Automated Performance Evaluation of Range Image Segmentation Algorithms

Jaesik Min, *Member, IEEE*, Mark Powell, *Member, IEEE*, and Kevin W. Bowyer, *Fellow, IEEE*

***Abstract*—Previous performance evaluation of range image segmentation algorithms has depended on manual tuning of algorithm parameters, and has lacked a basis for a test of the significance of differences between algorithms. We present an automated framework for evaluating the performance of range image segmentation algorithms. Automated tuning of algorithm parameters in this framework results in performance as good as that previously obtained with careful manual tuning by the algorithm developers. Use of multiple training and test sets of images provides the basis for a test of the significance of performance differences between algorithms. The framework implementation includes range images, ground truth overlays, program source code, and shell scripts. This framework should**

a) make it possible to objectively and reliably compare the performance of range image segmentation algorithms;
b) allow informed experimental feedback for the design of improved segmentation algorithms.

The framework is demonstrated using range images, but in principle it could be used to evaluate region segmentation algorithms for any type of image.

***Index Terms*—Performance evaluation, range image segmentation, region segmentation.**

I. Introduction

PERFORMANCE evaluation of computer vision algorithms has received increasing attention in recent years [1], [2], [3], [4], [5], [6]. This paper presents an automated framework for objective performance evaluation of region segmentation algorithms. While the framework is developed in the context of range images, in principle it is applicable to any type of imagery.

Earlier work in performance evaluation of range image segmentation compared four algorithms that segment images into planar regions [7]. In this comparison, the training to select parameter values for the algorithms was done manually by the algorithm developers. This work was extended to include algorithms that segment range images into curved-surface patches [8], to evaluate additional planar-surface algorithms [9], and to use an automated method of training to select algorithm parameters [10], [11]. None of these works provided a means to test for the significance of observed performance differences between algorithms.

Manuscript received July 1, 2002; revised January 5, 2003. This work was supported by National Science Foundation Grants IIS-9731821 and EIA-9729904. This paper was recommended by Associate Editor X. Jiang.

J. Min and K. W. Bowyer are with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: jmin@nd.edu; kwb@cse.nd.edu).

M. Powell is with the Mobility Systems Concept Development Section, Jet Propulsion Laboratory, Pasadena, CA 91109 USA.

Digital Object Identifier 10.1109/TSMCB.2003.811118

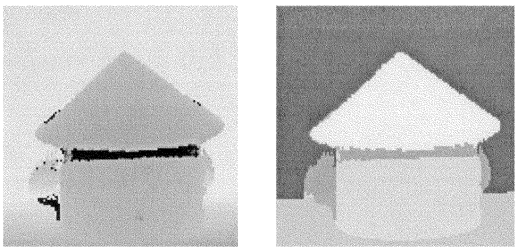
We introduce an automated framework for objective performance evaluation of region segmentation algorithms for range images. This framework includes image data sets for planar-surface and curved-surface scenes, corresponding manually-specified ground truth for the images, a tool for scoring of performance metrics, a tool for training to select algorithm parameters, source code for baseline algorithms for performance comparison, and a test for statistical significance of observed performance differences. The framework is available to the research community on our website http://www.nd.edu/~cvrl.

The remaining sections of this paper are organized as follows. The next section defines how instances of correct and incorrect region segmentation are scored. Section III then defines how these instances of correct and incorrect segmentation are summarized in a performance curve for a given algorithm and set of images. Section IV describes how the performance curve is used in an automated search of the segmenter's parameter space to determine the appropriate number of parameters and their settings. Section V outlines the framework for benchmarking the performance of a new algorithm and comparing it to a known baseline algorithm. Section VI steps through the details of the framework implementation and an example comparison of two algorithms.

II. Definition of Performance Metrics

The specific definition of region segmentation that we use is the same as used by Hoover *et al.* [7] and is repeated here for reference. A segmentation of an image R into regions $r_1, \ldots, r_n$ is defined by the following properties.

1) $r_1 \cup r_2 \cup \cdots \cup r_n = R$. Each pixel belongs to some region.
2) Every region is spatially connected. Our implementation currently uses four-connectedness as the definition of spatially connected.
3) $\forall r_i, r_j \in R \vdash i \neq j, r_i \cap r_j = \emptyset$. Regions do not overlap each other.
4) $\forall r_i \in R, P(r_i) = true$. All pixels in a region satisfy a specified similarity predicate; in the case of range images, they belong to the same scene surface.
5) $\forall r_i, r_j \in R \vdash i \neq j$ and r_i, r_j are four-connected and adjacent, $P(r_i \cup r_j) = false$. If two regions are four-connected and adjacent, then they represent different surfaces.
6) There are "artifact regions" in the image where no valid measurement was possible which all have the same label (violating rule 2) and for which rules 4 and 5 do not apply. In the ground truth, these generally represent sensor artifacts that the region segmentation algorithm is not expected to handle correctly as normal regions.



(a) Range Image (b) Ground Truth Image

Fig. 1. Example range image and corresponding ground truth image. The ground truth specifies four surface regions: two planar, one cylindrical, and one conical.

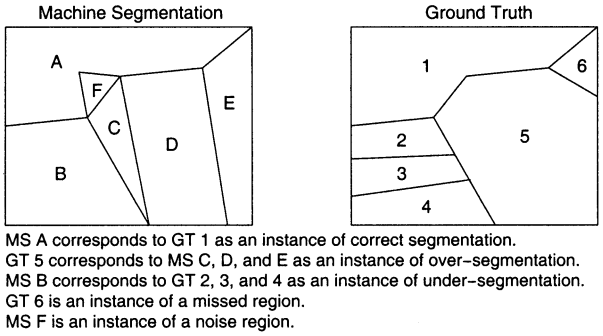


Fig. 2. Illustration of definitions for scoring region segmentation results.

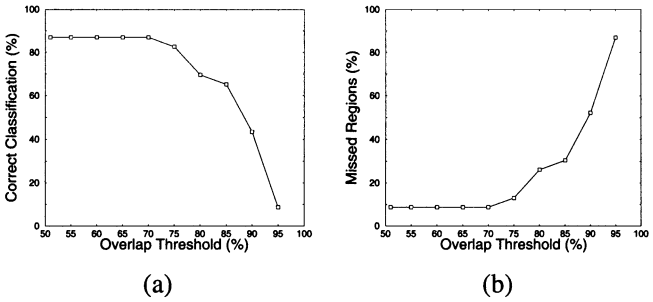


Fig. 3. Performance curve examples for instances of (a) correct segmentation and (b) missed regions.

For each image used in the train or test sets, a ground truth (GT) segmentation is manually specified using an interactive tool developed for this purpose. Fig. 1 shows an example range image and its corresponding ground truth. The scene has one conical surface and one cylindrical surface in the foreground, and two planar surfaces in the background. The GT contains a region for each of these surfaces, plus "artifact regions" for the areas that correspond to significant artifacts in the image. For example, the "shadow region" that is cast onto the cylindrical surface by the conical surface is marked as an artifact region in the GT. The shapes in the curved-surface scenes are all formed from quadratic surface patches, and the GT segmentation is in terms of these quadratic surface patches: planar, cylindrical, spherical, conical, and toroidal. This may present challenges for the evaluation of segmentation algorithms that describe a region patch using more general curved surfaces [12].

A machine segmentation (MS) of an image can be compared to the GT specification for that image to count instances of correct segmentation, under-segmentation, over-segmentation, missed regions, and noise regions. The definitions of these metrics are based on the degree of mutual overlap required between a region in the MS and a corresponding region in the GT. An instance of "correct segmentation" is recorded if and only if an MS region and its corresponding GT region have greater than the required threshold of mutual overlap. Multiple MS regions that correspond to one GT region constitute an instance of over-segmentation. One MS region that corresponds to several GT regions constitutes an instance of under-segmentation. A GT region that has no corresponding MS region constitutes an instance of a missed region. A MS region that has no corresponding GT region constitutes an instance of a noise region. Fig. 2 illustrates these definitions of the performance metrics. Results are automatically scored using a tool that compares an MS result to its corresponding GT at a specified overlap threshold.

III. Performance Curves for Region Segmentation

The meaningful range of required overlap between a given MS result and its corresponding GT image is $50\% < T \leq 100\%$. As the overlap threshold T is varied from lower (less strict) to higher (more strict) values, the number of instances of correct segmentation generally decreases. At the same time, the number of instances of the different errors generally increases. This is shown in Fig. 3. A performance curve can be created for each individual metric (correct segmentation, under-segmentation, etc.) for each image in a data set. The performance curve shows how the number of instances of the given metric changes for the given image as the overlap threshold varies over its meaningful range. Also, an average performance curve can be created for an image data set as a whole. Everingham *et al.* [13] proposed as a performance measure the use of Pareto front that allows trade-offs between multiple performance criteria in evaluation of image segmentation algorithms.

If algorithm A has consistently better performance than algorithm B, then its performance curve for the correct detections metric will lie above that of algorithm B. This comparison can be given a quantitative basis using the area under the curve. Performance curves can be normalized to a basis where the ideal curve has an area of 1. Thus the "area under the performance curve" (AUC) becomes an index in the range of $[0, 1]$, representing the average performance of an algorithm over a range of values for the overlap threshold. It is of course possible that the AUC index will obscure situations where, for example, algorithm A is better than algorithm B for low values of the overlap threshold, but worse at high values. Thus, in comparing two algorithms, it is important to also consider whether the performance curves cross each other.

The performance curve and the AUC metric as used here have a similarity to the receiver operating characteristic (ROC) curve and the area under the ROC curve [14]. However, precisely speaking, the performance curve as used here is not legitimately an instance of an ROC curve, or of a free-response ROC curve (FROC). The image segmentation problem as defined here is to specify a complete decomposition of the image into a set of regions. This is different from the problem definitions that give rise to the ROC or FROC curve. In essence, the

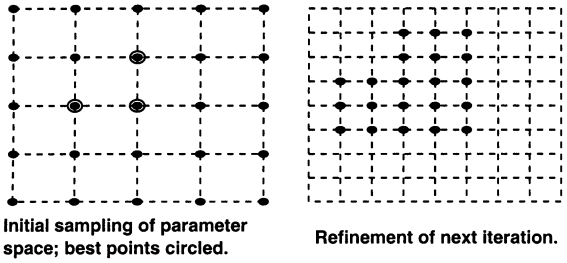


Fig. 4. Illustration of the adaptive sampling of parameter space in the training process.

problem definition underlying the ROC curve is to classify a given region as "positive" or "negative," and the definition underlying the FROC curve is to detect the "positive" regions in an image. Thus the problem underlying the ROC or FROC curve is in some sense simpler.

For experiments reported in this paper, the AUC values are computed using a trapezoid rule with overlap threshold sampled at ten values: 0.51, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, and 0.95. Our general experience is that the performance of current range segmentation algorithms drops rapidly with a threshold any stricter than 0.8, and so there is little value in sampling beyond 0.95.

IV. Manual Versus Automated Parameter Training

Manual training of algorithm parameters will produce results that are dependent on the knowledge, skill and effort of the experimenter. If the algorithm developer and a new algorithm user each spend equal time manually tuning the algorithm parameters, their results are likely to be different. Similarly, if two different users spend different amounts of time and effort in manual tuning, their results are likely to be different. For these reasons, an automated training procedure is preferable to manual training. This is especially true if automated training can be shown to produce performance at least as good as that produced by manual training done by algorithm developers well motivated to tune the performance of their algorithm.

We use the following adaptive search for our training procedure. Assume that the number of parameters to be trained, and the allowed range of each parameter, are specified. The range of each parameter is sampled by five evenly-spaced points. In case the parameter type is an ordered set, such as integer or Boolean, those five points will be rounded and redundant points will be deleted. If D parameters are trained, then there are 5^D initial parameter settings to be considered. The segmenter is run on each of the training images with each of these 5^D parameter settings. The segmentation results are evaluated against the ground truth using the comparison tool. Performance curves are constructed for the number of instances of correct region segmentation, and the areas under the curves are computed. The highest performing one percent of the 5^D initial parameter settings, as ranked by area under the performance curve on the training set of images, are selected for refinement in the next iteration (e.g., the top six settings carried forward in training four parameters).

The refinement in the next iteration creates a $3 \times 3 \times \cdots \times 3$ sampling around each of the parameter settings carried forward. See Fig. 4 for an illustration. In this way, the resolution of the parameter settings becomes finer with each iteration, even as the total number of parameter settings considered is reduced. The expanded set of points is then evaluated on the training set, and area under the performance curves again computed. The top-performing points are again selected to be carried forward to the next iteration. Iteration continues until the improvement in the area under the performance curve drops below 5% between iterations. (A value of 1% was also tried for this cutoff, but there was minimal change in the results and it was judged not to be worth the increased execution time.) Then the current top-performing point is selected as the trained parameter setting. Our search algorithm is a form of multi-locus hill climbing. The algorithm in its concept does not guarantee to find the global minima and that is why we set a larger number of initial points. We compared our algorithm to an exhaustive search at the same resolution of sampling parameter space and found no statistically significant difference in the performance of the selected points.

A more complex approach for searching the parameter space has recently been considered by Cinque *et al.* [15]. They explored an approach based on genetic algorithms. They suggest that the training of the University of Bern (UB) algorithm [16] for planar and curved-surface images is relatively sensitive to the composition of the training set. They do not report details of training execution times or composition of training sets used, and so we cannot make a direct comparison of our training approaches on these points.

Segmenters may vary in the number of parameters provided for performance tuning. The number of parameters trained is a major factor in the effort required in the training process. For example, training four parameters of the UB algorithm [17] on 10 six-image training sets (256×256 images) takes about one day as a background process on a Sun Ultra 5 workstation.

An important question is whether automated training can produce performance as good as manual training. Example comparisons of performance curves obtained by manual and automated training appear in Figs. 5 and 6. The performance curves representing the manually-tuned parameters of the four different segmentation algorithms for each of the two data sets are the same as those reported in [7]. The algorithm developers are assumed to have been well motivated in producing these results, as they were producing them for a public comparison of algorithm performance. The parameters of the UB algorithm [16] are also assumed to have been well-tuned because the algorithm performed well in this comparison. We also used the same UB algorithm with our automated parameter tuning algorithm to obtain a performance curve for each data set. This allows a direct comparison of results for manual versus automated tuning of parameters (see Fig. 7).

Note that the differences between the performance curves of the UB algorithm for manual versus automated training are small, with manual training having a slight advantage on one data set and automated training having a slight advantage on the other data set. Also important is the fact that the differences in performance here due to manual versus automated training are small in comparison to differences between segmenters. This indicates that automated training provides results comparable to those from manual training for the algorithms tested, and that

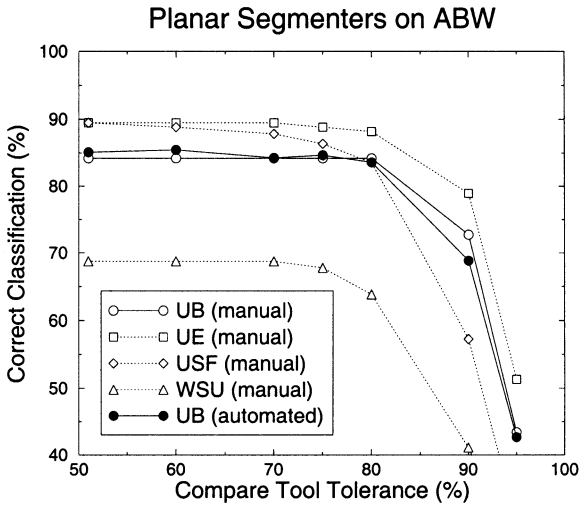


Fig. 5. Manual versus automated training to select parameter values.

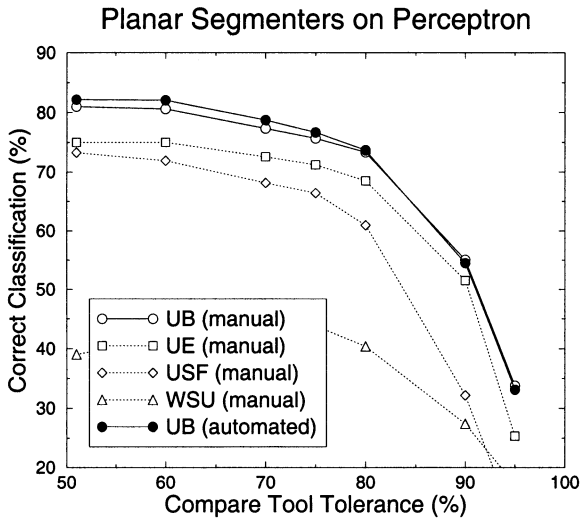


Fig. 6. Manual versus automated training to select parameter values.

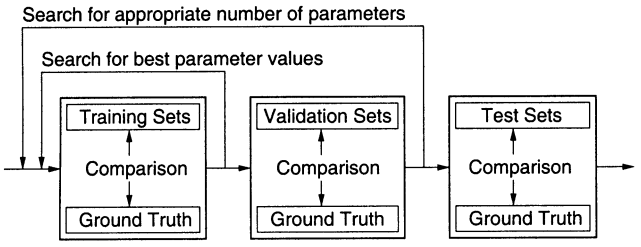


Fig. 7. Train, validation, and test performance evaluation framework.

automated training can reasonably replace manual training in algorithm comparisons.

The UB planar-surface segmenter has seven parameters that control its operation, as listed in Table I. These parameters are thresholds on various values in the segmentation algorithm. The table gives the values of the seven parameters as found by manual training [7]. The first two parameters are considered to be the most critical. For this experiment, automated training was done on the four most important parameters, and the three less important parameters were fixed at the same values as found by the manual training [7]. The parameter values found by automated training are generally close to those selected by manual training, but not identical. This appears to be simply an instance of different sets of parameter combinations resulting in similar performance on the test set. Note that the automated parameter tuning adaptively refines the resolution of the parameter sampling, and will not necessarily sample all the same values sampled in the manual tuning. Also, the automated tuning stops when the improvement falls below 5% between refinements.

TABLE I
MANUAL AND AUTOMATED TRAINING RESULTS FOR UB PLANAR-SURFACE SEGMENTER

	ABW		Perceptron	
Para.	manual	automated	manual	automated
T_1	1.25	1.20	1.75	2.75
T_2	2.25	2.90	3.25	3.50
t_1	4.0	4.50	4.0	1.70
t_2	0.1	0.75	0.1	0.125
t_3	3.0	3.0	3.0	3.0
t_4	0.1	0.1	0.2	0.2
t_5	100	100	150	150
AUC	81.31	79.56	71.26	71.99

V. VALIDATION SET TO CONTROL NUMBER OF PARAMETERS

Our performance evaluation framework uses separate sets of images for train, validation, and test. The training step searches for the "best" parameter settings. The validation step decides how many of the segmenter's parameters should have their value learned through training versus left at the default value. The test step determines performance curves to be used in comparing different segmenters. Because the selected parameter settings may vary based on the particular set of training images, we create multiple different training sets by random sampling from a larger pool of training images. Because the measured performance also may vary based on the particular images in the test set, we create multiple test sets.

In general, algorithms have a number of parameters that control their operation, and there are default values for each parameter. This introduces the question of how many of the available parameters should have their value set as a result of training versus left at their default value. Training on $N+1$ parameters naturally produces training results that are at least as good as training on N parameters. This can lead to over-training on the number of parameters, and reduced performance on the test set. The framework uses a validation step to avoid this over-training problem.

After training on a given number of parameters, the parameter values for each training set are run on each validation set. If there are T_{tr} training sets and V validation sets, then $T_{tr} \times V$ performance curves are produced. If the area under the validation performance curves is statistically significantly improved in going from $N-1$ to N parameters, and additional parameters are available, then training is repeated using $N+1$ parameters. (A sign test is used to test for statistical significance, as explained later.) If there was no significant improvement in going to N parameters available, then the $(N-1)$-parameter training result is kept. If there are no additional parameters, then the N-parameter result is kept. Although the current implementation of the

framework uses the validation sets only to avoid overtuning the number of parameters, the framework might be enhanced if the validation sets are also used in handling the values of parameters. That is, the optimization of a parameter can be stopped when the performance on the validation sets, rather than the training sets, start to decline.

Using training and validation sets, it is still possible that one algorithm could generalize better than another. For instance, in an evaluation of edge detection algorithms using the results of a structure-from-motion task, it was found that the ranking of different algorithms changed from the training results to the test results [18]. For this reason, once the right number of parameters and their appropriate settings are found through training and validation, the performance of the algorithm is then measured using separate test data. The final trained parameter values from each training set are run on each test set, resulting in $T_{tr} \times T_{te}$ performance curves. The areas under these curves are used as the basis of a test for statistical significance of an observed difference in performance between segmenters.

Note that the performance results of the baseline and challenger algorithms are "paired" according to the train and test sets. This is important for the comparison of algorithm performance. Relative performance of two algorithms should first be assessed by visual inspection of the corresponding performance curves. If there is a consistent pattern of one algorithm performing better than the other at lower values of the overlap threshold but worse at higher values, then the comparison becomes more problematic. However, in general, performance can be compared quantitatively and statistically by using the paired differences in the areas under the performance curves. Two types of statistical test are possible.

Assume that we are comparing a "challenger" algorithm to a "baseline" algorithm. The test statistic will be the difference between the areas under the corresponding performance curves, paired by train and test sets. The sign test can be used to check for statistical significance without requiring the assumption that the differences follow a normal distribution [19]. The null hypothesis is that there is no true difference in average performance between the algorithms. Under the null hypothesis, each algorithm has a 0.5 probability of generating the larger area under the performance curve on any given trial. The number of trials for which one algorithm generates a larger area than the other should follow a binomial distribution. A more powerful statistical test that may be applicable in some cases is the paired-t test [19]. The value $D_i = A_i - B_i$ is then the difference in the areas under the paired performance curves. The test statistic is the mean of the D_i divided by the standard deviation of the D_i. The test statistic can be compared against limits of the t distribution for the appropriate number of degrees of freedom and chosen level of confidence, and the null hypothesis rejected if the test statistic is sufficiently far from zero.

Our framework implementation automatically reports the results of a sign test rather than the paired-t test. This is because of the less strict assumptions about the distribution of the data. However, the paired-t test may be used if justified after examining the distribution of the differences.

One objection could be raised against the use of either statistical test in the current implementation of the framework. In the current implementation, a particular image from the pool of test images may appear in more than one test set. Also, parameters found with one training set are evaluated with more than one test set. Thus the possible objection is that the different trials used in the statistical test are not as independent as they should be. This objection could be addressed through using larger pools of train, validation, and test images.

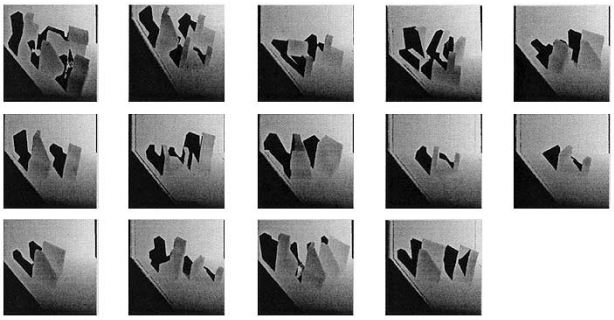


Fig. 8. Pool of 14 training images for planar-surface scenes.

VI. Implementation of the Evaluation Framework

The implementation of the framework consists of the various image datasets, the corresponding ground truth overlays, the comparison tool, the control structure for the framework, and baseline segmentation algorithms for planar-surface [16] and curved-surface [17] scenes. The implementation of the framework is available in the form of a compressed UNIX tar file. The programs are in the form of C source code and simple UNIX scripts. The framework should be able to be installed on most UNIX systems with minimal effort.

A. Image Data Sets

Two image data sets are used in the framework.[1] For planar scenes, we use the same set of forty ABW range images used in Hoover *et al.* [7]. For curved-surface scenes, we use a new dataset of forty Cyberware images acquired specifically for this purpose. (The image set used in [8], from the K2T structured-light scanner, was dropped from this study due to problems with data quality.) The average number of GT regions in an image is 16.5 for the ABW image set and 9.0 for the Cyberware image set.

Each set of forty images is divided into a pool of fourteen training images, thirteen validation images, and thirteen test images. Ten different training sets of six images each are created by random sampling from the pool of training images. Similarly, 10 validation sets of six images each are created by sampling from the pool of validation images, and 10 test sets of six images each are created by sampling from the pool of test images. Thus the decision about how many parameters to train is based on comparison across 100 (10×10) performance curves. Similarly, comparisons between segmenters would be based on 100 test performance curves. The pools of training, validation, and test images for planar-surface scenes are shown in Figs. 8–10, respectively. The pools of training, validation, and test images for curved-surface scenes are shown in Figs. 11–13, respectively.

[1]See www.abw-3d.de/home-e.html and www.cyberware.com for specifications on the scanners.

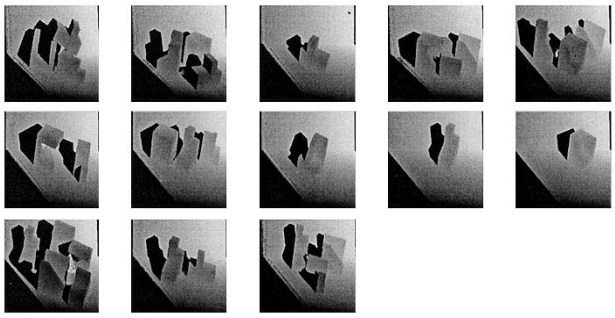


Fig. 9. Pool of 13 validation images of planar-surface scenes.

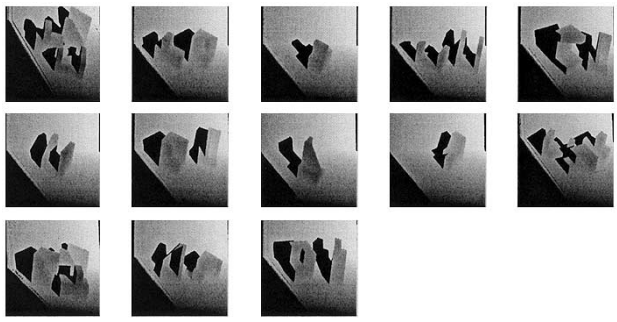


Fig. 10. Pool of 13 test images of planar-surface scenes.

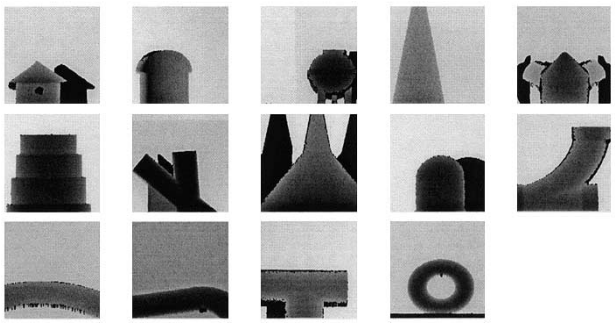


Fig. 11. Pool of 14 training images of curved-surface scenes.

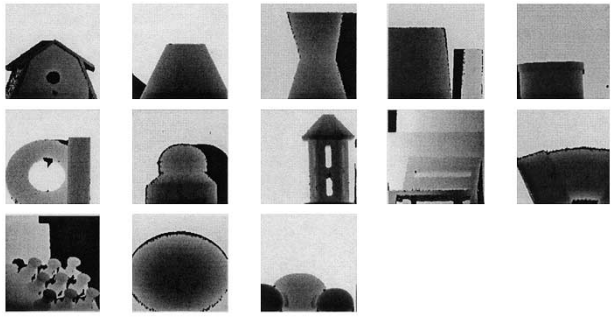


Fig. 12. Pool of 13 validation images of curved-surface scenes.

The framework is extensible in the sense that the number of images in the train, validation, and test sets can easily be increased, if desired. Increasing the number of images in the training, validation, or test pools requires additional experimental work in acquiring images and specifying the ground truth. Increasing the number of images in each training/validation/test set, or the number of sets used, translates into increasing the compute time required to use the framework.

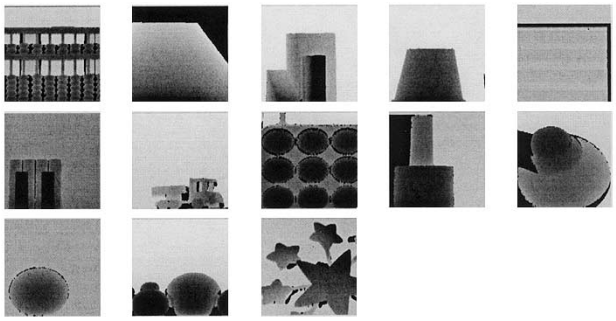


Fig. 13. Pool of 13 test images of curved-surface scenes.

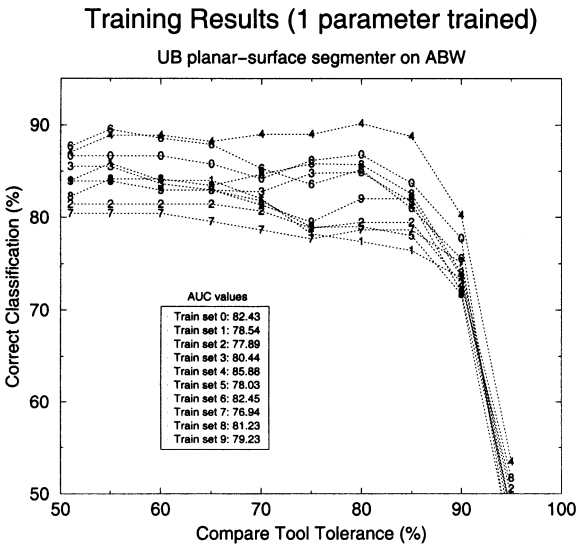


Fig. 14. Performance curves of UB planar-surface algorithm on the 10 training sets.

One possible motivation for increasing the size of the framework would be to make it possible to reliably measure smaller differences between algorithms. Possible motivations for changing the pools of images used might be to focus on a different type of range imaging technology, or a specific type of scene content.

B. Performance Results of the Baseline Algorithm

The first step in using the framework is to verify that it is installed correctly on the local system. This is done by running a script to train the baseline algorithm. This should reproduce the known training results for the baseline algorithm. The visible result of training either the baseline planar- or curved-surface algorithm is the set of performance curves shown in Figs. 14 or 15, respectively. Each curve in one of these figures represents segmenter performance on a different set of six images, with the segmenter parameter values trained for best performance on that set of images. For each curve, there is a corresponding area under the curve, and a corresponding set of trained parameter settings used to create the curve. The implementation checks the areas under the training curves against their known values to verify that the implementation is correctly installed. The area-under-the-curve values for the training of the baseline algorithms are listed in Table II. Values are listed for correct region segmentation for the baseline algorithm for planar-surface

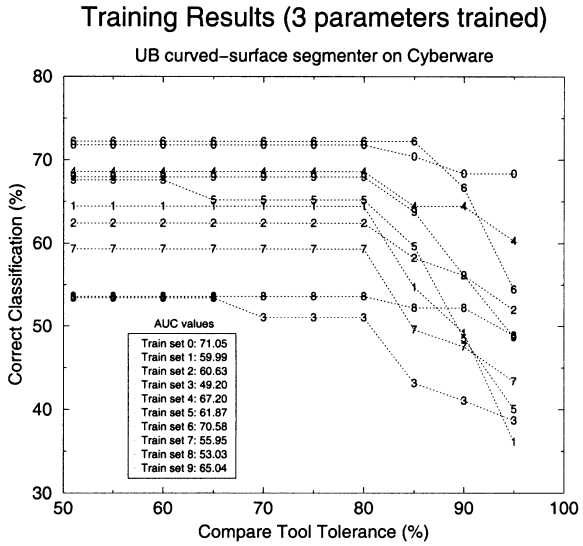


Fig. 15. Performance curves of UB curved-surface algorithm on the 10 training sets.

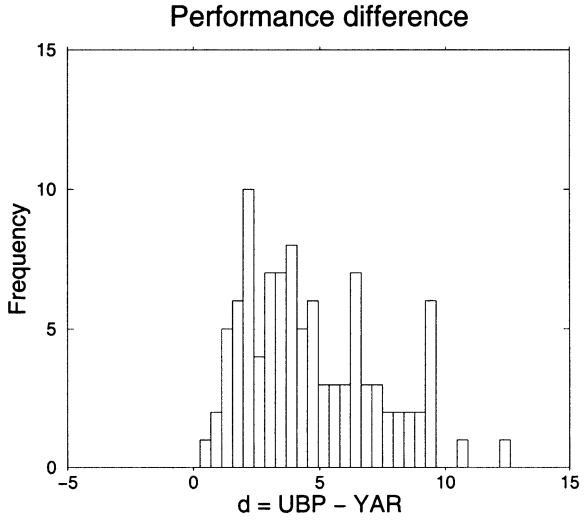


Fig. 16. Distribution of difference in test AUC values: (UB-YAR). Note that all of the differences are positive, indicating that the UB algorithm outperforms the YAR algorithm on each trial, and that the distribution does not appear Gaussian, as it has a long "tail" skewed toward the higher values.

TABLE II
AREAS UNDER THE 10 BASELINE ALGORITHM TRAINING CURVES

baseline algorithm on planar-surface scenes										
set	0	1	2	3	4	5	6	7	8	9
AUC	.82	.79	.78	.80	.86	.78	.82	.77	.81	.79
baseline algorithm on curved-surface scenes										
set	0	1	2	3	4	5	6	7	8	9
AUC	.71	.60	.61	.49	.67	.62	.71	.56	.53	.65

[16] and curved-surface [17] scenes, automatically trained on each of 10 different six-image training sets.

Once the local installation of the framework is verified to work correctly, it can be used in the development and comparison of a "challenger" algorithm. The first step is to train the challenger algorithm using the same process as used to reproduce the performance curves of the baseline algorithm. If the challenger algorithm does not show better performance on the training and validation sets than the baseline algorithm, then the next step is most likely to analyze the results and redesign the challenger algorithm. The framework provides information that should prove useful in re-design of algorithms. In addition to the performance curves for the number of correctly segmented regions, the framework also produces curves for instances of under-segmentation, over-segmentation, noise regions, and missed regions.

The performance curves shown for the curved-surface segmenter [17] in Fig. 15 appear somewhat "flat" over the range of about 0.51 to 0.8 for the overlap threshold. This is due to an interaction of the properties of the data set and the algorithm. In effect, many of the regions in the images are either "easy" or "hard" for the algorithm to segment correctly, regardless of the value of the overlap threshold. For example, a flat background region is typically segmented correctly over a broad range of the overlap threshold. Conversely, two separate quadratic patches that have a smooth join are difficult to segment correctly regardless of the overlap threshold. This problem is inherent to the algorithm using a general quadratic surface equation as its similarity predicate for defining a region.

C. Example Performance Comparison

As an example of comparing two segmentation algorithms, we step through a comparison of the yet another range (YAR)-segmentation algorithm [20] to the UB algorithm for segmenting planar-surface scenes. Fig. 16 shows a histogram of the 100 values of the difference in test AUC between the two algorithms. In general, a statistical test is needed to determine if the result is significant. However, in this particular comparison, the UB algorithm had at least a slightly higher AUC for each of the 100 paired values, and so the result of the statistical test is clear.

The test for significant difference in performance is also used during the training of the segmentation algorithm as part of determining the appropriate number of algorithm parameters to tune. The framework assumes that the parameters are specified in decreasing order of importance, and that each parameter has a default value. The training then begins with the first-specified parameter, and continues to add parameters to the training as long as there is a statistically significant increase in performance on the validation sets. Fig. 17 shows the distribution of AUC differences underlying the decision of whether or not to train the UB curved-surface segmenter on three or four parameters. This plot shows that the differences are centered near zero, with a few outliers on the negative side. Thus it is clear that the use of four parameters with the UB curved-surface segmenter offers no systematic performance advantage over the use of three parameters.

In general, under the null hypothesis, the challenger algorithm would be expected to show better performance than the baseline algorithm on fifty of the one hundred tests. The standard deviation would be $\sqrt{0.5 \times 0.5 \times 100} = 5$. Thus any result outside the range of forty to sixty (plus/minus two standard deviations from the mean) would provide evidence at the $\alpha = 0.05$ level of a statistically significant difference in performance between the baseline and challenger algorithms.

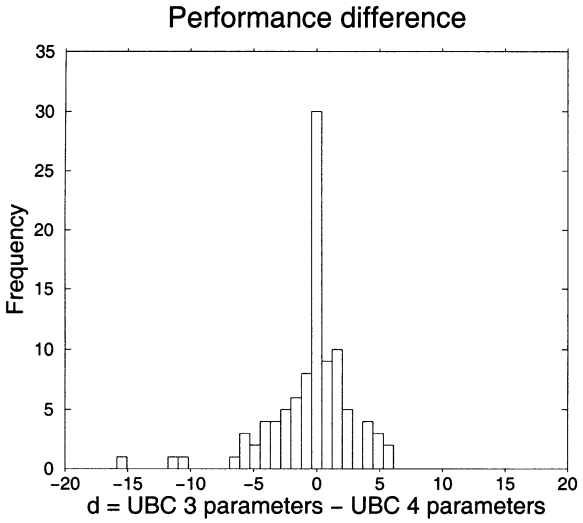


Fig. 17. Distribution of difference in validation AUC values: UB (3 parameter–4 parameter).

range-images-path :	ABW/
gt-images-path :	ABW/
ms-images-path :	ABW/
segmenter :	baseline/Segmenter
compare-tool :	train/compare
param-config-file :	baseline.paramconfig
number-of-train-sets :	10
number-of-val-sets :	10
train-image-set :	ABW/trnset
validation-image-set :	ABW/vldset
train-result-file :	baseline/trainresult
train-log-file :	baseline/trainlog
validation-result-file :	baseline/vldresult
stat-result-file :	baseline/statresult
eval-result-file :	baseline/evalresult

Fig. 18. Example of the contents of an evaluation configuration file.

D. Configuration Files and Execution

There are basically just two steps to configuring the framework for a challenger algorithm: a framework configuration file and a segmenter configuration file. The segmenter configuration file specifies the data type for the various parameters of the segmenter, the allowable range of values for each parameter, and its default value. The framework configuration file specifies the directories in which the various image data sets are located, along with the naming of the results files. See Fig. 18 for an example. Once the configuration files are created, the framework can be started up on one or more machines sharing the same distributed file system. If more than one machine is used, semaphore files are created to coordinate the execution of the segmenter on the various images.

VII. Conclusion

We have described an automated framework for objective performance evaluation of region segmentation algorithms. Parameters of a segmentation algorithm are automatically trained on a number of training sets. The number of segmenter parameters involved in the training is determined using validation sets. Finally, the benchmark performance of the algorithm is determined using separate test sets. The framework implementation comes with source code for baseline algorithms that segment range images into planar- and curved-surface regions. The baseline algorithms were shown in previous work to offer state-of-the-art performance [7], [8], [9]. A challenger algorithm's contribution in terms of improved experimental performance can be measured by improvement over the baseline algorithm's performance. The framework includes a statistical test for the significance of difference in performance between a challenger algorithm and the baseline algorithm.

There are several important contributions involved in the development of this framework. First, we have demonstrated that automated parameter tuning performs as well as manual tuning done by the algorithm developers. This should make it possible for algorithm tuning to be done in a more consistent and repeatable manner. Second, we have pointed out the need for using a validation set of images in order to avoid over-training on the number of parameters tuned to a value different from their default. From the perspective of sound pattern recognition methodology this may not be a "new" result, but it should be a "wake up call" for experimental methods in computer vision. Third, we have suggested an appropriate test for statistical significance of the performance difference between two segmenters. These contributions are implemented in a conceptual framework that can readily be extended in several directions. First, as mentioned, it is applicable to other types of imagery. For example, the set of texture images used in [21] might be used in the framework to evaluate texture-based segmentation algorithms. Also, the number of images and/or the number of training/validation/test sets can be increased. The current implementation should be more than sufficient for recognizing coarse-grain performance improvements, appropriate to the current state of the art in range image segmentation as demonstrated in [7]. However, as the general performance level of range image segmentation algorithms improves, it may become necessary to modify the framework by adding images of more and/or harder-to-segment scenes.

While we have focused on the performance metric for instances of correct segmentation, it is possible to also look at secondary error metrics such as over-segmentation, under-segmentation, missed and noise. The error metrics can be important, for example, in applications where the cost of the different types of errors varies significantly.

Lastly, we should emphasize that there are both short-term and long-term contributions to this work. Providing a means to compare the performance of existing algorithms is only the more visible short-term contribution. The longer term and more important contribution is to enable the design of better algorithms. Being able to measure the frequency of different types of errors in segmentation should make it possible to identify the important failure modes of existing algorithms. Once the failure modes of an existing algorithm are identified, it should be possible to design improved algorithms that address these failure modes. In this way, development of new algorithms can be guided by rigorous experimental performance data.

Acknowledgment

The authors would like to thank S. Sarkar and D. Goldgof, University of South Florida, and X. Jiang, Technical University of Berlin, for useful conversations about various elements of this work.

REFERENCES

[1] H. Christensen and W. Forstner, "Performance-characteristics of vision algorithms," *Mach. Vis. Applicat.*, vol. 9, no. 5–6, pp. 215–218, 1997.

[2] H. Christensen and P. Phillips, *Performance Characterization in Computer Vision*. Singapore: World Scientific, 2002.

[3] K. Bowyer and P. Phillips, *Empirical Evaluation Techniques in Computer Vision*. Los Alamitos, CA: IEEE Comp. Soc. Press, 1998.

[4] M. Viergever, *Performance Characterization and Evaluation of Computer Vision Algorithms*. Boston, MA: Kluwer, 2000.

[5] P. Courtney, N. Thacker, and A. Clark, "Algorithmic modeling for performance evaluation," *Mach. Vis. Applicat.*, vol. 9, pp. 219–228, 5–6, 1997.

[6] M. Heath, S. Sarkar, T. Sanocki, and K. Bowyer, "A robust visual method for assessing the relative performance of edge detection algorithms," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, pp. 1338–1359, Dec. 1997.

[7] A. Hoover, G. Jean-Baptiste, X. Jiang, P. Flynn, H. Bunke, D. Goldgof, K. Bowyer, D. Eggert, A. Fitzgibbon, and R. Fisher, "An experimental comparison of range image segmentation algorithms," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18, pp. 673–689, July 1996.

[8] M. Powell, K. Bowyer, X. Jiang, and H. Bunke, "Comparing curved-surface range image segmenters," in *Proc. Int. Conf. Comp. Vision*, 1998, pp. 286–291.

[9] X. Jiang, K. Bowyer, Y. Morioka, S. Hiura, K. Sato, S. Inokuchi, M. Bock, C. Guerra, R. Loke, and J. du Buf, "Some further results of experimental comparison of range image segmentation algorithms," in *Proc. Int. Conf. Pattern Recognition*, vol. IV, 2000, pp. 877–881.

[10] J. Min, M. Powell, and K. Bowyer, "Automated performance evaluation of range image segmentation," in *Workshop Applicat. Comp. Vision*, 2000, pp. 163–168.

[11] ——, "Progress in automated evaluation of curved surface range image segmentation," in *Proc. Int. Conf. Pattern Recog.*, vol. I, 2000, pp. 644–647.

[12] X. Jiang, H. Bunke, and U. Meier, "High-level feature based range image segmentation," *Image Vis. Comput.*, vol. 18, no. 10, pp. 817–822, July 2000.

[13] M. R. Everingham, H. Muller, and B. T. Thomas, "Evaluating image segmentation algorithms using the Pareto front," in *Proc. 7th Eur. Conf. Comp. Vision (ECCV2002), Part IV (LNCS 2353)*, June 2002, pp. 34–48.

[14] K. Bowyer, "Validation of medical image analysis techniques," in *Handbook of Medical Imaging, Vol. 2: Medical Image Processing and Analysis*. New York: SPIE, 2000.

[15] L. Cinque, R. Cucchiara, S. Levialdi, S. Martinz, and G. Pignalberi, "Optimal range segmentation parameters through genetic algorithms," in *Proc. Int. Conf. Pattern Recog.*, vol. I, 2000, pp. 474–477.

[16] X. Jiang and H. Bunke, "Fast segmentation of range images by scan line grouping," *Machine Vision Applicat.*, vol. 7, no. 2, pp. 115–122, 1994.

[17] X. Jiang, "An adaptive contour closure algorithm and its experimental evaluation," *IEEE Trans. Pattern Anal. Machine Intel.*, vol. 22, pp. 1252–1265, Nov. 2000.

[18] M. Shin, D. Goldgof, K. Bowyer, and S. Nikiforou, "Comparison of edge detection algorithms using a structure from motion task," *IEEE Trans. Syst., Man Cybern. B*, vol. 31, pp. 589–601, Aug. 2001.

[19] M. Bland, *An Introduction to Medical Statistics*. London, U.K.: Oxford Univ. Press, 1995.

[20] A. Hoover, "The space envelope representation of 3-D scenes," Ph.D. dissertation, Univ. South Florida, Tampa, 1996.

[21] K. Chang, K. Bowyer, and M. Sivagurunath, "Evaluation of texture segmentation algorithms," *Comput. Vis. Pattern Recognit.*, pp. I:294–299, 1999.

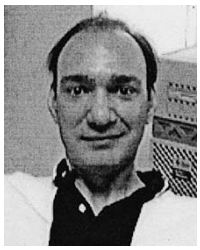
Jaesik Min (M'02) received the B.S. degree in mathematics from Seoul National University, Seoul, Korea, in 1990, the M.S. degree in computer science from the Pohang University of Science and Technology (POSTECH), Korea, in 1992, and the Ph.D. degree in computer science and engineering from the University of South Florida, Tampa, in 2002.

From 1992 to 1998, he was a Research Engineer at the Research Center, LG Electronics, Korea. He is a Postdoctoral Research Associate in the Department of Computer Science and Engineering, University of Notre Dame. His main research interests focus on computer vision, pattern recognition, and biometrics.

Dr. Min is a member of the IEEE Computer Society.

Mark Powell (M'97) received the B.Sc., M.Sc., and Ph.D. degrees in computer science and engineering the University of South Florida, Tampa, in 1992, 1997, and 2000, respectively.

He is a Member of Technical Staff, Mobility Systems Concept Development Section, Jet Propulsion Laboratory (JPL), Pasadena, CA, since 2001. His dissertation work was in the area of advanced illumination modeling, color and range image processing applied to robotics, and medical imaging. At JPL his area of focus is science data visualization and science planning for telerobotics. He is currently serving as a Software and Systems Engineer, contributing to the development of science planning software for the 2003 Mars Exploration Rovers and the JPL Mars Technology Program Field Integrated Design and Operations (FIDO) rover task.



Kevin W. Bowyer (F'97) currently serves as the Schubmehl-Prein Department Chair of the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN. He was North American Editor of the *Image and Vision Computing Journal*, and currently serves on the editorial boards of *Computer Vision and Image Understanding*, *Image and Vision Computing Journal*, *Machine Vision and Applications*, and the *International Journal of Pattern Recognition and Artificial Intelligence*. His research has been supported by grants from the National Science Foundation, Air Force Office of Scientific Research, Army Medical Research and Material Command, National Aeronautics and Space Administration, Sandia National Laboratories, Defense Advanced Research Projects Agency, and other institutions. He is author of the textbook *Ethics and Computing—Living Responsibly in a Computerized World* (New York: Wiley, 2001).

Dr. Bowyer received the *Outstanding Undergraduate Teaching Award* from the USF College of Engineering in 1991 and the *Teaching Incentive Program Awards* in 1994 and 1997. He was Editor-in-Chief of the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE.