Computer Science
Technical Report

Colorado
State
University

LiME Users Guide *

J. Ross Beveridge
Colorado State University
ross@cs.colostate.edu

November 20, 1997

Technical Report CS-97-122

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792   Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

LiME Users Guide

J. Ross Beveridge

Colorado State University

ross@cs.colostate.edu

November 20, 1997

Abstract

This Users Guide provides a brief overview of the software system LiME which performs optimal 2D line segment matching. LiME searches for the optimal many-to-many correspondence between a model and a set of data features, where each is expressed as a set of 2D line segments. The model may be rotated, translated and scaled relative to the data during matching. LiME is written in `c++` and is designed to be run as part of the DARPA Image Understanding Environment (IUE). There is, in addition, a stand alone version of LiME. Also distributed with LiME is a `Java` graphical user interface which will assist a user in running LiME. The roughly 65 user specified parameters read by LiME are laid out in a series of logically grouped control panels with accompanying on-line documentation. A brief tutorial in included at the end of this Users Guide.

Contents

List of Figures

1 Introduction

LiME, or more fully, the **Li**ne **M**atching **E**nvironment, is a set of algorithms for matching object models to image features where both are expressed as sets of 2D line segments. The LiME system itself is written in `c++`: there are roughly 20 source files which makeup LiME. The execution of LiME is typically controlled through a parameters file which contains 65 entries. Any of these parameters may alternatively be passed to the executable as command line arguments.

To assist users working with LiME, a graphical users interface for selecting and modifying control parameters has been developed in `Java`. This companion system, called **Limeade**, organizes the control parameters into logical sets, provides on-line documentation, and allows users to quickly change parameters and re-invoke LiME.

The purpose of this users guide is to assist someone installing LiME and to provide a general overview of the what LiME can do. It includes some tutorial examples which should help a user confirm that the system is correctly installed as well as gain familiarity using the system.

1.1 An Example of What LiME Does

LiME has emerged out a long tradition of object recognition algorithms used in the computer vision community which match geometric models of object to features extracted from imagery. Consequently, it does not match pictures of object to other pictures, but instead geometric features. LiME's definition of a feature is very simple: a feature is a 2D line segment.

An object model is simply a set of 2D line segments. These segments need not form a closed contour, in fact there are virtually no constraints on the segments. However, if one is to go beyond using LiME on the models and data supplied with it, then a means of deriving object models appropriate to one's chosen domain must be found. In general, models can be derived from 3D CAD models, extracted from cleaned up training images, or extracted from rendered images of 3D object models. A review of the technical papers written about the algorithms included within LiME will show examples of these and more.

Data is also represented simply as a set of straight line segments. In our own past work with LiME, line segments have been extracted from imagery using the Burns Algorithm [BHR86]. However, a variety of straight line extraction algorithms have been developed and might be used [NB80, LB82]. If you are using LiME as part of the IUE (DARPA Image Understanding Environment, see Section 1.2 below) then you will also have access to our implementation of the Burns algorithm.

LiME will use one of three heuristic optimization algorithms in an attempt to find the globally optimal many-to-many match between model and data segments. In the course of searching, the model will be rotated, translated and scaled so as to best fit the corresponding data. Figure 1 shows

a particularly challenging matching problem from the standpoint of cluttered and fragmented data.

1.2   LiME and the Image Understanding Environment

While LiME and Limeade have been developed to run in a stand-alone fashion without any dependence upon other systems, they are nevertheless much more useful in the context of a larger environment. For example, one is limited in using LiME if one does not possess an algorithm for extracting straight line segments from imagery. Consequently, LiME has been incorporated into the Image Understanding Environment (IUE). Specifically, a wrapper has been written to allow input in the form of IUE objects (`IUE_parametric_line_segment_2d`) stored in IUE data exchange files. More is said about LiME and IUE in Section 1.2.

For those readers not familiar with the Image Understanding Environment, it is a comprehensive set of `c++` object definitions and associated libraries designed to support image understanding research and technology transfer. The IUE is publicly available through a website maintained by Amerinex Applied Imaging: `http://www.aai.com/AAI/IUE/IUE.html`. It has been developed with the support of the DARPA Image Understanding Program and provides a common environment for the development and distribution of algorithms.

2   Installation

2.1   You Need Unix and the X11 Library

LiME runs on Unix workstations and it is currently known to operate properly under Solaris and Linux. LiME displays a window which shows visually the progress and final results of matching. The X11 library is used to create this window; no higher level windowing support is required. LiME should in principle port to any Unix workstation with X11. If you do port LiME to another architecture, please let us know.

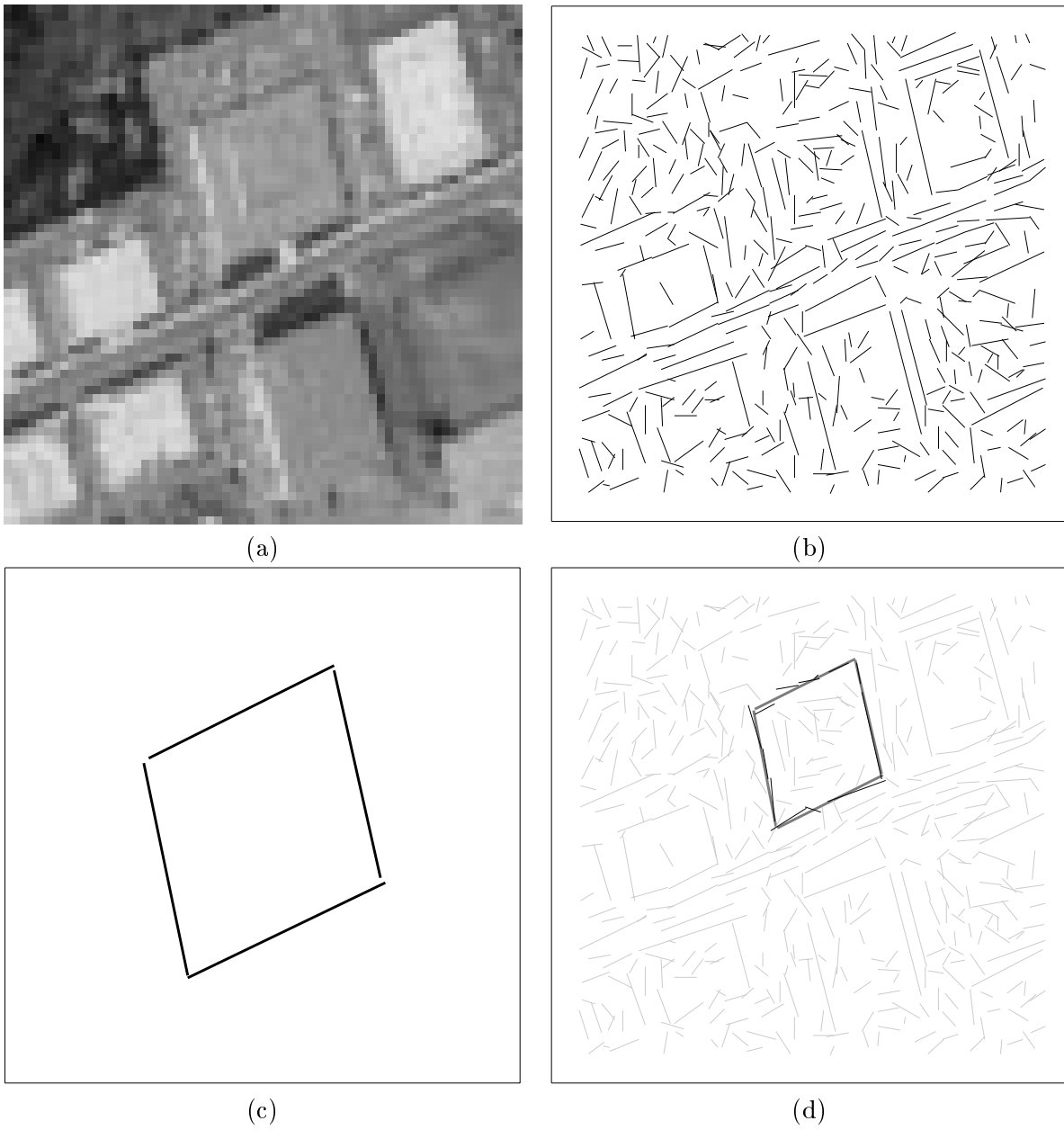A typical installation of LiME has the following directories.

(a)

(b)

(c)

(d)

Figure 1: Optimal match to fragmented data. a) aerial photograph, b) segments [BHR86], c) model, d) best match.

$LIME_HOME/SunOS5
$LIME_HOME/SunOS5/depends
$LIME_HOME/data
$LIME_HOME/java
$LIME_HOME/java/AnimationImages
$LIME_HOME/noniue
$LIME_HOME/src
$LIME_HOME/src/SunOS5
$LIME_HOME/src/SunOS5/depends
$LIME_HOME/src/SunOS5/depends/tmpl-inst
$LIME_HOME/src/SunOS5/tmpl-inst
$LIME_HOME/src/files
$LIME_HOME/tmpl-inst
$LIME_HOME/tutorial

The data directories contain examples of model and data line segment sets. In some cases they also include the imagery from which these models and data are derived. They are further described below. The scripts directory contains a small script which should be run upon login to customize your Unix environment for LiME. This script resides in the file `.lime_env`.

There are two source directories, one for the `*.c++` code which makes up LiME itself, and one for the `Java` code for the Limeade GUI which helps one to run LiME interactively.

2.2   The source files which makeup LiME.

The following is a complete list of the `*.cc` files which constitute the LiME distribution under the src directory which is used to build the IUE version of LiME.

limeBaseClass.cc	limeLineClass.cc	limeTableau.cc
limeCohenSutherlandClipping.cc	limeMain.cc	limeTableauBuild.cc
limeDEXIO.cc	limeMatcherClass.cc	limeTableauClass.cc
limeFit.cc	limeOmission.cc	limeUtils.cc
limeGenetic.cc	limePairClass.cc	limeVectorTransformations.cc
limeGrid.cc	limeParameters.cc	limeWindows.cc
limeGridClass.cc	limeSearch.cc	limeInterface.cc
limeSignal.cc	limeDEXIO.cc	

The following `*.h` files are also part of the LiME distribution:

limeBaseClass.h	limeIncludeSources.h	limeParameters.h
limeChromo.h	limeIncludes.h	limePointLineUtils.h
limeChromoVSz.h	limeInterface.h	limePointPointUtils.h
limeCohenSutherlandClipping.h	limeLineClass.h	limeSearch.h
limeConstants.h	limeLineUtils.h	limeSignal.h
limeDefines.h	limeMacros.h	limeTableau.h
limeFit.h	limeMatcherClass.h	limeTableauBuild.h
limeGenetic.h	limeMessyChromo.h	limeTableauClass.h
limeGlobals.h	limeMessyGene.h	limeUtils.h
limeGrid.h	limeOmission.h	limeVectorTransformations.h
limeGridClass.h	limePairClass.h	limeWindows.h

Under the `noniue` directory there are duplicates of several files which enable a user to build a stand alone version of LiME. These files are:

limeInterface.cc   limeMain.cc

2.3   The Source Files Which Makeup Limeade

Limeade is written in `Java` and consists of the following files:

CanvasLabel.java	CustomFileDialog.java	Data.java
DataCheckbox.java	DataChoice.java	DataTextField.java
FramedArea.java	HelpLabel.java	HelpSystem.java
LimeDisplayReportingFrame.java	LimeFileFrame.java	LimeMatchErrorFrame.java
LimeMatchSpaceFrame.java	LimeSearchFrame.java	LimeTabFrame.java
LimeadeTop.java	OpensFile.java	PathField.java
StringVector.java	TabPanel.java	TrivialApplication.java

These files should be compiled for the Java virtual machine. Under Solaris, this is done using the command `javac`. The Java compiler is smart enough that if you compile the root class, the file `Limeade.java`, the supporting classes will be compiled as well. This appears only to be true so long as the files reside in the same directory. Perhaps later versions of the java compiler will be better at handling class source files distributed across multiple directories.

2.4   Changes to your Run-Time Environment

There is a short script at the lime home directory called `lime_env`. This should be customized and sourced by your `.cshrc` in order to setup paths and environment variables to support LiME and Limeade. Figure 2 shows a copy of this script. After running the environment script, you should

be able to run LiME and Limeade from any directory.

There are several things done for you in this script. Perhaps the most important is that it defines an environment variable:

LIME_WORK

This should be be your current directory when running LiME. To reset this environment variable during your session the alias `limeworkhere` is provided. *If you have problems with LiME, a likely cause of difficulty is not have the proper binding for this environment variable.*

2.5 Compiling LiME

There are two Makefiles distributed with LiME, one for LiME as part of the IUE, and the other for the stand alone version of LiME.

2.5.1 Compiling LiME for use with the IUE

To build the IUE version of lime you should type `make` at the `LIME_HOME` directory. The subdirectory structure below `LIME_HOME` is consistent with use of the standard makefile distributed with the IUE. You may need to of course to make the standard path adjustments to match your site.

In keeping with IUE convention, a library is built for LiME which is then dynamically linked to the main executable which resides at `/s/parsons/a/fac/ross/vision/src/lime1.0/SunOS5` .

One key thing to keep in mind, the name of the actual executable is

limeIUE

2.5.2 Compiling LiME without IUE

A version of LiME may be built which does not use IUE. It uses most of the source files in the `src` directory and the additional files in the `noniue` directory. The name of the executable for the stand alone version of LiME is:

lime.

To build the stand alone version, invoke `make` with the argument `-f MakefileNonIUE`. The preamble of this makefile will probably need to be customized to reflect how each site name the paths to standard libraries.

```
#! /usr/local/bin/csh

## CHANGE THE NEXT LINE TO MATCH YOUR SYSTEM!
setenv LIME_HOME /s/bach/g/proj/vision/src/lime1.0

setenv LIME_SRC      $LIME_HOME
setenv LIME_BIN      $LIME_HOME/SunOS5
setenv LIME_WORK     ./
setenv LIME_DEMO     $LIME_HOME
setenv LIME_DATA     $LIME_HOME/data
setenv LIMEADE       $LIME_HOME/java

# Use this to change lime working directory to current one.
# LIME_WORK is where output of lime is directed.
alias limeworkhere 'setenv LIME_WORK 'pwd''

## The following aliases are simply here for convenience
## They are not actually used by either LiME or Limeade.
alias golime         'cd $LIME_HOME'
alias golimebin      'cd $LIME_BIN'
alias golimesrc      'cd $LIME_SRC'
alias golimework     'cd $LIME_WORK'
alias golimedemo     'cd $LIME_DEMO'
alias golimedata     'cd $LIME_DATA'
alias golimeade      'cd $LIMEADE_SRC'

# Add Limeade to the Java Class Path
setenv CLASSPATH  $CLASSPATH\:${LIMEADE}

# Add LIME_BIN and LIMEADE to path
echo $PATH | grep $LIME_BIN
if( $status ) then
 setenv PATH ${PATH}:$LIME_BIN
 setenv PATH ${PATH}:$LIMEADE
endif
```

Figure 2: Script that modifies run-time environemnt to support LiME

2.6   Compiling Limeade

Limeade is made up of a series of `Java` classes all kept at the directory

`LIME_HOME/java:`

Also at this directory is a short shell script that will call the `Javac` compiler to build the Limeade classes. The name of this script is `makeLimeade`.

3   A Brief Overview of the Three Matching Algorithms

LiME contains three quite different heuristic matching algorithms: a random starts local search algorithm, a key-feature algorithm and a messy genetic algorithm. All three algorithms seek to minimize a match error which in turn uses a quadratic fitting process to globally align models to data. An intuition for how this fitting and ranking process works is best developed by watching the visual display of matches provided by LiME. During local search, LiME can be instructed to draw each new best-fit configuration of the model relative to the data. Fitting and the associated match error are fully described in [Bev93].

Our work on optimal line segment matching began with the random starts local search algorithms and first considered rigid 2D objects [BWR89]. We quickly discovered that fitting 2D models to data is becomes simpler, not more complex, when scale is allowed to vary. This extension appeared in [BWR90], and while we did not realize this at the time, we soon discovered our fitting procedure is an extension of similar work by Ayache [AF86].

LiME has two different local search algorithms available, Hamming-distance 1 steepest descent and subset-convergent. An early comparison of the two appeared in [Bev92] and a more thorough comparison appears as part of [JRBG97]. This latter paper addresses the basic question of how run-time scales as a function of problem size. Both the algorithms and data used in this paper come as part of the LiME distribution.

The other two algorithms have been developed more recently. The key-feature algorithm is explained and compared to random starts local search in [JRBS97]. The messy genetic algorithm is explained and compared to random starts local search in [DWG97].

The key-feature algorithm begins by finding spatially proximate triples of model and data line segments and ranking these from best to worst according the match error. Then a simplified local search is used to fill-out a match from this initial triple. Triples are formed by identifying the two nearest neighbors to each model line in the model space and the two nearest neighbors to each data line in the data space. Then, for every possible pair of model and data segments, two triples are formed by combining these two nearest neighbors in pairwise matches.

The messy genetic algorithm uses the same spatially proximal triples as does the key-feature algorithm. However, rather than attempt to build matches independently off the k best triples, it places these triples into a population and uses a type of genetic algorithm, a Messy GA [GDKH93], to iteratively rank, select and recombine elements from the population until larger and better matches emerge.

It has been our own experience to date that the messy genetic algorithm appears to best combine speed with robust performance. However, one value of setting up matching problems within LiME is that you can easily experiment will all three algorithms and draw your own conclusions.

4   Understanding LiME's Parameters through Limeade

Limeade has been written in large part to hide the contents of the LiME parameters control file from beginning users: as well as from experienced users who grow tired of constantly editing a cryptic file. This section gives and overview of what you will see when starting up Limeade and how it interacts with LiME. The five panes used to specify logically related parameters are shown and described in general terms. Since Limeade provides on-line help for each selection, field by field descriptions are not provided here.

4.1   What Limeade Does

Limeade lets you run LiME interactively, quickly varying parameters and observing changes in performance. It is intended that you will create a directory in which you wish to run one or more interactive experiments and `cd` to that directory. Then, simply invoke Limeade by typing `Limeade`. You will be presented with the Limeade startup window which contains an animated Logo and a file path: the path to a LiME parameters file. An example of the startup window is shown in Figure 3.

The **choose** button may be used to invoke a file browser with which to select a LiME parameters file. The **open** button will open this file, which means it will read the file's contents and give you a GUI through which you can modify the file. There is a **Save & Run** option which lets you run LiME directly without leaving Limeade. In this fashion, you can quickly and easily modify the parameters and try different variations.

A warning, Limeade currently will show a default file name even if such a file does not exist, therefore you will want to copy a default parameter file from the LiME `LIME_HOME` directory to your working directory. Unfortunately, if you **open** an non-existing file the current version of Limeade just sits and does nothing. Future versions will hopefully at least give some warning that no file was found.

Figure 3: Limeade Startup Window.

4.2 The Five Limeade Parameter Panes

The LiME parameters controlled by Limeade are grouped into five distinct logical categories: file inputs, match error, match space, search and display/reporting. File inputs specify where LiME is to read the model, data and optionally and image used for display. Match error parameters alter the way the match error weights different terms. Match space parameters determine how LiME selects which model line segments might match which data line segments. Search parameters control which of the three search algorithms are used as well as selecting different types of behavior within each algorithm. Finally, display and reporting parameters alter how LiME allows the user to visualize matching progress, whether information about the matches found is stored to a log file, and what text is printed to standard out.

One moves between panes by selecting the desired pane name from along the top of the window. Along with the pane selection buttons, the **Save**, **Save as...**, **Save & Run** and **Done** buttons are always displayed regardless of the pane chosen. These are used to save the parameters file, save
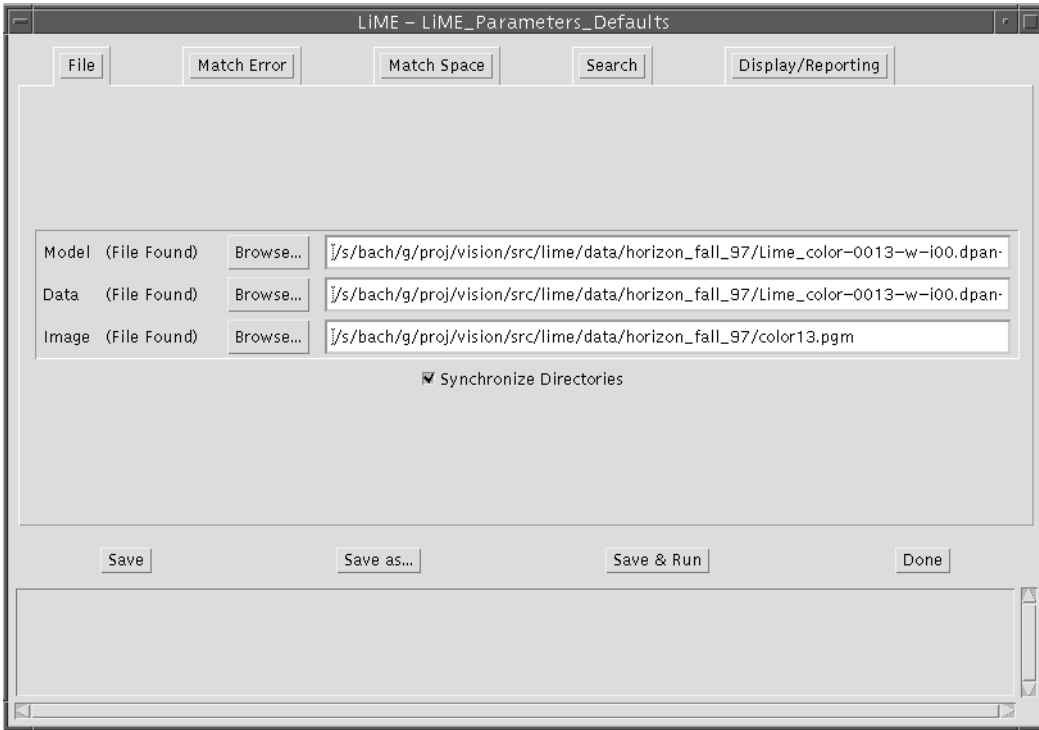
Figure 4: Limeade File Pane.

the parameters file to a file with a new name as specified through the File Browser, to save the parameters file and then invoke LiME, and finally close this window when finished experimenting with this parameters file.

At the bottom of this window is a help window which will display a short help message associated with whatever field the mouse is currently over. This is perhaps the best way to refresh your memory about what each of the parameters does.

4.3 The File Pane

The File Pane, Figure 4, allows a user to select model, data and image input files using the File Browser. When using the stand-alone version of LiME, the model and data files are simple ASCII files containing line segments endpoints. Here is an example of the file which contains the rectangle model from file $LIME_HOME/data/rectangle/Lime_rectangle-model-lines.dat:

```
9.310 9.310 9.310 85.120
```

11

```
9.310 85.120 94.430 85.120
94.430 85.120 94.430 9.310
94.430 9.310 9.310 9.310
```

When using the IUE version of LiME, the model and data files are IUE DEX files containing a sequence of `IUE_parameteric_line_segment_2d`.

The **Synchronize Directories** checkbox tells the File Browser to use a common pathway to find model, data and image files. This is convenient since typically these files are stored in a common directory.

The image file is not actually used by the matching algorithms. Instead, it is used by LiME in the search display window in order to show the original source image along with the line segments during matching. If display of an image is not desired or there is not image available, the string **None** is treated as a special flag to LiME indicating that there is no image to use in its display.

4.4 The Match Error Pane

The Match Error Pane, Figure 5, allows a user to vary the parameters which determine the exact form of the match error which is being optimized. While beginning users of LiME are not encouraged to play with all the degrees of freedom in the Match Error, it is important to understand that this is a parameterized error function which can, and usually should be customized for new domains.

Of all the parameters listed on this pane, the one which is most likely to require changing is **Maximum Displacement**. The units for this parameter are pixels, and maximum displacement in effect controls how far apart (measured by perpendicular distance) a model and data segment can be and still be considered a good match. What value to choose depends upon the degree of skew expected in the line segment data as well as the accuracy of the model.

The other parameter worthy of specific mention here is the **Scale Range**. There is a penalty term as part of the match error which penalizes matches which make a model grow either too large or too small. The penalty kicks in at the value of **Scale Range**. Hence, for a value of 2, there is no penalty so long as the model ranges between one half and twice its original size. Any scale change outside this range incurs a penalty. This parameter can be used to focus attention on matches which essentially leave the model the same size. One caution, however, if **Scale Range** is made too tight, then search can become nearly impossible due to all intermediate matches looking equally dreadful. Take care when using a value of less than 1.5 for the **Scale Range**.

All of the match error parameters correspond to parameters described in Beveridge's Thesis [Bev93]. To gain a solid understanding of how match error is formulated, computed and parameterized we recommending reading Chapters 3 and 4.
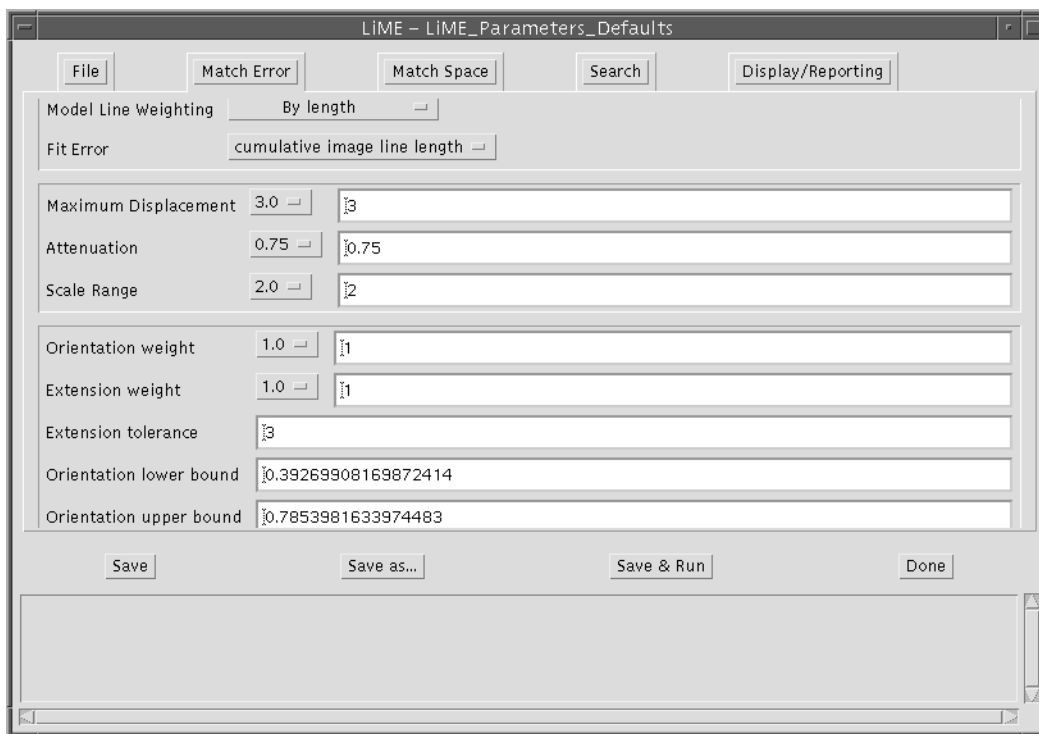
Figure 5: Limeade Match Error Pane.

4.5   The Match Space Pane

The Match Space Pane, Figure 6, allows a user to vary the parameters which determine the initial combinatorial space of possible matches to be searched. Once LiME reads in the model and data segments, it then constructs a set of pairs of possibly matching model and data segments. The search space for matching is then the powerset of this set of pairs.

In terms of how this search space is constructed, there are essentially two modes in which one can operate LiME. The first presumes that one lacks any prior constraint upon where the model appears in the image. In this case, any model segment might match any data segment and the complete set of model segments cross data segments form the basis for the search space. The second mode presumes some initial guess as to the placement of the model in the image. Based upon such a guess, it is possible to rule-out certain model-data pairings. For example, a vertical model line segment could be precluded from ever matching a horizontal data segment.

To reflect these different modes, there are actually three ways that LiME offers for constructing the initial set of possibly matching pairs. These are selected by the menu choice **Match Space**,
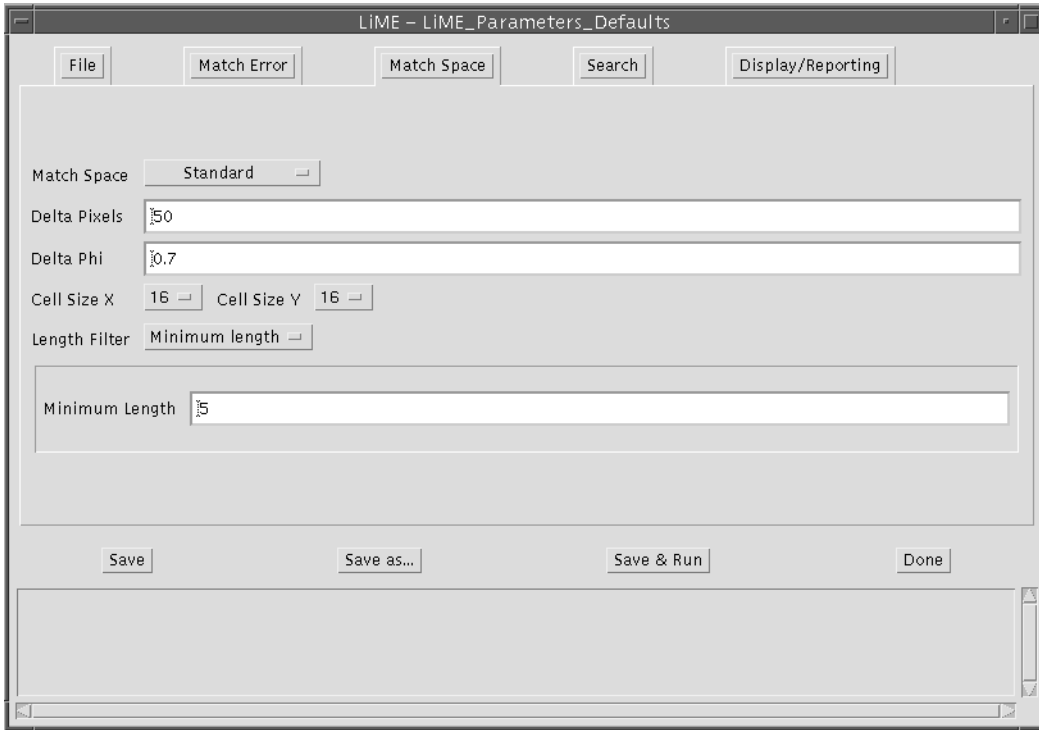
Figure 6: Limeade Match Space Pane.

which can be either **Complete**, **Standard** or **Constraint Based**. Choosing **Complete** builds the complete set of pairs in which all model segments possibly match to all data segments. This is the default and the proper setting if no prior knowledge of the model's placement in the image is available.

Both the **Standard** and **Constraint-based** modes use the initial placement of the model to filter out data segments. Specifically, there are two additional parameters which determine how the **Standard** mode operates: **Delta Pixels** and **Delta Phi**. Any pair of model and data segments with a minimum Euclidean distance between segments greater than **Delta Pixels** are removed from the set of possibly matching segments. Any pair of model and data segments which differ in orientation (unsigned) by more than **Delta Phi** are removed from the set of possibly matching segments. **Delta Phi** is expressed in radians.

The **Constraint-based** mode uses **Delta Pixels** and **Delta Phi** in the same way as the **Standard** mode. However, it applies an additional length constraint. If **Length Filter** is set to **Minimum Length**, then any data segment shorter than the specified minimum length is removed from any pairing with any model segment. If **Length Filter** is set to **Length ratio**, then a model-data
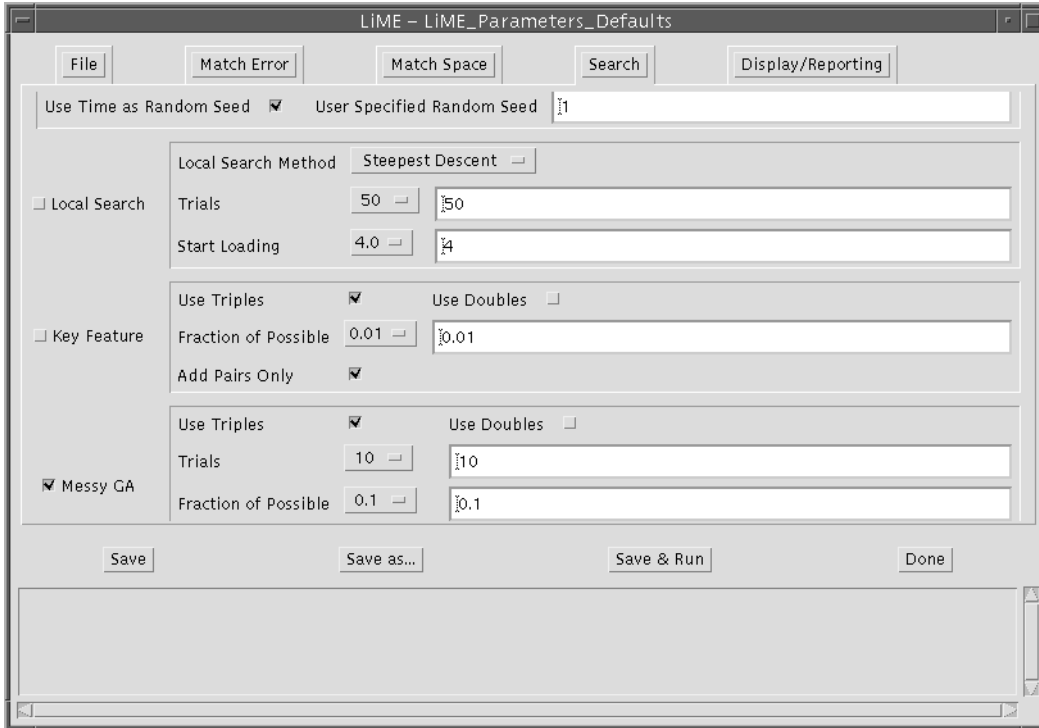
14

Figure 7: Limeade Search Pane.

pair is removed if the data segment is shorter than some percentage of the length of the associated model segment. It is also possible to specify that each model segment should consider matching only the K longest data segments that satisfy the **Delta Pixels** and **Delta Phi** constraints.

4.6 The Search Pane

The Search Pane, Figure 5, allows a user to select which of the search algorithms LiME will execute. The algorithm may be chosen either from the **Match Method** menu or by selecting the appropriate checkbox. Most parameters are algorithm specific and the pane is organized accordingly. The exception to this rule concerns the random number generator.

Both the Random Starts Local Search and Messy GA methods are non-deterministic and use the `srand48` random number generator. By default, each execution of LiME uses the time as the random seed, thus no two runs are likely to be the same. However, a user may specify a random seed. This is helpful if looking for a particular behavior which one wants to repeat either for debugging of illustration purposes.

There are three parameters which control the behavior of the random starts local search algorithm. The method is either **Steepest Descent** or **Subset Convergent**. Typically one runs between 5 and 100 trials when watching the system: anymore and the wait gets annoying. The parameter **Start Loading** determines, on average, how many data line segments are matched to a model segment in the randomly chosen initial matches.

Both the Key Feature Algorithm and Messy GA use spatially proximal triples and/or doubles to focus the attention of the search process. There are $2n$ proximal triples, where n is the number of possibly matching model and data segments. There are n proximal doubles. Thus, the total number of key features is $2n$ when using only triples, n when using only doubles, and $3n$ when using both.

Typically, the Key Feature algorithm uses only the best key features. How deep to go down into the ranked list of Key Features is controlled by **Fraction of Possible**. Recall the features are ranked from lowest to highest match error. Thus, a value of 0.10 instructs the algorithm to use only the best 10%: $0.2n$ when using triples only. The parameter **Add Pairs Only** indicates that the local search procedure used to fill out matches from each Key Feature will consider only the addition of pairs. If this flag is turned off, then local search will allow both additions and deletions, and local search will become capable of moving off of the Key Feature if a better alternative is found.

The Messy Genetic algorithm is typically run multiple times, since like random starts local search, it may not converge to the best match in any given run. However, the Messy GA is typically much less likely to become hung up on a local optima, and therefore far fewer trials are typically run. Typically between 1 and 10 trials are sufficient.

The size of the initial population used by the Messy GA is controlled by the parameter **Fraction of Possible**. This is analogous to the parameter of the same name used by the Key Feature algorithm. However, in this case it is expected that a larger value should be used: it is better to include more triples/doubles rather than fewer in the initial population.

The last parameter for the Messy GA is the maximum number of generations to run. The Messy GA uses the Genitor [WS90] selection strategy and therefore the definition of a generation may strike some as counter intuitive. In Genitor, a generation involves the selection and recombination of a *single* pair of parents. The number of generations should typically be in the thousands. The Messy GA may terminate before this number of generations if the population is determined to have converged.

4.7   The Display/Reporting Pane

The Display/Reporting Pane, Figure 8, presents parameters which alter how LiME displays the progress of search to the user as well as how the results of matching are logged to disk.
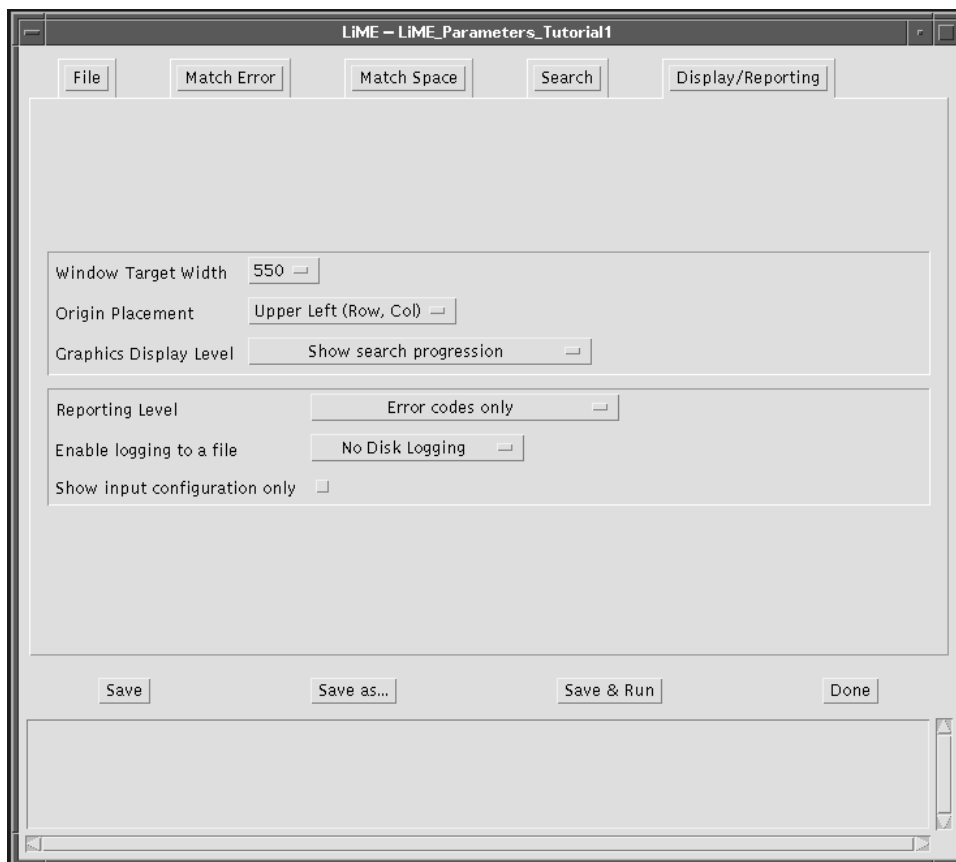
Figure 8: Limeade Display/Reporting Pane.

LiME has the ability to create a window in which it visually displays the matches being found. Whether this window is created, and what is displayed to it, is controlled by the choice parameter **Graphics Display Level**. The least interesting option is **no display**. This setting, is however very useful if you want to save a parameters file suitable for running LiME in a batch processing mode without any graphics.

When displaying the progress of the matching algorithm, there are four modes or levels of detail which can be requested. The lowest is to request that the best found yet be displayed. This is useful when the goal is to monitor how good of a match the system is finding. The next level is to request to see the result of every trial. This is somewhat harder to interpret, since good matches and bad matches will go by in rapid succession depending upon how each trials comes out. For the Key Feature algorithm, where there are no trials, the results of filling out each key feature will be displayed in turn.

The third level of detail is to request to see every match found by local search as it progresses from the initial to final (locally optimal) match. This mode makes the most sense for the random starts local search algorithm, and it is highly recommended that users use this mode with trials set to 1 or a small number in order to develop a sense of how the global fitting process positions the model during search and for which matches are favored over others.

There is a final level in which the user is prompted to hit the enter key before search will take the next step. Warning, this mode will only work if you invoke LiME from the command line, so you must save the parameters file and then run LiME from outside Limeade.

There is two other parameters which relates to the display of the matches, and this is the **Window Target Width**. This parameter controls the size of the display window on your screen. It can be adjusted to suit the users tastes and the size of their screen. The other is the **Origin Placement**. LiME currently allows the coordinate pairs (a, b) of points read from a file to be interpreted in two ways: either with the origin at the upper left of the window (row, col) or with the origin at the lower left (x, y).

The **Reporting Level** parameter controls the detail contained in messages printed by LiME to standard output. Again, at the risk of being redundant, these reports will only be seen when LiME is invoked from the Unix command line. The more typical way to capture the results of matching for later analysis is through the writing of Log files, and there is a parameter which enables logging. The file will appear in the directory from which LiME (or Limeade) is invoked. The name of the Log file is specified in the parameters file, but is not modifiable through Limeade.

There is one last flag in the Display/Reporting Pane that is very useful. Sometime you want to see the relative position of the model with respect to the data without actually doing any matching. This can be done by setting the **Show input configuration only** checkbox. If you then run LiME, you will see the model drawn exactly as it appears in the model file. Recall that in the standard mode of defining the search space, delta pixels and delta phi for data segments relative to model segments is defined relative to this initial configuration. One caution, with this box checked, LiME no longer performs matching (and may therefore appear broken if you have forgotten this option is chosen).

5   Tips, Trouble Shooting and Known Bugs

The separation between LiME and Limeade is in some respects unfortunate. It reflects more the process by which LiME came into existence then it does a conscious preference. However, along with disadvantages, some if which will be discussed here in regard to some tricks that make life easier, it is worth pointing out that the separation has the advantage that LiME is readily run under the control of Unix scripts with no GUI interface to get in the way.

5.1 Java is unable to find Limeade and/or Limeade cannot find files

Limeade is run by the Java virtual machine and it is intended to be invoked by a script `Limeade` kept in the LiME `bin` directory. This script expands to a single command line which calls the Java virtual machine, invokes the Limeade Java application, and passes two Unix directories into Limeade. The first is the directory where the Limeade `.java` and `.class` files are kept. The is actually only needed so that Limeade can find the images used to construct its initial animated Logo. Possible problems with Limeade may sometimes be traced to incorrect directory specifications.

5.2 Limeade Hides LiME text output including error message!

By design, when running LiME from Limeade, Limeade will parrot precisely the Unix command needed to duplicate how it would call LiME. This is hack to get around the unfortunate property that how Limeade tells you that there is a problem running LiME is by not saying anything at all. Granted this is very frustrating, but the only way out of this problem is to pass a text I/O stream into LiME from Limeade, and the interface is not to date that sophisticated.

If for any reason LiME does not appear when invoked from Limeade, the recommended procedure is to copy the parroted command into the Unix prompt and run LiME directly from the command line. In this way you will see whatever error message LiME may have intended you to see.

5.3 Limeade Does not Recognize you Quit LiME

When you run LiME from Limeade, the button that usually says **Save & Run** changes to **Kill Lime**. However, when you terminate the run of LiME by clicking on the **Quit** button in the LiME search display window, Limeade does not automatically switch back the label on the button to **Save & Run**. This is mostly a cosmetic annoyance, you can click on the **Kill Lime** button and it goes back to **Save & Run** and is ready to use again.

6 Tutorial Examples

The following tutorials will familiarize you with the use of LiME, Limeade and several interesting matching problems. These tutorials assume you are sitting down at a terminal with your own test directory and that you are setup to run Limeade. The tutorial will work equally well with either the IUE or NONIUE versions of lime. The only difference will be in whether you use `dex` files for the model and data line segments or the native `dat` files which work with the stand alone version of LiME.

The source data files that go with the tutorials can be found at the directory `LIME_HOME/tutorial/data` and the source Parameter files which include all the parameters settings as they should be set to start each tutorial are at `LIME_HOME/tutorial/Parameters` . The best way to start the tutorials is to create your own lime working directory under you own account and make your own copy of the Parameters File which correspondes to the tutorial you want to run.

Two *Warnings* before you start!

1. Be sure to run the script `limeworkhere` at your working directory before you attempt to run LiME or Limeade. If this script is not run in order to set the `LIME_WORK` environment variable, then LiME will not run properly.

2. Throughout the tutorial, all file names will have to be revised to match the paths for *your* installation of LiME. LiME and Limeade both store absolute paths in the Parameters Files. When LiME is installed, these match the paths at the previous site where LiME was assembled. You must fix this on a case by case basis.

6.1   Tutorial 1: The Rectangle and Random Starts Local Search

.

Copy the file `LiME_parameters_tutorial1` from the `$LIME_HOME/tutorial/parameters` directory into your test directory. Start Limeade by typing either `limeadeIUE` or `limeadeNONIUE` depending upon which version of LiME you wish to run. Just a note, the only difference between these two scripts is the name of the executable to run which is passed into Limeade. Limeade is otherwise the same.

Once the Limeade main window appears, use the **Browse** button to bring up the File Browser. From within the Browser select the file `LiME_parameters_tutorial1` from your current directory. Now open this file using the **Open** button. A note, in this browser you must double-click a selection so it appears as the selection at the top. Then to exit with this selection click on **Choose**.

Look at the File Pane and see if the Model and Data files have been found. This is indicated by a parenthetical comment next to the field labels: either **(File Found)** or **(Not a File)**. Probably the files have not been found, since the `$LIME_HOME` directory is going to be different for every installation, and the parameter file uses hard pathways rather than the `$LIME_HOME` environment variable.

Use the **Browse..** option to modify the pathways to the model and data files to reflect where this data is stored on your system. For this tutorial you want to select model file `Rectangle-model.dex` and data file `Rectangle-model.dex` (`Rectangle-model.dat` and `Rectangle-model.dat` if running LiME without IUE.) One other aspect of running with and without IUE. The placement of the
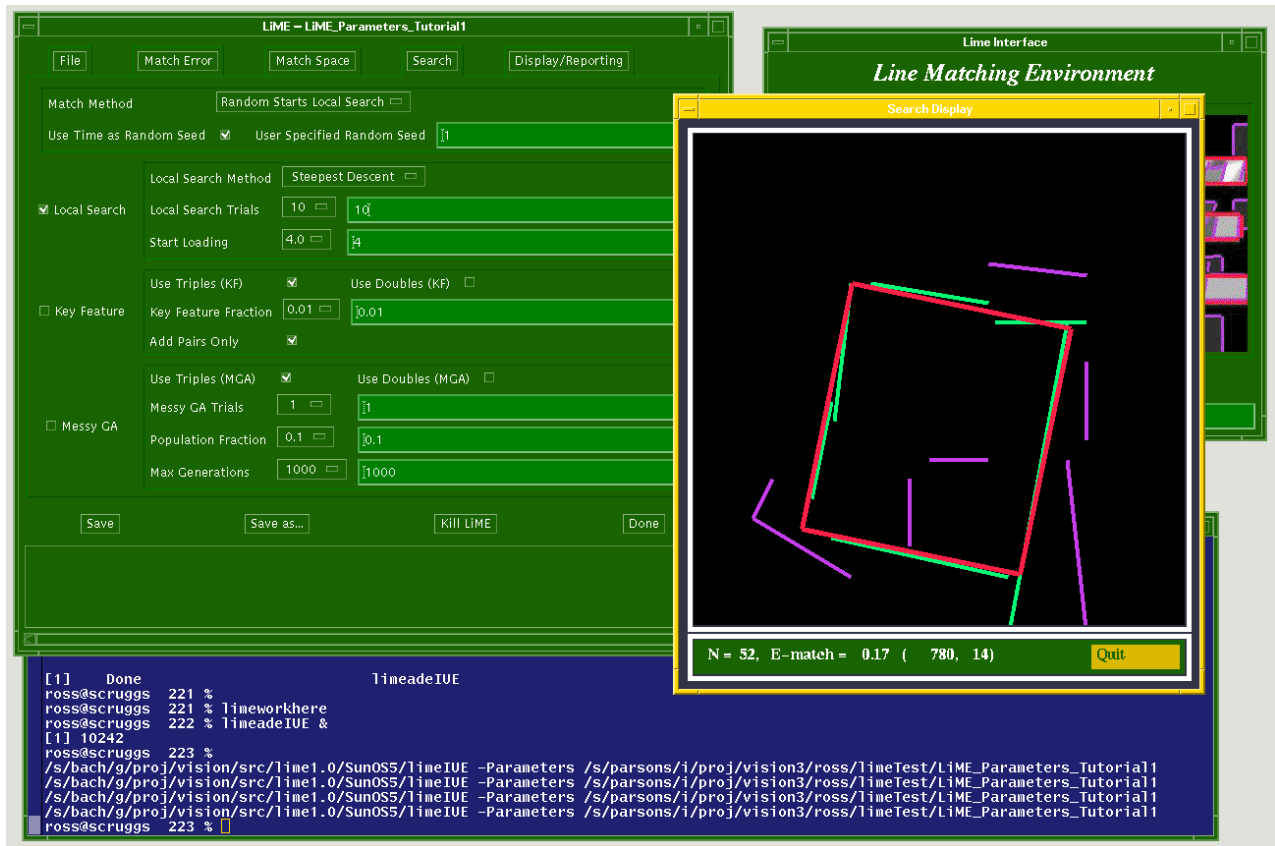
20

Figure 9: Example of all Windows Running LiME through Limeade.

origin of the images are different. The parameters **OriginPlacement** in the display and reporting pane can be used to select the correct origin.

The parameters file should be all set to run once you have specified the path to the model and data files. Confirm that the method selected is random starts local search and set the number of trials to 1. Then click on the **Save & Run** button. In several seconds you should see the LiME search display window appear. If you do not see LiME appear, then go to Section 5.2 and read about how to invoke LiME from the command line with the same arguments being used by Limeade in order to get messages from LiME about problems. Also, if you used the synchronized directories in the file Browser, odds are good you selected a path with no filename for the image and that this is causing the error. Go back to the File Pane and enter `None` in the image file field.

For reference, Figures9 show a snapshot of a screen for this tutorial. The mostly covered window in the upper right is the Limeade welcome window. The window on the top left is the Limeade

window in which LiME parameters are edited and from which Lime is run. The window in the lower right is the search display showing an optimal match for this rectangle example. Finally, the `xterm` from which Limeade was invoked is shown along the bottom. Note how the command line that would be needed to invoke LiME directly is parroted back to this terminal window.

6.1.1 Watching Single Trials of Local Search

Once you get LiME started properly, You will see the model (typically red) repeatedly drawn over top of the data for successively better matches. The button in the lower left of the search display window will say **Abort** while search is active. Once search terminates, this message changes to **Quit**. This switch of labels is an important cue letting you know matching has terminated.

Odds are small that the match you now see is truly optimal. You can gain some intuition for the nature of the local optima which trap the search algorithm by quitting and restarting LiME several times. You can do this either with the **Kill Lime** button in Limeade or with **Quit** button on the search display window.

6.1.2 Inspecting a Match

A nice feature of the search display window is that you can use it to inspect exactly which data segments have matched a given model segment as well as which data segments are candidate matches. Once matching has finished, the button now says **Quit**, you may click the mouse near a model segment and LiME will use different colors to indicate which data segments match. Any data segment found to match that model segment will turn bright green. The other segments which are candidate matches will turn dull green. At this point, a complete match space has been selected, so all data segments will be dull green.

You can inspect model segments repeatedly and thus go in turn through the entire match seeing visually exactly which data segments have been found to match which model segments. When you are finished, quit LiME.

This is also a good time to enable logging in the Display/Reporting Pane by selecting **Comprehensive Log** and run LiME. You should see a new `*.log` file in the test directory. Go ahead and load this file into an editor and inspect it. With detailed logging enabled the first thing to appear in the file is a copy of the parameters file. This keeps a complete record how the matches which follow were generated. Next comes a repetition of the model and data segments, followed by an enumeration of the set of possibly matching pairs. The remainder of the file lists the locally optimal matches found by LiME. A very short log file just summarizing which segments match can be generated by selecting **Log Match Summaries**.

6.1.3   Running Multiple Trials

Go to the Search Pane and select 10 trials of local search and re-run LiME. You will now see the complete progression of matches found by each of the 10 random trials of search. This can at first be a bit confusing. As soon as a locally optimal match is found, the display immediately switches to the random initial match of the next trial. However, after all trials are completed, the best match found by any trial is displayed. Now it is much more likely that the match you are seeing is in fact optimal.

If you only want to see when the system finds a match better than any it has seen yet, this can be done by dropping to a lower detail graphics level. Specifically, go to the Display/Reporting Pane and select **Show best currently found** for the **Graphics Display Level**. Now re-run LiME. Depending upon the machine on which you are running, you may now see a somewhat boring display, since on this problem the best match is quickly found and left up on the screen.

6.2   Tutorial 2: Key Feature algorithm for Horizon Matching

In this tutorial we will assume you have already completed Tutorial 1, and therefore be more concise with our instructions. Begin by copying the parameters file `LiME_Parameters_Tutorial2` to your working directory. Start Limeade and open this Parameters File. Go to the Search Pane and observe that the **Key Feature** algorithm has been selected. Also go to the Display/Reporting Pane and observe that the origin has been placed in the **Upper Left**. The line segments in the model and data have been generated with the Burns algorithm in the IUE and thus the choice of origin.

Also go to the Match Space Pane and observe that the **Standard** match space initialization technique has been selected. The means that only data segments within Delta Pixels of a model segment and Deltat Phi (in orientation) are considered to be candiate matches. Consequently, you should see "N = 90" displayed at the bottom of the search window when you run search: there are 90 candidate model-data pairs which are possible matches in this example.

Click on **Save & Run** and what you will see is an outdoor terrain image with a horizon line. The model in this case has been extracted from a rendered terrain map and the data is extracted from this image. The key feature algorithm will run a constrained local search from each of the top 21 spatially proximal triples. You will actually see each successive earch being carried out. Without attempting to explain all the details of the key feature algorithm, there are $1,067$ possible pairing between model and data segments, and twice that many possible spatially proximal triples: $2,134$. The **Key Feature Fraction** is set to 0.01, and therefore only the top 1% percent of the triples will be considered. One percent of $2,134$ is 21.

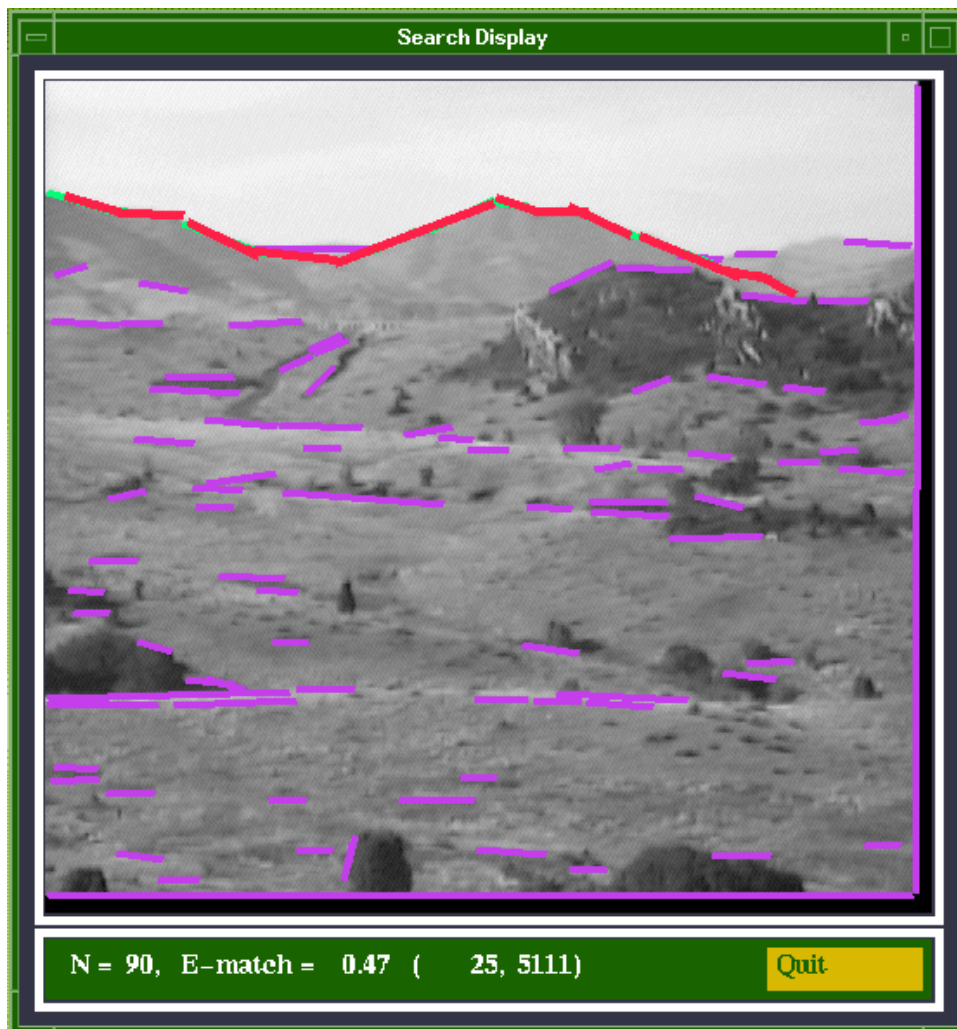Observe that for this example the Key Feature Matching algorithm has done well. The optimal

Figure 10: Horizon Match from Tutorial 2

match is shown in figure 10. You may compare the performance of the key feature algorithm to that of local search by selecting local search and re-running LiME. It is also instructive to run LiME from the command line by simply copying and executing the command line Limeade prints to standart output. In this way you can see the reports generated by lime as it runs successive searches. One word of caution, the messages talk about the Messy Genetic algorithm. This is a weakness in lime, the text output has not been fully updated to reflect each of the algorithms.

6.3   Tutorial 3: An Interesting Problem

This tutorial will consider the image and data for figure 1. This example has been used in several of our publications and it is one of the more challenging matching problems we have run across. Be prepared to make yourself a cup of coffe or tea at some points during this tutorial: to see interesting behavior make take several minutes of CPU time. Also, the data for this example is not in IUE Dex format, so you will be using the stand-alone version of LiME.

Begin by copying the parameters file `LiME_Parameters_Tutorial3_NONIUE` to your working directory. Start Limeade and open this Parameters File. The parameters are setup to constrain the search space around the center of the image. This will make it feasible to watch local search run on this problem. The search method is subset-convergent local search and the number of trials selected is 10. Also, the **Maximum Displacement** in the Match Error Pane is set to 2.0 pixels. This is important if the match shown in figure 1 is to be favored as the globally optimal match. The display mode is set to show only the best match found by each successive trial of random starts local search.

Go ahead and click on **Save & Run** and observe the variety of local optima which the algorithm finds in this data. If you are lucky, the best match may show up in the ten trials. However, it is not that likely. There are obviously *many* local optimal for the subset-convergent local search algorithm.

After you grow tired of watching the local search algorith, go to the Search Pane and select the **Messy GA**. For this problem, you should set the **Population Fraction** is set to 0.25 and run 1 trial of the Messy Genetic Algorithm. Also, in order to generate visual feedback on how the Messy GA is doing, select **Show Search Progression** from the Display/Reporting Pane. The Messy GA is a hybrid algorithm which runs local search periodically on individuals selected at random from the population. You will "see" these individuals being improved since these sort runs of local search will be displayed to the search window.

One trial of the Messy GA is not that likely to find the optimal match. Now comes the part where making coffee (or perhaps going out to lunch) domes in. You can run 10 trials of the Messy GA, and the probability that one of these will find the optimal match is fairly high. However, this may take between ten minutes to an hour depending upon your machine. Finally, you can also run the

key feature algorithm on this problem, and you will discover if you set the **Key Feature Fraction** low the algorithm quickly finds some sub-optimal matches. If you set the **Key Feature Fraction** to 1.0 then the algorithm will run for a long time and still only find sub-optimal matches.

7   More Advanced Features

The colors used by LiME in its search display can all be modified in the parameters file. These colors are specified using the hexadecimal notation commonly used by X windows. Here are examples of two different sets of colors. Also shown are the parameters which control such things as font color and line thickness. The set on the left are used in the tutorial.

-WindowFontColor	white	-WindowFontColor	white
-WindowBgColor	#196400	-WindowBgColor	#196400
-WindowFgColor	white	-WindowFgColor	white
-WindowFontColor	white	-WindowFontColor	white
-WindowButtonColor	#d8b800	-WindowButtonColor	#d8b800
-WindowButtonFontColor	#196400	-WindowButtonFontColor	#196400
-BgDataColor	#c43feb	-BgDataColor	#c43feb
-BgModelColor	#3fc4eb	-BgModelColor	#3fc4eb
-DataColor	#00ff77	-DataColor	#ff0000
-ModelColor	#fe2046	-ModelColor	#0000ff
-NormalColor	#b27020	-NormalColor	#b27020
-BgColor	#333346	-BgColor	#333346
-CurrentDataSpaceColor	#00cc88	-CurrentDataSpaceColor	#008888
-CurrentDataMatchColor	#00ff77	-CurrentDataMatchColor	#00ff00
-CurrentModelColor	#fe2046	-CurrentModelColor	#fe2046
-BgDataThickness	3	-BgDataThickness	4
-BgModelThickness	3	-BgModelThickness	4

References

[AF86]   N. Ayache and O. D. Faugeras. Hyper: A new approach for the recognition and positioning of 2-d objects. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 8(1):44 – 54, January 1986.

[Bev92]   J. Ross Beveridge. Comparing Subset-convergent and Variable-depth Local Search on Perspective Sensitive Landmark Recognition Problems. In *Proceedings: SPIE Intelligent Robots and Computer Vision XI: Algorithms, Techniques, and Active Vision*, volume 1825, pages 168 – 179. SPIE, November 1992.

[Bev93]   J. Ross Beveridge. *Local Search Algorithms for Geometric Object Recognition: Optimal Correspondence and Pose.* PhD thesis, University of Massachusetts at Amherst, May 1993.

[BHR86]   J. B. Burns, A. R. Hanson, and E. M. Riseman. Extracting straight lines. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI–8(4):425 – 456, July 1986.

[BWR89]   J. Ross Beveridge, Rich Weiss, and Edward M. Riseman. Optimization of 2-dimensional model matching. In *Proceedings: Image Understanding Workshop*, pages 815 – 830, Los Altos, CA, June 1989. DARPA, Morgan Kaufmann.

[BWR90]   J. Ross Beveridge, Rich Weiss, and Edward M. Riseman. Combinatorial Optimization Applied to Variable Scale 2D Model Matching. In *Proceedings of the IEEE International Conference on Pattern Recognition 1990, Atlantic City*, pages 18 – 23. IEEE, June 1990.

[DWG97]   C. Guerra-Salcedo D. Whitley, J. R. Beveridge and C. Graves. Messy Genetic Algorithms for Subset Feature Selection. In *Proc. 1997 International Conference on Genetic Algorithms*, pages 568 – 575, July 1997.

[GDKH93]   David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In Stephanie Forrest, editor, *Proc. 5th International Conference on Genetic Algorithms*, pages 56–64. Morgan-Kaufmann, 1993.

[JRBG97]   Edward M. Riseman J. Ross Beveridge and Christopher R. Graves. How Easy is Matching 2D Line Models Using Local Search? *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(6):564 – 579, June 1997.

[JRBS97]   Christopher R. Graves J. Ross Beveridge and Jim Steinborn. Comparing Random-Starts Local Search with Key-Feature matching. In *Proc. 1997 International Joint Conference on Artificial Intelligence*, page (to appear), August 1997.

[LB82]   David G. Lowe and T. O. Binford. Segmentation and Aggregation: An Approach To Figure Ground Phenomena. In *Proc. ARPA Image Understanding Workshop*, 1982.

[NB80]   R. Nevatia and R Babu. Linear feature extraction and description. *Computer Vision, Graphics, and Image Processing*, 13:257 – 269, 1980.

[WS90]   D. Whitley and T. Starkweather. Genitor ii: A distributed genetic algorithm. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(3):189 – 214, 1990.