Predicting Deviations in Software Quality by Using Relative Critical Value Deviation Metrics

Norman F. Schneidewind, Ph.D.
Division of Computer and Information
Sciences and Operations
Naval Postgraduate School
Monterey, CA 93943
Email: nschneid@nps.navy.mil

Allen P. Nikora, Ph.D.
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109-8099
Email: Allen.P.Nikora@jpl.nasa.gov

Abstract

We develop a new metric, Relative Critical Value Deviation (RCVD), for classifying and predicting software quality. The RCVD is based on the concept that the extent to which a metric's value deviates from its critical value, normalized by the scale of the metric, indicates the degree to which the item being measured does not conform to a specified norm. For example, the deviation in body temperature above 98.6 Fahrenheit degrees is a surrogate for fever. Similarly, the RCVD is a surrogate for the extent to which the quality of software deviates from acceptable norms (e.g., zero discrepancy reports). Early in development, surrogate metrics are needed to make predictions of quality before quality data are available. The RCVD can be computed for a single metric or multiple metrics. Its application is in assessing newly developed modules by their quality in the absence of quality data. The RCVD is a part of the larger framework of our measurement models that include the use of Boolean Discriminant Functions for classifying software quality. We demonstrate our concepts using Space Shuttle flight software data.

Keywords: Quality classification and prediction, relative critical value deviation metrics.

1. Introduction

Our goal is to provide models and processes to assist software managers in answering the following questions:

- How can I control the quality of my software?
- How can I predict the quality of my software?
- How shall I prioritize my effort to achieve my quality goals?
- How can I determine whether my quality goals are being met?
- How much will it cost to achieve my quality goals?

We develop quality control and prediction models that are used to identify modules requiring priority attention during development and maintenance. This is accomplished in two activities: *validation* and *application*. During *validation*, we use a build of the software that has been developed as the source of data to compute Boolean Discriminant Functions (BDFs), Relative Critical Value Deviation (RCVD) metrics, and regression equations that we use to retrospectively classify and predict quality with specified accuracy, by build and module. Using these functions and equations during *application*, we classify and predict the quality of new software that is being developed. This is the quality we expect to achieve during maintenance. During *validation*, both quality factor (e.g., discrepancy reports of deviations between requirements and implementation) and software metrics (e.g., size, structural) data are available; during *application*, only the latter are available. During *validation*, we construct Boolean discriminant functions (BDFs) comprised of a set of metrics and their critical values (i.e., thresholds) [1, 2]. We select the best BDF based on its ability to achieve the maximum relative incremental quality/cost ratio. During *application*, if at least one of the module's metrics has a value that exceeds its critical value, the module is identified as "high priority" (i.e., low quality); otherwise, it is identified as "low priority" (i.e., high quality). Our objective is to identify and correct quality problems during development, as opposed to waiting until maintenance when the cost of correction would be high. This process addresses the question: "How can I control the quality of my software?" Because BDFs only provide an *accept/reject* decision on module quality, during validation, we also construct RCVDs that are used to prioritize the effort applied to rejected modules. In other words, an RCVD measures the *degree* to which quality is low. This process addresses the question: "How shall I prioritize my effort to achieve my quality goals?

A RCVD is a derived metric, based on the normalized deviation between a metric's value and its critical value. It may be based on a single or multiple metrics. In our process, we: 1) identify the critical values of the metrics and 2) find the optimal BDF and RCVD based on their ability to

satisfy both *statistical* and *application* criteria. Statistical criteria refer to the ability to correctly classify the software (i.e., classify high quality software as high quality and low quality software as low quality). Application criteria refer to the ability to achieve a high quality/cost ratio. This process addresses the questions: "How can I determine whether my quality goals are being met?" and "How much will it cost to achieve my quality goals?"

RCVD values that exceeded the .80 percentile value were able to account for two-thirds of the discrepancy reports. To round out our approach, we use regression equations to predict quality limits. This is desirable because, although BDFs and RCVDs control and predict quality based on expected values, they are not capable of predicting the range of quality values.

We show that it is important to perform a marginal analysis (i.e., identification of the incremental contribution of each metric to improving quality) when making a decision about how many metrics to include in the BDFs and RCVDs. If many metrics are added to the set at once, the contribution of individual metrics is obscured. Also, the marginal analysis provides an effective rule for deciding when to stop adding metrics.

The contributions of this research are the following: 1) the Relative Critical Value Deviation (RCVD) is a new metric for classifying and predicting software quality; 2) the RCVDs in combination with the BDFs we previously developed, allow the software manager to both control quality and prioritize the effort required to achieve quality goals; 3) BDFs, RCVDs, and regression equations are *integrated* into a process to assist the software manager in answering the questions posed in the introduction; and 4) the data and most of the calculations are implemented in a spreadsheet for easy transfer to practitioners.

1.1 Related Research

Our models are in the class of models concerned with the classification, control, and prediction of quality. Other researchers have had similar objectives but different approaches. Porter and Selby used classification trees to partition multiple metric value space so that a sequence of metrics and their critical values could be identified that were associated with either high quality or low quality software [3]. This technique is closely related to our approach of identifying a set of metrics and their critical values that will satisfy quality and cost criteria. However, we use statistical analysis to make the identification.

Briand et al. used logistic regression to classify modules as fault-prone or not fault-prone as a function of various object oriented metrics [4]. In another example of logistic regression, Khoshgoftaar and Allen used it to classify modules as fault-prone or not fault-prone as a function of faults, requirements, performance, and documentation software trouble report metrics [5]. While one of our objectives is similar -- classify modules as either high quality or low quality -- we derive from this *binary* classification several predictive *continuous* quality and cost metrics, including the RCVDs. These metrics are used to predict the quality of software that will be delivered by development to maintenance and the cost of achieving it.

Khoshgoftaar et al. used nonparametric discriminant analysis in each iteration of a military system project to predict fault-prone modules in the next iteration [6]. This approach provided early indication of reliability and the risk of implementing the next iteration. They conducted a similar study involving a telecommunications application, again using nonparametric discriminant analysis, to classify modules as either fault-prone or not fault-prone [7]. Our approach has the same objective but we produce BDFs and RCVDs in terms of the original metrics as opposed to using density functions as discriminators.

Khoshgoftaar and Allen have also developed models for ranking modules for reliability improvement according to their degree of fault-proneness as opposed to whether they are fault-prone or not [8]. They used Alberg Diagrams [9] that predict percentage of faults as a function of percentage of modules by ordering modules in decreasing order of faults and noting the cumulative number of faults corresponding to various percentages of modules. Our approach is similar but we accomplish the same objective by sorting the modules by RCVD and finding its percentile distribution and the corresponding *drcount* percentile distribution, as we explain later.

2. Discriminative Power Model

2.1. Discriminative Power Validation

Using our metrics validation methodology [10, 11], and the *Space Shuttle* flight software metrics and discrepancy reports (DRs), we validate metrics with respect to the quality factor *drcount*. This is the number of discrepancy reports written against a module. In brief, this involves conducting statistical tests to determine whether there is a high degree of association between *drcount* and candidate metrics. As shown in Figure 1, we validate metrics on Build 1 (1397 modules) and apply them to Build 2 (846 modules) of the *Space Shuttle* flight software. Nikora and Munson argue for the need of a measurement baseline against which evolving systems may be compared [12]. Our baseline is Build 1 in Figure 1. The measurement results from Build 1 provide the data source for controlling and predicting the quality delivered to maintenance and for comparing predicted with actual quality, once the latter is known. Next, we define *Discriminative Power*.

2.1.1. Discriminative Power

Given the elements M_{ij} of a matrix of **n** modules and **m** metrics (i.e., **nm** metric values), the elements MCj of a vector of **m** metric critical values, the elements F_i of a vector of **n** quality factor values, and scalar FC of quality

factor critical value, M_{ij} must be able to discriminate with respect to F_i, for a specified FC, as shown below:

$$M_{ij} > M_j \leftrightarrow F_i > FC \text{ and } M_{ij} \leq M_j \leftrightarrow F_i \leq FC \quad (1)$$

for i=1,2,...,n, and j=1,2,...,m with specified α, where α is the significance level of various statistical tests that are used for estimating the degree to which a set of metrics can correctly classify software quality. In other words, do the indicated metric relations imply corresponding quality factor relations in (1)? This criterion assesses whether MC_j has sufficient *Discriminative Power* to be capable of distinguishing a set of high quality modules from a set of low quality modules. If so, we use the critical values in Quality Control and Prediction described below. The validation process is illustrated in Figure 1, where the critical values MC_j are produced during the Test phase of Build 1 by using the metrics M_{ij} from the Design phase and the quality factor F_i (e.g., *drcount*) available in the Test phase. (Discrepancy Reports are written against the software throughout development but they are not significantly complete until the end of the Test phase during which failures are observed). The desired quality level is set by the choice of FC. The lower its value, the higher the quality requirement; conversely, the higher its value, the lower the requirement. A value of zero is appropriate for safety-critical systems like the *Space Shuttle*.

2.2. Relative Critical Value Deviation (RCVD) Metric

The RCVD is based on the concept that the extent to which a metric's value deviates from its critical value, normalized by the scale of the metric, is an indicator of the degree to which the entity being measured does not conform to a specified norm. For example, the extent to which body temperature exceeds 98.6 degrees Fahrenheit is an indicator of the deviation from an established norm of human health. Measurement involves using surrogates: the deviation in temperature above 98.6 degrees is a surrogate for fever. Similarly, the RCVD is a surrogate for the extent that software quality deviates from acceptable norms (e.g., zero discrepancy reports). The concept of the RCVD is shown in Figure 2, where the metric and quality scales are shown, defined by the maximum (MX_j) and minimum (MN_j) metric boundaries and the maximum (FX) and minimum (FN) quality boundaries, respectively. The theory of the RCVD is given by the following relation:

$$RCVD_{ij} = (M_{ij} - MC_j)/(MX_j - MN_j) \leftrightarrow (F_i - FC)/(FX - FN) \quad (2)$$

This means that the deviation of a metric from its critical value, normalized by metric length, is related to the degree of quality, as represented by the normalized deviation of a quality factor (e.g., *drcount*) from its critical values: increasing positive deviations are related to decreasing quality and increasing negative deviations are related to increasing quality. It should not be inferred that the relationship is linear or proportional; in fact, it is non-linear. In the idealized diagram in Figure 2, the worst quality corresponds to MX_j and FX, the best quality to MN_j and FN, and acceptable quality to MC_j and FC. Also, Figure 2 does not indicate the mathematical form of F_i. If FN is equal to zero and F_c is set equal to zero, which is frequently the case, F_i and FX can be replaced by the sum of the quality factor across a set of modules and the total quality factor, respectively. This quantity is the proportion of *drcount* computed across a set of modules. An RCVD can also be comprised of multiple metrics by computing their mean. Note that although it would not be valid to compute the mean of metrics, the mean of RCVDs is another story since these are normalized dimensionless quantities. We experimented with both single and multiple metric RCVDs, as we explain later.

2.3. Quality Control and Prediction

Quality control is the evaluation of modules with respect to predetermined critical values of metrics. The purpose of quality control is identify software that does not meet quality requirements early in the development process so corrective action can be taken when the cost is low. Quality control is applied during the Design phase of Build 2 in Figure 1 to flag software for detailed inspection that is below quality limits. The validated BDFs, comprised of the metrics M_{ij} and their critical values MC_j that are obtained from Build 1, are used to either accept or reject the modules of Build 2 [1, 2]. At this point during the development of Build 2, only the metric data M_{ij} and MC_j are available. The validated RCVDs are used to prioritize the attention and effort devoted to modules that are rejected by the BDFs. Details are given later.

Quality predictions are used by the developer to anticipate rather than react to quality problems. Figure 1 shows the metrics controlling and predicting the quality of software that will be delivered to maintenance *early* in the development of Build 2. Accompanied by rigorous inspection and test, this process will result in improved quality of Build 2 and the software that is released to maintenance. Once all of the quality factor data F_i (e.g., *drcount*) have been collected for Build 2, at the end of the Test phase as shown in Figure 1, the quality of Build 2 would be known. This, then, becomes the actual quality of Build 2 in the maintained software. Regression equations $F_i=f(M_{ij})$ are developed during the Test phase of Build 1 and applied to predicting quality limits during the Design Phase of Build 2, as shown in Figure 1. This process addresses the question: "How can I predict the quality of my software?"

3. Validation Methodology

We use a five stage process to select metrics and metric functions for quality control and prediction: 1) com-

pute critical values of the candidate metrics; 2) for the set of candidate metrics and critical values, find the optimal BDF based on statistical and application criteria; 3) apply a stopping rule for adding metrics; 4) identify the best RCVD for prioritizing quality assurance effort; and 5) develop a regression equation that will accurately predict quality limits (e.g., limits of *drcount*). Table 1 provides a functional description of each stage. The five stages take place during the Test Phase of Build 1 of Figure 1, once all the quality factor data F_i (e.g., *drcount*) are available. The next sections describe the analysis for each stage.

3.1. Stage 1: Compute Critical Values

Critical values MC_j are computed based on the Kolmogorov-Smirnov (K-S) test [1, 2]. Table 1 shows the metric definitions, critical values MC_j, and K-S distances for six metrics of Build 1. These metrics were selected based on their relatively high K-S distance compared to other metrics that had been collected on the *Space Shuttle*. The test statistic is the maximum vertical difference between the CDFs of two complementary sets of data (e.g., the CDFs of M_{ij} for *drcount*≤FC and *drcount*>FC). If the difference is significant (i.e., $\alpha \leq .005$), the value of M_{ij} corresponding to maximum CDF difference is used for MC_j. This relationship is expressed in equation (3). Metrics are added to the BDF in order of their K-S Distance.

$$K\text{-}S(MC_i) = \max\left\{\left[CDF\ (M_{ij}/(F_i \leq FC))\right] - \left[CDF\ (M_{ij}/(F_i > FC))\right]\right\} \quad (3)$$

3.2. Stage 2: Form a Set of Boolean Discriminate Functions (BDFs)

For each BDF identified in Stage 1 we use Table 2 to further evaluate the ability of the functions to discriminate high quality from low quality, from both statistical (e.g., misclassification rates) and application (e.g., ability of the metric set to correctly classify low quality modules) standpoints. In Table 2, MC_j and FC classify modules into one of four categories. The left column contains modules where none of the metrics exceeds its critical value; this condition is expressed with a Boolean AND function of the metrics. This is the *ACCEPT* column, meaning that according to the classification decision made by the metrics, these modules have acceptable quality. The right column contains modules where at least one metric exceeds its critical value; this condition is expressed by a Boolean *OR* function of the metrics. This is the *REJECT* column, meaning that according to the classification decision made by the metrics, these modules have unacceptable quality. The top row contains modules that are high quality; these modules have a quality factor that does not exceed its critical value (e.g., *drcount*=0). The bottom row contains modules that are low quality; these modules have a quality factor that exceeds its critical value (e.g., *drcount*>0).

Equation (4) gives the algorithms for making the cell counts, using the BDFs of F_i and M_{ij} that are calculated over the n modules for m metrics. This equation is an implementation of the relation given in (1).

$$\begin{aligned}
C_{11} &= \underset{i=1}{\overset{n}{\text{COUNT}}} \text{ FOR } ((F_i \leq FC) \wedge (M_{i1} \leq MC_1) \ldots \wedge (M_{ij} \leq MC_j) \ldots \wedge (M_{im} \leq MC_m)) \\
C_{12} &= \underset{i=1}{\overset{n}{\text{COUNT}}} \text{ FOR } ((F_i \leq FC) \wedge ((M_{i1} > MC_1) \ldots \vee (M_{ij} > MC_j) \ldots \vee (M_{im} > MC_m))) \\
C_{21} &= \underset{i=1}{\overset{n}{\text{COUNT}}} \text{ FOR } ((F_i > FC) \wedge (M_{i1} \leq MC_1) \ldots \wedge (M_{ij} \leq MC_j) \ldots \wedge (M_{im} \leq MC_m)) \\
C_{22} &= \underset{i=1}{\overset{n}{\text{COUNT}}} \text{ FOR } ((F_i > FC) \wedge ((M_{i1} > MC_1) \ldots \vee (M_{ij} > MC_j) \ldots \vee (M_{im} > MC_m)))
\end{aligned} \quad (4)$$

for j=1,...,m, and where COUNT(i)=COUNT(i-1)+1 FOR Boolean expression *true* and COUNT(i)=COUNT(i-1), otherwise; COUNT(0)=0. The counts (C_{11}, C_{12}, C_{21}, and C_{22}) correspond to the cells of Table 2, where row and column totals are also shown: n, n_1, n_2, N_1, and N_2.

In addition to counting modules in Table 2, we must also count the quality factor (e.g., *drcount*) that is incorrectly classified. This is shown as Remaining Factor, RF, in the *ACCEPT* column. This is the quality factor count on modules that should have been rejected. Also shown is Total Factor, TF, the total quality factor count on all the modules in the build. Table 2 and subsequent equations show an example validation, where the combination of metrics from Table 1 and their critical values for Build 1 is *prologue size* (P) with a critical value of 63, *statements* (S) with a critical value of 27, and *eta2* (E2) with a critical value of 45. This is the optimal BDF. Later we will explain how we arrived at this particular combination of metrics as the optimal set. The results of the following calculations for the optimal BDF are shown in Table 3.

3.2.1. Statistical Criteria

We validate a BDF statistically by demonstrating that it partitions Table 2 so that C_{11} and C_{22} are large relative to C_{12} and C_{21}. If this is the case, a large number of high quality modules (e.g., modules with *drcount*=0) would have $M_{ij} \leq MC_j$ and would be correctly classified as high quality. Similarly, a large number of low quality modules (e.g., modules with *drcount*>0) would have $M_{ij} > MC_j$ and would be correctly classified as low quality. We evaluate partitioning ability using the misclassification rates.

3.2.2. Misclassification

We compute the degree of misclassification in Table 2 by noting that ideally $C_{11}=n_1=N_1$, $C_{12}=0$, $C_{21}=0$, $C_{22}=n_2=N_2$. The extent to which this is not the case is estimated by *Type 1* misclassifications (i.e., the module has *Low Quality* and the metrics "say" it has *High Quality*) and *Type 2* misclassifications (i.e., the module has *High Quality* and the metrics "say" it has *Low Quality*). Thus, we define the following measures of misclassification:

$$\text{Proportion of Type 1: } P_1 = C_{21}/n \quad \text{For the example, } P_1 = (35/1397)*100 = 2.51\% \tag{5}$$

$$\text{Proportion of Type 2: } P_2 = C_{12}/n \quad \text{For the example, } P_1 = (344/1397)*100 = 24.62\% \tag{6}$$

3.2.3. Application Criteria

Because it is the performance of the metrics in the application context that counts, we also validate metrics with respect to the application criteria *Quality and Inspection*, which are related to quality achieved and the cost to achieve it, respectively [1, 2]. During the Design phase of Build 2 in Figure 1, we predict that the quality computed by equations (7)--(9) will be delivered to maintenance, assuming that the modules rejected by the quality control process are inspected and tested and that the problems that are found are corrected. Furthermore, we predict that the degree of inspection computed by equation (10) will be required to achieve this quality. In addition to controlling and predicting quality, equations (7)--(9) can be used to address the question: "How can I determine whether my quality goals are being met?" For example, if a quality goal is ≤3% residual defects, the achievement of this goal can be measured by RFP -- equation (9). Also, the degree of rigorous inspection -- equation (10) can be used to address the question: "How much will it cost to achieve my quality goals?"

3.2.4. Quality

First, we estimate the metrics' ability to correctly classify quality, given that the quality is known to be low:

$$\text{LQC: proportion of low quality (e.g., drcount} > 0) \quad \text{software correctly classified} = C_{22}/n_2 \tag{7}$$

For the example, LQC=(541/576)*100=93.92%.

Second, we estimate the metrics' ability to correctly classify quality, given that the BDF has classified modules as *ACCEPT*. This is done by summing quality factor in the *ACCEPT* column in Table 2 to produce Remaining Factor, RF (e.g., remaining *drcount*), given by equation (8).

$$RF = \sum_{i=1}^{n} F_i \text{ FOR } ((F_i > FC) \wedge (M_{i1} \le MC_1) \ldots \wedge (M_{ij} \le MC_j) \ldots \wedge (M_{im} \le MC_m)) \tag{8}$$

for j=1,...,m. This is the sum of F_i (e.g., *drcount*) on modules incorrectly classified as high quality because, for these modules, $(F_i>FC)\wedge(M_{ij}\le MC_j)$.

We estimate the proportion of RF by equation (9), where TF is the total F_i for the build.

$$RFP = RF/TF \tag{9}$$

For the example, from Table 2 there are 56 DRs on 35 modules that are incorrectly classified (i.e., RF=56). The total number of DRs for the 1397 modules is 2579. Therefore, RFP=(56/2579)*100=2.17%.

3.2.5. Inspection

Inspection is one of the costs of high quality. We are interested in weighing inspection requirements (i.e., percent of modules rejected and subjected to detailed inspection) against the quality that is achieved, for various BDFs. We estimate inspection requirements by noting that all modules in the *REJECT* column of Table 2 must be inspected; this is the count $C_{12}+C_{22}$. Thus, the proportion of modules that must be inspected is given by:

$$I = (C_{12} + C_{22})/n \tag{10}$$

For the example, I=((344+541)/1397)*100=63.35% and the percentage accepted is 1-I = 36.65%.

3.2.6. Summary of Validation Results

Table 3 summarizes the results of the validation example. The properties of *dominance* and *concordance* are evident in these validation results and in other data we have analyzed from the *Space Shuttle*. That is, a point is reached in adding metrics where *Discriminative Power* is not increased because: 1) the contribution of the dominant metrics in correctly classifying quality has already taken effect and 2) additional metrics essentially replicate the classification results of the dominant metrics -- the *concordance* effect. This result is due to the property of the BDF used as an OR function, causing a module to be rejected if only one of its metrics exceeds its critical value.

3.3. Stage 3: Apply a Stopping Rule for Adding Metrics

It is important to strike a balance between quality and cost (i.e., between RFP and I). Thus we add metrics until the ratio of the relative change in RFP to the relative change in I is maximum, as given by the *Quality Inspection Ratio* in equation (11), where *i* refers to the previous RFP and I:

$$QIR = (|\Delta RFP|/RFP_i)/(\Delta I/I_i) \tag{11}$$

For the example, QIR(P,S→P,S, E2)= ((|.2.17-2.95|)/2.95)/((63.35-60.13)/60.13)=4.91. Therefore, we stop adding metrics after *eta2* (E2) has been added.

3.3.1. Comparison of BDF Validation with Application Results

In order to compare validation with application results, we first show how BDF Table looks in the Design phase of Build 2 in Figure 1, when only the metrics M_{ij} and their critical values MC_j are available. This is shown in Table 4, where the "?" indicates that the quality factor data F_i are not available when the validated metrics are used in the quality control function of Build 2. During the Design phase of Build 2, modules are classified according

to the criteria that have been described. Whereas 36.65% (512/1397) and 63.35% (885/1397) modules were accepted and rejected, respectively, during Build 1 (see Table 2), 26.95% (228/846) and 73.05 % (618/846) modules were accepted and rejected, respectively, during Build 2 (see Table 4). The rejected modules would be given priority attention (i.e., subjected to rigorous inspection).

A comparison of the Validation (Build 1) with the Application (Build 2) with respect to statistical and application criteria are shown in Table 5. To have a basis for comparison with the validation results, we computed the values shown in Table 5 retrospectively (i.e., after Build 2 was far enough along to be able to collect all of the quality factor data at the conclusion of the Test phase). The values for Build 2 are the actual quality delivered to maintenance, as shown during the Test phase of Figure 1. The results of the two builds are comparable. Note that the same critical values computed during Build 1 were used on Build 2. This procedure is necessary because the quality factor data that is used in the K-S test in Stage 1 is not available during the Design Phase of Build 2 in Figure 1. This transferability of model parameters is key to our process because the point of validation is to apply its results to other but similar software when the quality factor data is not available for the latter. Also, we have found that to apply this approach, Build 2 does not have to be a direct descendant of Build 1. Builds 1 and 2 do not have this relationship.

3.4. Stage 4: Form a Set of Relative Critical Value Metrics (RCVD)

Granularity of data is an issue that does not seem to have been discussed much in the literature but one that we have found to be of great importance in metrics analysis. By granularity we refer to the level of data (e.g., module, module sets, build) that will yield useful results when the data are used in a model. This was an issue in our research to develop an RCVD suitable for use as a second level discriminant in controlling and predicting quality. By second level we mean that the RCVD comes into play after the optimal BDF has done its job of either accepting or rejecting a module. Although the BDF is very useful, it does not indicate the *degree* of quality (e.g., number of DRs) on a rejected module or set of rejected modules. Our original objective was to provide discrimination at the module level (i.e., rank the *drcount* in modules by RCVD). Due to the large number of modules with zero DRs (58.77% and 50.59% for Build 1 and Build 2, respectively) and the large variability of the data, this did not prove feasible. However, by sorting the modules by RCVD and finding its percentile distribution and the corresponding *drcount* percentile distribution, we were able to identify key points in the plots of these distributions. We call these points *break points*. These are points in the percentile distributions where the slope of the percentile curve starts to increase sharply. An example is shown in Figure 3, where percentile *drcount* is plotted against percentile *prologue size*. A break point occurs at .80 percentile (80%) on the X-axis. This corresponds to RCVD (*prologue size*)=0.517. This value corresponds to a Y-axis value of .35 (35%). Thus for values of RCVD greater than .0517, we estimate that the RCVD would identify 65% of the *drcount*. Thus we see that a difference of only .20 percentile (1.00-.80) of the RCVD accounts for a difference in .65 percentile (1.00-.35) of the *drcount*. In order to implement this process, we validate function (12) for sets of metrics during the Test Phase of Build 2, in Figure 1, when the quality factor data F_i are available. Then we apply function (12) during the Design Phase of Build 2, when no quality factor date is available for Build 2.

$$\vee (M_{ij} > MC_j) \wedge RCVD_{jj} \qquad (12)$$

This means that in addition to rejecting modules -- the function performed by the BDF -- there is further classification performed by the RCVD. Any modules that evaluate to *true* in (12), would receive special attention because the likelihood is that they would contain multiple DRs. This is illustrated in Table 6 where 65.37% of the *drcount* is identified by RCVD (*prologue size*) in combination with the BDF on Build 1, corresponding to a *drcount* density of 6.08. This is in contrast with a density of .80 on modules where (12) does not evaluate to *true* and 2.85 when the BDF alone is used. Similar results are observed for Build 2 in Table 6. These results indicate the quality that would be delivered to maintenance unless action is taken in inspection and test to correct the defects.

We experimented with using all six metrics of Table 1 in the RCVD. We used all six in order to have sufficient data to make the computation feasible. $\overline{\text{RCVD}}$ was worse than RCVD (*prologue size*), as can be seen in Table 6, in terms of both percentage of *drcount* classified and *drcount* density. Since RCVD (*prologue size*) is much easier to compute, it was the preferred RCVD to apply to Build 2, as shown in Table 6. This result is due to the *dominance* and *concordance* properties of metrics mentioned earlier. In addition, the result is due to the fact that *prologue size* contains a thorough change history comprised of the following notations in the program listing: module; purpose of the module; specification reference; change request; discrepancy report; release; release date; revision level; programmer; description of change; listing of statements affected by the change; indication of whether a statement is added, deleted, or changed; and program comments. We use *prologue size* as a predictor of *drcount* in the *aggregate* (i.e., the cumulative quantity of entries in the program), not on a one-for-one basis of a change possibly resulting in a DR.

A seemingly trivial but yet important aspect of this stage of the analysis was demonstrating the usefulness of sorting data to examine their distributions and the flexibility for doing this provided by a spreadsheet program.

3.5. Stage 5: Identify Quality Limit Predictors

The final stage of the analysis involves identifying regression equations for predicting the average and limits of quality (e.g., *drcount*) of module sets, $F_i=f(M_{ij})$, during the Test Phase of Build 1, as shown in Figure 1. This process is desirable because BDFs and RCVDs are not capable of predicting quality limits. During the Test phase of Build 1, regression coefficients are estimated and the resultant equation is applied, during the Design Phase of Build 2, to predict the quality limits that would be delivered to maintenance unless action is taken to correct the defects. As in the case of forming the RCVDs, granularity of data was an issue. Again, because of the large number of modules with zero *drcount* and the large variability of the data, prediction at the individual module level was not feasible. However, applying our earlier regression work for the *Space Shuttle* [13], where we found that if we divided the data into the appropriate number of frequency classes (i.e., modules sets), according to Sturges' rule [14], usable regression equations could be developed based on the averages computed for the classes. In that work, we only predicted average values. We now extend the approach to include predicting quality limits. We experimented with various sets of predictor variables. The model results are shown in Table 7. The equation we selected is the exponential function using average statements (ave S):

$$avedrcount = \exp(0.1137 + 0.0056697 * \text{aveS}) \qquad (13)$$

This equation was selected for application to Build 2 for the following reasons: 1) lowest Mean Square Error (MSE) in Table 7; 2) fair accuracy in predicting Build 1 *drcount*; 3) theoretical consideration that the rate of change of *drcount* with module size would vary with module size (property of exponential distribution); and the relative ease of collecting size data. Although the F-ratio and R^2 are impressive for the linear function using *nodes*, this equation has a relatively high MSE and the collection of *nodes* requires the use of a metrics analyzer.

Prediction results are shown in Figures 4 -- 7. The figures show the following for average *drcount* for sets of 100 modules (1 -- 100, 101 -- 200, etc.): Figure 4, actual and predicted values for Build 1; Figure 5, actual and predicted limits for Build 1; Figure 6, actual and predicted values for Build 2; and Figure 7, actual and predicted limits for Build 2. Figure 7 shows that the prediction limits bracket the actual values for Build 2. This is another example of retrospective analysis: once the quality factor data F_i are available during the Test Phase of Build 2, Figure 1, the actual *drcount* can be compared with the predictions. In the application of the prediction equation, the software manager would compute the average size of sets of modules and predict the *drcount* and the limits of *drcount* for each module set, as shown in Figures 6 and 7, respectively.

4. Summary and Conclusions

We developed a new metric, Relative Critical Value Deviation (RCVD), for classifying and predicting software quality. When the granularity of data was considered, the RCVD proved to be a useful indicator of the degree to which software quality deviates from a specified norm. We discovered that the major application of the RCVD was to prioritize the effort required to achieve quality goals. At the outset we posed several questions that the software manager wants answered concerning software quality. We provided an integrated set of models based on Boolean discriminant functions, RCVDs, and regression equations to address these questions. We made a thorough evaluation of two builds - one was used for validation and the other for application -- using a five-stage analysis approach. In the three areas of our modeling effort, the predictions for the application build were close to the actual values. Based on these preliminary results and the fact that we have done analysis on additional *Space Shuttle* data, we feel that the *models*, not the specific numerical results, are transferable to other organizations, if the models are applied within and not across application domains. However, to increase our confidence in the results, in future research we will examine several additional builds of the *Space Shuttle* flight software. Finally, we found that mundane aspects of the analysis like data sorting to discover information about distributions of data and the use of spreadsheet calculations significantly aided the analysis.

5. Acknowledgments

The research described in this paper was carried out at the Naval Postgraduate School and the Jet Propulsion Laboratory, California Institute of Technology. We wish to acknowledge the support provided for this project by Dr. William Farr of the Naval Surface Warfare Center and the National Aeronautics and Space Administration's IV&V Facility; the data provided by Prof. John Munson of the University of Idaho; the data and assistance provided by Ms. Julie Barnard of United Space Alliance; and the helpful comments of Dr. Linda Rosenberg of NASA's Goddard Space Flight Center.

6. References

[1] Norman F. Schneidewind, " A Software Metrics Model for Quality Control", Proceedings of the International Metrics Symposium, Albuquerque, New Mexico, November 7, 1997, pp. 127-136.

[2] Norman F. Schneidewind, "A Software Metrics Model for Integrating Quality Control and Prediction", Proceedings of the International Symposium on Software Reliability Engineering, Albuquerque, New Mexico, November 4, 1997, pp. 402-415.

[3] A. A. Porter and R. W. Selby, "Empirically Guided Software Development Using Metric-Based Classification Trees", IEEE Software, Vol. 7, No. 2, March 1990, pp. 46-54.

[4] Lionel C. Briand, John Daly, Victor Porter, and Jürgen Wüst, "Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems", Proceedings of the Ninth International Symposium on Software Reliability Engineering, Paderborn, Germany, November 4-7, 1998, pp. 334-343.

[5] Taghi M. Khoshgoftaar and Edward B. Allen. "Logistic Regression Modeling of Software Quality", Department of Computer Science & Engineering, Florida Atlantic University, TR-CSE-97-24, March, 1997.

[6] Taghi M. Khoshgoftaar, Edward B. Allen, Robert Halstead, and Gary P. Trio, "Detection of Fault-Prone Software Modules During a Spiral Life Cycle", Proceedings of the International Conference on Software Maintenance, Monterey, California, November 4-8, 1996, pp. 69-76.

[7] Taghi. M. Khoshgoftaar, Edward B. Allen, Kalai Kalaichelvan, and Nishith Goel, "Early Quality Prediction: A case Study in Telecommunications", IEEE Software, Vol. 13, No. 1, January 1996, pp. 65-71.

[8] Taghi M. Khoshgoftaar and Edward B. Allen, "Predicting the Order of Fault-Prone Modules in Legacy Software", Proceedings of the Ninth International Symposium on Software Reliability Engineering, Paderborn, Germany, November 4-7, 1998, pp. 344-353.

[9] Niclas Ohlsson and Hans Alberg, "Predicting Fault-Prone Software Modules in Telephone Switches", IEEE Transactions on Software Engineering, Vol. 22, No. 12, December 1996, pp. 886-894.

[10] Standard for a Software Quality Metrics Methodology, Revision, IEEE Std 1061-1998, 31 December, 1998.

[11] Norman F. Schneidewind, "Methodology for Validating Software Metrics", IEEE Transactions on Software Engineering, Vol. 18, No. 5, May 1992, pp. 410-422.

[12] Allen P. Nikora and John C. Munson, "Determining Fault Insertion Rates for Evolving Software Systems", Proceedings of the Ninth International Symposium on Software Reliability Engineering, Paderborn, Germany, November 4-7, 1998, pp. 306-315.

[13] Norman F. Schneidewind, "Software Metrics Validation: Space Shuttle Flight Software Example", Annals of Software Engineering, J. C. Baltzer AG, Science Publishers, 1(1995)287-309.

[14] J. D. Jobson, Applied Multivariate Data Analysis, Volume II:, Springer-Verlag, 1992.

Table 1: Kolmogorov-Smirnov Distance for *drcount*=0 vs. *drcount*>0 Validation: Build 1 (n=1397 modules)					
Metric (symbol)	Definition (counts per module)	Critical Value	Distance	α	Rank
prologue size (P)	change history line count in module listing	63	0.592	0.005	1
statements (S)	executable statement count	27	0.505	0.005	2
eta2 (E2)	unique operand count	45	0.472	0.005	3
loc (L)	non-commented lines of code count	29	0.462	0.005	4
eta1 (E1)	unique operator count	9	0.430	0.005	5
nodes (N)	node count (in control graph)	17	0.427	0.005	6

Table 2: Boolean Discriminant Function: Validation (Build 1)			
	$\wedge(M_{ij} \le MC_j)$ $P_i \le 63 \wedge S_i \le 27 \wedge E2_i \le 45$	$\vee(M_{ij} > MC_j)$ $P_i > 63 \vee S_i > 27 \vee E2_i > 45$	
High Quality $F_i \le FC$ *drcount*=0	C_{11}=477	C_{12}=344 Type 2	n_1=821
Low Quality $F_i > FC$ *drcount*>0	C_{21}=35 Type 1	C_{22}=541	n_2=576
	N_1=512 RF=56	N_2=885	n=1397 TF=2579
	ACCEPT	REJECT	

Table 3: *Discriminative Power* Validity Evaluation (Build 1, n=1397 modules)										
	Critical Values				**Statistical Criteria**		**Application Criteria**			
Metric Set	**P**	**S**	**E2**	**L**	**P_1 %**	**P_2 %**	**LQC %**	**RFP %**	**QIR**	**I %**
P	63				6.23	15.10	84.90	6.13	-	50.11
P, S	63	27			3.22	22.12	92.19	2.95	2.59	60.13
P, S, E2	63	27	45		2.51	24.62	93.92	2.17	**4.91**	63.35
P, S, E2, L	63	27	45	29	2.00	29.35	95.14	1.78	2.16	68.58
K-S Distance	0.592	0.505	0.472	0.462						
P: prologue size, S: statements, E2: eta2, L: lines of code										

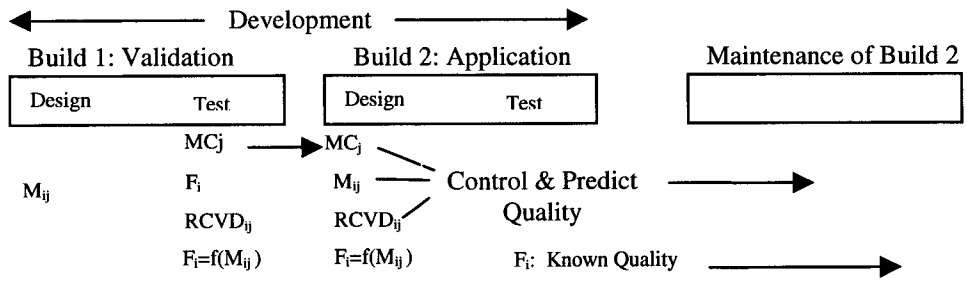
Table 4: Boolean Discriminant Function: Application (Build 2)			
	$\wedge(M_{ij} \le MC_j)$ $P_i \le 63 \wedge S_i \le 27 \wedge E2_i \le 45$	$\vee(M_{ij} > MC_j)$ $P_i > 63 \vee S_i > 27 \vee E2_i > 45$	
High Quality ?	?	Type 2 ?	?
Low Quality ?	Type 1 ?	?	?
	N_1=228	N_2=618	n=846
	ACCEPT	REJECT	

Table 5: Comparison of Validation (Build 1, n=1397 modules) with Application (Build 2, n=846 modules)									
	Critical Values			Statistical Criteria		Application Criteria			
Metric Set	**P**	**S**	**E2**	**P_1 %**	**P_2 %**	**LQC %**	**RFP %**	**QIR**	**I %**
Validation P, S, E2	63	27	45	2.51	24.62	93.92	2.17	4.91	63.35
Application P, S, E2	63	27	45	3.07	26.71	93.78	2.69	9.11	73.05
P: prologue size, S: statements, E2: eta2									

Table 6: Comparison of Relative Critical Value Deviation (RCVD) *Discriminative Power*			
	Build 1 (Validation)		Build 2 (Application)
	$\overline{\text{RCVD}}$ (six metrics)	RCVD (*prologue size*)	RCVD (*prologue size*)
.80 Percentile RCVD Value (Break Point)	.1026	.0517	.0777
BDF ∧ RCVD	((P>63)∨(S>27)∨(E2>45)) ∧(RCVD>.1026)	((P>63)∨(S>27)∨(E2>45)) ∧(RCVD>.0517)	((P>63)∨(S>27)∨(E2>45)) ∧(RCVD>.0777)
drcount identified (percent)	1400 (54.28)	1686 (65.37)	1002 (62.74)
modules with *drcount* identified (percent)	263 (18.83)	280 (20.04)	173 (20.45)
drcount density (*drcount*/module)	5.32	6.02	5.79
drcount density for other modules	1.04	.80	.88
BDF	((P>63)∨(S>27)∨(E2>45))		
drcount density	2.85		2.51

1. $\overline{\text{RCVD}}$ (six metrics): mean of RCVDs of six metrics in Table 1
2. *drcount* identified: count of DRs on modules rejected by BDF ∧ RCVD; percent of total DRs
3. modules with *drcount* identified: count of modules rejected by BDF ∧ RCVD; percent of total modules
4. *drcount* density: *drcount*/module count
5. *drcount density* for other modules: modules other than those rejected by BDF ∧ RCVD

Table 7: Regression Equation Summary for Predicting ave*drcount*							
Predictor Variables	Type	F	R^2	MSE	Mean Residual	Predicted Build *drcount*	Actual Build *drcount*
Build 1: Validation							
aveS	Exponential	56.94	.851	0.702	.0000	2377	2579
aveN	Linear	283.13	.966	1.545	.0000	2241	2579
aveS, aveN	Exponential	39.84	.899	0.754	.0000	2404	2579
Build 2: Application							
aveS	Exponential	56.94	.851	0.437		1637	1597
S: statements, N: nodes, MSE: mean square error computed between predicted and actual *drcount*							



M_{ij} : Metric j on Module i
MC_j : Metric j Critical Value
F_i : Quality Factor on Module i
$RCVD_{ij}$: Relative Critical Value Deviation for Metric j on Module i
$F_i=f(M_{ij})$: Quality Limits Predictor

Figure 1. Measurement Process

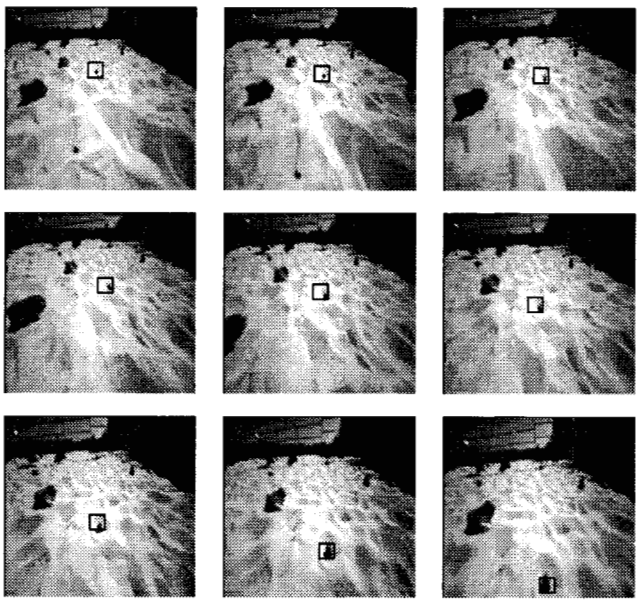


Fig. 5. Sample tracking sequence.

have also successfully placed the instrument arm onto a boulder over five meters away. However, since the visual tracking algorithm servos on the local elevation maximum, only targets on the top of rocks were specified at this time.

Among several runs that succeeded, there were a few runs that did not complete. The two primary reasons are:

- The visual tracker loses its target. This occurs when either the target leaves the camera FOV, no range data is available due to lighting conditions, multiple targets are visible inside the search window, target is outside the search window, or target is same color as background. Using 14 different datasets, the visual tracker succeeded in maintaining target lock through 10 complete sequences. Correcting the threshold increased this to 13 successful datasets.
- The visual tracking succeeds but the rover cannot stabilize about the goal point. Since we rely on the mobility system, positioning resolution of the vehicle is less than our goal tolerances. This is mainly apparent on sandy ground where vehicle maneuvering introduces much positional uncertainty.

VII. Future Work

We are planning to improve the robustness of the visual tracking algorithm (reducing its dependency on the brightness-based filter) by matching the entire shape of the terrain around the target. We also plan to improve the position and pose estimates using visual feature tracking on the whole scene[5]. These improvements should allow tracking of targets anywhere on a rock, which would enable a more general mast placement capability. Another area that we will be addressing is the elimination of the instabilities that result from the imprecise vehicle motions on loose terrain. Improving the coordination between the vehicle and the arm trajectories will improve the overall system. We also like to introduce obstacle avoidance in the arc planning strategies and test the continuous operation of the rover while images are being acquired and processed.

VIII. Acknowledgments

We wish to thank the Long Range Science Rover (LRSR) team for providing the *Rocky 7* rover and for their assistance and support during the development of this work, especially Samad Hayati, Richard Volpe, Bob Balaram, Robert Ivlev, Sharon Laubach, Alex Martin-Alvarez, Larry Matthies, Clark Olson, Richard Petras, Robert Steele, and Yalin Xiong. The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract to the National Aeronautics and Space Administration.

References

[1] P.K. Allen. Automated tracking and grasping of a moving object with a robotic hand-eye system. *IEEE Transactions on Robotics and Automation*, 9(2):152–165, 1993.

[2] Gregory D. Hager, Gerhard Grunwald, and Kentaro Toyama. *Intelligent Robotic Systems*, chapter Feature-Based Visual Servoing and its Application to Telerobotics. Elsevier, Amsterdam, 1995. V. Graefe, editor.

[3] R. Horaud, F. Dornaika, and B. Espiau. Visually guided object grasping. *IEEE Transactions on Robotics and Automation*, 14(4), August 1998.

[4] Mark Maimone, Issa Nesnas, and Hari Das. Autonomous rock tracking and acquisition from a mars rover. In *International Symposium on Artificial Intelligence for Robotic Systems in Space*, Noordwijk, Netherlands, June 1999. http://robotics.jpl.nasa.gov/tasks/pdm/papers/isairas99/.

[5] Larry Matthies. *Dynamic Stereo Vision*. PhD thesis, Carnegie Mellon University Computer Science Department, October 1989. CMU-CS-89-195.

[6] I.A. Nesnas and M.M. Stanišić. A robotic software developed using object-oriented design. In *ASME Design Automation Conference*, Minnesota, 1994.

[7] H.K. Nishihara, H. Thomas, E. Huber, and C.A. Reid. Real-time tracking using stereo and motion: Visual perception for spa ce robotics. In *International Symposium on Artificial Intelligence for Robotic Systems in Space*, pages 331–334, 1994.

[8] N. Papanikolopoulos and P.K. Khosla. Adaptive robotic visual tracking: theory and experiments. *IEEE Transactions on Automatic Control*, 38:1249–1254, March 1993.

[9] L. Pedersen, D. Apostolopoulos, W. Whittaker, T. Roush, and G. Benedix. Sensing and data classification for robotic meteorite search. In *SPIE Photonics East*, Boston, 1998.

[10] S.B. Skaar, W.H. Brockman, and W.S. Jang. Three dimensional camera space manipulation. *International Journal of Robotics Research*, 9(4):22–39, 1990.

[11] D.A. Theobald, W.J. Hong, A. Madhani, B. Hoffman, G. Niemeyer, L. Cadapan, J.J.-E. Slotine, and J.K. Salisbury. Autonomous rock acquisition. In *AIAA Forum on Advanced Development in Space Robotics*, Madison, WI, August 1996.

[12] Richard Volpe. Navigation results from desert field tests of the Rocky 7 mars rover prototype. *International Journal of Robotics Research*, Accepted for Publication, Special Issue on Field and Service Robots 1999. http://robotics.jpl.nasa.gov/people/volpe/papers/JnavMay.pdf.

[13] David Wettergreen, Hans Thomas, and Maria Bualat. Initial results from vision-based control of the Ames Marsokhod rover. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1377–1382, Grenoble, France, September 1997. http://img.arc.nasa.gov/ papers/iros97.pdf.

[14] Yalin Xiong and Larry Matthies. Error analysis of a real-time stereo system. In *Computer Vision and Pattern Recognition*, pages 1087–1093, 1997. http://www.cs.cmu.edu/~yx/ papers/StereoError97.pdf.

1.	Acquire stereo image pair with body navigation cameras
2.	Send left image over wireless network to host
3.	Scientist/Operator selects target rock on left image
4.	Target location and intensity threshold sent to rover
	All subsequent processing occurs on-board
5.	Identify 3-D location of rock based on calibrated camera models and on-board stereo image processing
6.	Compute single-arc rover trajectory to target
7.	Drive rover toward target
8.	Periodically (every 10 cm) poll tracking software to update target location using new stereo pair & current odometry
9.	Redirect rover toward new target location using new single-arc trajectory, and repeat until target is within 1 cm of goal position
10.	Deploy sampling arm, sense and pick up rock.

TABLE I

ALGORITHM FOR SMALL-ROCK ACQUISITION USING THE ARM

1.	Acquire stereo image pair with mast cameras
2.	Send the left image over wireless network to host
3.	Scientist/Operator selects target on left image
4.	Target location and intensity threshold sent to rover
	All subsequent processing occurs on-board
5.	Identify 3-D location of target region based on calibrated camera models and on-board stereo image processing
6.	Compute single-arc rover trajectory to target
7.	Drive rover toward target
8.	Periodically (every 50 cm) poll the target tracking software to update target location using new stereo pair and current odometry
9.	Redirect rover toward the new target location using new single-arc trajectory, and repeat until target is within 1 m from goal
10.	Compute a 2-arc trajectory to a point 0.5 m from target with a final rover orientation to match the target's surface normal
11.	Drive rover along the two-arc trajectory
12.	Poll target tracking software for update on target location and surface normal
13.	If target is out of instrument's reach, move closer to target and update location
14.	Deploy mast arm instrument toward target
15.	Servo mast along surface normal until mast touches the rock

TABLE II

ALGORITHM FOR INSTRUMENT PLACEMENT USING THE MAST

E. Target Grasping

Once the arm's workspace is centered to within 1 cm of the target, the arm is deployed. The scoops are opened and the arm moves downwards toward the ground sensing obstacles along its trajectory. Sensing is done by monitoring changes between the desired and actual trajectories of the arm's shoulder joints. The arm stops when either the target or the ground are sensed. The arm then goes into a grasping mode. As the scoops sense resistance, the arm is raised in small amounts while the scoops continue to close. The arm exits this mode when either a stable grasp is achieved, the scoops are completely closed, or the algorithm times out. This algorithm ensures that the gripper has a good hold on the target.

F. Instrument Placement

Table II describes the algorithm used for instrument placement. The general strategy is similar to the rock sample acquisition, except that the rover must approach the target and place the instrument at a specific orientation determined by the target's surface normal. In addition, the instrument placement can be started from a distance of more than five meters away. Because of this long distance, we use the narrow field-of-view cameras of the mast to track the target. We drive the vehicle with its mast halfway up to continuously monitor the target. Every 50 cm the rover stops and acquires a new stereo pair for the target tracking algorithm. When the rover is within 1 m of its target, it stops and plans a two-arc trajectory to adjust its final approach toward the target. The final approach is determined by the surface normal which is computed from the range data of the target area. The rover drives along the two-arc trajectory and stops in front of the boulder. The mast fully deploys and approaches the boulder. It stops at about 20 cm from the target's surface. Instrument sensing is enabled and the mast moves along a straight line toward the target until the instrument touches the rock. The mast then stops and the instrument takes its measurements. The mast retracts and stows and the rover moves away from the target area.

VI. EXPERIMENTAL RESULTS

We have performed several experiments in JPL's Mars Yard[1], and successfully demonstrated the acquisition of small rocks (3-5 cm) located over 1 meter in front of the rover. Figure 5 shows a sample tracking sequence, with the target indicated in each frame by a dark square. We

[1]http://marscam.jpl.nasa.gov/