Fast Terrain Classification Using Variable-Length Representation for Autonomous Navigation

Anelia Angelova[1] Larry Matthies[2] Daniel Helmick[2] Pietro Perona[1]
[1]Department of Computer Science
California Institute of Technology
anelia,perona@caltech.edu
[2]Jet Propulsion Laboratory
California Institute of Technology
lhm,dhelmick@jpl.nasa.gov

Abstract

We propose a method for learning using a set of feature representations which retrieve different amounts of information at different costs. The goal is to create a more efficient terrain classification algorithm which can be used in real-time, onboard an autonomous vehicle.

Instead of building a monolithic classifier with uniformly complex representation for each class, the main idea here is to actively consider the labels or misclassification cost while constructing the classifier. For example, some terrain classes might be easily separable from the rest, so very simple representation will be sufficient to learn and detect these classes. This is taken advantage of during learning, so the algorithm automatically builds a variable-length visual representation which varies according to the complexity of the classification task. This enables fast recognition of different terrain types during testing. We also show how to select a set of feature representations so that the desired terrain classification task is accomplished with high accuracy and is at the same time efficient. The proposed approach achieves a good trade-off between recognition performance and speedup on data collected by an autonomous robot.

1. Introduction

Our goal is to build a learning algorithm that can recognize various natural terrains automatically from visual information. The problem emerges in the context of autonomous navigation, in which some terrain types may negatively affect rover mobility. For example, the rover might get stuck in mud or sand, so it needs to learn to recognize such terrains in order to avoid them. In a more general context, autonomous robots would need to perceive their environment and understand what the effect of their interaction with different objects or materials will be in their surroundings, so that they can accomplish autonomous tasks, e.g. assist humans, work in cooperation with other robots, etc.

Among the challenges in this application domain is the significant intra-class variability in the appearance of natural terrains (Figure 1). Additionally, terrain classes which are very similar in appearance but affect the rover mobility differently need to be discriminated correctly. To address these challenges, the terrain classification algorithm has to use more complex representations than average color or color histograms, commonly applied in current onboard systems [3], [11], [15].

The most significant challenge for an onboard system, however, is that it has to process the abundant information from onboard sensors using very limited computational resources. Some sensors are fast to acquire and fast to process, e.g. range data, some might require more computationally intensive algorithms to process, e.g. image texture, and some are expensive to obtain and process, but might be invaluable in performing fine distinction between some materials, e.g. a high resolution, small field-of-view camera, which can focus on small portions of the terrain.

Analogously, when looking at a single sensor, e.g. color imagery, there are feature representations (or classifiers) of varying complexity which achieve different levels of success in classification. A very simple classifier might be sufficient to discriminate between some classes, e.g. recognizing grass from soil could be done by using only color. Conversely, a very complicated one might be needed for classes which are very similar but would incur a lot of penalty, if not discriminated properly, e.g. soil and sand. In summary, the terrain classification algorithm or representation does not have to be uniformly complex for all classes.

Based on these observations, we propose to build a terrain classifier in a hierarchical fashion, in which the description for each class is learned based on the complexity of the classification task. The hierarchy is automatically built by using representations of different complexity at each level and by making decisions if further processing is needed to classify some examples, or if the classification task can be subdivided. In this way, the classification task is split into smaller, possibly harder, but more focused subtasks

and some classification decisions are made early using simple and fast to evaluate classifiers. Thus, the representation of each class will be of variable length depending on the complexity of the task and its confusion with other classes. While in previous hierarchical classification methods [5] the hierarchy is built in a bottom-up fashion, we construct it in the reverse way, starting from simple classifiers. The reason is that some classifications can be done satisfactorily well with cheap sensors, *without* the need to invoke more complicated or expensive processing. The learned variable-length feature representation gives significant leverage during detection. Note that previous texture and object recognition approaches use the same complexity of representation for all classes [8], [9], [16].

Additionally, we show a general algorithm which performs efficient selection of a set of classifiers or 'sensors' which can achieve a particular classification task within a limited time. This is a generalization of the method of Viola and Jones [17] who proposed to use classifiers in a sequence. No principled approach for selecting or adjusting the complexity of the classifiers was provided in [17].

The idea for selectively processing visual information for the purposes of terrain recognition, proposed here, can be extended to using multiple onboard sensors of various capabilities and computational costs. It can also find applications in the learning of a large number of objects or visual categories, where a uniform size description for all objects will be impractical and inefficient.

2. Previous work

Previous terrain recognition approaches have focused on recognizing classes that are relatively easily discriminable, such as 'sky', 'grass', 'road' [11], or have been limited to only detecting the drivable dirt road [1], [3]. In contrast, we consider a larger variety of terrain types, some of which might be visually quite similar, such as sand, soil and gravel, and would be all considered to be in a 'dirt road' category in the abovementioned approaches. These classes, however, induce very different robot mobility, especially when driving on slopes, and therefore need to be correctly recognized. Unlike conventional methods for terrain classification which classify individual pixels [3] or pixel neighborhoods [11], here the proposal is to look at larger terrain patches. In this way, not only better statistics of occurrence of typical texture features can be built, as shown in [16], but also a speedup can be achieved by applying the proposed complexity-dependent processing of patches.

Previous texture [8], [10], [16] and scene [9] recognition approaches apply a fixed, uniform representation for all classes or construct the features without regard to the existence of other classes. Our approach is, in that sense, orthogonal to them because, as a result of the proposed hierarchical representation, the final feature representation for each class is of variable length depending on how hard a particular discrimination task is.

Hierarchical classification has become popular with classifying data which is naturally hierarchically organized, e.g. large corpora of documents, web sites or news topics [7]. It has also been applied to digit [5] and object recognition [12]. These methods require an already built hierarchy which can be obtained prior to learning, e.g. by agglomerative clustering of classes according to their similarity [5]. These techniques work from bottom up and assume that a sufficiently good object representation exists [5], [12]. Our proposal is to build the hierarchy in the reverse direction, starting from simple classifiers and not evaluating more complicated or costly classifiers, unless necessary. The hierarchy here is also built as a part of the learning process. Some similar ideas to subdivide the classification problem have appeared in [18] in which a nearest neighbor classifier finds crude clusters of similar classes and then more precise classifiers, e.g. SVM, are used to discriminate among them.

The idea of using classifiers of increasing complexity has been previously used in [6], [17]. These methods exploit the extremely skewed distribution of face vs nonface in an average image to build a classifier which quickly discards large areas which do not contain a face. Here we consider multi-class recognition and propose a mechanism for automatically subdividing the classification into more focused sub-tasks which work on fewer and more similar to one another classes. We also provide an approach for selecting a subset of classifiers to achieve the desired task.

3. Selecting an optimal set of sensors

In this section we provide an algorithm which efficiently selects a set of classifiers which can work in a sequence to achieve the classification task.

Suppose we are given a set of N sensors, or classifiers, $C_i, i = 1, ..., N$, for each of which we know how well they perform on the data (i.e. classification errors $e_i, i = 1...N$) and how much time t_i will be spent if they are tested on a dataset of a particular unit size. Additionally, let us assume that each classifier has a mechanism for classifying a particular portion of the data r_i with large confidence, so these examples are not to be evaluated further in the sequence (r_i is an estimate of the portion of examples discarded by either of the techniques proposed in Section 4). The goal is to determine a subset of classifiers which work in succession, so as to minimize some criterion, e.g. classification error or computational time. For example, we might want to know the optimal sequence of classifiers, if any, which can run within a predefined time limit and what minimal error to expect of it. The information needed, i.e. e_i, t_i, r_i, can be obtained by running each classification algorithm individually and measuring their performance prior to the optimization. The general subset selection problem is NP-complete, so we

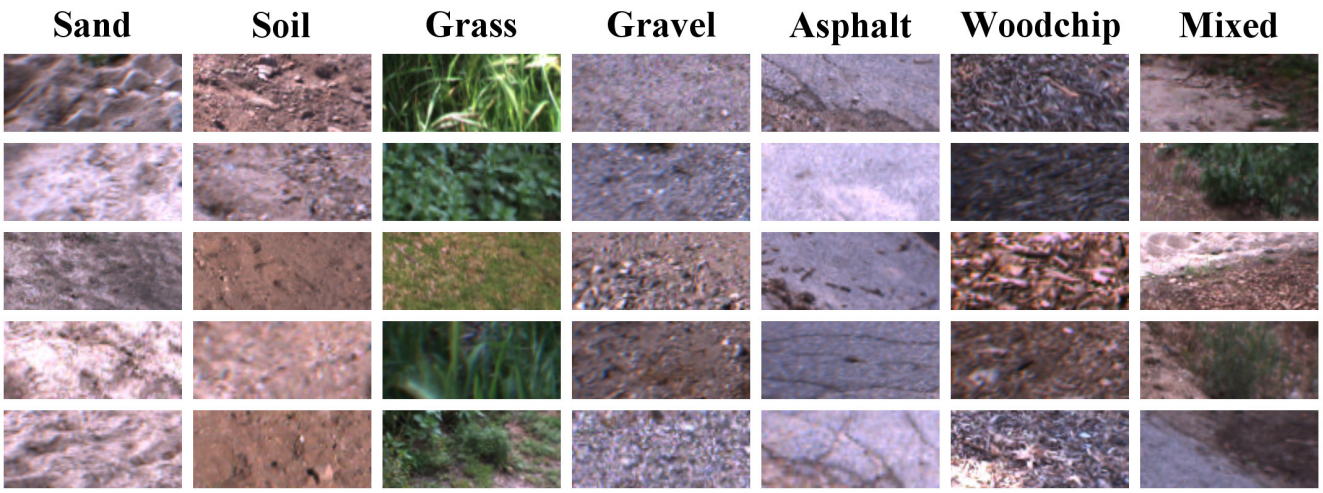

Figure 1. Patches from each of the classes in the dataset collected by an autonomous robot on natural terrains. The variability in texture appearance is one of the challenges present in our application domain. To simplify the task we have manually removed the 'mixed' terrain patches (most right column) from the training data, but they will no doubt be present in the test sequences of the robot.

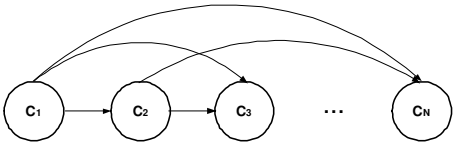

Figure 2. A set of classifiers, ordered by increasing complexity and classification power. The algorithm automatically selects a subset of them which, when used together, solve the final classification task with minimum error and in limited time.

assume that the order in which the classifiers are selected is known (Figure 2). The classifiers here are ordered according to their complexity which, unless overfitting occurs, can be thought of as ordering by decreasing classification error. In practice, this ordering will be also correlated with the increasing computational time of the classifiers. The key requirement underlying this assumption is that in this ordering, the portion of examples which can be successfully removed by each classifier r_i is preserved, independently of the previously removed examples.

Now that the classifiers are ordered by complexity we wish to find a subset (in that particular order) so as to minimize the classification error and at the same time limit the computational time. This is solved algorithmically in the following way: let us define the function $E_i^n(T,R)$ which returns the minimum error accumulated through the optimal sequence of classifiers with indices among $i,...,n$, running within time limit T and processing R portion of the whole data. $E_i^n(T,R)$ is computed using the following recursion:

$$E_i^n(T,R) = \min(e_i r_i + E_{i+1}^n(T - Rt_i, R - r_i), E_{i+1}^n(T,R))$$

with the bottom of the recursion being $E_n^n(T,R) = e_n R$. The final function that needs to be estimated is $E_N(T_{limit}) = min_{n \leq N}(E_1^n(T_{limit}, 1))$, i.e. we would like to select a set of classifiers, among the first N which can classify all the examples (R=1) with minimal error within the time limit T_{limit} (the choice of T_{limit} is guided by the application requirements). In our particular case, we solve it recursively, as we have a small N. However, for large N it is conceivable to quantize the parameters T and R and solve it efficiently using dynamic programming. Note that in the formulation of the problem, apart from trying to minimize the classification error and limit the time, the portions of examples which are expected to be discarded at each level also play a role in selecting the optimal sequence.

3.1. Case study: terrain recognition

As an example, let us consider the following classifiers in the context of terrain recognition:

0) **Average red** - the average normalized red in a patch

1) **Average color** - the average normalized R,G in a patch

2) **Color histogram** - a 3D color histogram, which builds statistics of pixel values in a patch

3) **Texton based** - a 1D histogram of occurrence of a set of learned atomic texture features ('textons'), similar to [16]; we use 20 textons per class.

4) **Texton based, slow** - same as 3) but using 40, instead of 20, textons per class.

The average color representation is not very accurate for the six terrain classes we are interested in (Figure 1), but has been used in alternative applications to discriminate between terrains such as sky, grass, dirt road [11] and has been preferred for its speed. The color histogram representation [15] considers statistics of occurring pixel values. It provides better representation than the average color and is also fast but cannot capture the dependency of neighboring

Table 1. Classification performance of each of the base algorithms.

Algorithm	Error ($e_i, \%$)	Time (t_i,sec.)
0) Average red	40.9±1.7	0.06±0.01
1) Average color	17.5±2.6	0.06±0.01
2) Color histogram	14.0±3.3	0.57±0.03
3) Texton based	8.1±1.5	4.21±0.32
4) Texton based, slow	7.9±2.4	6.26±0.42

pixels. The texton based representation considers a texture as a union of features with specific appearances, without regard to their location [16]. This type of representation, known as 'bag-of-features', has become very popular with object recognition [2], but the concept is more akin to textures. The texton based representation used here has some small differences to [16]: when working on large image patches we perform random feature sampling, which provides a certain speedup during testing. Table 1 compares the average test errors and times of the abovementioned classifiers for 100 runs on 200 randomly sampled test patches[1]. We consider the performance of the texton based classifier satisfactory, as the data is quite challenging. However, its computational time is not acceptable for a real-time system.

The results of running the algorithm with T_{limit}=3 seconds are the following[2]. It has selected classifiers 1), 2), 3) as the optimal sequence; 0) is left out as its cost is similar to 1) but its performance is much worse, so it is not cost effective to include it; 4) is left out as it has prohibitive computational time, but 3) is possible to include because it will process only a portion of the examples. The expected error of the selected sequence is 11.7%.

The above example is to illustrate that if we have available multiple classifiers of different capabilities with respect to our particular task, we can automatically select a subset of them which 1) have subsumed redundant and inefficient classifiers 2) will work more efficiently in succession. This algorithm can be viewed as a formal method for selecting the complexities of each of the classifiers in a cascade, instead of selecting them in an ad-hoc way, as in [17].

4. Learning a variable-length representation

The previous section showed how to select a subset of the available classifiers so as to minimize the classification error and at the same time guarantee that the computational time would not exceed a particular, predefined time. In this section we show how to create a variable-length representation using the selected sequence of classifiers. We build a hierarchical classifier, composed of feature representations of generally increasing complexity, at each level of which, a decision is made if the recognition task can be subdivided into smaller sub-tasks, or if some terrain classes do not need further classification. Note that the labels take part in this decision. Figure 3 shows a schematic of the algorithm. Because of this representation, an important speedup can be achieved during testing, since the slowest to compute parts of the feature representation would not need to be evaluated for all of the classes. Additionally, if some examples are classified with high confidence, they are not evaluated by the subsequent classifiers.

We use the feature representations corresponding to the classifiers, selected in Section 3.1: 1) **Average color**; 2) **Color histogram**; 3) **Texton based**. In this case, the simplest representation residing at the top level is only two dimensional, the medium complexity representation is a three dimensional histogram of pixel color appearances, while the most complex and accurate, but slowest to compute, representation is a histogram of textons detected within a patch. The classifier at each level of the hierarchy is a decision tree performed in the feature space for this particular level. A nearest neighbor classifier is used only in the last stage.

4.1. Building the hierarchy

At each level of the hierarchy we wish to determine if it is possible to subdivide the terrain recognition task into non-overlapping classes. After performing training with the classifier at this level, its classification performance is evaluated. If there are two subgroups of classes which are not misclassified with classes outside the group, then the classifier at the next level can be trained on each subgroup independently. A similar technique is applied after classification in the other intermediate levels of the hierarchy.

At each level of the hierarchy we test the newly built classifier on a validation set and construct a graph of R nodes, each node of which represents a terrain class, and each edge $m(i,j)$ represents the portion of examples of class i, misclassified as class j, $1 \leq i,j \leq R$. Instead of a graph, we can equivalently use the confusion matrix $M = M_{RxR}$ which results from this particular classification. Now the problem of finding non-overlapping classes is reduced to a min-cut problem, and in particular we will use a normalized min-cut problem which favors more balanced sizes of the two groups. Instead of solving the exact normalized min-cut problem, which is known to be NP-complete, we will apply the approximate version [13]. Using the approximate normalized min-cut [13], we compute the matrix:

$$A = D^{-1/2}(D - M)D^{-1/2},$$

where $D(i,i) = \sum_{j=1}^{R} M_{i,j}, D(i,j) = 0$, for $i \neq j$. Then

[1] Computational times are machine dependent and should be considered in relative terms.

[2] For the purposes of this example, we have set r_i to 0.01, 0.2, 0.3, 0.4, and 0.5 for $i = 0, ..., 4$ respectively, although in practice r_0 is 0 and will be immediately discarded by the optimization.

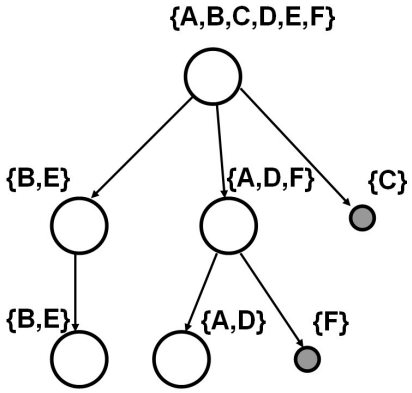

Figure 3. Schematic of the proposed hierarchical algorithm. A decision whether to subdivide the recognition task into several groups of non-overlapping classifications is made at each level. The terminal nodes are shaded; these classes do not need to be further trained or classified.

the coordinates of the second smallest eigenvector of A are clustered into two groups. This is trivial as the unique separation is found by the largest jump in the eigenvector. The elements which correspond to these two clusters are the groups of classes with the minimal normalized cut.

After the min-cut is found, the amount of misclassification between the two selected groups is computed. If there is only negligible misclassification, the data is split into subgroups of classes, which are trained recursively, independently of one another, at the next more complex level. This procedure is applied until the bottom level of the hierarchy is reached or until a perfect classification is achieved. This particular approach is undertaken, since a simple and fast to compute algorithm can be sufficient for classifying perfectly some classes or at least for simplifying the classification task. With this procedure, some groups of examples can be classified without resorting to the classifiers residing at the bottom levels of the hierarchy, especially if they involve some very inefficient computations. Conversely, if there are no useful splits, this means that the feature space is not reliable enough for classification and the classifier needs to proceed to the next level of complexity. In the case where one of the groups is of a single class, it will be a terminal node in the hierarchy and no more training and testing will need to be done for it. We have limited the subdivisions to two groups only, although recursive subdivision is possible.

Instead of using the raw confusion matrix M, some domain knowledge can be introduced. A cost matrix C_{RxR}, which determines the cost of misclassification of different terrains, can be constructed for a particular application. For example, misclassifying sand for soil is very dangerous as the rover might get stuck in sand, but confusing terrains of similar rover mobility would not incur a significant penalty. Given C, the normalized min-cut is performed on the matrix $M_C = M.C$ (element-wise multiplication).

4.2. Finding confident classifications

An additional mechanism of the algorithm is to perform *early abandon* of examples which have confident classifications. That is, such examples will not be evaluated at the later stages of the hierarchy. For that purpose we put the algorithm in a probabilistic framework. At each intermediate level we wish to determine the probability of a particular classification (assignment to a class w_i) given an example: $P(w_i|X) = \frac{p(X|w_i)p(w_i)}{\sum_{j=1}^{R} p(X|w_j)p(w_j)}$, $1 \leq i \leq R$.

Each example, for which the risk of misclassification $\sum_{j \neq i} P(w_j|X)$=$1 - P(w_i|X)$ [3] is small, will be discarded.

The prior probabilities are set according to some knowledge about the terrain. For example, if the terrain contains mostly soil and grass, the soil and grass will have higher priors than the other terrains. Here we set equal priors because we have extensive driving on predominantly asphalt, sand, gravel and woodchip terrains too. So now the problem is reduced to computing $p(X|w_i)$ for $i = 1, ..., K$. As we have mentioned, the classifier at the intermediate levels is a decision tree. Also note that some of the feature representation might be high dimensional e.g. 2) and 3), if 3) were selected to be an intermediate level. To compute the required probability we use the following non-parametric density estimation method, proposed by [14]. For each example X, the probability $p(X|w_i)$ is approximated by:

$$p(X|w_i) = \frac{1}{N_i} \sum_{s=1}^{N_i} \prod_{k \in path} \frac{1}{h_k} K\left(\frac{X^k - X_s^k}{h_k}\right),$$

where X^k are the values of X along the dimensions selected along the path from the root to the leaf of the tree where this particular example is classified, $X_s, s = 1, ..., N_i$ are the training examples belonging to class w_i, K is the kernel function, and h_k is the kernel width. In this way, instead of performing a density estimation in high dimensional spaces, only the dimensions which matter for the example are used.

At each level we evaluate a threshold such that if $P(w_i|X) \geq \Theta$ for some example X, then it will be classified as belonging to class w_i and will not be evaluated in the consequent levels. The rest of the examples are re-evaluated by the supposedly more accurate classifier at the next level.

4.3. Discussion

Building the classifier in a hierarchical way has the following advantages. Firstly, classes which are far away in appearance space or otherwise easily discriminable, will be classified correctly early on, or at least subdivided into groups where more powerful classifiers can focus on essentially more complex classification tasks. This strategy

[3] If a misclassification cost matrix C is available, the risk of misclassification $R(i|X) = \sum_{j=1}^{R} C_{i,j} P(w_j|X)$ will be used instead.

could be considered as an alternative to the 'one-vs-all' and 'one-vs-one' classifications when learning a large number of classes simultaneously. Secondly, there is no need to build complex description for all classes and perform the same comparison among all classes. So, the description lengths of each class can be different, which gives significant leverage during testing. Thirdly, classifications which are confident will be abandoned early during the detection phase, which will give additional speed advantage. A drawback of hierarchical learning is that a mistake in the decision while using simple classes can be very costly, so for that purpose we make a decision only if the classification is correct with high probability.

The key element of the method is that the class labels are taking active part in building the hierarchy and therefore creating the variable-length representation. This is in contrast to previous approaches which have done the feature extraction disregarding the class label [8], [9], [10], [16].

Although the proposed hierarchical construction shares the general idea of a decision tree of subdividing the task into smaller subtasks, the proposed hierarchy operates differently. More complicated classifiers reside at each level rather than simple attributes, as is in a decision tree. This requires a more complicated attribute selection or, in our case classifier selection, strategy as proposed in Section 3. Some previous criteria for subdividing the training data into subclasses have been applied for decision trees [4], but they involve combinatorial number of trials to determine the optimal subset of classes per node. Instead, we propose an efficient solution using normalized min-cut (Section 4.1).

For an arbitrary learning task, there is no guarantee that a split in the learning of classes will occur at the earlier stages. In that case, the algorithm presented in Section 3 ensures that the hierarchical classifier converges to the largest complexity classifier at the bottom level, rather than building a composite representation at multiple levels. In particular, the algorithm in Section 3 takes into consideration the time that can be saved by classifying examples early on and the overhead of computing additional shorter length representations and selects (in a greedy way) the optimal sequence of classifiers, if any. The framework is advantageous for problems in which the complexity of discrimination among classes is non-uniform, e.g. for easily identifiable classes or groups of classes which can be assigned short description lengths and, conversely, for sets of classes which are very similar and more complex representations are needed to make fine distinctions among them.

5. Experimental evaluation

The proposed algorithm has been applied to terrain recognition for the purposes of autonomous navigation. The dataset has been collected by an autonomous LAGR[4] robot while driving on six different off-road terrains: soil, sand, gravel, asphalt, grass, and woodchips. We consider the texture in image patches which correspond to map cells. The patches are 100 pixels across for map cells visible at close ranges (1-2m) and 10-15 pixels across for cells at far ranges (5-6m). Figure 1 shows some examples from the best resolution patches available from all the terrains. The data is quite challenging as it is obtained in outdoor environments.

We compare the classification performance and the speed of each of the baseline (flat) classifiers with the hierarchical classifier. The experimental setup is such that all the classifiers are evaluated on the exact same split of the data into training, test and validation subsets. The average performance and time from multiple runs or across multiple frames is reported below. The algorithm depends on two parameters: 1) the portion of examples g_1 misclassified between two groups before a split is allowed (here g_1=0.03). 2) the portion of examples g_2 misclassified by a high confidence early abandon technique (here g_2=0.06),

Two experiments are performed:

Experiment 1. We take $\sim$600 patches collected from the rover at close range. They are distributed equally by random sampling into independent training, validation and test sets and each of the algorithms is tested on them. The results of this experiment, comparing the baseline algorithms to the hierarchical one are shown in Figures 4 and 5. The hierarchical classifier decreases the computational time of the texton classifier more than twice at the expense of slight increase in the test error. We also compared the performance of the hierarchical classifier, in the cases when only splitting is allowed, and when only examples with significant confidences are discarded (without doing any splitting). When only splitting in the hierarchy is performed, the computational time decreased, but not dramatically, which is due to the fact that in our particular application five out of six classes will reach the bottom level (a typical hierarchy discards the grass class after the first level and then after the second level splits the rest of the classes into two groups: $\{$sand, soil, woodchip$\}$ and $\{$gravel, asphalt$\}$). The important point is that in this case a decrease in computational time is achievable without compromising classification performance. A more notable decrease in test time comes from discarding examples with large confidence, which however introduces some error. The final hierarchical classifier benefits from both mechanisms.

Test results of the hierarchical classifier when trained with different values of the parameter g_1, varying from g_1=0 (no hierarchical split allowed) to g_1=0.1, are shown in Figure 6. As seen, there is a certain range for which the proposed classifier can decrease the computational time no-

[4]LAGR stands for Learning Applied to Ground Robots and is an experimental all-terrain vehicle program funded by DARPA

tably with almost no increase in classification error. A version of the hierarchical classifier in which examples of high confidence are not allowed to be eliminated (g_2=0) is also shown. The initial large error, corresponding to g_1=0, is due to evaluating all the representations for all of the examples. The error starts decreasing as soon as the hierarchical splitting is allowed.

Experiment 2. We test the algorithm on 512x384 image sequences collected by the rover. As the image resolution decreases with range, a texture classifier trained at close range does not generalize well at far ranges. To solve the problem, we train two independent classifiers for patches observed at close ($\leq$3m) and far ($>$ 3m) ranges. The performance is evaluated on a $\sim$1600 frame sequence containing all six terrains, testing every tenth frame of the sequence. A local voting among the decisions on the neighboring map cells is done to remove occasional misclassification errors.

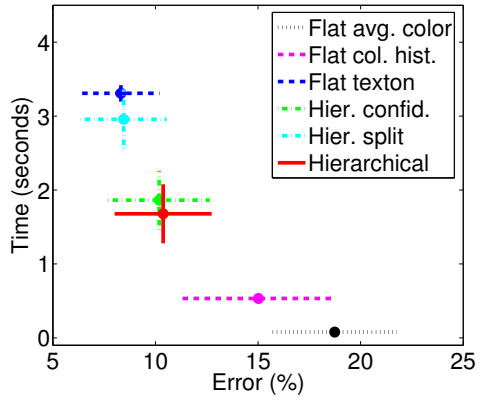

Figure 4. **Experiment 1.** Average test results evaluated on $\sim$200 randomly sampled best resolution test patches (100 runs).

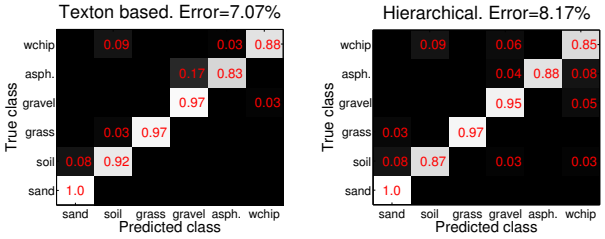

Figure 5. **Experiment 1.** Confusion matrices for one of the runs from the results in Figure 4. Texton based (left), hierarchical (right). Only the non-zero elements are displayed.

Summary results are shown in Table 2. The hierarchical classifier simultaneously achieves very good performance (only slightly outperformed by the texton approach) and decreases the computational time by more than a factor of two. To further analyze the results, we restricted the classification to the patches at close and far ranges (Table 3). At close ranges, the texton approach has higher classification rate than the hierarchical one, as expected, but is much

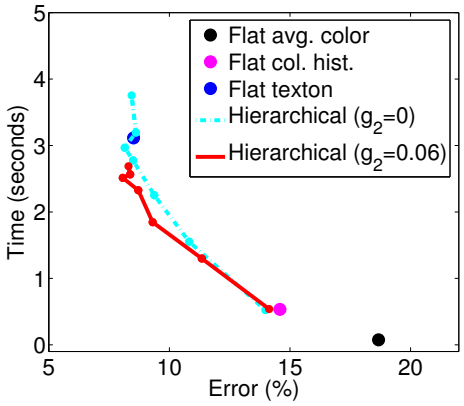

Figure 6. **Experiment 1.** Test results for different values of the parameter g_1 (averaged over 50 runs).

slower. The deterioration of the texton classifier performance at far ranges is because the farther patches take a much smaller portion of the image and do not contain sufficient texture information. The hierarchical classifier, on the other hand, takes advantage of the other baseline methods which rely mostly on color to achieve better classification performance. Furthermore, when comparing the computational time, we can observe that the hierarchical classifier is significantly faster than the texton approach at close range, which is because map cells take larger portions of the image and are more informative. As a result, they are more likely to be correctly classified with simpler methods, and whenever they are classified early, a significant speedup is achieved. The hierarchical classifier spends more computational time at far ranges, since the patches are less informative and therefore the algorithm cannot be very certain in making a split during training or in making an early decision during testing. Figure 7 shows the terrain classification results and the amount of time spent to test each map cell on a frame collected on gravel terrain.

Table 2. **Experiment 2.** Average classification rate and time on image sequences. Classifies all cells of the forthcoming terrain; includes all image resolutions. The test time is evaluated per frame.

Algorithm	Classif. (%)	Time (sec.)
Texton based [16]	78.65	3.47
Hierarchical (proposed here)	76.58	1.48

6. Conclusions and future work

We propose to efficiently process color imagery using a hierarchy of classifiers ('sensors') which retrieve different amounts of information at different costs. First, a subset

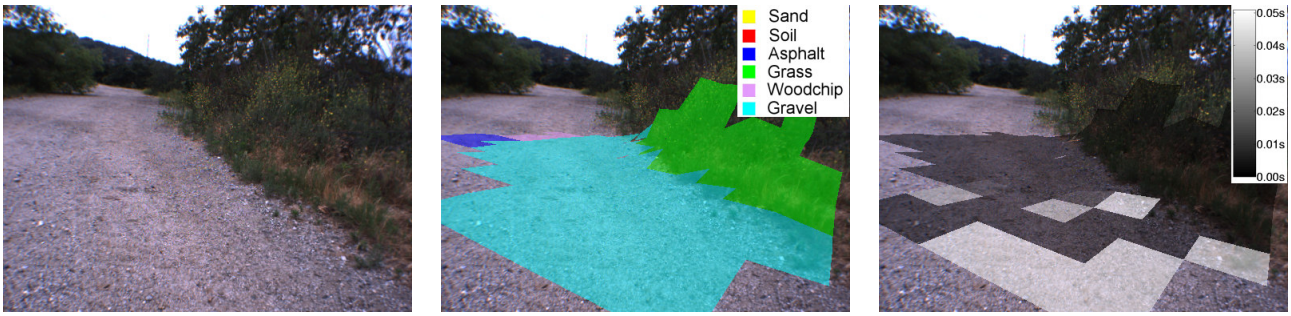

Figure 7. **Experiment 2.** Input color image (left), terrain classification results of the hierarchical classifier in a frame (middle) and the amount of computations performed on each cell (right) overlayed on the original image. The algorithm gains speed advantage from classifying some patches at close range with much less computation than the baseline method. Gravel terrain.

Table 3. **Experiment 2.** Classification performance on image sequences, evaluating separately patches at close and far ranges.

Algorithm	Classif. (%)	Time (sec.)
Texton based (ranges $\leq$ 3 m)	78.85	2.32
Hierarchical (ranges $\leq$ 3 m)	77.40	0.70
Texton based (ranges $>$ 3 m)	64.18	1.10
Hierarchical (ranges $>$ 3 m)	65.68	0.82

of classifiers is selected as a response to the needs of the classification task. That is, classifiers which are redundant, inefficient, or simply not useful regarding a particular classification task are not selected by the algorithm. Second, a hierarchical classifier is built, taking into consideration the labels and the complexity of the classification task. As a result, a variable-length representation for each terrain class is learned, which gives significant leverage during detection. The outcome is a very competitive in terms of performance terrain classifier which also runs faster.

A natural extension is to consider more complex representations from high resolution or multi-spectral cameras which have better discriminative capabilities for some classes and will improve the overall performance. Another important next step is to construct the hierarchy while performing the optimization proposed in Section 3. This will provide more accurate estimates for the values e_i, t_i, r_i.

Acknowledgment. This research was carried out by the Jet Propulsion Laboratory, California Institute of Technology with funding from the NASA's Mars Technology Program. We thank Max Bajracharya and the anonymous reviewers for providing very useful comments on the paper.

References

[1] Y. Alon, A. Ferencz, and A. Shashua. Off-road path following using region classification and geometric projection constraints. *CVPR*, 2006.

[2] A. Berg, T. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondences. *CVPR*, 2005.

[3] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski. Self-supervised monocular road detection in desert terrain. *Robotics: Science & Systems*, 2006.

[4] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley & Sons, 2001.

[5] X. Fan. Efficient multiclass object detection by a hierarchy of classifiers. *CVPR*, 2005.

[6] F. Fleuret and D. Geman. Coarse-to-fine face detection. *International Journal of Computer Vision (IJCV)*, 2001.

[7] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. *International Conference on Machine learning (ICML)*, pages 170–178, 1997.

[8] S. Lazebnik, C. Schmid, and J. Ponce. A sparse texture representation using affine invariant regions. *CVPR*, 2003.

[9] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. *CVPR*, 2006.

[10] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *IJCV*, 43(1), 2001.

[11] R. Manduchi. Bayesian fusion of color and texture segmentations. *ICCV*, 1999.

[12] K. Mikolajczyk, B. Leibe, and B. Schiele. Multiple object class detection with a generative model. *CVPR*, 2006.

[13] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. on PAMI*, 2000.

[14] P. Smyth, A. Gray, and U. Fayyad. Retrofitting decision tree classifiers using kernel density estimation. *ICML*, 1995.

[15] B. Upcroft et al. Multi-level state estimation in an outdoor decentralised sensor network. *Proceedings of the Int. Symp. on Experimental Robotics*, 2000.

[16] M. Varma and A. Zisserman. Texture classification: Are filter banks necessary? *CVPR*, 2003.

[17] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *CVPR*, 2001.

[18] H. Zhang, A. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. *CVPR*, 2006.