for a single processor system is shown in Figure 2(b).

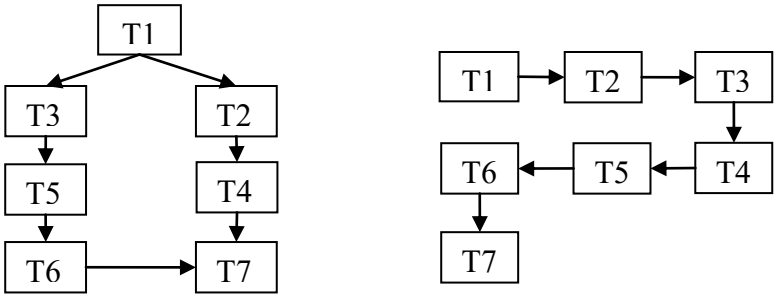

Figure 2. (a) Task graph of LNT (b) task graph for a single processor system

Table 1. Overview of tasks in LNT

Task	**Computation**
T1	Decimation
T2	Computation of mean and variance of ROI
T3	Bit shift and building histogram
T4	Horizon detection
T5	K-means clustering
T6	Creation and updation of Look-Up-Table
T7	Scoring

4.1 Exploiting data and task level parallelism

To exploit data level parallelism, the input image is partitioned as illustrated in Figure 3. In this figure, Level 1 comprises of the input and the decimated images. In Level 2, the input image is partitioned into sub-images and the decimated image is partitioned into ROI and 'd_image'. In Level 3, the 'd_image' is further partitioned as shown in the figure. For exploiting task level parallelism, concurrent tasks are identified which can be executed in parallel. For example, the tasks of K-means clustering and horizon detection can occur in parallel.

The first task of decimation is performed on the input image of size 460x640 on R, G and B space. Mean and variance of the ROI are computed. Next, values of pixels in the ROI are bit-shifted. A three dimensional histogram is built using the bit shifted values. Using the mean and variance of the ROI, horizon detection commences on each of the sub-images (1 to n2) independently and in parallel.

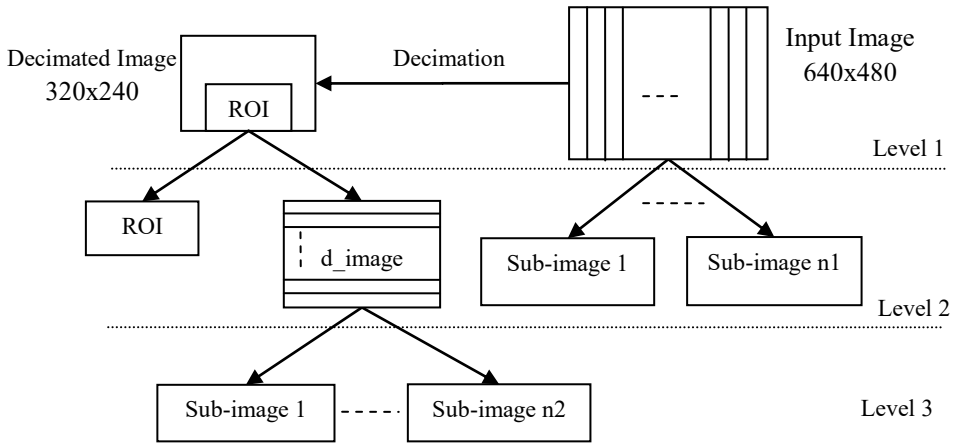

Figure 3. Division of Images for learning navigable terrain

A dedicated task (T^{HD}) is included to consolidate the results obtained from each of the sub-tasks of T4. Horizon detection and K-means clustering commence in parallel. K-means clustering is followed by creation and updation of LUT. Once these tasks are completed, the information in the LUT is used for scoring the pixels. Scoring is done on each of the sub-images (1 to n1) in parallel and a task is assigned (T^{S}) to consolidate the results. The updated task graph is shown in Figure 4.

To accelerate computationally intensive stages of the algorithm, co-processors are designed. The stages targeted and names of the co-processors are as follows:

Gaussian filtering – 'RGBfil'
Computing mean of the ROI – 'RGBM'
Computing variance of the ROI – 'RGBV'
Horizon detection – 'HD'
Assigning of clusters in K-means clustering – 'AC'
Computation of covariance matrix sum– 'CMS'
Scoring –'S'

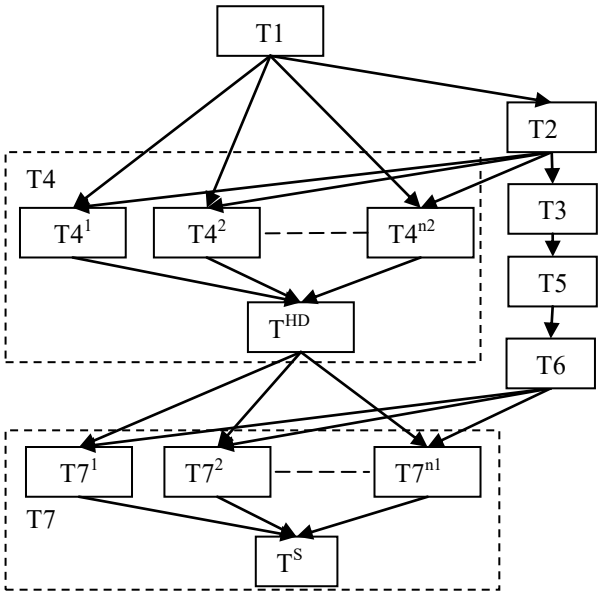

Figure 4.Task graph exploiting task and data level parallelism

An overview of sub-tasks is shown in Table 2 and the updated task graph is shown in Figure 5.

Table 2. An overview of tasks and their sub-tasks

Task	Sub-Tasks	Function
T1	T^{RGBfil}	Gaussian filtering on R, G and B space
T2	$T2^{RGBM}$,$T2^{RGBV}$	Computing mean and variance respectively
T4	T^{HD1} to T^{HDn2}	Horizon detection on the sub-image 1 to n2 respectively
T5	T^{AC},T^{CMS}	Assigning clusters and computing CMS respectively.
T7	T^{S1} to T^{Sn1}	Scoring on sub-images 1 to n1 respectively.

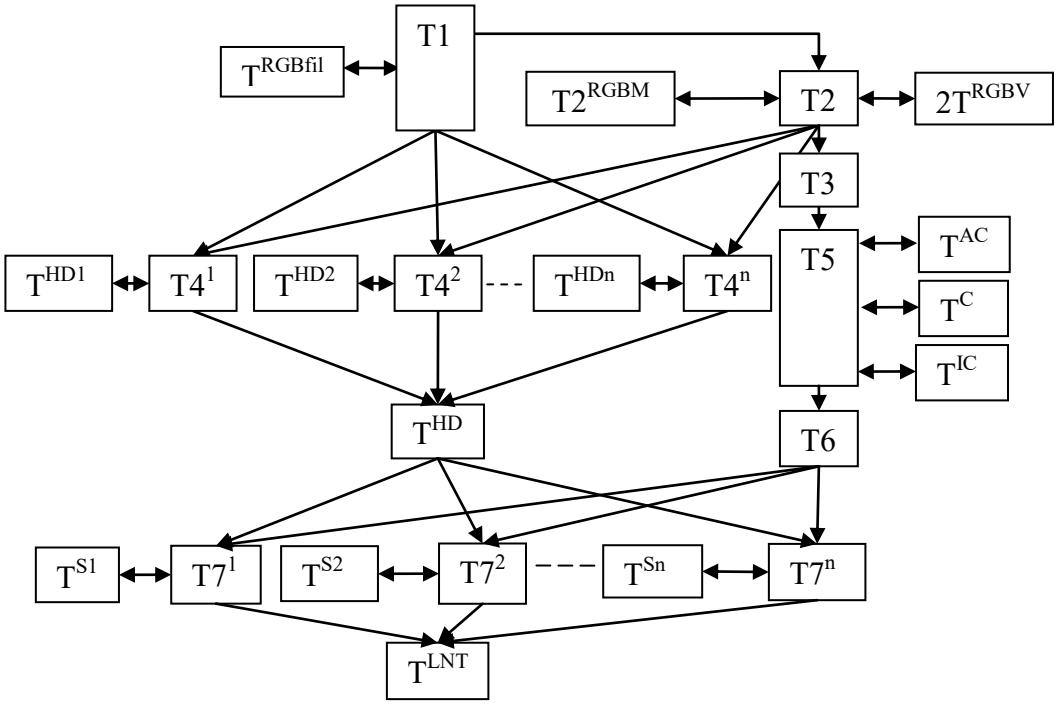

Figure 5. Task graph incorporating sub-tasks for the application of learning navigable terrain

5. Architectures

In this section the three variants of the architecture i.e multi-processor based architecture using soft-core processors, co-processor based architecture and a custom architecture.

5.1 Variant 1 - MPSoC based Architecture

An overview of the multi-processor based architecture for appearance based terrain learning is shown in Figure 6. It includes a data acquisition system to acquire images from the camera and store it in the external memory.

The architecture on the FPGA includes a main Processing Unit (PU0), a data distribution system comprising of AXI Interconnect Busses and a Central Direct Memory Access (CDMA), and a Processing Systems PS containing 3 PUs. The data distribution system uses double buffering mechanism to overcome the

'Readers-Writers' problem between the direct memory access system which feed data from on-board memory to on-chip memory and the Processing Units which process this data.

Each Processing Unit (PU) comprises of a soft core processor (Microblaze) and a local memory (a bank of BRAMs of size 64Kb) connected to it using dedicated Data and Instruction Local Memory Busses [Xilinx]. The Microblaze processor is clocked at 100MHz.

Each Processing System (PS) contains a local memory (M, Ma and Mb each of 32KB) apart from the PUs. The image stored in the external memory is accessed by PU0 and is partitioned and the sub-images are transferred to the internal memories of PS. The number of sub-images chosen here is two. Therefore the number of processing units is three, where one PU is dedicated to computations on ROI and the other PUs are dedicated to processing on sub-images 1 and 2. Two pairs of memories Ma and Mb are dedicated to each of the processing units PU2 and PU3. The partitioned images are stored in Ma and Mb, where as the ROI image is stored in memory M.

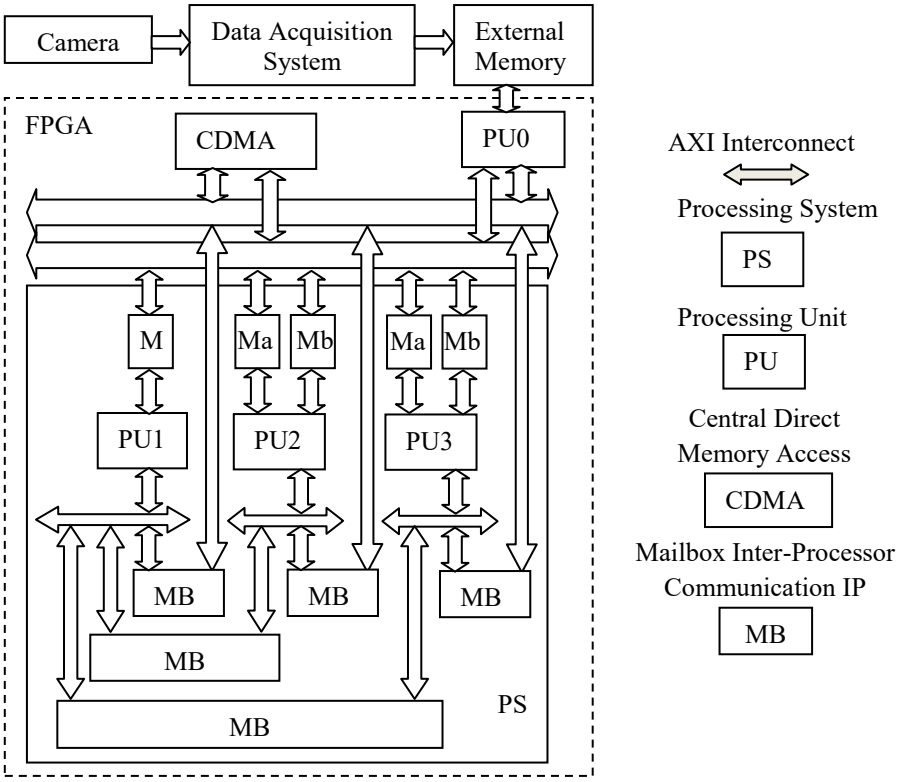

Figure 6. Overview of the architecture for appearance based hybrid mapping. Ba and Bb are of size 4KB and Ma, Mb and M are of size 32KB

An updated task graph of LNT is shown in Figure 7(a). Mapping of the tasks onto the processing units and a schedule is illustrated in Figure 7(b) and Figure 8 respectively. The first task corresponding to accessing the input image stored in the external memory, decimation, partitioning of the input image and sending the sub-images to the local memory of PS commences in PU0. Next, the task of

computing mean and variance of the ROI image commences in PU1. This is followed by bit shift operations and building of the histogram. In parallel to this, horizon detection is implemented in PU2 and PU3. In PU1, K-means clustering and modification of LUT follows the task of construction of histogram. Once the data required to score the pixels is ready, it is sent to PU2 and PU3 where the task of scoring on the sub-images commence.

Task mapping to processing units is performed based on constrained optimization of area and throughput. This provides the basis for selecting the number of processing units in the architecture. Inter-processing unit communication and synchronization is achieved by using a mailbox IP.

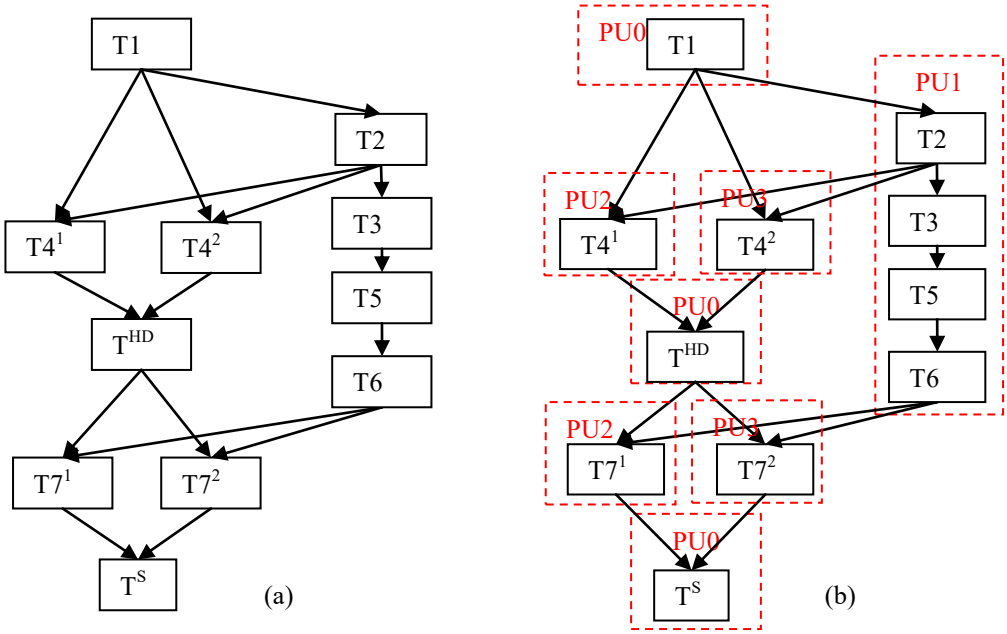

Figure 7. Task graph of LNT when number of sub-images is two

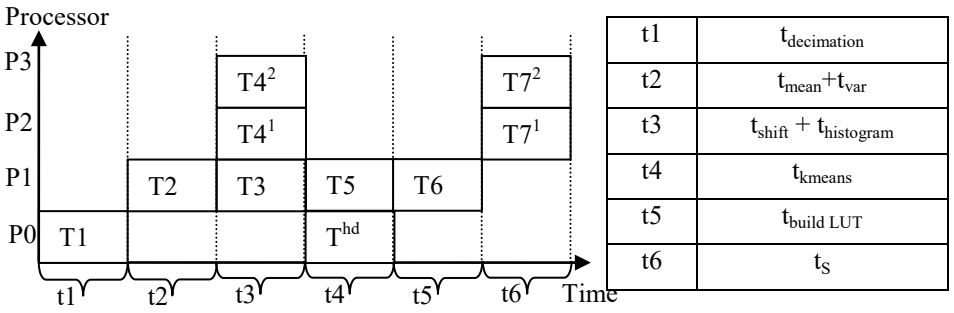

Figure 8. Schedule on the MPSoC based architecture

5.1.1 Methods incorporated to decrease computations

The R, G and B values range from 0 to 31 on the bit-shifted ROI image. For an RGB image, the maximum memory requirement to store the histogram is $32x32x32xn_p$ bits (where n_p is the number of pixels in each bin). The maximum

value that n_p can take is 8192 which takes 14 bit for representation (128x64 pixels in the ROI). The memory requirement for this case is 32x32x32x14 = 4,58,752 bits. The method used here reduces the memory required because the number of bins which have non-zero values cannot exceed 8192, as the ROI image is of size 128x64.

Using this information, two one-dimensional arrays Array-1(H_{T_W}) and Array-2($H_{T_R}, H_{T_G}, H_{T_B}$) are initialized. The first array stores the number of pixels in each bin and is of size 8192. The second array also of size 8192, stores the concatenated values of R, G and B (5 bits for each color). The memory space required using this method is 128x64x14+128x64x15=2, 37,568 bits. The steps in building a histogram using the proposed method is presented below.

1. Compare R, G, B values of the pixel in the ROI image with the pixel values of Array-2.

2. If the same pixel values are found in the array, update the number of pixels in the corresponding index in Array-1.

3. If the pixel values are not found in Array-2, update Array-2 by adding the new pixel values by incrementing the index. Increment the value in the corresponding index in Array-1.

To compute the Covariance Matrix (CM) as described before, the elements in Covariance Matrix Sum (CMS) are computed first. To compute the Inverse of Covariance Matrix (ICM), Adjoint of CMS (ACMS) and Determinant of CMS (DCMS) are computed first. Thus, CM and ICM can be represented as shown in equation 8 where 'W' denotes weight. This is used to simplify operations in the next stages of computations.

$$\mathrm{CM} = \frac{\mathrm{CMS}}{\mathrm{W}} \qquad (21)$$

$$\mathrm{ICM} = \frac{\mathrm{W}*\mathrm{ACMS}}{\mathrm{DCMS}} \qquad (22)$$

As part of the computations in LUT updation, a condition shown in equation 14 has to be satisfied. Equation 23 is updated using 22 and as shown in equation 24.

$$[\mu_{L_R} - \mu_{T_R} \quad \mu_{L_G} - \mu_{T_G} \quad \mu_{L_B} - \mu_{T_B}][\mathrm{CM_L} + \mathrm{CM_T}]^{-1}\begin{bmatrix}\mu_{L_R} - \mu_{T_R}\\ \mu_{L_G} - \mu_{T_G}\\ \mu_{L_B} - \mu_{T_B}\end{bmatrix} \leq 1 \quad (23)$$

$$[\mu_{L_R} - \mu_{T_R} \quad \mu_{L_G} - \mu_{T_G} \quad \mu_{L_B} - \mu_{T_B}](w_L * w_T)[\mathrm{ACMS_L}]\begin{bmatrix}\mu_{L_R} - \mu_{T_R}\\ \mu_{L_G} - \mu_{T_G}\\ \mu_{L_B} - \mu_{T_B}\end{bmatrix} \leq \mathrm{DCMS_L} \quad (24)$$

Where $\mathrm{CMS_L} = \mathrm{CMS_L} * \mathrm{W_T} + \mathrm{CMS_T} * \mathrm{W_L}$

The computations involved in scoring given in 20 are computed using 22 and as shown in equation 25.

$$[f_R - \mu_{L_R} \quad f_G - \mu_{L_G} \quad f_B - \mu_{L_B}](w)[ACMS]\begin{bmatrix} f_R - \mu_{L_R} \\ f_G - \mu_{L_G} \\ f_B - \mu_{L_B} \end{bmatrix} \leq th * DCMS \quad (25)$$

If a covariance matrix sum, weights and means in the LUT (l) have to be updated using new values (t), computations are illustrated in equations (26) – (28)

$$CMS_L = CMS_L + CMS_T \quad (26)$$

$$W_L = W_L + W_T \quad (27)$$

$$\mu_L = \frac{\mu_L W_T + \mu_T W_L}{W_L + W_T} \quad (28)$$

Where CMS and ACMS are given as in equation 29 and 30 respectively

$$CMS = \begin{bmatrix} CMS_{RR} & CMS_{RG} & CMS_{RB} \\ CMS_{RG} & CMS_{GG} & CMS_{GB} \\ CMS_{RB} & CMS_{GB} & CMS_{BB} \end{bmatrix} \quad (29)$$

$$ACMS = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{01} & A_{11} & A_{12} \\ A_{02} & A_{12} & A_{22} \end{bmatrix} \quad (30)$$

Where, subscripts L, T refer to learned models and training models respectively.When the above equations are used division operation is restricted to only updation of means in LUT, as shown in equation 13. All computations follow integer arithmetic. An overview of the computational flow in processing units PU0, PU1, PU2 and PU3 for the application of learning navigable terrain is illustrated in Figures 9 and 10.

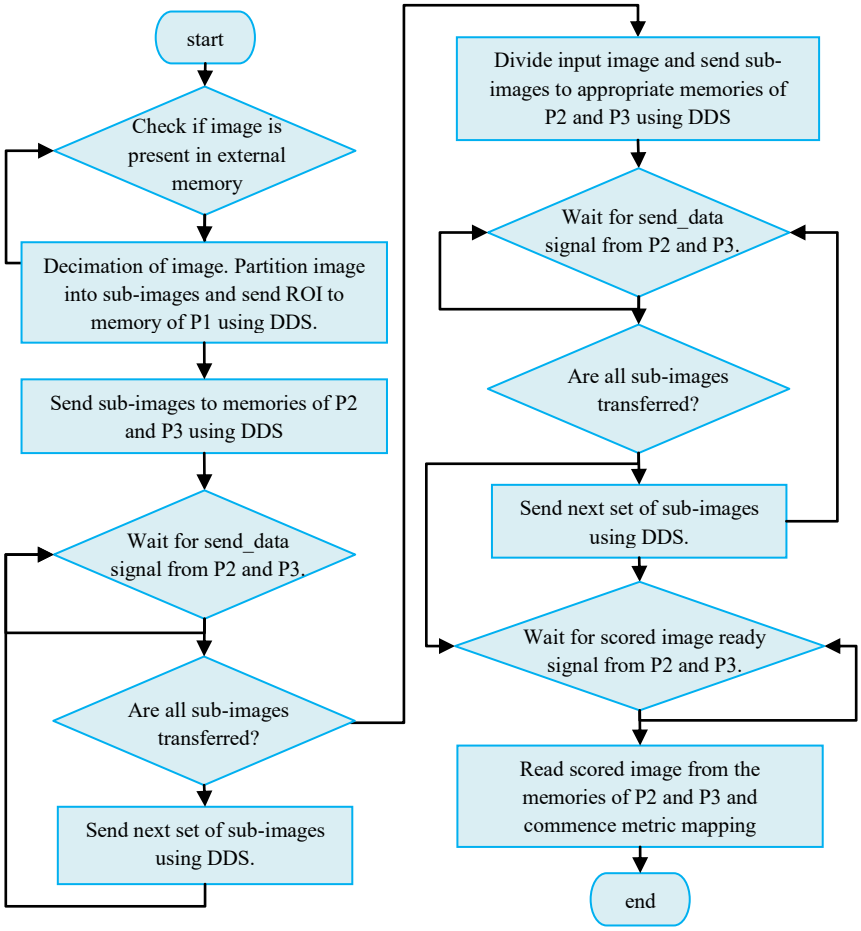

Figure 9. Overview of computational flow in PU0

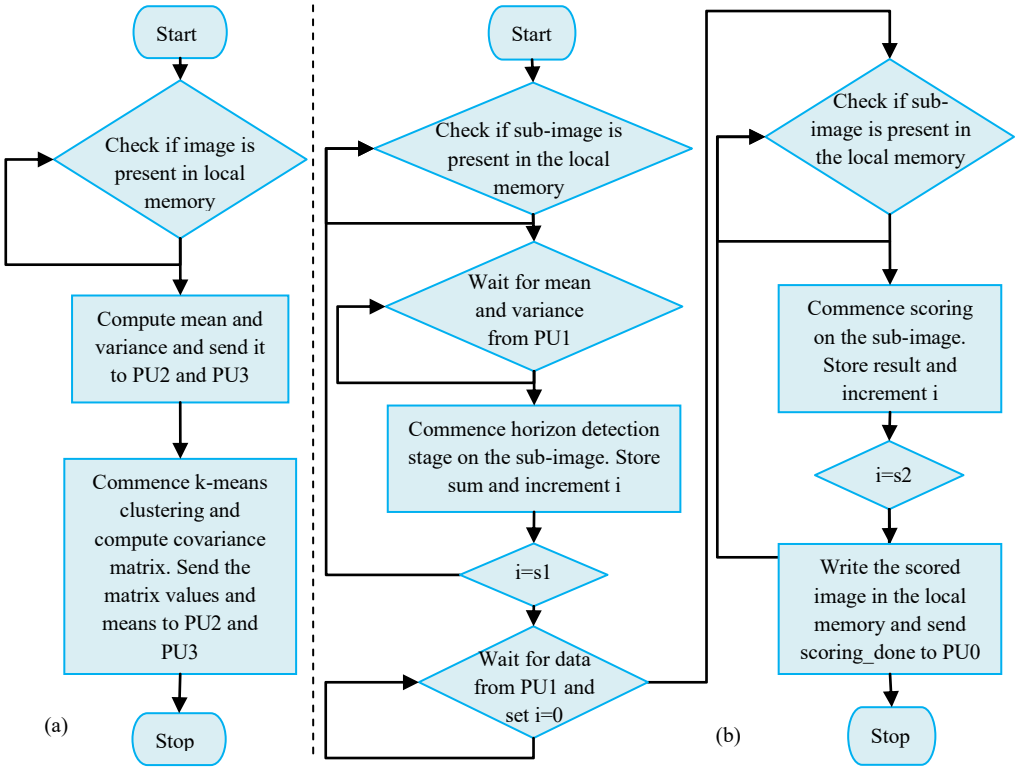

Figure 10. Overview of computational flow in (a) PU1 (b) PU2, PU3

5.2 Variant 2 - Co-processor based architecture

Three and four co-processors are designed to accelerate computations in PU0 and PU1 respectively. Two co-processors each have been designed for PU2 and PU3. The soft-core processors communicate with its co-processors through Fast Simplex Link Busses. Each co-processor comprises of an FSL interface, registers to store inputs, intermediate results and outputs, a Finite State Machine (FSM) which has been designed to sequence the data flow in accordance to the task being executed and one or more DSP48E1 IP slices[DSP48E datasheet] as the main computational engine. The general architecture of the DSP IP slice used to design the Co-processor is illustrated in Figure 11.

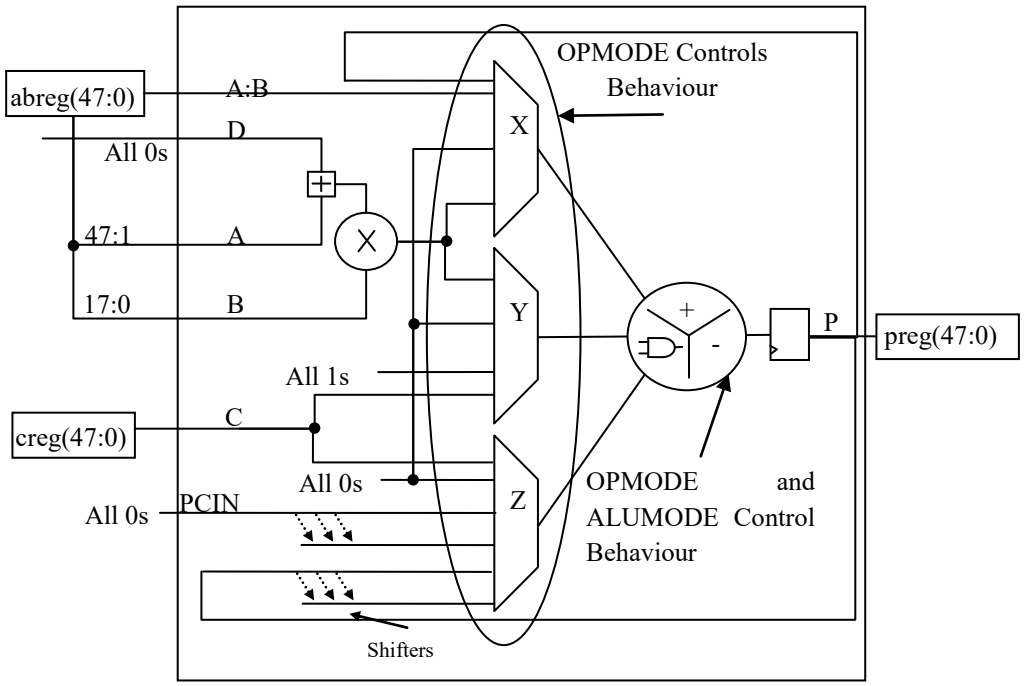

Figure 11. Overview of the architecture of the DSP slice as present in [datasheet]

The arithmetic portion of the DSP48E1 IP slice consists of a 25-bit pre-adder, a 25-bit * 18-bit two's complement multiplier followed by three 48-bit data path multiplexers X, Y, and Z. The multiplexers are controlled by OPMODE signals. This is followed by a three-input add/subtract block or a two-input logic unit which is controlled by the ALUMODE and OPMODE signals. When using the two-input logic unit, the multiplier cannot be used. A brief description of the ports and the values assigned to them when incorporated in the co-processor design is shown in Table 3

Table 3. Overview of ports of the DSP slice

Name	Bit Width	Description	Values
A	[29:0]	Input to the multiplier [24:0] and most significant bits of A:B [29:0]	abreg [47:18]
B	[17:0]	Input to the multiplier [17:0] and least significant bits of A:B	abreg[17:0]
C	[47:0]	Data input	creg

D	[24:0]	Data input to the pre-adder	All 0s
P	[47:0]	Data output	preg
PCIN	[47:0]	Cascaded data input from PCOUT of previous DSP48E1 slice to adder	All 0s

In case of multiplication and addition/subtraction operation, A and B inputs are multiplied and the result is added to, or subtracted from the C register. Selecting the multiplier function consumes both X and Y multiplexer outputs to feed the adder. The two 43-bit partial products from the multiplier are sign extended to 48 bits before being sent to the add/subtract block. When not using the first stage multiplier, the 48-bit, dual input, bit-wise logic function implements AND, OR, NOT, NAND, NOR, XOR, and XNOR. The inputs to these functions are A:B, C, P, or PCIN selected through the X and Z multiplexers, with the Y multiplexer selecting either all 1s or all 0s depending on the logic operation.

One of the attributes of the DSP slice includes a Single Instruction Multiple Data (SIMD) Mode control which selects the mode of operation for the add/subtract block. The attribute setting supports One 48-bit adder mode (ONE48), or two 24-bit adder mode (TWO24), or four 12-bit adder mode (FOUR12) and precludes use of multiplier in first stage.

An example of a co-processor architecture built around a single DSP slice is illustrated in Figure 12. A 'start' signal from the FSL is sent to the controller to signal the start of the operations. An overview of the co-processor based architecture is illustrated in Figure 13.

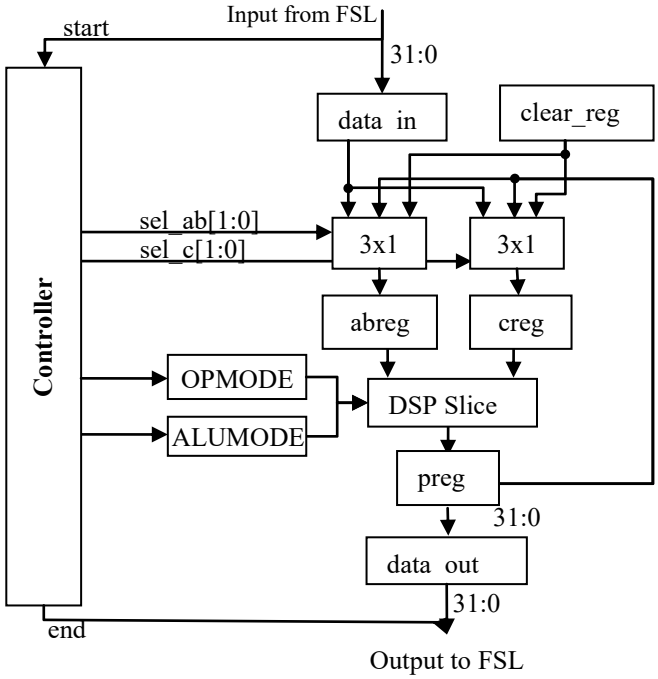

Figure 12. Overview of the co-processor architecture

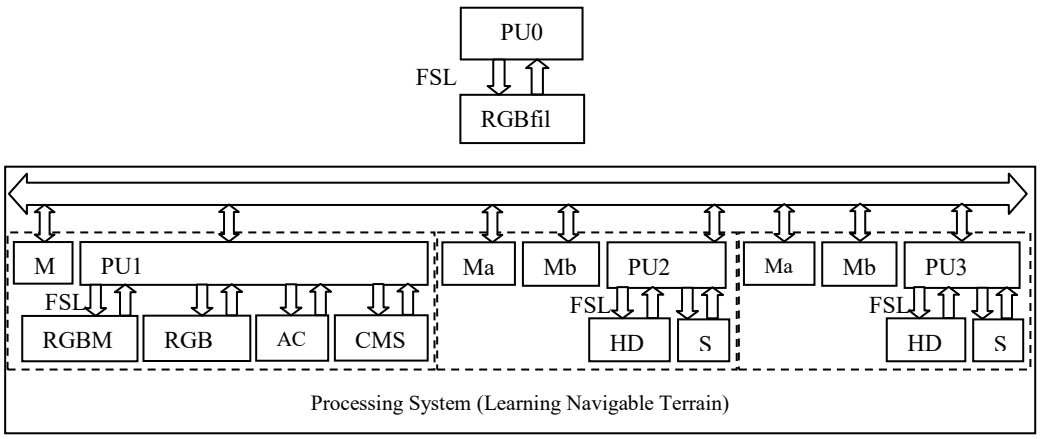

Figure 13. Co-processors of PU0 and in PS

The data from the FSL is stored in a register 'data_in' and is used for further processing in the data path. The data inputs to the DSP slice 'abreg' and 'creg' are selected by two multiplexers. The multiplexers select between data stored in registers 'data_in', 'clear_reg' and 'preg'. The OPMODE and ALUMODE registers are set depending on the nature of computations involved. The output is stored in the register 'data_out' and is sent along with the 'end signal' from the controller to the FSL bus.
One of the attributes of the DSP slice used includes a Single Instruction Multiple Data (SIMD) Mode control which selects the mode of operation for the add/subtract block. The attribute includes a 48-bit adder mode (ONE48), two 24-bit adder mode (TWO24) and four 12-bit adder mode (FOUR12). Depending on the computations, appropriate modes are selected to utilize the DSP slice efficiently.

A description of the co-processors used in this architecture is presented below.

'RGBfil' co-processor: Dedicated to computations involved in filtering using Gaussian kernel. The computations include addition and bit shift operations. This co-processor is built around a single DSP slice.

'RGBM' co-processor: Dedicated to compute mean of R, G and B values of ROI. This co-processor consists of 3 DSP slices, each dedicated to compute Rmean, Gmean and Bmean.

'RGBV' co-processor: Dedicated to compute variance of R, G and B values. This co-processor consists of 4 DSP slices. An overview of computations is shown in Figure 12. One DSP slice is dedicated to perform subtraction operation using SIMD mode of the processor. Other DSP slices are dedicated to perform multiplication and addition operation as shown in the Figure 14.

(Size of ROI = 64X128)

$$\sigma_R^2 = \frac{\sum\left(X_R - \mu_R\right)^2}{64X128}$$

$$\sigma_G^2 = \frac{\sum\left(X_G - \mu_G\right)^2}{64X128}$$

$$\sigma_B^2 = \frac{\sum(X_B - \mu_B)^2}{64X128}$$

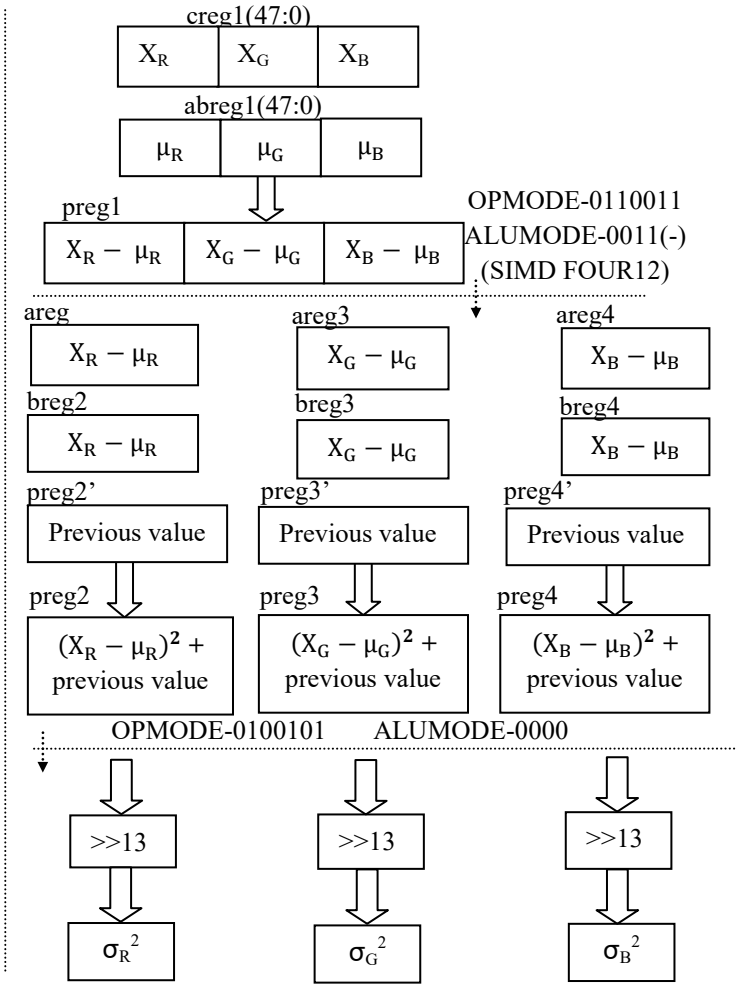

Figure 14. Computational flow of RGBV co-processor

'AC' co-processor: Dedicated to computations involved in assigning clusters (part of K-means clustering). This co-processor consists of 3 DSP slices. An overview of computations is shown in Figure 15.

$$\text{cluster number} = \min\begin{pmatrix}(|H_{T_R} - \mu_{T1_R}| + |H_{T_G} - \mu_{T1_G}| + |H_{T_B} - \mu_{T1_B}|),\\ (|H_{T_R} - \mu_{T2_R}| + |H_{T_G} - \mu_{T2_G}| + |H_{T_B} - \mu_{T2_B}|),\\ (|H_{T_R} - \mu_{T3_R}| + |H_{T_G} - \mu_{T3_G}| + |H_{T_B} - \mu_{T3_B}|)\end{pmatrix}$$

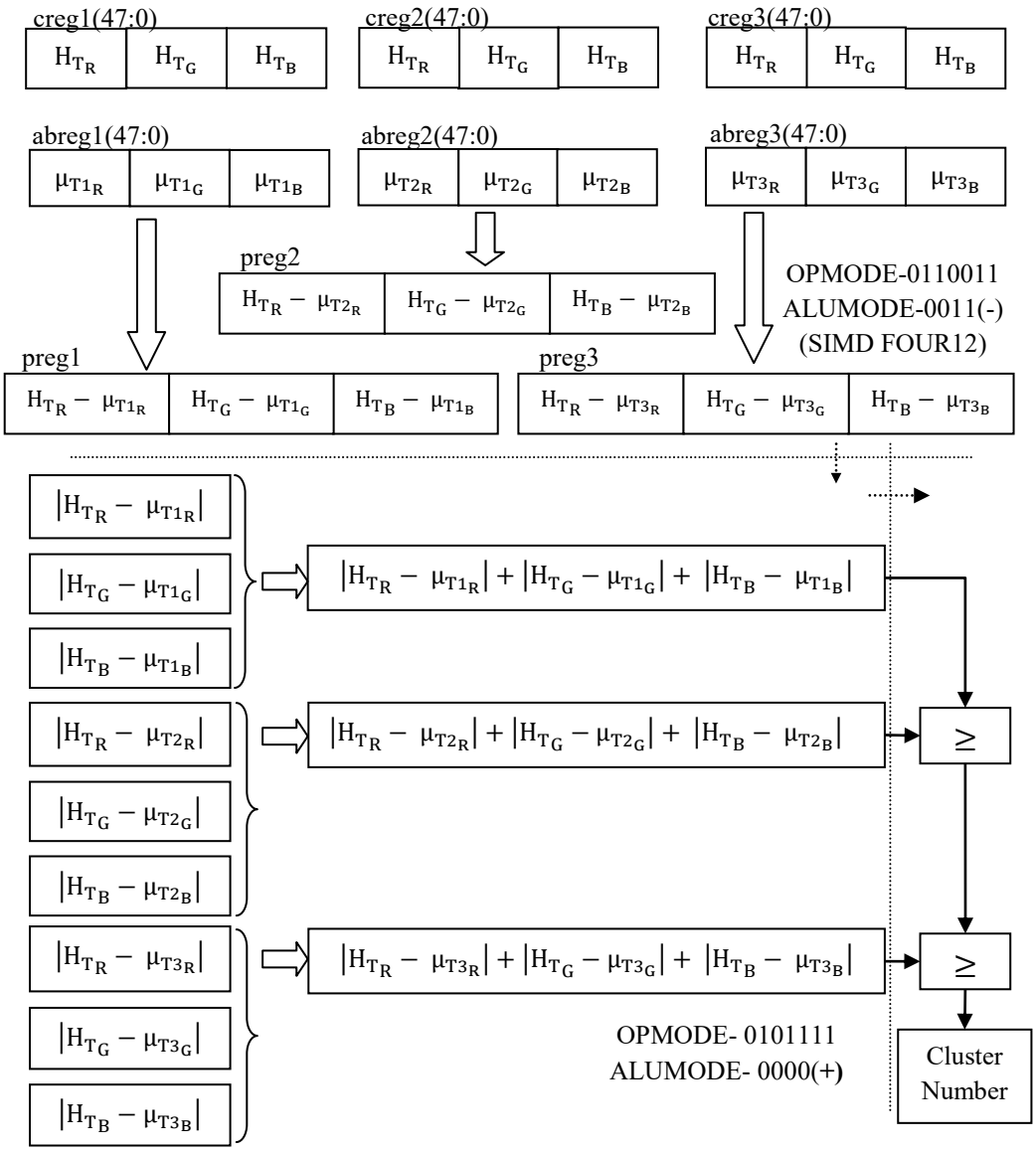

Figure 15. Computational flow in AC co-processor

'CMS' Co-processor: This co-processor consists of 13 DSP slices. One DSP slice is dedicated to perform subtraction operation using SIMD mode of the processor. Two DSP slices are present to compute each of the unique elements in CMS. An overview of the computational flow is shown in Figure 16.

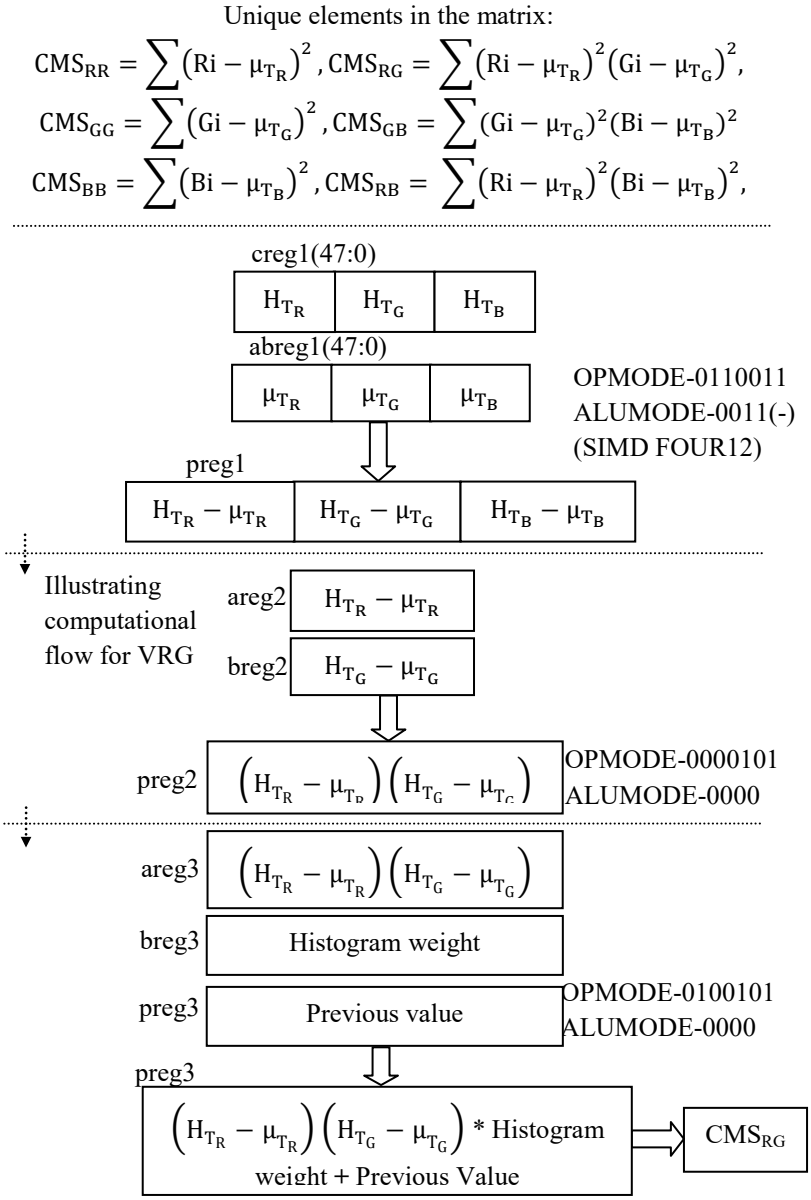

Figure 16. Computational flow in CMS co-processor

'HD' Co-Processor: This co-processor consists of 4 DSP slices. The computational architecture is similar to the 'RGBV' co-processors. Instead of summation of the products, each of the products is compared against variance values using a subtraction operation. The soft-core processor sends the mean and variance values to the co-processor. This is followed by sending R, G and B values of each pixel. The co-processor evaluates the condition and communicates the status to the processor.

'S' Co-Processor: This co-processor is dedicated to perform computations in the scoring stage. This is explained in detail in the next section.

5.3 Variant-3: Architecture using Custom designed data path units for hardware acceleration

An overview of the custom architecture is shown in Figure 17. The architecture comprises of three Custom Processing Elements (CPE1 to CPE3) as shown in the Figure. CPE1 is dedicated to computations on ROI, CPE2 and CPE3 are dedicated to computations of horizon detection and scoring.

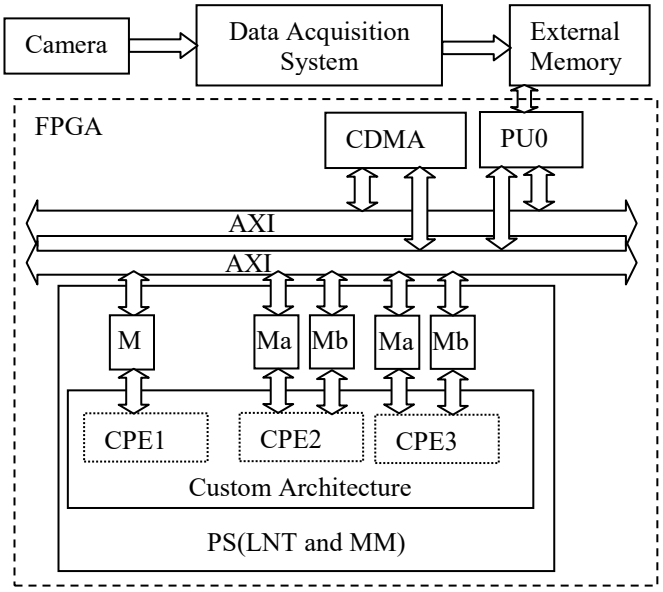

Figure 17. Overview of custom architecture of LNT

The computations involved in each of the custom processing elements and the interdependencies are shown in Figure 18. The custom architecture to compute mean and variance is shown in Figure 19. Three values (R, G and B) each of 8 bit is accessed from memory 'M' to compute mean. Once means are computed (μ_R, μ_G, μ_B), computations involved in variance commence. The mean and variance are bit-shifted thrice before sending to CPE2 and CPE 3.

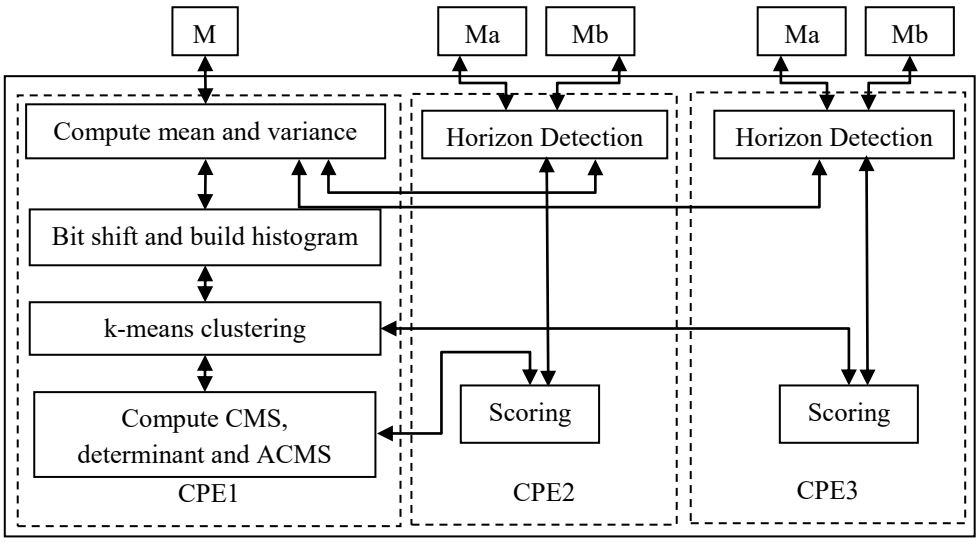

Figure 18. Overview of computations involved in CPE1, CPE2 and CPE3.

An overview of the custom architecture to perform horizon detection on one row is shown in Figure 20(a). Pixel values are accessed from Ma/Mb, where a sub-image of size 30x320 is stored. Five such 'HD' modules are instantiated, such that computations on each of the five rows are performed in parallel as shown in Figure 20(b). When the sum in each row is obtained, subtractions of sums of adjacent rows are done. Next, the results are fed to a comparator to determine the maximum difference. Once all the computations are done on the entire image, the differences are compared once again to determine the maximum difference. The row number of the corresponding difference corresponds to the horizon.

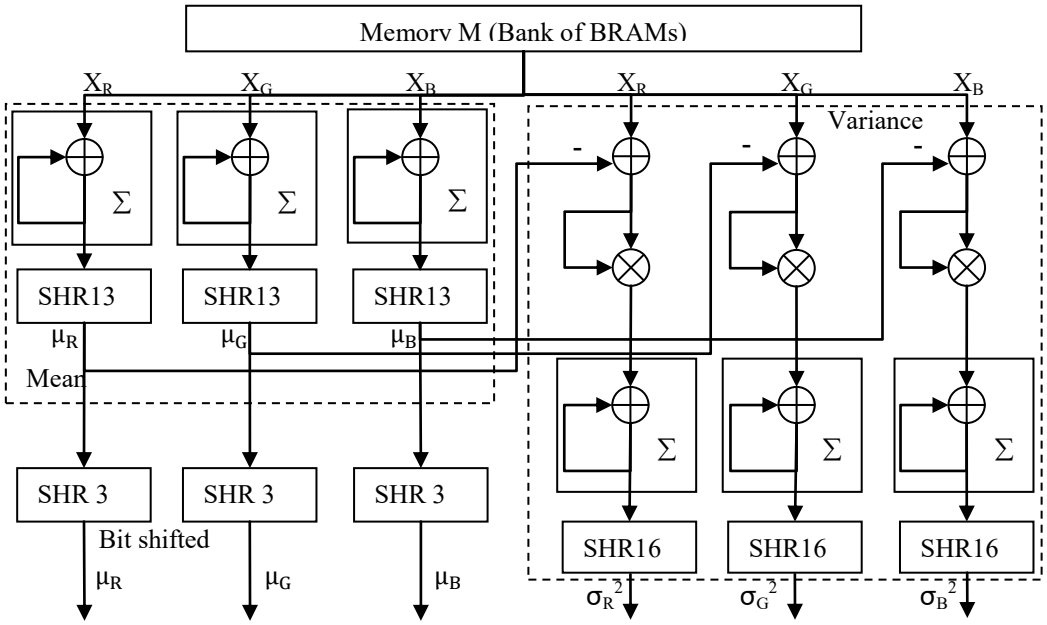

Figure 19. Custom architecture to compute mean and variance

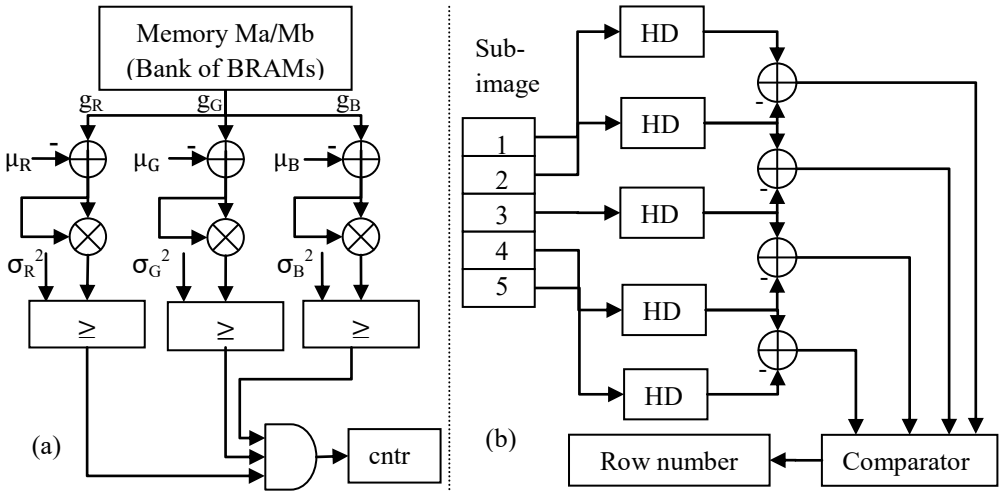

Figure 20. Custom architecture in Horizon Detection stage (a) for a single row (b) for a sub-image

Next a histogram is built using the method described before. This is followed by assigning clusters to each pixel value in the ROI. Once the color models are computed, covariance matrix sum is computed. An overview of the custom architecture to compute elements CMS and ACMS are illustrated in Figures 21 and 22 respectively.

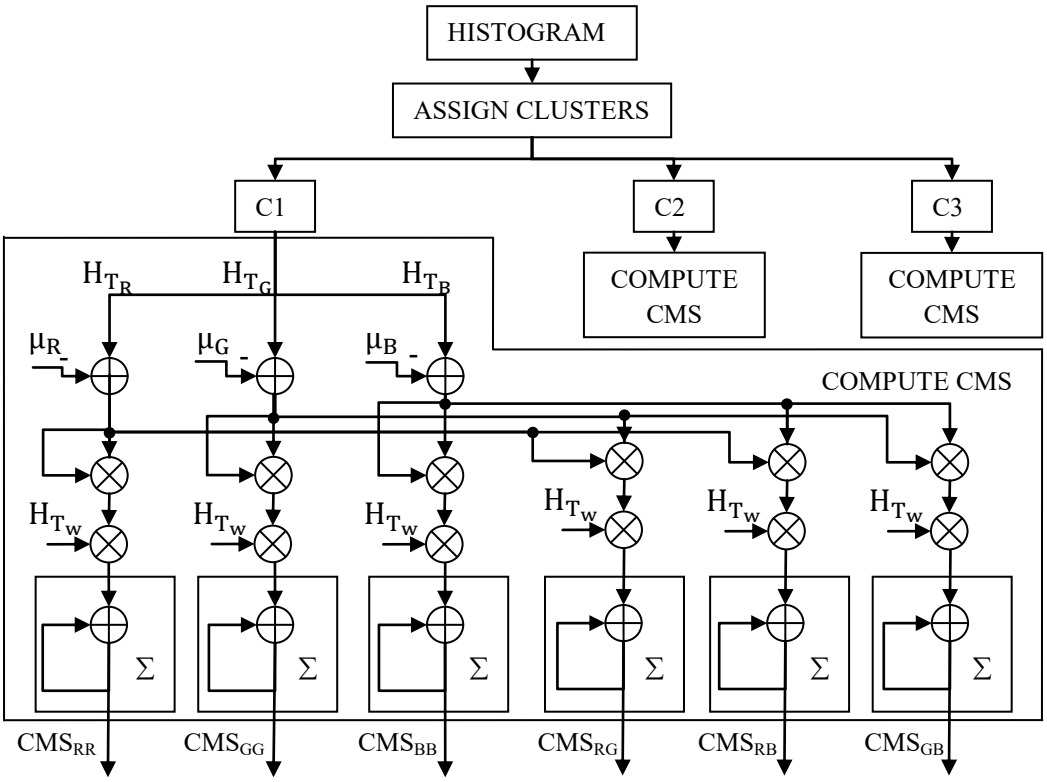

Figure 21. Custom architecture to compute Covariance Matrix Sum (CMS) matrix for each cluster

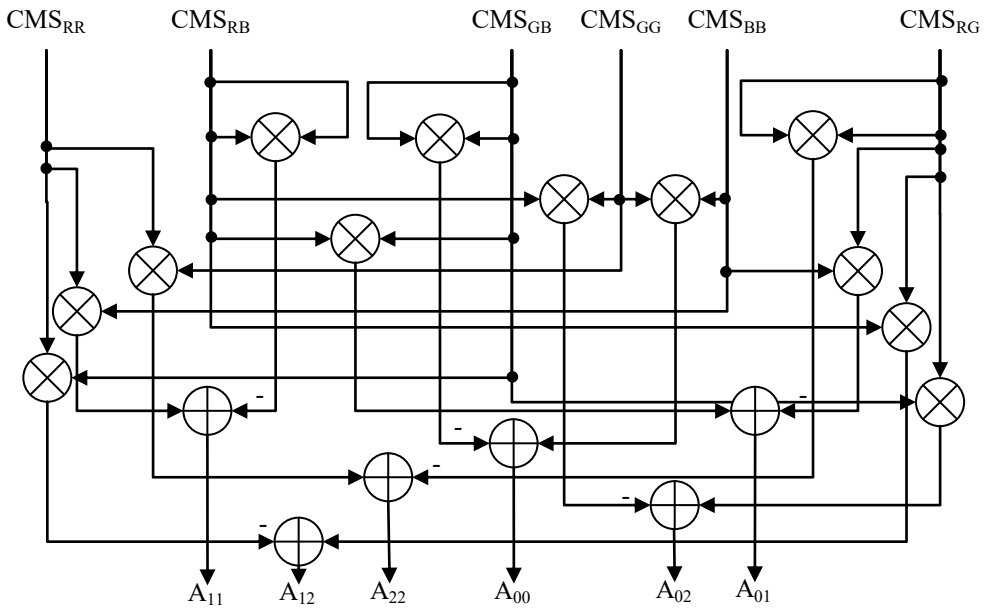

Figure 22. Custom architecture to compute values of ACMS

An overview of the architecture to implement the scoring stage is shown in Figure 23. Scoring is done on a sub-image of size 480x20 which is stored in Ma/Mb at a time. The scoring module is instantiated five times such that scoring on five columns occurs in parallel. The architecture of a co-processor dedicated to scoring includes the architecture shown in Figures 22 and 23.

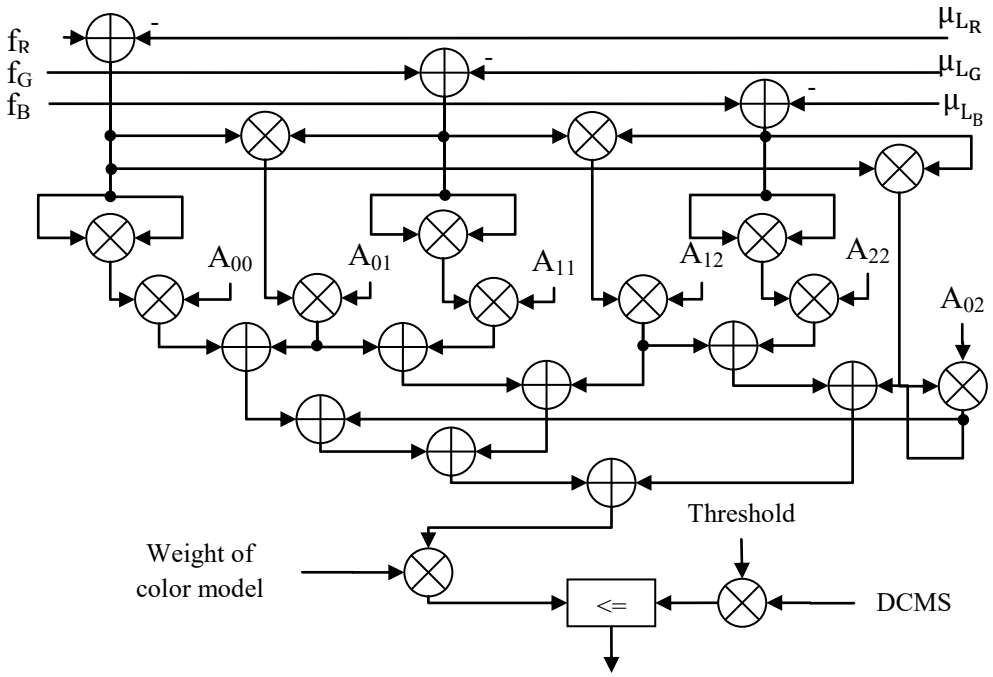

Figure 23. Custom architecture for computations involved in scoring. (IM_R, IM_G, IM_B) – Pixels in the input image.

6. Results and Analysis

The three variants of architectures described till now i.e. multi-processor based architecture (V1) , co-processor based architecture (V2) and custom architecture (V3) have been validated on a Virtex 6 FPGA (XUPV6LX240T, ML605 board). The entire computational flow is divided into ten stages and the computation time overhead of each stage, on the three architectures is presented in this section. The ten stages are:

S1: Decimation
S2: Computing mean and building histogram on the ROI image
S3: Computing variance
S4: Updating initial K-means
S5: K-means clustering
S6: Computing CMS
S7: Computing determinant of CMS and ACMS
S8: LUT Creation
S9: Horizon detection
S10: Scoring

Computation time overhead of S1 and S2 are shown in Figures 24(a) and (b) respectively. The presence of a co-processor contributes to the decrease in time overhead in S1 as shown in Figure 24(a). In the method of histogram building employed here, time overhead increases if depth of the histogram (number of bins which has non-zero values) increases. This is shown in Figure 24(b) for different values of depth. The incorporation of the mean co-processor does not contribute

significantly in the co-processor based architecture because the time taken to build the histogram by the soft-core processor is very high. The time required by the custom architecture is significantly less as seen in Figure 24(b).

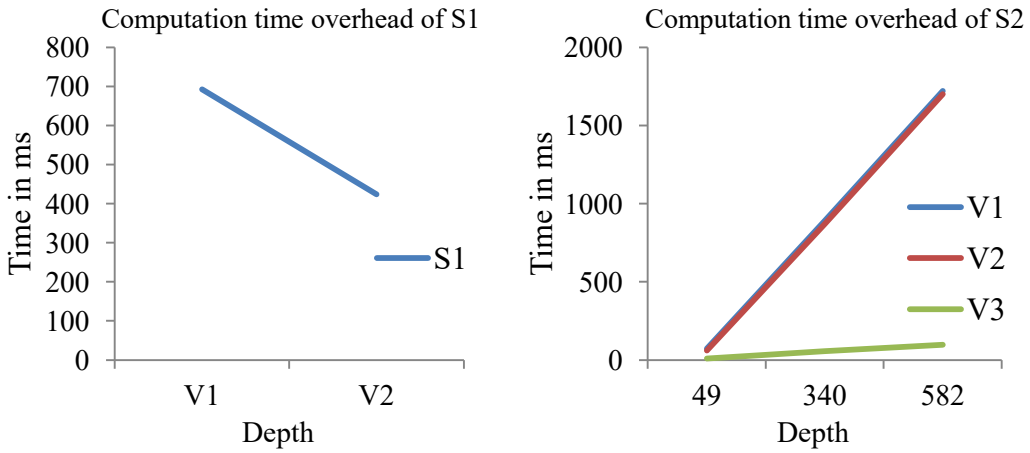

Figure 24. Comparison of computation time overhead of (a) S1 and (b) S2

Computation time overhead of S3 is shown in Figure 25(a). Time overhead on V1 is the highest and overhead on V1 and V2 are comparable. Time overhead of S4 is shown in Figure 25(b). The overhead on V1 and V2 is the same as there is no co-processor in this stage in V2. The overhead on V3 is significantly less as seen Figure 25(b).

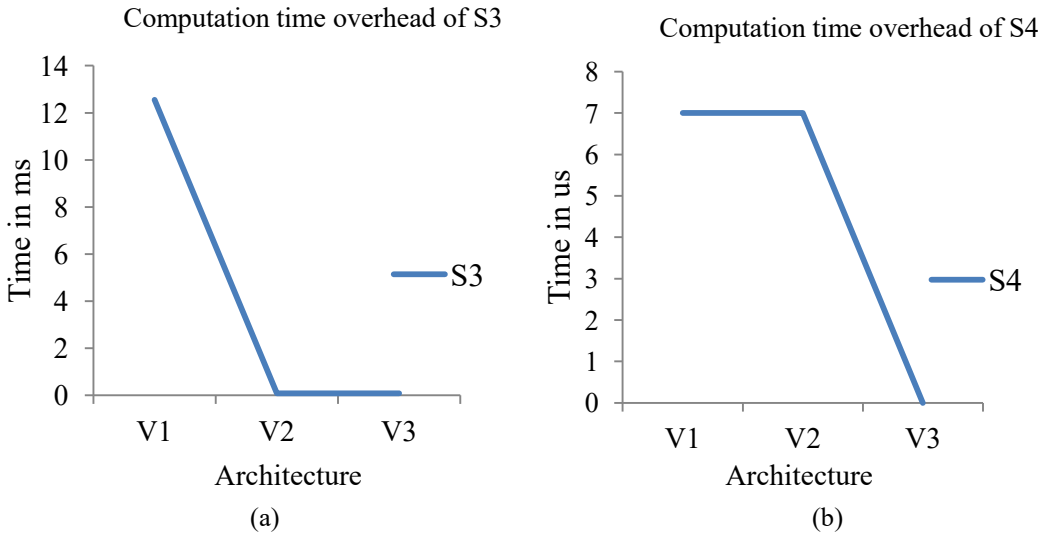

Figure 25. Comparison of computation time overhead of (a) S3 and (b) S4

Computation time overhead of S5 on V1, V2 and V3 for different values of depth is shown in Figure 26(a). The same trend follows here. The overhead on V1 is significantly high and on V3 is the lowest.

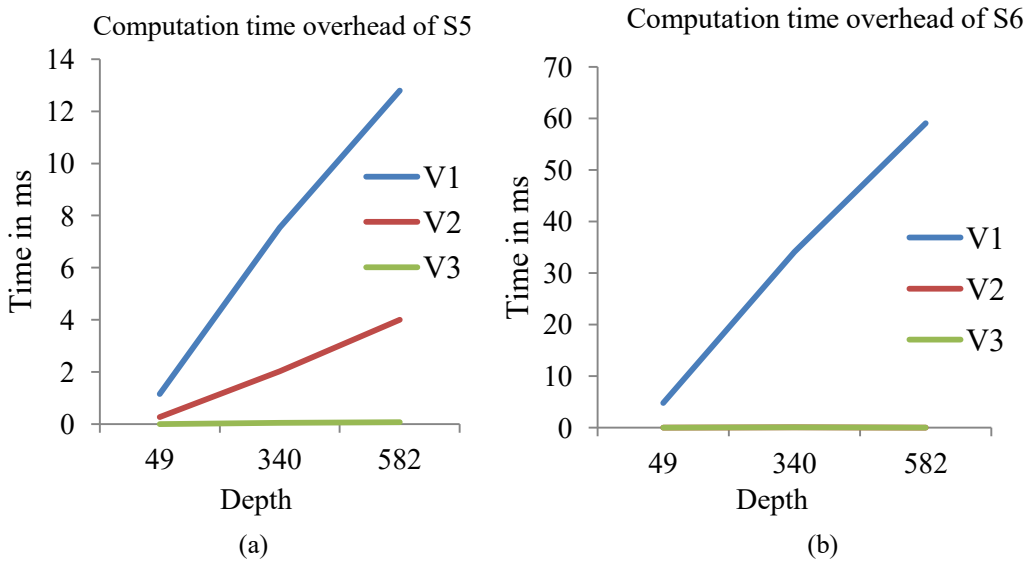

Figure 26. Comparison of computation time overhead of (a) S5 and (b) S6

The overhead of S6 and S7 are shown in Figure 26(b) and Figure 27(a) respectively. Time overhead on V1 is significantly high and overhead on V2 and V3 are comparable as shown in the two figures. Time overhead of S8 is shown in Figure 27(b). The time overhead on S9 and S10 are shown in Figures 28(a) and (b) respectively. It is seen that the overhead on V1 is the highest and the overheads on V2 and V3 are comparable.

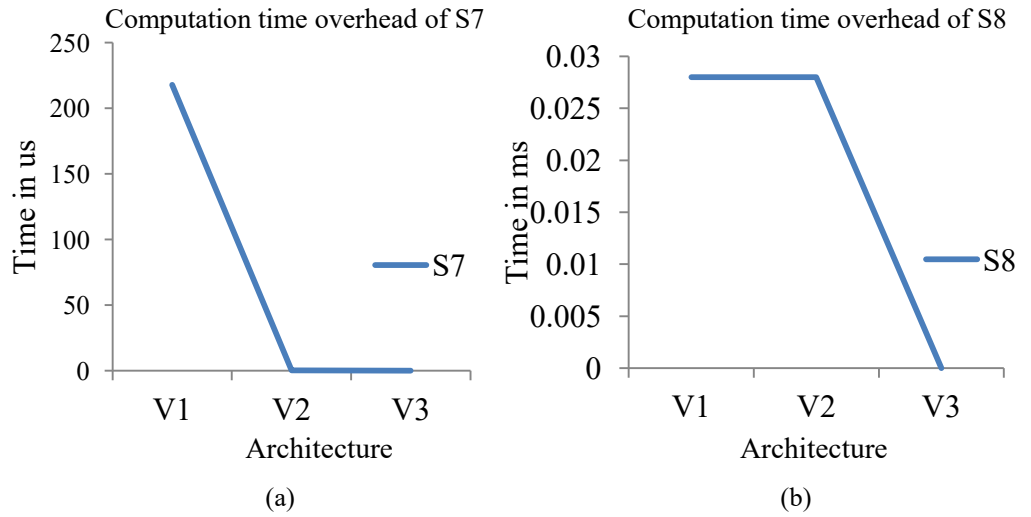

Figure 27. Comparison of computation time overhead of (a) S7 and (b) S8

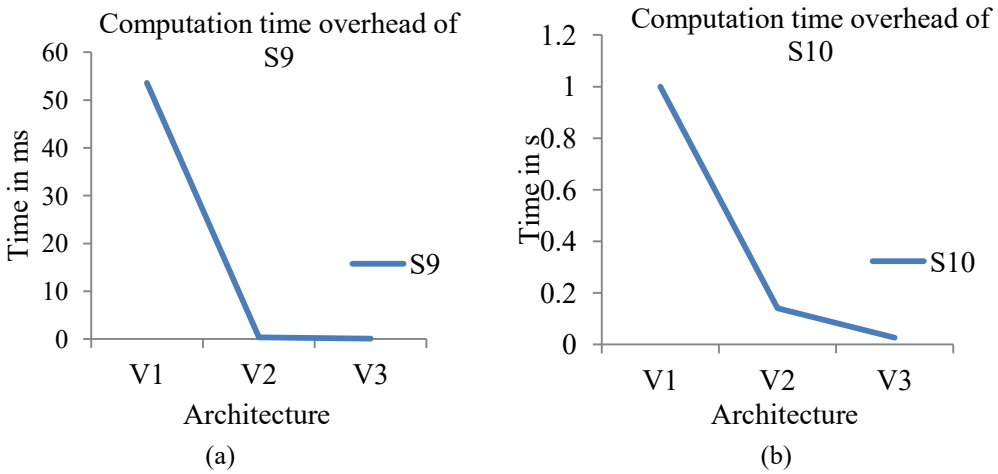

Figure 28. Comparison of computation time overhead of (a) S9 and (b) S10

Resources utilized by the three architectures dedicated to only learning navigable terrain is shown in Figures 29 and 30. The common modules in all the three architectures include PU0, DDS and the local memories of the processing system. Comparison of resources utilized by only PS in all the architectures is shown in Figure 30.

Comparison of consumption of resources which include slice registers, slice LUTs and BRAMs is shown in Figure 29(a). Architecture V2 consumes higher resources than V1 due to the presence of co-processors and V3 consumes the least resources. Comparison of consumption of BRAM and DSP Slices is shown in Figure 29(b). The number of DSP slices consumed by V3 is higher than V1 and V2 because most of the computations in the custom architecture are implemented using DSP slices. Comparison of total resources consumed is shown in Figure 30.

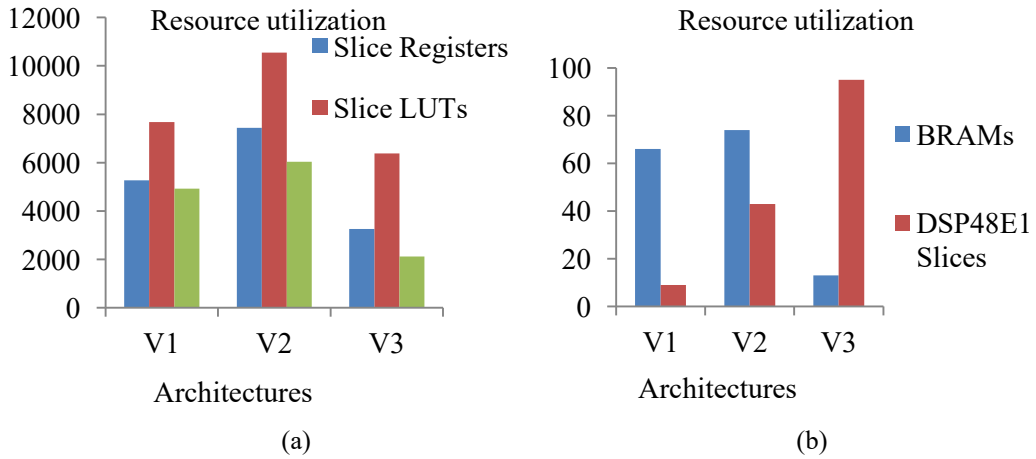

Figure 29. Comparison of resources utilized by PS

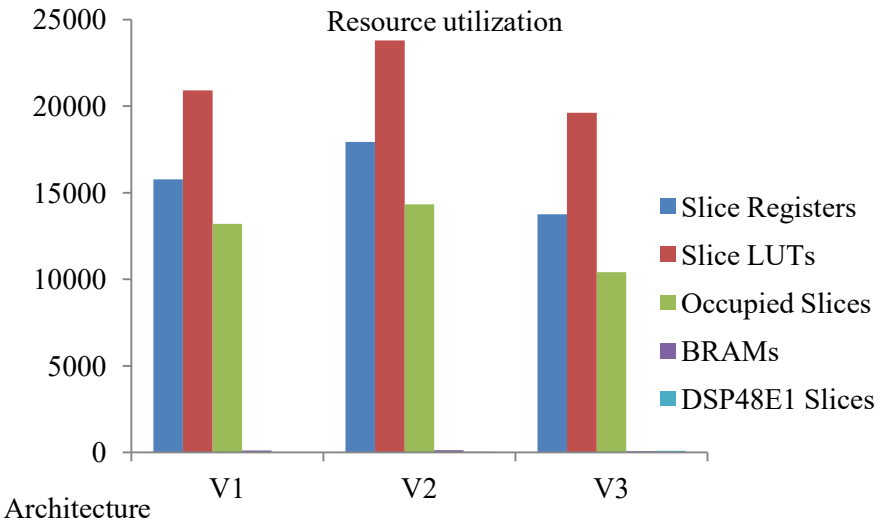

Figure 30. Overview of resource consumption

Dynamic power consumption by the three architectures is shown in Figure 31. The leakage power is 3.451W. Architecture V2 consumes the highest power followed by V1 and V3 as shown in the figure.

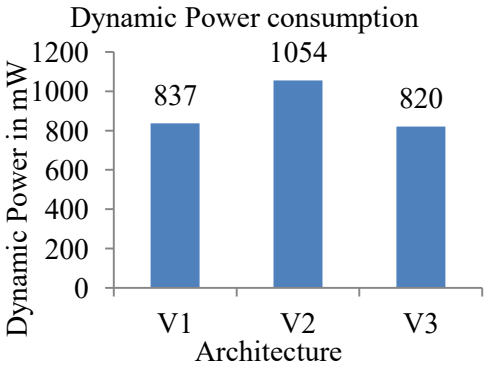

Figure 31. Comparison of dynamic power consumption

7. Conclusion and Future Work

In this paper, we described a computationally inexpensive algorithm to identify and learn navigable terrain around the rover, using a color based clustering approach. The proposed scheme was completely implemented using an integer data-path which employs bit-shifts and accumulate operations. The proposed perception method has significantly faster per frame throughput as compared to the approach detailed in literature. The computation time is reduced by orders of magnitude, by learning dominant terrain color in bit shifted perception sub-space, at the coarse level of an octave pyramid.

We presented three variants of embedded computational architectures which exploit the potential of task and data level parallelism of this algorithm. The methods incorporated to decrease computations in building 3-D histograms and matrix computations were also illustrated. Based on the results obtained, it is seen

that for applications which demand high performance with severe constraints on resources and power, architecture variant three which consists of custom data path is a more feasible option than the other two architectures. For applications demanding high performance with constraints on resources architecture variant two is a feasible option. Depending on the nature of the demands of the application and the resources available the number of sub-image can be increased to achieve higher performance. Our future work comprises of designing embedded computational architectures for the application of appearance based metric map building which uses depth information and information from the scored image obtained in the learning navigable terrain module.

8. References

[1] Angelova, A; Matthies, L.; Helmick, D.; Perona, P., "Fast Terrain Classification Using Variable-Length Representation for Autonomous Navigation," *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on* , vol., no., pp.1,8, 17-22 June 2007. doi: 10.1109/CVPR.2007.383024

[2] Astuti, G., Gaetano Giudice, Domenico Longo, C. Donato Melita, Giovanni Muscato, and Angelo Orlando. "An overview of the "Volcan Project": An UAS for exploration of volcanic environments." *Journal of Intelligent and Robotic systems* 54, no. 1-3 (2009): 471-494.

[3] Bajracharya, Max, Andrew Howard, Larry H. Matthies, Benyang Tang, and Michael Turmon. "Autonomous off-road navigation with end-to-end learning for the LAGR program." *Journal of Field Robotics* 26, no. 1 (2009): 3-25. http://dx.doi.org/10.1002/rob.20269

[4] Bernard, Markus, Konstantin Kondak, Ivan Maza, and Anibal Ollero. "Autonomous transportation and deployment with aerial robots for search and rescue missions." *Journal of Field Robotics* 28, no. 6 (2011): 914-931. http://dx.doi.org/10.1002/rob.20401

[5] Burt, P.J.; Adelson, E.H., "The Laplacian Pyramid as a Compact Image Code," *Communications, IEEE Transactions on* , vol.31, no.4, pp.532,540, Apr 1983. doi: 10.1109/TCOM.1983.1095851

[6] Crane Iii, Carl D., David G. Armstrong Ii, Robert Touchton, Tom Galluzzo, Sanjay Solanki, Jaesang Lee, Daniel Kent et al. "Team CIMAR's NaviGator: an unmanned ground vehicle for the 2005 DARPA grand challenge." In *The 2005 DARPA Grand Challenge*, pp. 311-347. Springer Berlin Heidelberg, 2007. http://dx.doi.org/10.1007/978-3-540-73429-1_10

[7] Dahlkamp, Hendrik, Adrian Kaehler, David Stavens, Sebastian Thrun, and Gary R. Bradski. "Self-supervised Monocular Road Detection in Desert Terrain." In*Robotics: science and systems*. 2006. http://www.robot.cc/papers/dahlkamp.adaptvision06.pdf

[8] David Ratter, "FPGAs on Mars", XcellJournal, Issue 50,Fall 2004,cover story, Pages 8-11

[9] DeSouza, G.N.; Kak, AC., "Vision for mobile robot navigation: a survey," Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.24, no.2, pp.237,267, Feb 2002. doi: 10.1109/34.982903

[10] Dima, C.S.; Vandapel, N.; Hebert, M., "Classifier fusion for outdoor obstacle detection," *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on* , vol.1, no., pp.665,671 Vol.1, 26 April-1 May 2004. doi: 10.1109/ROBOT.2004.1307225

[11] Dongshin Kim; Sang Min Oh; Rehg, J.M., "Traversability classification for UGV navigation: a comparison of patch and superpixel representations," *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on* , vol., no., pp.3166,3173, Oct. 29 2007-Nov. 2 2007. doi: 10.1109/IROS.2007.4399610

[12] Duda, Hart, and Peter Hart. "Stork, Pattern Classification." 2001.

[13] Filitchkin, P.; Byl, K., "Feature-based terrain classification for LittleDog," *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* , vol., no., pp.1387,1392, 7-12 Oct. 2012. doi: 10.1109/IROS.2012.6386042

[14] Goldberg, S.B.; Maimone, M.W.; Matthies, L., "Stereo vision and rover navigation software for planetary exploration," *Aerospace Conference Proceedings, 2002. IEEE* , vol.5, no., pp.5-2025,5-2036 vol.5, 2002. doi: 10.1109/AERO.2002.1035370

[15] Habib, Maki K. "Humanitarian demining: Reality and the challenge of technology-the state of the arts." *International Journal of Advanced Robotic Systems* 4, no. 2 (2007): 151-172. http://cdn.intechopen.com/pdfs-wm/4223.pdf

[16] Hadsell, Raia, Pierre Sermanet, Jan Ben, Ayse Erkan, Marco Scoffier, Koray Kavukcuoglu, Urs Muller, and Yann LeCun. "Learning long-range vision for autonomous off-road driving." *Journal of Field Robotics* 26, no. 2 (2009): 120-144. http://dx.doi.org/10.1002/rob.20276

[17] Hanaa M. Hussain, Khaled Benkrid, Ali Ebrahim, Ahmet T. Erdogan, Huseyin Seker: Novel Dynamic Partial Reconfiguration Implementation of **K-**Means Clustering on FPGAs: Comparative Results with GPPs and GPUs. Int. J. Reconfig. Comp. 2012 Bailey, D. G. (2011) Image Processing, in Design for Embedded Image Processing on FPGAs, John Wiley & Sons (Asia) Pte Ltd, Singapore. doi: 10.1002/9780470828519.ch1

[18] Happold, Michael, Mark Ollis, and Nikolas Johnson. "Enhancing Supervised Terrain Classification with Predictive Unsupervised Learning." In *Robotics: science and systems*. 2006. http://www.roboticsproceedings.org/rss02/p06.pdf

[19] K. Morris, FPGAs in Space, Tech Focus Media, FPGA and Structured ASIC Journal 2004

[20] Khan, Y.N.; Komma, P.; Zell, A, "High resolution visual terrain classification for outdoor robots," *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on* , vol., no., pp.1014,1021, 6-13 Nov. 2011.doi: 0.1109/ICCVW.2011.6130362

[21] Lalonde, Jean-François, Nicolas Vandapel, Daniel F. Huber, and Martial Hebert. "Natural terrain classification using three-dimensional ladar data for ground robot mobility." *Journal of field robotics* 23, no. 10 (2006): 839-861. http://dx.doi.org/10.1002/rob.20134

[22] Larry Matthies et.al, “Computer Vision on Mars”, International Journal of Computer Vision, October 2007, Volume 75, Issue 1, pp 67-92, D.O.I 10.1007/s11263-007-0046-z

[23] Liu, Yugang, and Goldie Nejat. "Robotic urban search and rescue: A survey from the control perspective." *Journal of Intelligent & Robotic Systems* 72, no. 2 (2013): 147-165. http://dx.doi.org/10.1007/s10846-013-9822-x

[24] Maimone, Mark, Jeffrey Biesiadecki, Edward Tunstel, Yang Cheng, and Chris Leger. "Surface navigation and mobility intelligence on the Mars Exploration Rovers." *Intelligence for Space Robotics* (2006): 45-69. http://www-robotics.jpl.nasa.gov/publications/Mark_Maimone/05_Chapter3_final.pdf

[25] Montemerlo, Michael, and Sebastian Thrun. "A multi-resolution pyramid for outdoor robot terrain perception." In *AAAI*, vol. 4, pp. 464-469. 2004. http://www.aaai.org/Papers/AAAI/2004/AAAI04-074.pdf

[26] Mars Exploration Rover Mission: Science. http://mars.jpl.nasa.gov/mer/science/objectives.html

[27] Objectives - Mars Science Laboratory. http://mars.jpl.nasa.gov/msl/mission/science/objectives/

[28] Mishra, P.; Viswanathan, A, "Computationally inexpensive labeling of appearance based navigable terrain for autonomous rovers," *Computational Intelligence in Vehicles and*

Transportation Systems (CIVTS), 2013 IEEE Symposium on , vol., no., pp.87,92, 16-19 April 2013. doi: 10.1109/CIVTS.2013.6612294

[29] Ostler, P.S.; Caffrey, M.P.; Gibelyou, D.S.; Graham, P.S.; Morgan, K.S.; Pratt, B.H.; Quinn, H.M.; Wirthlin, M.J., "SRAM FPGA Reliability Analysis for Harsh Radiation Environments,"*Nuclear Science, IEEE Transactions on* , vol.56, no.6, pp.3519,3526, Dec. 2009. doi: 10.1109/TNS.2009.2033381

[30] Panagiotis Papadakis, Terrain traversability analysis methods for unmanned ground vehicles: A survey, Engineering Applications of Artificial Intelligence, Volume 26, Issue 4, April 2013, Pages 1373-1385, ISSN 0952-1976, http://dx.doi.org/10.1016/j.engappai.2013.01.006.

[31] Singh, S.; Simmons, R.; Smith, T.; Stentz, A; Verma, V.; Yahja, A; Schwehr, K., "Recent progress in local and global traversability for planetary rovers," Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on , vol.2, no., pp.1194,1200 vol.2, 2000. doi: 10.1109/ROBOT.2000.844761

[32] Steve Leibson," Xilinx of Mars: NASA's Curiosity rover shoots Martian eclipse using Smarter Vision", Xcell Daily Blog

[33] Szeliski R, "Computer vision: algorithms and applications." Springer, 2010

[34] Thrun, Sebastian, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong et al. "Stanley: The robot that won the DARPA Grand Challenge." *Journal of field Robotics* 23, no. 9 (2006): 661-692. http://dx.doi.org/10.1002/rob.20147

[35] Urmson, Chris, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan et al. "Autonomous driving in urban environments: Boss and the urban challenge." *Journal of Field Robotics* 25, no. 8 (2008): 425-466. http://dx.doi.org/10.1002/rob.20255

[36] Vandapel, N.; Huber, D.F.; Kapuria, A; Hebert, M., "Natural terrain classification using 3-d ladar data," *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on* , vol.5, no., pp.5117,5122, April 26 2004-May 1 2004. doi: 10.1109/ROBOT.2004.1302529

[37] Vernaza, Paul; Taskar, B.; Lee, D.D., "Online, self-supervised terrain classification via discriminatively trained submodular Markov random fields," *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on* , vol., no., pp.2750,2757, 19-23 May 2008. doi: 10.1109/ROBOT.2008.4543627