Real-Time Horizon Line Detection based on Fusion of Classification and Clustering

Ali Pour Yazdanpanah
Electrical and Computer Engineering Department
University of Nevada, Las Vegas
4505 South Maryland Parkway Las Vegas, NV, 89154

Emma E. Regentova
Electrical and Computer Engineering Department
University of Nevada, Las Vegas
4505 South Maryland Parkway Las Vegas, NV, 89154

Venkatesan Muthukumar
Electrical and Computer Engineering Department
University of Nevada, Las Vegas
4505 South Maryland Parkway Las Vegas, NV, 89154

George Bebis
Computer Science and Engineering Department.
University of Nevada, Reno, 1664 N. Virginia Street, Reno, NV 89557NV, 89557
bebis@cse.unr.edu

ABSTRACT
Horizon line detection is a demanding problem in various tasks associated with planet exploration, because no standard approaches such as global positioning system is available. Horizon line is a boundary line defined between a sky and non-sky region in 2D images, and it is an important visual clue that can be utilized for calculating the rover's position and orientation during planetary missions. The problem of segmenting an image into sky and non-sky regions is classically referred as "horizon line detection". Subsequently, the localization problem can be solved by matching the detected horizon line in 2D images with virtually generated horizon lines from 3D terrain patterns. In this paper, we propose a new real-time horizon line detection (HLD) method by coupling clustering and classifications, as well as implementing the algorithm on the NVIDIA's compute unified device architecture (CUDA). The proposed method has been evaluated on NASA Basalt Hill dataset and on a set of mountainous images that have been collected from the web. The experiments demonstrate high accuracy in determining the horizon line that is proven by provided Receiver Operating Characteristic (ROC) curves.

General Terms
Computer vision, machine learning, CUDA

Keywords
Horizon line detection, skyline extraction, sky segmentation, CUDA, k-means clustering, neural network, fusion.

1. INTRODUCTION
Horizon line detection or sky segmentation is the problem of finding a boundary between sky and non-sky regions in still images or video sequences. This task is of demand in navigating small unmanned aerial vehicles (UAVs) [1, 2, 3] and micro air vehicles (MAVs) [4, 5, 6], visual geo-localization [7, 8], ship detection [9, 10] and outdoor robot/vehicle localization [11, 12, 13]. The horizon line can be detected by (i) modeling sky and non-sky regions using machine learning [1, 2, 4, 5, 6, 9, 22] and (ii) employing edge detection [14, 15]. McGee et al. [2] used SVM classifier trained on color information for detecting sky and non-sky regions. They have used horizon line segmentation for obstacle detection in UAV flight. Their underlying assumption was a linearity of the horizon boundary; this is frequently violated and probably is acceptable only for UAVs navigation. Ettinger et al. [5] proposed a flight stability and control system for micro air vehicles (MAVs) that relies on the horizon line detection. The authors model the sky and non-sky regions by Gaussian distributions and found an optimum boundary between them. Two assumptions made are a) the horizon boundary is linear and b) there are only two regions significantly different in appearance (i.e., sky and non-sky). However, these assumptions might not always be true. The approach of Fefilatyev et al. [9] also assumed a linear horizon boundary and used color and texture features, specifically, mean intensity, entropy, smoothness, uniformity and other features to train various classifiers. Croon et al. [4] extended the features used in [9] by including cornerness, grayness to train a shallow decision tree classifier. The proposed method was able to detect non-linear horizon boundaries, and has been tested in the context of MAVs obstacle avoidance. Todorovic et al. [6] circumvented the assumption of the horizon boundary being linear in [5] by building priors for sky and non-sky regions based on color hue and texture features derived by complex wavelet transform to model the priors. The latter was a required step because of great appearance variations between sky and non-sky regions. A Hidden Markov Tree model has been able to detect non-linear horizon boundaries. Authors of [1] presented a clustering based approach for the horizon line detection for UAV navigation. The main assumption was the presence of a dominant light field between sky and ground regions, which they detect by clustering the intensity information. The authors identified cases when their method requires modifications of clustering process, because the assumption about the light field does not hold in general. Thurrowgood et al. [3] has found an optimum threshold to segment preliminary transformed from RGB space images based on histograms and priors about sky and non-sky regions. Their approach is applicable only to UAV navigation due to the assumption that sky and ground pixels are equiprobable.

Among methods which employ edge detection approach, a most prominent method is that of Lie et al. [14]. The horizon detection is formulated as a graph search problem. The method assumes a consistent edge boundary between sky and non-sky regions. The detected edge map is represented as a multi-stage graph where each column of the image is a stage of the graph and each edge pixel is a node. The shortest path is then found extending from the left-most column to the right-most one using dynamic programming. Since the edge boundaries are not consistent, the method in [14] suggests a gap-filling parameterized approach which makes assumptions that a full horizon line across the image does exist and that it lies in the upper half of the image. The complexity of the algorithm depends on the number of edges detected in an image and the performance can be affected greatly by nearby edges or by edge gaps. Dynamic programming hinders the real-time implementation of this method.

The rest of this paper is organized as follows: Section 2 describes the proposed HLD algorithm. Section 3 presents the parallel implementation of the HLD algorithm and the architecture behind it. Section 4 discusses implementation and presents testing results. Section 5 concludes the work and highlights perspectives of the future work.

2. HLD- HORIZON LINE DETECTION ALGORITHM

The proposed HLD method is based on fusion of results of clustering and classification. The complete algorithm comprises texture feature calculation, classification using a Neural Network (NN), pixel intensity clustering, and fusion of the results of the last two steps followed by a post-processing as depicted in Figure 1, which shows the processing flow. In the next section a detailed description is provided for each step of the proposed method.

2.1 Feature Extraction

The sky and non-sky regions of images are characterized by different textures, and thus texture features are chosen for the classifier input. The gray level co-occurrence probabilities show a second order statistics for calculating the texture features. The co-occurrence matrix (GLCM) introduced by Haralick [16, 17] includes the conditional joint probabilities of all pair-wise mixtures of gray levels given these parameters: inter-pixel distance (d) and inter-pixel orientation (θ). The probability can be introduced as [18]:

$$P_r(x) = \{P_{ij}|(d,\theta)\}$$

Where P_{ij} is defined as:

$$P_{ij} = \frac{H_{ij}}{\sum_{i,j=1}^{G} H_{ij}}.$$

H_{ij} denotes the number of occurrences of gray levels g_i and g_j , and G is the total number of gray levels. The sum in the denominator of P_{ij} denotes the total number of gray level pairs within a window given (d, θ). A different GLCM is required for each (d, θ). Typically only four orientations, i.e., 0, 45, 90 and 135 degrees are used for computation. Various features are extracted from GLCM, such as μ - the mean value of P, μ_x, μ_y - means of P_x, P_y. $P_x(i)$- defined as the i[th] entry obtained by summing the rows of P_{ij}.

$$P_x(i) = \sum_{j=1}^{G} P_{ij} \text{ and } P_y(j) = \sum_{i=1}^{G} P_{ij}$$

$$\mu_x = \sum_{j=1}^{G} i.P_x(i) \text{ and } \mu_y = \sum_{i=1}^{G} j.P_y(j).$$

Hx, Hy are the entropies of P_x, P_y and

$$H = -\sum_i \sum_j P_x(i)P_y(j)log\{P_x(i)P_y(j)\}$$

$$Hxy = -\sum_i \sum_j P_{ij} log\{P_{ij}\}$$

A number of statistical measures can be determined from each GLCM, those used in our method are listed in Table 1. These features are fed to the NN and image is labeled accordingly.

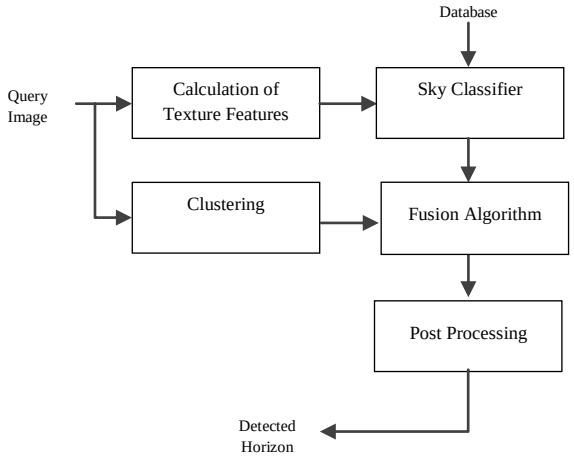

Fig 1: Flow diagram of the proposed HLD method.

2.2 Clustering

The assumption that the horizon boundary is a consistent edge boundary is not always true in practice due to environmental conditions (e.g., clouds) and foreground variations. Clustering partially addresses this problem as a means for obtaining a consistent edge boundary in the clusters boundaries.

Table 1. Texture statistics

Energy	$\sum_{i,j=1}^{G} P_{ij}^2$
Dissimilarity	$\sum_{i,j=1}^{G} P_{ij}\,\lvert i-j\rvert$
Information Measure of Correlation	$(1-e^{-2(H-Hxy)})^{0.5}$
Cluster Shade	$\sum_{i=1}^{G}\sum_{j=1}^{G}(i+j-\mu_x-\mu_y)^3.\ P_{ij}$
Cluster Prominence	$\sum_{i=1}^{G}\sum_{j=1}^{G}(i+j-\mu_x-\mu_y)^4.\ P_{ij}$

K-means clustering partitions n observations into k clusters, where each observation belongs to the cluster with a nearest mean [19]. For images, each pixel is assigned a label of the cluster it belongs to.

In order to have an accurate consistent edge in cluster boundaries, the value of means is supposed to be chosen correctly; this number mostly depends on the resolution of images under processing.

A number of disjoint clusters can be assigned the same label i ($i = 1 \ldots Num$, where Num is a number of clusters), so generally there could be S disjoint regions with the label i,

that is, the output of clustering sharing label i is $R_j = \{R_1, R_2, \dots, R_{s_i}\}$ – disjoint clusters i. In Figure 2, the clustering result with $Num = 10$ is shown for one of the test images.

Fig 2: Top: input test image, bottom: clustering result.

2.3 Classification

The Neural Network (NN) is used for classifying image blocks as belonging to sky or to the rest of the image. The classifier with a single hidden layer of 20 neurons was trained using the gradient descent method. The features were extracted from 9 x 9 non-overlapping blocks from 9 test images selected randomly from our dataset. The classifier was trained using pixel intensities and 5 texture statistics of Table 1.

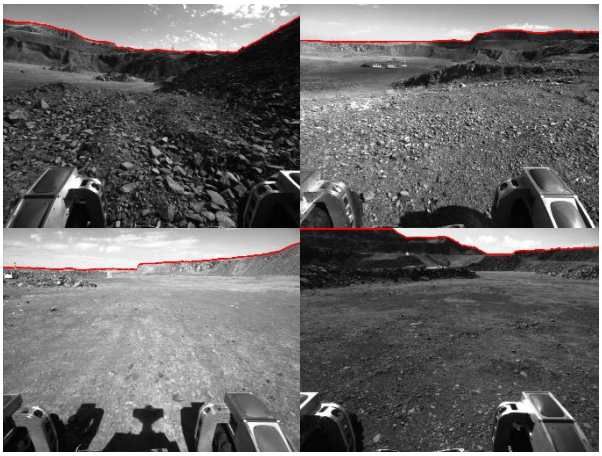

Fig 3: Ground truth (red lines) from Basalt Hill data.

For training, we manually labeled the horizon line in each training image and selected a number of blocks around key points from the sky image above it. Figure 3 shows a few images with ground truth marked by a red line. The same way, negative samples are blocks selected from non-sky regions. The classification result is a binary map of sky/non-sky region. Figure 4 shows the classifier output for an example image.

2.4 Fusion Algorithm and Post-Processing

The fusion algorithm checks the intersection between clusters and a spatially corresponding region that was classified as sky. If the latter is a certain part of the cluster, then the algorithm merges it into the sky, provided that the relative size of the intersection is a greater than a threshold, T_h. We will show how the performance of the method is affected by the choice of the threshold in section 4. Next, all sky regions are merged together to produce the final binary map - F. The following are the parameters defined for implementing the procedure:

Num – Number of clusters. The default value is 10.
F – Final segmentation matrix (initially, zero matrix).
T_h – Tunable threshold (between 0 and 1). The default value is $T_h = 0.5$.
N – Output matrix for classifier stage after post processing.
$N_j = \{N_1, N_2, \dots, N_{s_i}\}$ – Clusters in matrix N paired with/one to one relationship with $\{R_1, R_2, \dots, R_{s_i}\}$ clusters.
$F_j = \{F_1, F_2, \dots, F_{s_i}\}$ – Clusters in matrix F paired with/one to one relationship with $\{R_1, R_2, \dots, R_{s_i}\}$ clusters.
E_j – Total number of pixels in R_j.
P – Total number of pixels in image *x*.

Figure 5 shows the pseudo-code of the fusion procedure.

The post-processing stage is intended to reducing the number of faulty regions without affecting actual edges between sky and non-sky regions. Simple rules defined for this procedure are as follows:

1. There is a minimum size for the whole sky region (The default value depends on the image resolution and is 0.05% of total pixel numbers in image.

2. Any region classified as sky, but surrounded only by non-sky regions should be labeled as non-sky and vice versa.

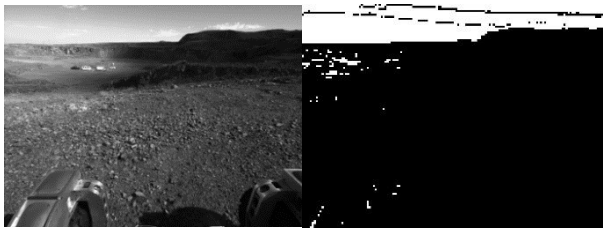

Fig 4: Left: input test image, right: classifier output (binary map).

3. HLD PARALLEL IMPLEMENTATION

We assign each thread process to a single pixel to have all pixels simultaneously processed. The parallel version was implemented using four CUDA kernels: (1) feature extraction, (2) classification, (3) clustering, and (4) fusion. The post-processing stage is run on the CPU. The flow chart is shown in Figure 6, where memory transfers are shown using wide arrows. The kernels running in GPU are invoked by the host. To increase the performance, we have used two general optimization techniques including memory management overhead reduction and the memory transfer overhead reduction [20].

In CUDA, memory allocations (cudaMalloc and cudaFree) are more intensive operations than standard C functions (malloc and free). Therefore we have allocated the GPU memory just once at the beginning and then we accessed and changed that memory in any kernel calls, finally at the end we just brought the results back from the GPU memory to the host just one time. For overhead reduction in memory transfer, we have to avoid unnecessary data transfers between GPU and CPU during execution, therefore we perform most of the computationally expensive procedures in GPU. In the following subsections we describe the CUDA implementation of each stage of the HLD algorithm.

Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_{Num} \in R^n$
Repeat until converge: {
For every k, set
$$c^k := arg\ min_k \left\| x^k - \mu_l \right\|^2$$
For each l, set
$$\mu_l := \frac{\sum_{k=1}^{P} 1\{c^k = l\} x^k}{\sum_{k=1}^{P} 1\{c^k = l\}}$$
}
For $i = 1 \dots Num$ ***{***
For $j = 1 \dots S_i$ ***{***
$R_j \leftarrow 0$ // *for* $\forall$ *pixels in* R_j
if $T = R_j \cup N_j \neq 0 \,\&\, T \geq Th \times E_j$ ***{***
$F_j \leftarrow 1$ // *for* $\forall$ *pixels in* F_j
}
}
}

Fig 5: Pseudo code of fusion algorithm

3.1 Parallel Feature Extraction

Due to the data parallelism of feature calculation, which is calculated per image block, we store block pixel values in 2D on-chip texture memory. Multiple threads then can run in parallel. The feature extraction kernel is implemented on a grid of *W/32* × *H/32* thread blocks, for *H* x *W* images. Before any arithmetic operation, data accessed by each thread block are read first from the global into the texture memory, that is on-chip. For minimizing the repetitive access to off-chip memory, each thread reads neighborhood pixels in 9 by 9 blocks into texture caches and calculates texture statistics.

3.2 Parallel Clustering

Dhillon et al. [23] presented a parallel implementation of k-means clustering on multiprocessors. In the labeling stage, the pixel value set *x* is divided equally between processors. Each processor is assigned to find the labels of the subset of *x*. Authors in [23] showed that the speedup increases with the number of processors have been used in the implementation. The communication cost is defined as a performance metric between the processors. The method shows high efficiency and simplicity so we have adopted this implementation.

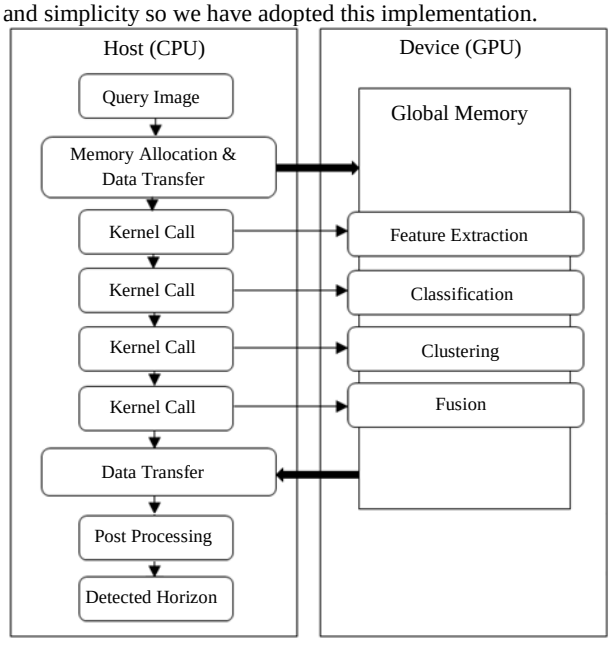

Fig 6: Flow chart of CUDA-based HLD algorithm.

3.3 Parallel Fusion

To parallelize the fusion procedure, we have used *S* disjoint blocks corresponding to *S* disjoint regions for each cluster, if more than one region is defined per cluster. We have assigned a block to a region and pixels inside the regions to threads inside the blocks. Figure 7 provides the CUDA-based pseudo code.

3.4 Parallel Classification

In classification stage we assign a blockID for each feature and each threadID for each neuron on a feature vector. Every feature vector is handled by a block and each pixel in it is handled by a thread. The weights are used as an array for each network layer. There is a CUDA kernel handling the computation of neuron values of that layer.

4. EXPERIMENTS AND RESULTS

4.1 HLD Evaluation

We have experimented with two different data sets: NASA Basalt Hill data and Web data set. . The Basalt Hill data set consists of 45 grayscale images (1038x1388 pixels) chosen from a larger data set based on a field study for rovers taken by two Hazcams (hazard avoidance cameras) cameras on NASA's rovers. Hazcams are photographic cameras sensitive to the visible light. They have a wide field of view (approximately 120° both horizontally and vertically) to allow a large amount of terrain to be visible. They are mounted on the front and rear of NASA's rovers. These images are used by the rover to autonomously navigate hazards terrains. The Web data set consists of 80 mountainous images (519x1388 pixels) that have been randomly collected from the web. This data set includes various viewpoints, geographical and seasonal variations. The classifier has been trained on basalt Hill data and applied to WEB data in our experiments described in the following section.

For $i = 1 \dots Num$ ***{***
// *we have used* S_i *different blocks*
if $threadId = 0$ ***{***
$R_j \leftarrow 0$ // *for all j*
}
synchronize threads
if $T = R_j \cup N_j \neq 0 \,\&\, T \geq Th \times E_j$ // *for all j* ***{***
$F_j \leftarrow 0$ // *for all j* ***}***
synchronize threads
}

Fig 7: CUDA Pseudo code for fusion stage.

In this section, we provide a comparison between the proposed method and the approach of Lie et. al [14] that outperforms all previous methods. In each case, the detected and true horizon lines are compared by calculating a pixel-wise absolute distances S as shown below.

$$S = \frac{1}{N} \sum_{j=1}^{N} \left| L_{d(j)} - L_{g(j)} \right|$$

where $L_{d(j)}$ and $L_{g(j)}$ are row indices of the detected and true horizon pixels in column j, and N is the number of columns in the test image. For each column, the absolute distance between the detected and ground truth pixels is computed and summed over the entire number of columns in the image.

Hence, there is a one-to-one correspondence between the pixel locations in the true and detected horizon pixel locations.

For each method, we compute the average and the standard deviations over all images in the data set. As it can be concluded from Table 2, the proposed HLD its counterpart.

Table 2. Average absolute errors for methods under comparison

Approach	NASA Basalt Hill		Web	
	Mean	STD. Dev.	Mean	STD. Dev.
Lie et al.	5.5548	9.4599	9.1500	17.9195
Proposed	**1.6077**	**2.8493**	**4.2964**	**6.1581**

The proposed method is evaluated by calculating: true positive (TP), false negative (FN), false positive (FP), and true negative (TN) rates. The TPR and FPR rates for M images in the database are calculated as follows:

$$TP_{total} = \sum_{k=1}^{M}(TP)_k \quad FN_{total} = \sum_{k=1}^{M}(FN)_k$$

$$FP_{total} = \sum_{k=1}^{M}(FP)_k \quad TN_{total} = \sum_{k=1}^{M}(TN)_k$$

$$TPR = \frac{TP_{total}}{TP_{total}+FN_{total}} \quad FPR = \frac{FP_{total}}{FP_{total}+TN_{total}}$$

The ROC curves for NASA Basalt Hill data are shown in Figure 8. As it can be observed, the proposed HLD method shows high performance. TPR=0.9956 and FPR=0.0282 are the max TPR and corresponding FPR obtained for NASA Basalt Hill dataset. Also, the effect of the number of clusters, k on the accuracy can be observed in Figure 9. A highest performance is attained for k = 10, so this value is set for both experiments. Figure 10 shows the ROC for Web data that are normalized and histogram-wise equalized. We used for the experiment same parameters applied towards NASA Basalt Hill data.

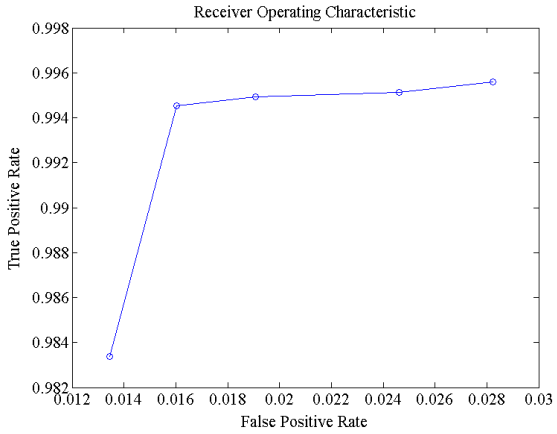

Fig 8: ROC curve obtained by testing the HLD on the NASA Basalt Hill data set.

Figure 11, displays several examples from each data set where the HLD method has successfully detected the horizon line. The complete real-time result for video sequences can be accessed at [21]. Figure 12 shows examples where our method fails to detect certain portions of the horizon line. The main reason for these failures is a small set of training images from Basalt Hill data (i.e., only 9 images). Adding more data to the training set including web images would improve the detection accuracy.

4.2 Parallel Implementation Result

In this section, we present the performance evaluation of the HLD algorithm on both CPU and GPU. As for the CPU implementation, the algorithm is tested using single-threaded implementation in both Matlab and C++. Both CPU and CUDA implementations of the algorithm run on a machine with 4 Gigabyte RAM, Intel core 2 Duo E8400 processor and with the NVIDIA Tesla C2050 GPU board with 448 CUDA cores. We compare the execution times of the Matlab, C++, and CUDA versions of the HLD method for two data sets in Table 3 to illustrate the accelerating performance of CUDA. Tabulated execution times of CPU and CUDA implementations are average values over ten executions. The speedups achieved with the proposed CUDA implementation range from ~9x to ~70x, as illustrated in Table 3. According to the experimental results in Table 3, we observe that the parallel algorithm on the GPU through the CUDA environment improved the performance significantly compared to the serial single-threaded algorithm running on the CPU.

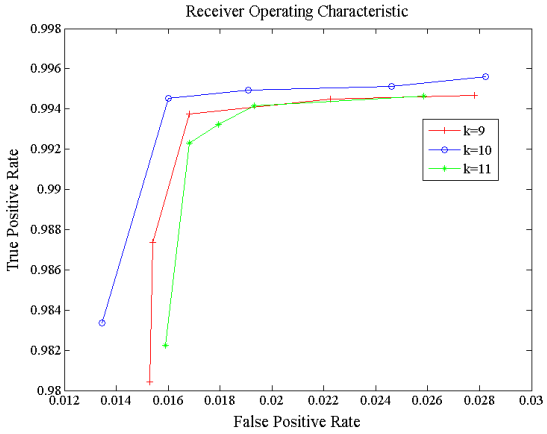

Fig 9: ROCs with three different values of *NUM* for NASA Basalt Hill data set

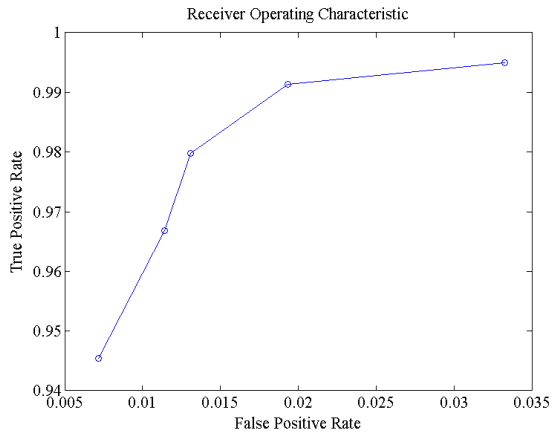

Fig 10: ROC curve obtained by testing the HLD on the Web data set

For the input image of 1038 x 1388 pixels of Basalt Hill data set, the speed up is up to 70 times compared to Matlab implementation and up to 10 times compared to C++ implementation, and for the input image of 513 x 1388 pixels from Web data set, the speed up is up to 53 times compared to Matlab implementation and up to 9 times compared to C++ implementation.

5. CONCLUSION

We have designed a real-time horizon line detection (HLD) method based on fusion of clustering and classification. We consider local texture features for training sky/non-sky classifier. Further processing involves fusion of the classifier outcome with the clustering result. Also we develop the parallel implementation of the algorithm for CUDA to meet the requirement for real-time horizon detection in video sequences. The proposed method has been evaluated on NASA Basalt Hill data set and a set of images that have been collected from the web. Our experimental results show considerable accuracy in determining the horizon line location in two challenging data sets. For the future work, we intend to explore various other classification features and more powerful classifiers to reduce the post-processing step of the algorithm and also consider the horizon line as a localization cue in planetary environment.

Table 3. Average execution times for each frame in both data sets using Matlab, C++, and CUDA implementation

Implementation	NASA Basalt Hill	Web
CUDA	~1014 (ms)	~549 (ms)
CPU (C++)	~10123 (ms)	~4627 (ms)
CPU (Matlab)	~70000 (ms)	~29000 (ms)

6. ACKNOWLEDGMENTS

This research was supported by NASA EPSCoR under cooperative agreement No. NNX10AR89A.

7. REFERENCES

[1] Boroujeni N. S., Etemad S. A., Whitehead A. 2012 Robust Horizon Detection Using Segmentation for UAV Applications. Proceedings of IEEE Ninth Conference on Computer and Robot Vision.

[2] McGee T. G., Sengupta R., and Hedrick K. 2005 Obstacle Detection for Small Autonomous Aircraft Using Sky Segmentation. Proceedings of International Conference on Robotics and Automation (ICRA).

[3] Thurrowgood S., Soccol D., Moore R. J. D., Bland D., and Srinivasan M. V. 2009 A Vision Based System for Altitude Estimation of UAVs. Proceedings of International Conference on Intelligent Robots and Systems (IEEE/RSJ).

[4] De Croon G. C. H. E., Remes B. D. W., DeWagter C., and Ruijsink R. 2011 Sky Segmentation Approach to Obstacle Avoidance. IEEE Aerospace Conference.

[5] Ettinger S. M., Nechyba M. C., Ifju P. G., and Waszak M. 2002 Vision- Guided Flight Stability and Control for Micro Air Vehicles. Proceedings of International Conference on Intelligent Robots and Systems (IEEE/RSJ).

[6] Todorovic S., Nechyba M. C., Ifju P. G. 2003 Sky/Ground Modeling for Autonomous MAV Flight. Proceedings of International Conference on Robotics and Automation (ICRA).

[7] Baatz G., Saurer O., Koser K., and Pollefeys M. 2012 Large Scale Visual Geo-Localization of Images in Mountainous Terrain Proceedings of European Conference on Computer Vision.

[8] Liu W. and Su C. 2014 Automatic Peak Recognition for Mountain Images Advanced Technologies, Embedded and Multimedia for Human-centric Computing.

[9] Fefilatyev S., Smarodzinava V., Hall L. O., Goldgof D. B. 2006 Horizon Detection Using Machine Learning Techniques. International Conference on Machine Learning and Applications, 17-21.

[10] Gershikov E., Libe T., Kosolapov S.: Horizon Line Detection in Marine Images. 2013 Which Method to Choose? International Journal on Advances in Intelligent Systems, 6(1-2): 79 - 88.

[11] Gupta V. and Brennan S. 2008 Terrain Based Vehicle Orientation Estimation Combining Vision and Inertial Measurements. Journal of Field Robotics, 25(3):181 - 202.

[12] Ho N. and Chakravarty P. 2014 Localization on Freeways using the Horizon Line Signature. Proceedings of International Conference on Robotics and Automation (ICRA).

[13] Dumble S. J. and Gibbens P. 2014 Efficient Terrain-Aided Visual Horizon Based Attitude Estimation and Localization. Journal of Intelligent and Robotic Systems.

[14] Lie W., Lin T. C.-I., Lin T., and Hung K.-S.. 2005 A robust dynamic programming algorithm to extract skyline in images for navigation, in Pattern Recognition Letters, 26(2) 221–230.

[15] Kim B., Shin J., Nam H. and Kim J. 2011 Skyline Extraction using a Multistage Edge Filtering World Academy of Science, Engineering and Technology.

[16] Haralick R.M. 1979 Statistical and structural approaches to texture, in: Proceedings of the IEEE, vol. 67, 786–804.

[17] Harlick R.M., Shanmugam K., Dinstein I. 1973 Textural Features for Image Classification. IEEE Trans, System. Man Cybernetic. 610–621.

[18] Barber D.G., LeDrew E.F. 1991 SAR sea ice discrimination using texture statistics: a multivariate approach, Photogrammetric Engineering and Remote Sensing 57 (4) 385–395.

[19] Seber, G.A.F. 1984 Multivariate Observations. John Wiley & Sons, Inc., Hoboken.

[20] Boyer M., Tarjan, D., Acton S.T., Skadron K. 2009 Accelerating leukocyte tracking using CUDA: A case study in leveraging many core coprocessors," IEEE International Symposium on Parallel & Distributed Processing, 1–12.

[21] http://www.ee.unlv.edu/~yazdan/projects.html

[22] Pour Yazdanpanah A., Regentova E. E., Mandava A. K., Ahmad T., Bebis G. 2013 Sky Segmentation by Fusing Clustering with Neural Networks. 9th International Symposium on Visual Computing (ISVC), 663-672.

[23] Dhillon I. S. and Modha D. S. 2000 A data clustering algorithm on distributed memory multiprocessors. In Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence, 245–260.

8. APPENDIX

Fig 11: Horizon Line detection by the proposed method (green line); NASA Basalt Hill data [rows 1-2], Web data [rows 3-4].

Fig 12: Examples where the proposed method has missed true or added false regions of sky.