

# **Video Performance Evaluation Resource**

## **Performance Evaluation Manual**

**November 25, 2002**

# Table of Contents

1	Introduction.....	3
2	ViPER-PE's Performance Evaluation Methods.....	3
2.1	The Problem of Performance Evaluation.....	3
2.2	The ViPER Solution .....	3
2.3	Object Analysis – Matching Candidates to Targets .....	4
2.3.1	Detection.....	6
2.3.2	Localization.....	6
2.3.3	Statistical Comparison .....	7
2.3.4	Target Matching.....	7
2.4	Framewise – Evaluation on the Frame Level.....	8
2.5	Tracking – Evaluation Given the Starting Frame .....	8
3	The viper-pe Tool.....	9
3.1	Installation.....	9
3.2	Using the viper-pe Tool .....	9
3.3	Setting the Evaluation Parameters .....	10
3.3.1	Equivalence .....	11
3.3.2	Evaluation.....	11
3.3.3	Filtering.....	13
4	Appendix I.....	15
4.1	Terminology.....	15
4.2	Property Names and Command Line Arguments .....	15
4.3	Filter Types .....	18
4.4	Metrics .....	18
4.5	File Formats .....	19
4.5.1	Properties File.....	19
4.5.2	Evaluation Parameters File .....	19
4.5.3	Human Readable Output File.....	20
4.5.4	Raw Output File.....	21
4.6	Tools .....	22
5	Appendix II: Notes.....	23
5.1	FAQ.....	23

# ViPER-Performance Evaluation Manual

## 1 Introduction

Note: This document is but one in a set of documents describing the entire ViPER toolkit:

0. Quick Start: Describes how to install and set up ViPER.
1. ViPER-GT: Describes the Ground Truthing tool and the data formats.
2. ViPER-PE: This document, describes the performance evaluation tool and its metrics.
3. Scripting ViPER: Describes various scripts to deal with multiple evaluations and make graphs displaying the results.
4. Case Studies: Describes several use cases of ViPER, from design of GTF to evaluation.

The goal of the Performance Evaluation component of the Video Performance Evaluation Resource, or ViPER-PE, is to take the result data from some analysis, compare it to some ground truth, and to produce some useful data describing the success or failure of the analysis, hopefully in such a way that results are repeatable, valid and comparable. This document describes the goals of the ViPER-Performance Evaluation software, how to use it, and how to interpret its results. The three different major types of analysis are object, framewise and tracking.

## 2 ViPER-PE's Performance Evaluation Methods

### 2.1 The Problem of Performance Evaluation

Performance evaluation is required for empirical research into video analysis. Without performance evaluation, researchers and users cannot measure improvement over older methods or determine the best algorithm for a task. Good performance evaluation can provide baselines, deliverables, and other artifacts useful not only for the good of science, but also for securing grants and funding.

Unfortunately, performance evaluation of video is inherently difficult. It is labor intensive, often requiring the creation of custom tools and the generation of ground truth. The custom tools may contain flaws, as they are often not given as much attention as required. The ground truth itself is more likely to contain flaws, especially for video, where generating ground truth is thankless and repetitive. It is often subjective, with metrics and data sets tailored for the tested algorithms. The published results may not be statistically valid or correct, and the reader is rarely given the tools to verify results.

### 2.2 The ViPER Solution

Essential components of an evaluation system include:

- 1) A standard method of representing analysis results and ground truth. This is accomplished with the Ground Truth file formats described in the ViPER-GT Manual. This will allow publication of ground truth and algorithm output, allowing interested parties to perform evaluation using their own tools, or directly compare a different algorithm to one included in a paper.
- 2) A system for configuring an evaluation system. ViPER uses the Evaluation Parameters File (EPF) in combination with a set of properties to configure a specific evaluation. The EPF tells the system how to perform an evaluation of candidate data against a target, and is readable enough to let a reader know what an evaluation means while providing a method for duplicating the experiment.
- 3) A tool to perform the evaluation. This is accomplished with the `viper-pe` java-based command line tool.
- 4) A meaningful way to present results. ViPER-PE provides two output formats, a human readable text file, and a space delimited one for machine reading.<sup>1</sup> The `makeGraph` tool can convert sets of the machine-readable output format into graphs, while the `RunEvaluation` tool can compare several sets of experiments. For information about display, see the Scripting ViPER manual.

With the above four items, ViPER provides repeatability, comparability and the ability for an experimenter to determine validity. Users and researchers in a given field of video analysis can decide on a ViPER-PE configuration; this will allow duplication of the experiments. Used properly, ViPER-PE can track progress during development of a video understanding system, compare the features of existing systems and otherwise help evaluate the state of the art.

Performance evaluation involves comparing generated data, called the **candidate** or result data, with the ground truth, or **target**, data. They are called this since any given algorithm is trying to generate the target data as accurately as possible, producing a set of candidate objects, or **descriptors**. Given sets of candidate and target data, with appropriate configuration options, the Performance Evaluation tool generates a description of how well the candidates (results) match the targets (truth) given the parameters.

Given that the field of video understanding is wide open, little agreement exists upon a proper set of performance metrics. As such, ViPER-PE includes several different types of analysis, each of which is configurable. The three major analysis types included in ViPER are **object analysis**, **framewise** and **tracking**.

### 2.3 Object Analysis – Matching Candidates to Targets

Object analysis attempts to match candidate objects to target objects. It attempts to answer the question, “How close are two descriptors?” Using this idea, it determines which targets and candidates are close together, and then generates the precision and recall of the number of candidates matching targets. This involves defining some distance

---

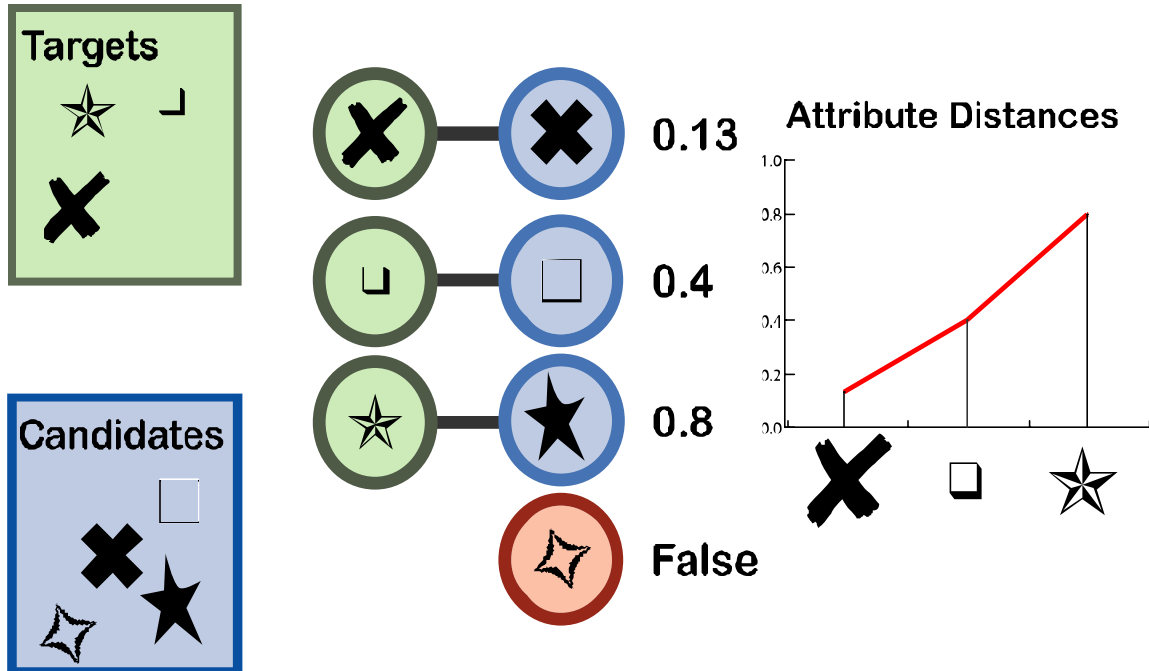
<sup>1</sup> Two additional methods have been suggested: a ground truth file that displays the results like the Overlay scripts, and an xml version of the raw output file that could be used to present the data with XSLT.

space for descriptors, preferably a metric one. For ViPER-PE, all distances are normalized between zero and one. A complete listing of the available metrics is included in the appendix.

**Table 1: Common Distance Metrics**

Metric	Definition
E	Equality – two attributes are close if they are equivalent.
Dice	Dice coefficient – twice the shared area over the sum. This avoids the asymmetry of the Overlap metric.
Target Overlap	The fraction of the target that overlaps the candidate is used as the distance.
Maximum Deviation	The maximum of either the target or candidate overlap distance.
String Edit Distance	Strings that require more atomic edit operations (insert or delete character) are farther apart.

Object Analysis is divided into multiple phases: detection, localization, statistical comparison and target matching. Each phase is a finer level of analysis. Detection determines that some object was found in a set of frames for each target. Localization and statistical comparison make sure that a possible match has similar attributes. Finally, the target matching phase checks many-many candidate-target pairings, dealing with split and merged objects.



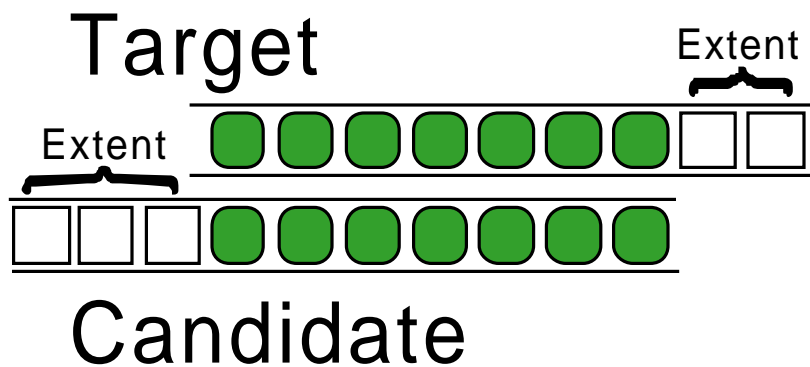
**Precision:  $3/4 = 0.75$**

**Recall:  $3/3 = 1.0$**

### 2.3.1 Detection

In the Detection phase, ViPER-PE looks at descriptor types and the frames in which the objects occur. If a candidate overlaps the appropriate number of frames as any target descriptor, then both candidate and target are counted as detected. The chosen distance metric and tolerance define the number of frames required for a pair to count as a detection. The metric, which defaults to a dice coefficient, determines how to compute the distance between two frame spans. If the computed distance is less than or equal to the tolerance, the pair is detected.

Detection could very well be enough analysis for an application. For example, a program that counts the number of faces on a screen does not need to check to see where they are. Recognition, frame detection and object counts all require no more analysis than this.



Dice Coefficient

$$\frac{2 \cdot 7}{10 + 9} = 0.74$$

### 2.3.2 Localization

The second phase, localization, works much like detection. Instead of looking at all frames of the objects, it compares only those frames where the attributes are similar. Attribute similarity, like frame range similarity, is user-defined. For a given candidate-target pair, dissimilar frames are counted as missed or false in the range distance calculation. Each attribute distance is calculated separately in the properties and the evaluation parameters file (EPF).

For example, when localizing faces, it is common to use a dice metric on the face with a tolerance of about 0.5. If we also wanted to check the identity of the face, using a string attribute "Name", we would also add an equality metric for that attribute. For a possible match, only frames with the appropriate face name and meet the dice threshold will count as similar. The frame range metric will be computed as if the dissimilar frames that did not overlap.

### 2.3.3 Statistical Comparison

The third level is statistical comparison. From the localized matches, it computes the average, median, minimum, or maximum distance, and then thresholds that value against another tolerance value, specified as the `level3_tol` property. This is useful for descriptors that cross several frames.

### 2.3.4 Target Matching

After these different measures, ViPER-PE has a list of candidates that match targets. Many candidates may match a single target, and vice versa. While this may be perfectly acceptable, it is often not enough to leave it at this. There are many situations where having two objects match one do not make sense. Allowing only one candidate for each target is a possible solution. For many cases, such as text detection, a scheme of splitting or merging descriptors so that only appropriate multiple groupings are made may be preferable. Since both have their use, ViPER-PE provides both methods.

It should be noted that either method requires an ability to rank target-candidate matches. While the previous steps only generated a distance measure for individual attributes, target matching requires a single distance measure for a candidate-target pair. For detected and localized pairs, the distance is the frame range distance. If ViPER-PE was set to use statistical comparison, the match distance is calculated using the average of the statistic for each evaluated attribute, with missed and falsely detected frames are counted as having a distance of one.

For one-to-one target matching, it would be best to minimize the sum of the distances over all targets and candidates. While this is possible and ViPER-PE offers it as a solution, it has exponential difficulty in the many-many matches and may not even be the most instructive solution. A better solution may be to select the best pair, then the next best pair from the remaining matches, and so on. Set the `'target_match'` property to `SINGLE-OPTIMUM` for the optimum search and `SINGLE` for the simpler solution.

Setting the `'target_match'` property to `MULTIPLE` will aggregate descriptors together. The aggregation algorithm is iterative. It examines targets that match the same candidate, then aggregates them if aggregation would improve the distance. Then, it looks at candidates that match the same target and repeats. This iterates until no improvement in the distance is gained. Aggregation is only defined for some of the attribute types. Aggregation of frame spans and circles are unions. Boxes, oriented boxes, and, in the near future, polygons can be combined with each other, also. In order to make the output prettier, strings will be concatenated, but this is unlikely to be the appropriate aggregation solution; a better technique would perhaps take a bag of words.<sup>2</sup>

---

<sup>2</sup> Another appropriate technique would be to add another localization after multiple matching, as Eliza suggested.

## 2.4 Framewise – Evaluation on the Frame Level

It may be apparent now that object analysis is not a panacea. It is very complex, and often does not produce eminently usable data. Framewise analysis is in direct response to these issues. Instead of looking at objects across frames, it examines them frame-by-frame. It generates a list of metrics for each frame, including pixel precision and recall. It supports most of the metrics defined in Mariano, Park, Min and Kasturi's paper on performance evaluation; the tracking evaluation provides those that it does not.

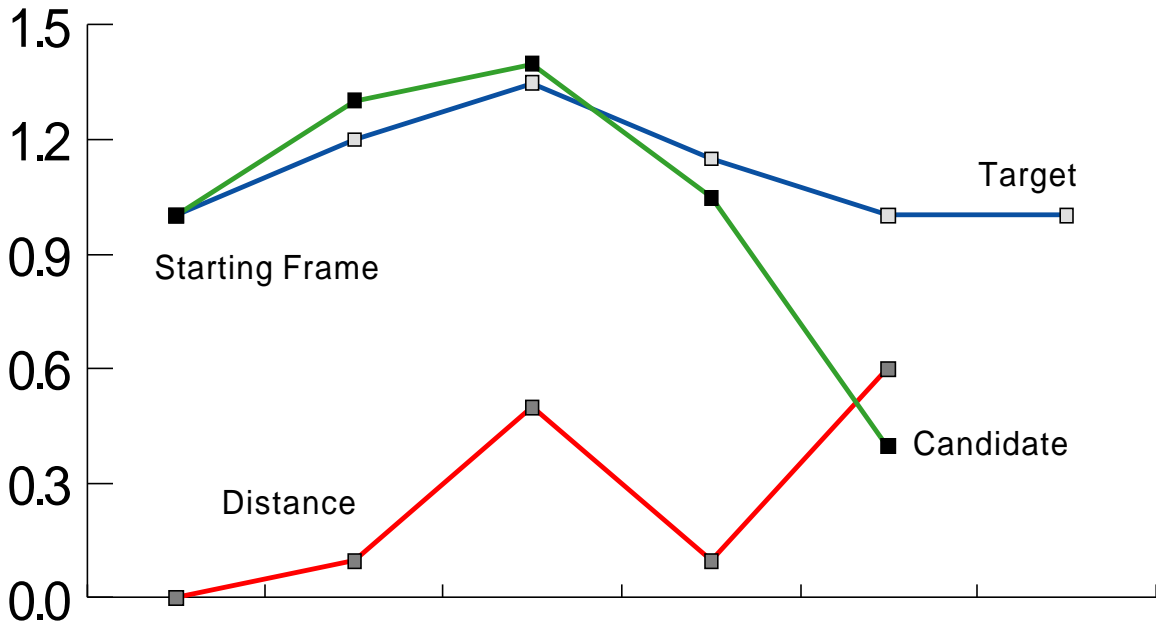


## 2.5 Tracking – Evaluation Given the Starting Frame

Tracking Analysis assumes that there already exists some mapping between candidates and targets such that no one candidate or target is mapped to more than one descriptor. The simplest way to achieve this is to associate a unique attribute with each descriptor in the ground truth, and use that to identify the candidate descriptors. An experiment here will usually involve distributing first-frame data sets, generated using `gtf2gtf`'s `-clip` option.<sup>3</sup>

---

<sup>3</sup> For information on this and other tools, see the Scripting ViPER manual.



Average Distance for ObjectN: 0.4

### 3 The viper-pe Tool

#### 3.1 Installation

For information about how to install ViPER-PE, refer to the Quick Start Guide.

#### 3.2 Using the viper-pe Tool

The `viper-pe` command takes a series of properties to configure the system. These can either be specified on the command line using their short names or property names or in a properties file. The command needs at least two properties, the target file and the candidate file, specified with the `-g` option and the `-r` option, respectively, although the analysis will likely be garbage without an evaluation parameters file and at least the `target_match` property set.

Table 2: Frequently Used Properties - For Complete List, See Appendix

Command Line Switch	Property Name	Value
<code>-g</code>	<code>gt_file</code>	The file name of the truth file (the file containing the target data set).
<code>-r</code>	<code>results_file</code>	The file name of the results file (the file containing the candidate data set).
<code>-epf</code>	<code>epf_file</code>	The evaluation parameters file,

		including equivalency, evaluation, and filter information.
-o	output_file	Where to print the human readable output data. Defaults to standard output. Set to '-' for standard output, and the empty string for none.
-raw	raw_file	The file to receive the raw data output. Defaults to none. Set to '-' for standard output, and the empty string for none.
-P<propertyname>		Specify any property by its long name.
-pr		The file name of the properties file.
	target_match	If using an object evaluation in your EPF, this parameter specifies the methods of object analysis: ALL, SINGLE, SINGLE-BEST, or MULTIPLE.

Given a valid set of properties, ViPER-PE will generate readable output in the file specified in the '-o' option and machine readable data in the file from the '-raw' option. It will display error messages to the system's error stream, and informative messages to the stream specified in the '-l' option.

From a Unix install, you should be able to invoke the command `viper-pe` from any location, assuming you have set the `PATH` variable by sourcing the `viper.config` file from `csh`, or dotting the `viper-config.sh` script from `sh`. From Windows, you will have to write a batch file, or invoke the slightly more complex command `java -jar viper-pe.jar`. For example, a simple command would be:

```
Viper-pe -pr textdetect.pr -epf dice-graphic.epf -g all.gtf.xml -r
UMD/RDF/all.rdf.xml -o dice-graphic.out -raw dice-graphic.raw
```

### 3.3 Setting the Evaluation Parameters

The evaluation parameters file is given as a list of sections, delimited with

```
#BEGIN_EQUIVALENCE

TargetDescriptor : CandidateDesc
TargetAttribute :CandidateAttr

#END_EQUIVALENCE
```

**Figure 2: Simple Equivalence Section**

```
#BEGIN_EQUIVALENCE

PROFILE : Male Female
DIRECT : Male Female

#END_EQUIVALENCE
```

**Figure 1: Many-Many Match**

#BEGIN\_<section> and #END\_<section> lines. The sections are EQUIVALENCE, EVALUATION, and the four FILTER sections. The Equivalence section matches names in the target data to names in the candidate data; the Evaluation section specifies both the type of evaluation and how the results will be computed; and the Filters specify which descriptors to evaluate and which ones to ignore.

### 3.3.1 Equivalence

One common issue during evaluation is a disagreement between the descriptor and attribute names of the candidates and targets. Even worse, several different candidate descriptor types may match one target type. The Equivalence section addresses the problem with a simple list of matches. It is possible to map a single target name to multiple candidate names on one line; simply place all of the candidate names in a space delimited list on the right side of the colon. If you want to map multiple target names to a single candidate name, or a list of candidate names, the same line must be repeated with different candidate names.

### 3.3.2 Evaluation

There are three possible evaluation sections; there is one for each type of evaluation. While they look similar, each has a slightly different layout. All three operate on the ground truth names for the descriptors and attributes, and work by listing the descriptors to evaluate, with a list of metrics following each line. Attributes and descriptors that are not listed in the EPF file are ignored.

Evaluation Type	Sample Evaluation Block
Object Evaluation	OBJECT Descriptor1 [dice .99] AttributeAlpha : [maxdev -]
Frame-wise Evaluation	OBJECT Descriptor1 BBOX : dice [arearecall .6] [areaprecision .6]\ <areaprecision .6>
Tracking Evaluation	OBJECT PERSON BBOX : dice extents [maxdev -] * NAME

#### 3.3.2.1 Object Evaluation

The Object Evaluation section specifies which descriptors and attributes to evaluate. It is a list of the descriptors to evaluate, and information about how to evaluate them. After each attribute name is a colon and a square-bracketed metric-threshold pair. The metrics for each attribute type is listed in the appendix. It should also be noted that the frame span gets a similar treatment, with its pair placed on the first line, after the name of the descriptor. The frame span metric and threshold are used for the matching and localization steps, with the attribute metrics and thresholds used for the statistical localization and target matching steps. Dashes may be used to indicate the default metric or threshold level.

The evaluation is performed as specified in the section on Object Analysis from section 2 of this manual, using the *level* and *target\_match* properties to determine the level of analysis, such as detection or statistical comparison, and the matching heuristic, such as MULTIPLE or SINGLE\_BEST.

For example, the object evaluation listed in the above table is designed to match target descriptors to candidates that overlap somewhat in time and have a maximum deviation distance that for the *AttributeAlpha* attribute that is less than the default for that data type. The evaluation will first match all Descriptor1 candidate-target pairs that overlap in time with a dice distance of less than or equal to .99; this will be most pairs that share a few frames. ViPER-PE will then check each frame of each matching; for each frame of a match, if all attributes, in this case *AttributeAlpha*, are not above their thresholds for the given metric, the frame counts as a miss, and the dice metric is evaluated again on the two frame spans to determine validity of a matching. The third level, statistical comparison, takes the statistic set in the *stat\_metric* property over all the frames of each remaining pairing and compares the average over all attribute results of this to the *stat\_tol* property. The matching is then performed as set in the *target\_match* property; if set to SINGLE-GREEDY or SINGLE-BEST, some pairings are cropped, and if set to MULTIPLE, some pairings are merged with some pairings that are deemed unnecessary are dropped. Precision and recall are then calculated using the remaining pairings.

### 3.3.2.2 *Framewise Evaluation*

Framewise evaluations follow each attribute line of the descriptor you wish to evaluate. Unlike Tracking or Object, there is no metrics for the frame span. A FRAMEWISE\_EVALUATION takes three kinds of metrics: normal metrics, which just return a distance; localizers, in the form [metric threshold], which return a precision/recall; and don't care matchers, in the form < metric threshold >, which mark the candidate as don't care if it matches any given target that is marked as don't care by the ground output filters.

The distance metrics vary for each attribute. These include dice, areaprecision, and arearecall for shape attributes, as well as edit distances for string attributes. For each frame, one number is returned for each of these metrics. This number is either an average or a sum for each descriptor in a frame, depending on the definition of the metric. For example, the "matched" metric returns the sum of all matched pixels in a frame, while "fragmentation" compares each target shape to all candidate shapes and returns an average over all targets in the frame.

Localizers are represented as [distance threshold] pairs in square brackets. These work by taking a distance for each candidate to the targets or vice-versa, and counts the match as successful if the distance is either above or beneath the threshold, depending if the distance is actually a distance, like dice, or a similarity measure, like areaprecision or arearecall. The number returned is a ratio of correctly matched to number of possibles.

The third possibility, in angle brackets, is used for ignore filtering, which is described more fully in a later section. The basic idea is that certain target descriptors may be

marked as "don't care". This is fine for distances, but for localizers and the three constant metrics, object count accuracy, precision and recall, there needs to be some way to determine that the descriptor and not just some or all of its value should be ignored. Viper-pe evaluates the metric for all candidates against each ignored target; candidates whose distances or similarity measures fall below or above the threshold for some specific ignored target are themselves ignored. Note: currently only one filter metric is accepted per line.

The first thing that occurs is the marking of regions as don't care. All target and candidate regions that are marked as ignored by the OUTPUT\_FILTERs are marked as don't care, as are all candidates that return an area precision of greater than .6 for a target that is to be ignored. Object count accuracy, precision, and recall is determined as the count of all descriptors that are not ignored. Then a dice coefficient is run between the set of all candidate and target pixels less those marked as ignored. For each non-ignored target, it counts as localized by the first localizer if more than 60% of its region is recalled by the union of candidate boxes; the ratio of these targets to all possible targets is the localized area recall. Each non-ignored candidate counts as localized by the second localizer if it is precise enough; that is, it more than sixty percent of it (not including ignored regions) is uncovered by target boxes; the ratio of localized boxes to all boxes is the result of the third metric.

### 3.3.2.3 Tracking Evaluation

Tracking Evaluation has a complicated evaluation format as well, as it must take into account what attributes to mark as the key attributes, as well as how to handle the fallback case when attributes have no keys. It contains all the elements of the object evaluation section; in this case, the object evaluation is used in the fallback situation. In addition, it accepts lists of metrics; these are used for the tracking evaluation itself. Finally, the key attribute is marked with an asterisk before it. It does not need any metric information after it.

### 3.3.3 Filtering

Often when using a given set of ground truth, it is preferable to only examine a subset of the data. For example, it is often instructive to compare how well an algorithm performs on text above a certain size, or objects that are not occluded. The Evaluation section

<pre>#BEGIN_GROUND_FILTER OBJECT Descriptor1 : contains "1:90"   IntAttrib : &gt; "10" &amp;&amp; &lt; "20" #END_ GROUND_FILTER</pre>	<pre>#BEGIN_GROUND_OUTPUT_FILTER OBJECT Descriptor1   FloatAttrib : &gt; "1"    &lt; "-1.5" #END_ GROUND_OUTPUT_FILTER</pre>
---	--

specifies the metrics to use and which descriptor types to evaluate, and the Filter sections provide precise control of which descriptors to evaluate.

The Filter sections are divided two ways, into input and output filters, and candidate and target filters. Input filters prevent some descriptors from being read into ViPER-PE; as far as ViPER-PE is concerned, descriptors that do not match an input filter (if one is given for that descriptor type) do not exist. The output filters specify that they, and any matching descriptors, are not to be output; they are also called “Don’t Care” filters. Each different attribute type has a different set of possible filters. For a complete list, see the appendix.

Input filtering makes it appear to the evaluation program that the specified Descriptors are simply not included in the data. Since the evaluation will only occur on descriptors that you specify in the EPF file, leaving them out will skip them as well. The rule system was developed to skip items based on their static attributes. If the attribute is dynamic, the attribute counts as passing if any of the values it takes passes the filter. A descriptor passes a filter if each of its attributes passes. The frame span can also be filtered, as demonstrated in Figure 4.

Since input filtering simply prevents descriptors from being included in the performance evaluation, it results in odd evaluations. Output filters provide a more intelligent, if more processor and memory intensive, approach. Instead of filtering descriptors during parsing, the output filter acts after the computations have been completed. For Object Analysis, matchings that involve a filtered object are not counted towards the precision and recall, and are not printed out as detected. In Pixel Analysis and Tracking Analysis, these objects are treated as “Don’t Care” objects; the regions they cover are not included in many of the pixel metrics. For a complete understanding of the “Don’t Care” methodology, and how to choose the ‘Don’t Care Threshold,’ see [Mariano].

## 4 Appendix I

### 4.1 Terminology

Term	Definition
Candidate data	Also known as result data, this is the set of descriptors generated by some algorithm that will be compared to the target data
Detection	An object is classified as 'Detected' if one of its type is found on the frame. For example, if you want to retrieve all frames containing faces, you may find that 'Detection' is the only required depth of analysis.
Results data	See <i>Candidate data</i>
Target data	Also known as truth data, this is the set of descriptors that represent the true content of the media file
Truth data	See <i>Target data</i> .

### 4.2 Property Names and Command Line Arguments

Command Line Switch	Property Name	Value
-g	gt_file	The file name of the truth file (the file containing the target data set).
-gc	gtconfig_file	Using the old file format, which is still supported, it is possible to specify the descriptor configuration in a separate file. This is the truth configuration information. The property will be ignored if the ground file is in the xml format.
-r	results_file	The file name of the results file (the file containing the candidate data set).
-rc	resultsconfig_file	This is the result configuration information; it is only necessary if the result data is in the old format and does not include a configuration header.
-epf	epf_file	The evaluation parameters file, including equivalency (the map between candidate and target names), evaluation, and filter information.
-o	output_file	Where to print the human readable output data. Defaults to standard output. Set to '-' for standard output, and the empty string for none.
-raw	raw_file	The file to receive the raw data output. Set to '-' for standard output, and the empty string for none. Defaults to none.

-b	base	The base file name for all the other files. Sets all of the above to this stem, except for the configuration files. The file suffixes are .gtf, .rdf, .epf, and .out. For example, setting -b temp will load the target data from temp.gtf, candidate data from temp.rdf, set the evaluation parameters from temp.epf, and output data to temp.out.
-L	level	For object analyses, sets the type of comparison to do. See the object analysis section for a complete description of the following options: 1 = Detection: The objects overlap temporally better than some threshold. (See range_tol, rmetric_default) 2 = Localization: Performs detection using only the frames whose attributes meet necessary thresholds. 3 = Statistical Comparison: The average/ median/max/min, depending on which was selected, meets a certain threshold.
-P<property>		Specify any property by its long name.
-pr		The file name of the properties file.
	verbose	
	attrib_width	Specifies the number of columns in the output file before it will clip the attribute information.
	target_match	The type of evaluation to perform:  PIXELWISE = This is not really a setting of how to crop, but instead changes the way the program fundamentally evaluates. The program will not perform an object level analysis, but print out a pixel/box level analysis. Turning this option on will cause the software to ignore all of the variables related to object-level analysis, such as the various tolerances and metrics.  BOX_TRACKING = Like pixelwise, this turns on a different type of evaluation. This requires that each

		<p>object match exactly one other by localization, and then gives results based on the boxes in each frame.</p> <p>(To perform object analysis, indicate how you want to treat many-many matches.)</p> <p>ALL, DEFAULT = Allow them.</p> <p>SINGLE-GREEDY = Only allow one Candidate for each Target.</p> <p>SINGLE-BEST = Select the one-to-one matching that minimizes the sum of the distances.</p> <p>MULTIPLE = Allow many-to-many mappings using aggregation.</p>
	range_metric	The default distance metric to use for the frame overlap test.
	range_tol	Comparisons that indicate the distance between two descriptor's frame span is greater than this will be dropped.
	level3_metric	The statistic to use when doing statistical localization.
	level3_tol	The tolerance to place on the above measure for each attribute for descriptor target/candidate pairs to count as localized.
	<attribute type>_tol	The default tolerance for a given attribute type for object localization. For a list of metrics for each type, see the table in the Object Analysis section.
	<attribute type>_metric	The default metric type to compute attribute distance during object analysis.
	pixel_threshold	Used for PIXELWISE evaluation, this determines the threshold for localized object precision and recall.
	dont_care_threshold	Used for PIXELWISE evaluation, this number is the threshold for candidate objects to be counted out from pixel evaluation. If the percentage that a given candidate object overlaps a given 'Don't Care' object is greater than this threshold, the candidate object is also counted as 'Don't Care'.

### 4.3 Filter Types

Rule	Description	Applicable Attributes
==	Equivalence: The attribute must be equal to the given value.	All
!=	Non-equivalence: The attribute must not equal the specified value.	All
>, >=, <=, <	Relational values. The attribute is either greater than, greater than or equal to, less than or equal to, or strictly less than the specified value.	dvalue fvalue – less than svalue – lexicographical ordering
contains, intersects, excludes	Set values. - Contains: The rule value is completely covered by the attribute value. - Intersects: The rule value and the attribute value share at least one pixel, frame, element, etc. - Excludes: The rule value and the attribute value share no frames/pixels/elements.	Frame Spans - Specify a range of frames.  Oriented boxes Bounding Boxes - For boxes, use the same format for the value as the GTF format for the given type. Eg, “10 10 20 20” for bboxes.

### 4.4 Metrics

Metric	Attributes	Definition	Formula
E	All Attribute Types	Equivalence	0 if target equals candidate, 1 otherwise.
Dice	Frame Span Circle Box Oriented Box	Twice the shared area, divided by the sum of the two areas.	$1 - 2 * sz(T \cap C) / (sz(T) + sz(C))$
Overlap	Frame Span Circle Box Oriented Box	The fraction of the target that the candidate overlaps.	$1 - sz(T \cap C) / sz(T)$
MaxDev	Circle Box Oriented Box	The maximum amount either the candidate object or the target deviates from the shared region.	Maximum of: $sz(C - T) / sz(C)$  $sz(T - C) / sz(T)$
L	String Value	Normalized Levenshtein (Edit) distance	E = Edit distance. Normalized using normalization factor alpha as follows: $1 - e^{-\alpha E}$
H	String Value	Hamming Distance	1 if length is different. Otherwise, for D = number

			of characters that are different, L = Length, returns: D/L
Euclidean	Point	Normalized Euclidean distance	E = Euclidean distance, normalized as follows: $1 - e^{-E}$
Manhattan	Point	Normalized Manhattan distance	M = Manhattan distance, normalized as follows: $1 - e^{-M}$
Difference	Dvalue Fvalue	Normalized Difference	For D = abs(C - T), Alpha = normalization factor, then: $1 - e^{-\alpha D}$

## 4.5 File Formats

The video data file formats are described in the ViPER-GT Manual.

### 4.5.1 Properties File

The properties file is simply a list of <property> = <value> pairs, with optional comments, that run from a '#' to the end of the line.

<pre># Level of analysis level = 3 target_match = MULTIPLE range_metric = dice</pre>
<b>Figure 3: Example of Properties File</b>

### 4.5.2 Evaluation Parameters File

The Evaluation Parameters file defines three items:

**Equivalencies:** Marked with #BEGIN\_EQUIVALENCIES and #END\_EQUIVALENCIES, the equivalency section defines the mapping between names in the ground truth file and names in the candidate data file. Each line is of the form **<truth name> : <candidate name>**

**Filters:** The input filters determine what data is parsed, while the output filters determine which data is used in the final calculations. The four properties section are delimited with #BEGIN\_ and #END\_ markers as well, in this case

<pre>#BEGIN_EQUIVALENCE   TextBlock : Text   BBOX : LOCATION #END_EQUIVALENCE #BEGIN_GROUND_OUTPUT_FILTER   OBJECT Text : includes "1:1" #END_GROUND_OUTPUT_FILTER #BEGIN_EVALUATION OBJECT Text [- -] LOCATION : Edges_001</pre>
<b>Figure 4: Example Evaluation Parameters File</b>

GROUND\_FILTER and RESULT\_FILTER for the truth and candidate input filters, respectively, and GROUND\_OUTPUT\_FILTER and RESULT\_OUTPUT\_FILTER for the truth and candidate output filters. Inside the section delimiters, the filters are a list of descriptors in format, with a colon and the filters following each line.

*Evaluation:* The Evaluation section selects which descriptors and attributes to examine. Each block is a descriptor. Descriptors and attributes that are not listed are not evaluated. For object analysis, include [metric tolerance] information after the attribute. The frame range metric and tolerance is set after the main descriptor line.

Note that C++ style // comments are acceptable.

### 4.5.3 Human Readable Output File

Divided into two main sections, it first describes the parameters for the evaluation, then displays the results. The “INPUT PARAMETERS” section lists various properties. The four “FILTER” sections display what, if any, filters were used. Finally, the “METRICS” section includes the Descriptors and Attributes that are to be evaluated, including their metric and tolerances. The input parameters establish how the file was generated, and the repeat of the parameters are also good for troubleshooting possible error in file format or understanding of how to set the parameters of an evaluation.

For Object Analysis, the rest of the file describes which level a FALSE or MISSED descriptor got to, and then lists the detection matches. If a descriptor is counted as false or missed on level 0, that means that no other object of that type occurs in a given file. If a descriptor falls out at level 1, this descriptor did not meet the frame range metric and tolerance restrictions for any other object of the same type. Level 2 indicates localization failures, 3 are statistical failures, and 3c are descriptors that matched an object not as well as another, as described in the given type of target\_match.

The Pixelwise output gives a frame-by-frame commentary, listing several properties for each frame. Briefly, these are the number of pixels matched in the frame, the number missed by the candidate set, the number the algorithm detected mistakenly, a pixel accuracy and an object accuracy, a fragmentation measure, and object, average box, and localized box precision and recall.

The Tracking metric lists results for each truth object, listed by input file and id number. These numbers are temporal precision and recall, positional accuracy, size accuracy, and angle difference

It should be noted that all three of the types of evaluation end with a summary of some sort. Object Analysis returns precision/recall data, Pixelwise gives a total, and Tracking gives various coefficients. If these are not printed, then there is a flaw in the code, in the input parameters and data, or the computation requires too much memory to finish.

The format is self-explanatory. For an example, see the included case studies.

#### 4.5.4 Raw Output File

The raw file is a list of space and line delimited numbers describing the results of an evaluation. Each raw file also includes information about how the evaluation was performed. Like the Evaluation Parameters file and the old Ground Truth file formats, #BEGIN\_<SECTION> and #END\_<SECTION> markers divide the raw file format is divided into sections. It may include C++ style // comments.

PARAMETERS: A set of <property> = <value> pairs. Currently, this includes config\_file, gt\_file, result\_file, epf\_file, log\_file, output\_file, and level.

GROUND\_FILTER, RESULT\_FILTER: The input target and candidate filter settings, respectively.

METRICS: A list, basically corresponding to the EVALUATIONS section of the EPF file. It is of the form: <Descriptor> <metric> <tol> \n \*( \\* <attrname> <metric> <tol> \n)

GTF\_INFORMATION, RDF\_INFORMATION: Data concerning the target file and the candidate file, expressed as name = value pairs. Currently NUMFRAMES and NUMFILES are the only two values in both.

#### <RESULTS TYPE>

RESULTS: Object Analysis results. Line is of the format <Type> <ID> (MISSED|FALSE|DETECT) <level> <Detected Comparison Values>. The <Type> is the descriptor type specified in the ground truth file. For MISSED and FALSE, level indicates on which level the descriptor was marked as incorrect; for example, a value of 3 indicates that the descriptor did not pass statistical localization. All DETECTED should indicate the same level. The <values> is of the form <frame span distance between target and union of candidates> <Number of candidates (1 if target match is set to anything other than ALL)> +\ [<cand id(s)> <frame range distance> <attribute distances>] If ALL or DEFAULT is the target\_match, there can be several bracketed matches. If MULTIPLE is the specified match, instead of object ID numbers, there may be bracketed comma delimited lists of ID numbers; note that in this case, <Number of Candidates> is set to one.

SUMMARY: Also included in Object Analysis, the Summary section lists the result by descriptor type, followed by a sum. It is in the form <Descriptor or TOTAL> <# of targets> <# of candidates> <precision> <recall>.

PIXEL\_RESULTS: Each line contains the frame number, followed by the pixelwise evaluation for the frame. The last line starts with the String 'Total,' then includes the sum over the whole file. <ID/"Total"> <Matched Pix> <Missed Pix> <False Pix> <Object Count Accuracy> <Average Fragmentation> <Average Object Recall> <Average Object Precision> <Localized Object Recall> <Localized Object Precision> <Object Count Recall> <Object Count Precision>

TRACKING\_RESULTS: Each line contains the tracking result for a single object, except for the last line, which contains the total coefficients. <ID/"Total">

<Temporal Precision> <Temporal Recall> <Positional Accuracy> <Size Coefficient> <Orientation Coefficient>

#### 4.6 Tools

gtf2xml, gtf2gtf, xml2gtf, xml2xml: Takes a ground truth file from the input stream, or a list of files passed as command line arguments, and converts them to the specified output type. The gtf2gtf and xml2xml tools serve two purposes, to clean up files that contain improper formatting, or to combine multiple files into a single file. To merge multiple files in the older GTF format, make sure that they contain the FILE Information descriptor, described in the Ground Truth File Format section of the ViPER-GT manual. They also take two optional arguments, -split and -clip. Splitting splits objects that are not contiguous into separate objects, while clip returns the first frame of every object. Both are useful in evaluation circumstances; -split for evaluating algorithms that do not support tracking across occlusions, and -clip for generating first-frame starting points for TRACKING evaluations.

viper-pe: *The command to run a performance evaluation, its options are described in a separate section. FAQ/Trouble Shooting*

## **5 Appendix II: Notes**

### 5.1 FAQ