The University of Saskatchewan
Department of Computer Science

Technical Report #2006-01

SkyD: Recognizing Stars in Astro-Imagery Based on Stable Local Geometric Patterns

Jeff Smith
Email: jeff.smith@usask.ca

Imaging, Multimedia and Graphics Lab
Computer Science Dept.
University of Saskatchewan

December, 2004

Contents

1 Introduction **3**

2 State of the Art **5**
2.1 Professional Projects . . . 5
2.2 Relevant Research . . . 6

3 The SkyD Algorithm **7**
3.1 Element Extraction . . . 7
3.1.1 SExtractor . . . 9
3.1.2 DAOPHOT . . . 10
3.1.3 Flood Search . . . 10
3.1.4 Analysis: Compute Time . . . 12
3.2 Pattern Construction . . . 13
3.2.1 The Obvious Pattern . . . 13
3.2.2 Scaling Invariance . . . 14
3.2.3 Luminance Response Invariance . . . 15
Magnitude . . . 16
3.2.4 Rotational Invariance . . . 17
Analysis: Trusting Little Brother . . . 18
3.2.5 Reflection Invariance . . . 19
3.3 Image Registration . . . 19
3.3.1 Analysis: The Scope of the Beast . . . 20
3.4 Pattern Matching . . . 21
3.4.1 Comparing Patterns . . . 21
Analysis: The Promiscuous Ray Problem . . . 22
3.4.2 Cleaving and Marginal Rejection . . . 23
Analysis: Shrimpy Little Brother . . . 24

4 Results **25**
4.1 Project Code . . . 25
4.2 Early Tests . . . 25
4.3 Test Data . . . 26
4.4 Verifying Success . . . 27
4.5 Analysis: Flaws in the Ointment . . . 28
4.5.1 Confirmation Confusion . . . 28

5 Image Matching **30**
5.1 Adaptive Source Weighting . . . 30
5.2 Iterated Pruning . . . 30
5.3 Aggregation Leverage . . . 31

6 Conclusions and Future Work **32**
6.1 Combinatorial Leverage . . . 32

List of Figures

3.1 Sample Input Image 8
3.2 Normalized Input Image 9
3.3 SExtractor Element Extraction 9
3.4 Overlapping Bounding Regions 11
3.5 Floodsearch Element Extraction 12
3.6 Neighbor Patterns for Cassiopeia 13
3.7 Bounding Size Fallacy 14
3.8 Primary Radiant 15
3.9 Neighbor Patterns for Rotated Cassiopeia 18
3.10 Comparing Patterns 21
3.11 Pattern Comparison Flaw 22
3.12 Pattern Cleaving 24

4.1 Test Data Details 26
4.2 Placement of Training Images In the Night Sky 27
4.3 Trial Successes 27

Chapter 1

Introduction

In the world of professional astronomy the processing of captured, digital images is quickly replacing the tradition of the lonely searcher with his eye pressed to the objective lens. The communication and analysis of such images is also maturing rapidly. Standard file formats (such as FITS) containing contextual information about the capturing technology, optics and sky position are in wide use. Access to massive, historical databases (see Section 2.1) and state-of-the-art processing tools (see Section 3.1.1) is fast and easy.

The amateur field, on the other hand, is a different story. Amateur sky watchers capture hundreds of images every minute, from every quadrant of the globe, ranging across every conceivable nook and cranny of the night sky. But the vast majority of these images serve little purpose other than to pad out the collection of the enthusiast who took it.

Many of the people who engage in this hobby would love to contribute in some meaningful way to the global body of knowledge, if only there was a way. Ideally that method of integration should require as little expertise as possible, to maximize the number of potential participants.

Throughout recent history, many of the more exciting discoveries in astronomy have been found by comparing images of the same region of sky taken at different times. In addition to the famous case of the discovery of Pluto, many supernovae, comets and near-earth objects have been discovered using this technique.

The objective of this larger context[1] is to develop an image analysis technique and registration database that will make it possible to compare images of the night sky, identify images of overlapping regions, and identify anomalous components of those regions that warrant further scrutiny.

[1] Beyond the scope of this project.

Ideally, such a system should be capable of comparing images, regardless of: camera orientation; telescope and lens optics; and image resolution. And in a perfect world, the contributors to this system should not be required to include any information at all regarding the region of sky depicted in their images.

I make these drastic fundamental assumptions for two reasons. First, we want to make the system function for people with minimum effort on their part. Many participants will have a vast library of images on file. If we require them to first enter contextual details about each image, participation will drop exponentially. Second, by requiring no contextual information, we eliminate the problems of dealing with, or even rejecting outright, images with inaccurate data attached.

The purpose of this specific class project was to solve the difficult portions of the larger project: developing the algorithms for successfully constructing these star patterns, registering them in a database and identifying registered patterns that match against specific input patterns taken from new input images. If this can be done successfully, the process of identifying registered overlapping regions between registered images and those newly input images becomes relatively simple.

Chapter 2

State of the Art

2.1 Professional Projects

There are a number of existing sky survey projects, scanning the heavens at various resolutions and wavelengths in order to create detailed baseline images for analysis and future comparison[1][**?**][2]. Indeed, there are even some formalized sky-search projects[1] whose aims are to identify anomalous phenomena for subsequent study.

But these projects are only really accessible to professional practitioners, using state-of-the-art equipment – multimillion dollar telescopes, located on remote mountain tops at high altitude and staffed by reasonably well-paid academics. Such projects are above the level at which amateur enthusiasts can readily participate.

These projects are also predicated on databases which relate a great deal of contextual data to each image. Information such as the capture optics, the coordinates of coverage, the atmospheric conditions, wavelengths, duration of exposure and so forth are all critical pieces of the data set that are used to match images. Requiring such rigorous contextual data would place a significant burden on amateur participants and would greatly increase the likelihood of error as well as presenting a barrier to participation.

It is for these reasons that I conceived this project, basing the image matching algorithm on nothing but the pixular content of the image. This represents the greatest possible simplification of the process and encourages amateur participation.

[1] such as SpaceWatch, SpaceGuard and the NEO (Near Earth Object) Project

2.2 Relevant Research

Little is apparently being done in the area of astronomic pattern recognition in the sense of identifying stars by the geometric pattern they form with their local neighbors. A great deal of work is being done on the classification front - classifying phenomena within images into categories such as nebula, star cluster, galaxy, etc.

One project[3] appeared to be promising. The Galileo explorer craft uses an imaging device to determine spacecraft orientation by identifying stars in its field of view. But upon closer examination, it turns out that the orientation camera on Galileo is comparing analog signals from a rotating slit-scanner and matching the 1-dimensional signal against a very restricted list of candidate stars. There is no two-dimensional pattern recognition at all.

Some work has been done (such as that of Boxer[4]) on the more generalized problems of matching geometric patterns in noisy point sets. But these algorithms do not include support for reflection invariance nor do they make use of point attributes such as magnitude. Adapting some of these neural net algorithms might have been feasible, but I felt that neural nets were a little too unintuitive to allow me to debug them if they didn't work *out of the box*. Consequently, this research has had no real influence over my own efforts.

Chapter 3

The SkyD Algorithm

In broad terms, the SkyD algorithm will construct geometric patterns by associating a target star with appropriate neighbor stars. The rays connecting the target star to these neighbors form a starburst pattern with a distinctive shape. These patterns will be computed for each star in each image from a set of test images. Those images and their patterns will then be registered in a central database from which subsequent queries can be extracted.

After training over the series of images, the system will be able to query that database for matches against patterns taken from a new test image. The matching process will consist of comparing the target pattern to an appropriate subset of patterns from the database, and scoring each with respect to its similarity with the target pattern. The highest scoring pattern from the database will be taken as the best match.

This process is decomposed into the following four steps: Element Extraction, Pattern Construction, Image Registration and Pattern Matching. A fourth stage, Image Matching, is discussed briefly as it is the ultimate goal of the larger system. However, this project is focused on the harder problem represented by those first four steps.

3.1 Element Extraction

Before we can do any fancy evaluation of the stars in an image, we must first isolate them from each other and from the background noise in the source image.

Figure 3.1 shows a typical input image, in this case of the constellation Cassiopeia. Note the overall haziness

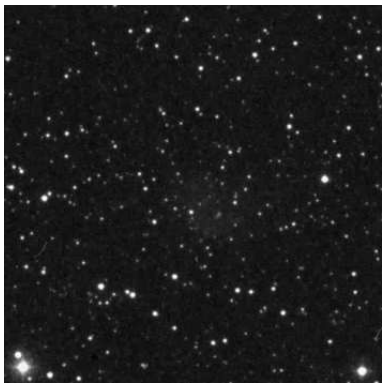

Figure 3.1: Sample Input Image

of the image. This is typical of astronomical images. The haze represents a great number of very faint artifacts as well as some amount of background noise introduced by the optical and mechanical processes used to capture the image[1].

In order to ensure that we are comparing apples to apples, I decided that input images would undergo a normalization process. The purpose of this project is to identify correspondence between images. Consequently, it is not necessary to retain **all** of the information in the image to do so. It will be sufficient to keep the brighter and more well defined elements of the image that, presumably, have a higher degree of fidelity than do the elements that are skirting around the lower threshold of the capture system's sensitivity and are also more severely compromised by the presence of background noise.

Furthermore, by reducing the background noise, patterns between the brighter elements of the scene will be more readily accessible, as there will be less likelihood of including those lesser elements as part of the pattern.

To that end, all input images are first run through a normalization process manually. The darkest 50% of the data is thrown out and the remaining, brighter components are then renormalized from 0 to 255[2].

Notice in Figure 3.2 that the scene has been visually simplified. In addition to reducing the combinatorial complexity of the potential patterns in the image, this processing has also created a consistent background for the image which will come in handy when I subsequently decide to implement my own element extraction code. But before I reached that decision, I surveyed some of the available tools that I hoped might be able to handle the element extraction phase for me.

[1] Even digitally captured images have some amount of noise introduced by issues such as thermal leakage.
[2] The filter used was *pgmnorm -bvalue 128 -value 255*

Figure 3.2: Normalized Input Image

3.1.1 SExtractor

SExtractor is a ubiquitous tool in the industry, used primarily to classify hazy elements from images into various classes of nebulae, galaxies and fuzzy stars.

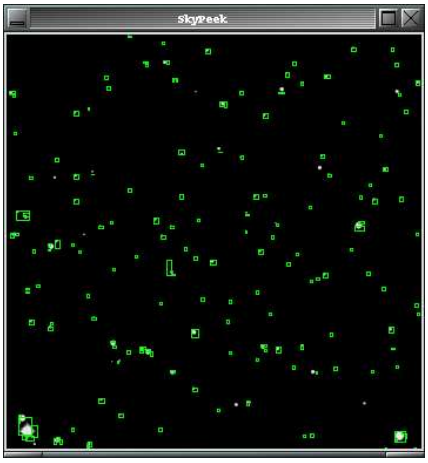

Figure 3.3: SExtractor Element Extraction

When applied to an image containing a relatively sparse, point-like field of stars, SExtractor performs weakly. Figure 3.3 shows the result of sampling the example image of the constellation Cassiopeia using SExtracto's *starfield* configuration. Notice the relative frequency with which one feature box will overlap or even entirely surround another feature box. Note also the number of stars that appear to have no feature box at all and another class which appear to have degenerate feature boxes of zero width or height.

3.1.2 DAOPHOT

Another freely available system that looked promising was DAOPHOT, which stands for Dominion Astrophysical Observatory Photometry software. The primary purpose of DAOPHOT is to extract and classify stars from densely populated star field images, such as those found in globular clusters and galaxy cores.

Unfortunately, discussion of the tools online suggests that it performs rather poorly for sparsely populated images. So when I was unable to get the software to compile in my environment, I did not pursue it any further.

3.1.3 Flood Search

The quest to find an existing solution for extracting star data from images proved ultimately fruitless. After exploring these packages for several days it became apparent that developing my own algorithm would be faster and more customizable.

An examination of the normalized images in the test suite suggested that a simple approach might work. I devised what I call the *flood search* algorithm to tackle the job. Flood search scans the image in left-to-right, scanline order, looking for pixels that are of greater intensity than the background[3]. Whenever a non-background pixel is encountered, a modified flood fill search is employed to ascertain the region of neighboring non-background pixels connected to that first pixel. I was able to simplify the search even further because I don't need an exhaustive list of pixels, just the extents of the bounding box of the non-background region.

Of course, when bright stars are near each other in an image, there is a chance that their non-background halos will overlap, causing them to be extracted as a single region. This is why I chose such an aggressive normalization filter – dropping fully half of the image's content in the hope that this would ensure a clear separation of halos for all but the brightest star pairs. In the few cases where they are still non-distinct, I have assumed that this will always be the case and so they will always be extracted as a single region. As long as the algorithm is consistent in this way, it should introduce no errors in the subsequent pattern matching.

In addition to computing the bounding box, it was also possible to keep track of the total pixel energy contained within the box[4] as well as the location of the brightest pixel in the region and its value.

After completing the isolation of a given non-background region, which I refer to as a SkyBox, control passes

[3] As a result of the normalization process, the background will be exactly zero.
[4] Computed as the sum of the pixel gray values

back to the scan-line traversal loop, but this traversal is smart enough to never re-examine a pixel that has already been included in an existing region.

It is important to note that it would be incorrect to simply preclude all pixels that are contained within an existing SkyBox bounding box. It is possible for disjoint, approximately circular regions to have overlapping bounding boxes. By eliminating all pixels in a bounding box from consideration, rather than just those already explicitly found in a previous floodsearch, it would become possible to miss a smaller element entirely.

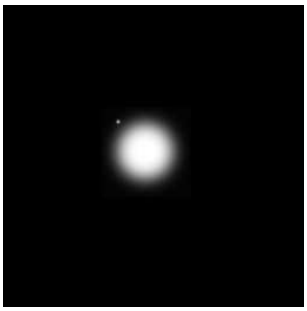

Figure 3.4: Overlapping Bounding Regions

Consider the case illustrated in Figure 3.4. Clearly the bounding box of the larger star encompasses that of the smaller companion, even though the two are completely disjoint elements.

The result of this floodsearch extraction process is a list of SkyBox records. During the extraction process, we accumulate a number of quantitative features about the region's content which presumably pertain to the star contained within its bounds.

Listing 3.1: SkyBox structure

```
typedef struct region
{
        int xmin;
        int ymin;
        int xmax;
        int ymax;
        double magnitude;
        int littleBrother;
        double xcentroid;
        double ycentroid;
        double xpeak;
        double ypeak;
} SkyBox;
```

In this structure (a small number of bookkeeping elements have been removed from this listing in the interest of clarity), one can see that we are tracking the bounding box (expressed in image pixel coordinates), the magnitude (expressed as a floating point value that I will discuss presently), the centroid of the star (expressed as a pixel coordinate with subpixel accuracy) and the location of the peak, or brightest pixel in the region. The only other element is the id of the *little brother* star, which will also be discussed later.

Although simplistic, this extraction algorithm produces surprisingly satisfying results, as can be seen in Figure 3.5. Clearly this is a stronger set of candidate elements from which to construct feature patterns[5] than are those produced by the SExtractor tool.

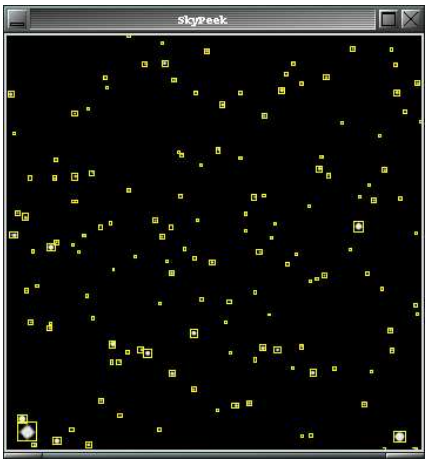

Figure 3.5: Floodsearch Element Extraction

3.1.4 Analysis: Compute Time

The decision to drop 50% of the image detail at the start of the extraction process was somewhat arbitrary. The more stars that remain in the image, the higher the compute times required to proces the image. I arrived at 50% experimentally as an appropriate tradeoff. However, in more star-crowded images, such as the image of Canis Major, even a 50% reduction still leaves an enormous number of stars to process.

I have decided, for the purposes of consistency, to stick with this 50% value. However, it occurs to me that an adaptive detail-culling process might be more appropriate. If images were culled until they achieved some maximum threshold number of stars per pixel, this would perhaps be even more consistent (across different regions of sky) than using some arbitrary, global value.

[5]Clearly this algorithm is not particularly robust over non-stellar images such as gaseous nebulae or galaxies.

3.2 Pattern Construction

Once we have a list of isolated star elements, and we have their coarse attributes such as position and magnitude, we can begin to look for pattern structures that we can use to recognize the same feature in other images.

3.2.1 The Obvious Pattern

Humans have been constructing geometric patterns from neighborhoods of proximate stars since we developed the capacity for symbolic thought. It seems perfectly natural to want to base our pattern recognition system on a similar process.

Figure 3.6 shows five such patterns superimposed on the now-familiar Cassiopeia image. Each image is composed of ten rays extending from a target star to a collection of its nearby neighbors. The only dilemma is in how to select those neighbors in a way that will be invariant over the various types of differences we anticipate between the images provided. I classify these dimensions as differences in: scale, luminance response, rotation and reflection. The algorithm for comparing images in SkyD must be tolerant of these kinds of image discrepancies. To accomodate this, the pattern must be encoded in such a way as to minimize the influence of these types of variation in the patterns obtained.

The remainder of this section is devoted to the details of each of these varying dimensions and the steps taken to make the engine impervious to them.

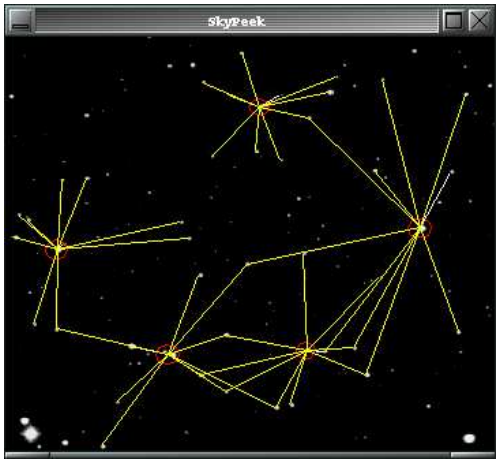

Figure 3.6: Neighbor Patterns for Cassiopeia

3.2.2 Scaling Invariance

The single most commonly expected problem will be issues of scaling. Images are captured at different magnifications, using differently sized telescopes and differently configured capture hardware. Each of these dimensions creates an opportunity for two otherwise similar images to be of completely different pixel dimensions. As a result and without the availability of capture context information, when a pattern is extracted from image A it is impossible to predict how many pixels that same pattern might span in a separately captured image B.

So clearly, for the sake of comparison, our patterns cannot be expressed in terms of pixels. Nor can they be expressed in terms of arc-seconds, since we have no capture context about the image from which to compute such values. The only thing we can do is to express size in some normalized form.

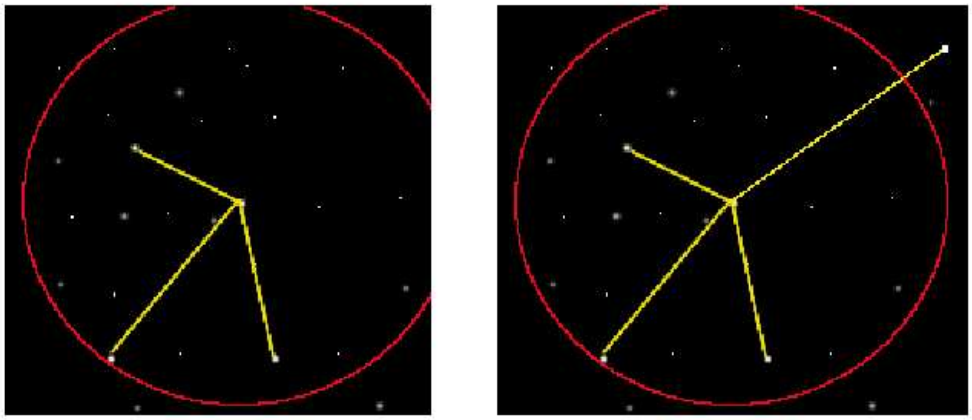

Figure 3.7: Bounding Size Fallacy

The most obvious course would be to express the dimensions of the pattern in values relative to the size of its bounding box or bounding circle. Figure 3.7 demonstrates why this won't work. Consider two images, identical in every aspect except that, the left hand image does not extend quite as far to the right in the night sky as the other image. The second image contains a brightish star in the upper right corner that is beyond the edge of the first image.

When that truncated image is processed, the bounding circle extends as far as the most distant neighbor (seen in the lower left) and establishes the canonical distance by which the relative lengths of the pattern rays will be encoded. However, the bounding circle for the right-hand image would be larger, thus establishing a larger canonical length and resulting in smaller normalized values for the lengths of all the other rays. (Consider that the lower left ray would be encoded with a length of 1.0 in the left-hand case and about 0.85

in the right-hand case.)

Clearly, since the edges of images can occur at arbitrary points in the night sky, basing our canonical normalizing distance on the star furthest from the pattern's center is ill-conceived. Instead, we want to do just the opposite. When considering any given target star, the neighbor star least likely to be missing from other images is the target's **nearest** neighbor.

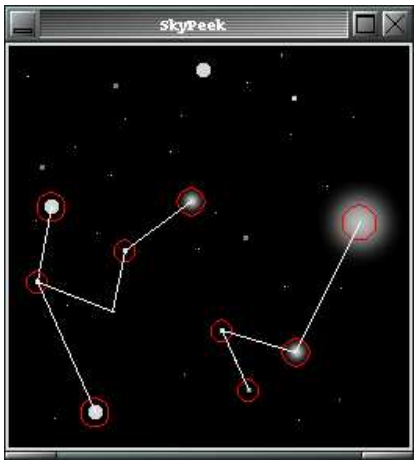

Figure 3.8: Primary Radiant

Figure 3.8 demonstrates an artificial sample image with a number of pattern stars selected. The white rays extending from each circled *pattern-target* represent the nearest neighbor star, used to determine the canonical distance for that pattern. I refer to this star in the code as the target star's *little brother* although there is in fact no implied relationship between the two stars' sizes or magnitudes.

3.2.3 Luminance Response Invariance

The observant reader might ask why the pattern-target stars depicted in Figure 3.8 have neighbor stars that are clearly closer to the center than are the indicated little brothers. Aren't little brothers supposed to be the nearest neighbor?

The answer lies in our approach to luminance response invariance. But first I'll explain what I mean by that term.

Images anticipated in our system are expected to come from a wide variety of sources: CCD captures and scanned photographs being the two most common. But considering the degree of processing that may or may not have been applied to the images before they arrive, and considering also the wide variability in

responsiveness of the various capture systems to differing levels of incoming light, it is impossible to say whether or not we can assume a linear relationship between the brightness of incoming light and the amount of corresponding pixel energy represented in the image.

For example, two stars might have apparent brightnesses (in the sky) that differ by 30%. In a short duration, wide angle image, those two stars might be some of the dimmest stars shown and may register with a difference of 26% in the brightness of their pixels. However, in a more highly magnified and longer exposed image, those same two stars might be more significant components of the image. They might even be the center of interest in that second, narrower field image. Registering now at the brighter end of the luminance response curve, the capture system might be overly sensitive to the brighter of the two stars and may exhibit bias, showing a more extreme 34% difference in recorded pixel values.

Clearly, if such non-linearities are present in the capture system, or in the combination of hardware and software used to transform photographic images into digital form, then our algorithm must express the star patterns in a manner that will be relatively consistent regardless of the capture system used.

The one thing we can rely on is that, over a small enough range of luminance values, the response of the capture system is at least approximately piece-wise linear. Stars with relative differences of 30% are much more likely to be plagued by the problem of nonlinear luminance response than are stars that differ by only 5%. So if we could constrain our pattern engine to working only with stars of the same approximate luminance levels, then we could be reasonably sure of linear behavior regardless of where those energies sat in the response curve of the capture hardware.

So this is exactly what I've done. Rather than choose **all** of the nearby neighbors to construct the pattern, I take only those stars that have a computed magnitude within ±10% of that computed for the pattern-target star.

Magnitude

All this talk of relative luminance response curves is meaningless of course, unless we have a way to compute the energy in the first place. During the extraction phase, we compute a sum for each StarBox region: a sum of all the non-background pixel values in the connected region. We assume (for want of a better value that would require all that pesky context information) that this sum is an adequate approximation of the total energy received by the capture system from the associated star.

Unfortunately, this results in an approximately linear value which does not conform to the human logarithmic sensitivity to light energy. If we were to use these raw energy levels we would find very few neighbor stars

within our ±10% selection window, even though there are quite a few neighbors that appear to the eye to be very similar in brightness.

Since the human eye has a logarithmic response to light, I decided to apply the same quantization process in SkyD. The resulting *magnitude* value is used in all comparison functions and is in fact the log of the total pixel energy computed in the sum. As a result of this effective scale compression, the region of selected pattern neighbors is small enough to be practical.

In addition to ensuring luminance response invariance within a given image, we must also ensure that the computed magnitudes for a given star are consistent across multiple images as well. Again we lack the necessary context information that might allow us to compute an absolute magnitude from the image. So we must find a way to express our magnitudes in a way that is both repeatable and consistent, regardless of the image context.

To do this, we rely once again on the little brother. By encoding the magnitude by its value relative to the computed magnitude of the little brother, I hope that this will behave linearly enough that it will be reproducable.

3.2.4 Rotational Invariance

In addition to images having different sizes and brightnesses, I also expect to find images of different rotational orientations. Consider two images of the Big Dipper: one captured just after sunset and the other captured at midnight. One of these images will appear to be rotated by 90°, relative to the other, because the sky will have rotated with respect to the capture system's local sense of *up*.

Figure 3.9 is simply a copy of Figure 3.6, rotated 30°. Notice that relative to the orientation of the stars within the image, the star patterns are identical in the two images. But with respect to the image borders, the patterns appear different: each separate ray pattern is rotated about its center, with respect to its corresponding pattern in the other image. While these differences are almost trivial for the human eye to ignore, it will take some careful planning to make those differences invisible to our pattern comparison engine.

That engine must be able to match these apparently different patterns if the system is going to be of any value. This implies that the patterns must be expressed in some canonical form that is not dependent upon the image's sense of *up*.

The rays of our patterns are going to be stored as weighted polar coordinates, comsisting of an angle, a

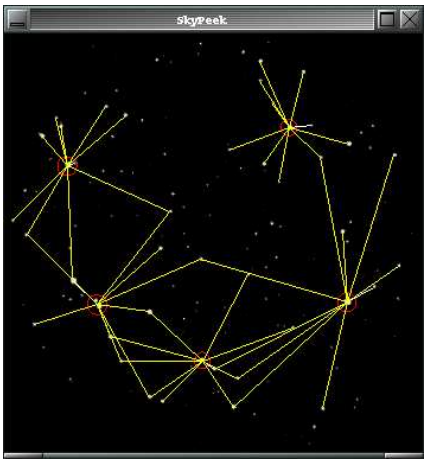

Figure 3.9: Neighbor Patterns for Rotated Cassiopeia

length and a magnitude. We've already discussed the facts that the length and magnitude are normalized. The key question remaining is the question of the prime angle. Where do we get a primary direction from which to measure the angles of all the rays in the pattern? And how do we ensure that this primary direction is intrinsic to the pattern and therefore repeatably constructible regardless of the image orientation?

It turns out we've already computed this. In addition to providing the canonical unit of length, the little brother ray can also be used as the primary ray from which all other ray angles are measured. By doing this, we ensure (for as long as we can consistently choose the correct nearest neighbor to serve as little brother) that the angles associated with rays will also be consistent, regardless of the orientation of the capture system.

There is one caveat to this. It is possible, particularly in very inexpensive capture systems, that a degree of anisotropic bias might exist in which the pixel densities differ between the horizontal and vertical. Should this actually be the case, it will be impossible to detect and images coming from such systems will be unusable. However, since it is the case that angular distances are an important aspect of astronomic images, I suspect that is is exceedingly unlikely that we will encounter such images with any great frequency.

Analysis: Trusting Little Brother

It is worth noting that our entire pattern construction and comparison algorithm is critically dependant on our ability to select the same little brother primary ray every time we inspect any image of a given star. Should the little brother turn out to be a pulsar, or be occluded by a wisp of cloud, the pattern constructed from that image, based on an erroneously selected little brother, will be completely unmatchable.

3.2.5 Reflection Invariance

There is one last aspect in which images of identical regions of space might be expected to differ. Reflection. Depending upon the details of the capture system used, sometimes the images will be mirror inverses of the actual sky. It would be preferable if our pattern matching system were able to successfully match such mirror-twin images.

My first implementation of SkyD encoded the ray angles as simply the cosine of the actual angle. (A value easy to achieve from the cross product of the two ray vectors.) This neatly encoded the angle in a left-right-invariant form (since a negative cosine curve is identical to a positive cosine curve). By comparing the rays via their cosines, we were implicitly comparing in a reflection-invariant manner.

Unfortunately, this meant that for each ray record there were in fact two legitimate rays that could have produced it. By comparing the rays between target and candidate patterns in this manner, I was actually accepting any one of the 2^n patterns that can generate the same signature. As the number of rays included in the pattern sets was increased, the accuracy of the system plunged exponentially.

Fortunately the mistake was identified and I was able to re-implement that section of code. In the new implementation, the actual angle, not the cosine, is encoded. The target and candidate patterns are compared twice at the pattern level, but not twice for each ray comparison, as was implicitly being done in the cosine technique. Instead, one complete pass of the candidate is made with angles exactly as they are recorded. Then a second pass of the candidate is made but with all its candidate angles negated, effectively swapping a clockwise angular sweep for a counter-clockwise sweep. In this way, there are only two possible patterns that could produce a match, rather than 2^n.

While this technique appears to work and provide the intended reflection invariance, it should be noted that the sample images being used for testing were all of the same non-reflected class. Since the reflection invariance doubled the compute and debugging times, I have turned it off to facilitate faster execution and the use of test images of sufficient resolution to provide interesting results.

3.3 Image Registration

In order to be able to search input images for matches against previously registered images, we need to construct a database to record them. Since a practical implementation of this project would ultimately require a robust data store, I decided to implement a MySQL database to handle storage.

The database design is fairly simple, involving just three related tables:

Images: The image table contains just two fields: a unique id and the URL of the file containing the image. A URL was chosen so that the database does not have to handle the actual storage of the image data. This would facilitate construction of an enormous searchable engine without the associated image storage costs. In a practical implementation, a regular confirmation engine would be run to confirm the continued availability of the remotely stored images. Searches of the database could then return results that only included currently accessible images.

Stars: The star table has a unique id for each star seen and a reference to the imageid for the image from which the star was encoded. It should be noted that there is currently no attempt made to re-use these starid values across multiple images containing the same star. However, a map table could be added that linked a given star to all other occurrences of the same star in other images. This would be an interesting way to optimize the query process. In addition to

Components: The final table contains all the rays from all the star patterns. Each record contains a unique ray id, a reference to the starid for the star from which the ray pattern was extracted and the three component values of the ray: angle, length and magnitude.

3.3.1 Analysis: The Scope of the Beast

The pattern matching process involves making SQL calls to the database to extract pattern records for comparison. These calls, over Internet channels, represent a significant bottleneck for the algorithm even though it provides a highly robust storage mechanism.

One approach I've considered is applying a fingerprinting technique that can be applied via SQL on the server to prefilter the patterns that get returned to the querying client.

The simplest idea I've considered is simply creating a bitmap to represent the presence or absence of rays in each of 16 angular sectors. If such a mask is constructed from the target pattern and included in the SQL call, patterns whose ray distribution strongly conflict with the target fingerprint need not be returned.

I have not, however, attempted to implement this algorithm and there are certain to be subtleties involved that I have not anticipated.

3.4 Pattern Matching

The SkyD application has two primary modes of operation: star query mode and image query mode.

In star query mode, an image is displayed in which the user can click on stars of interest. The application attempts to find the closest match for the selected star from the database and opens the corresponding image, highlighting the matched star.

The purpose of this project was to develop and refine the star query mode. While a sketch implementation of the image-query mode has been undertaken, it will not be discussed at any great length.

3.4.1 Comparing Patterns

Scoring the comparison of two patterns is an immensely variable process. There are dozens of ways to approach the problem, but most of them boil down to a series of pairwise ray comparisons and some mechanism for accumulating these ray-pair scores into an overall match score.

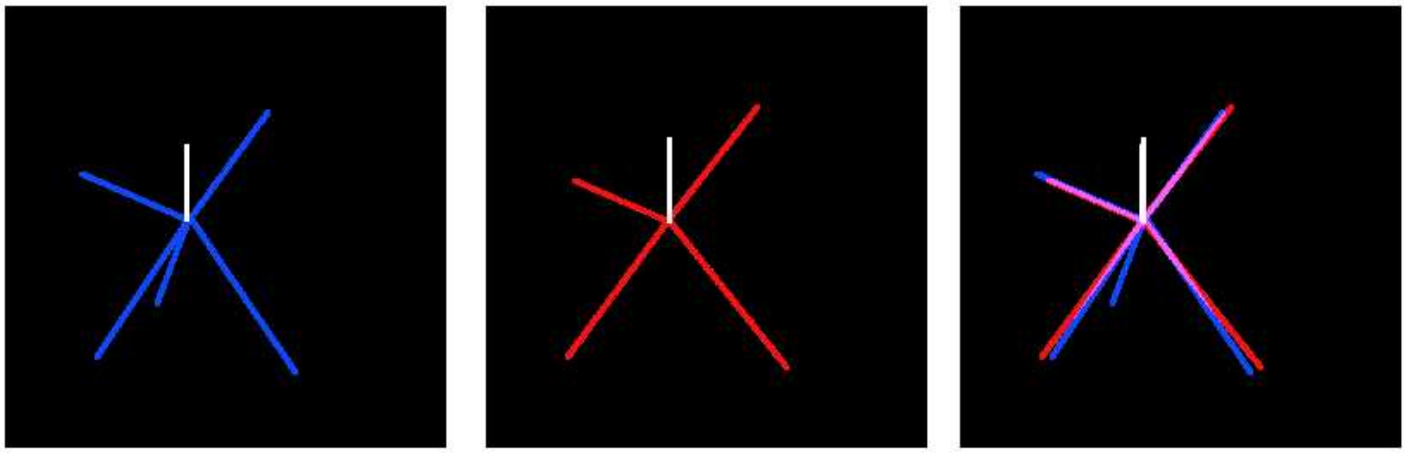

Figure 3.10: Comparing Patterns

Figure 3.10 shows the process. The red pattern represents the target for which we seek a match. The blue pattern represents a candidate from the database that we are going to score.

In both images, the white ray is the little brother. The two patterns probably don't look exactly like this in the images from which they're drawn. These patterns are being displayed in their canonical size and orientation, after having filtered out the image-specific orientation and dimensional information, to make the comparison process more visually understandable.

The comparison process marches around the target pattern and for each ray, compares it to all of the rays in the candidate pattern. Each ray-ray comparison yields a score in the range of $0 \cdots 1$. The highest score from comparing to each candidate ray is taken as the match score for the current ray in our sweep around the target pattern.

As we determine the best match score for each target ray, we incorporate it into the overall pattern score, by simply multiplying the new score into the accumulation buffer. This provides an extremely aggressive penalty for less than perfect matches and may in fact be **too** aggressive.

The ray-to-ray match score is computed as follows. Each ray is in fact a three-dimensional vector comprised of an angle, a length and a magnitude. The measure of correspondence between two vectors is their mutual dot product, divided by the product of their lengths.

In the example shown in Figure 3.10, the correspondence between the two patterns is quite good, except that the red pattern is missing one ray. This might indicate the presence of an anomaly in the image from which the blue pattern was taken – perhaps an asteroid – located just below and to the left of the target star. Of course, it might also just be a coincidentally similar pattern belonging to a completely different star. It is impossible to be sure which, and so we rely on the hope that no two star images will have strongly corresponding patterns unless they are images of the same star.

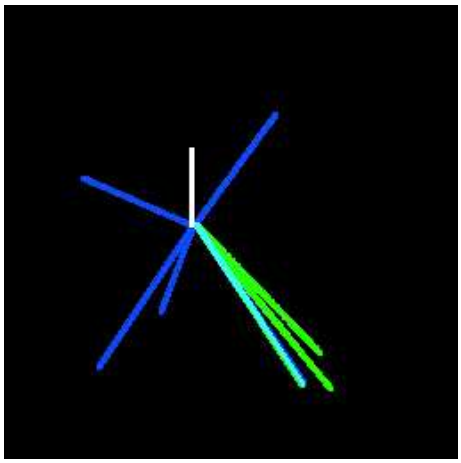

Figure 3.11: Pattern Comparison Flaw

Analysis: The Promiscuous Ray Problem

This algorithm, as related, has a flaw. Consider the case shown in Figure 3.11. The green pattern is the target pattern and is being compared to the candidate blue pattern. By chance, the green pattern has only

three rays, all of which are clustered around a single ray of the candidate. Clearly this should be a horrible match. Yet the score for each of the rays as we sweep around the green pattern is high. The result is an artificially high score for what should be a dismal failure.

The problem is that one single ray in the candidate is being allowed to match against multiple rays from the target. Each of the matched rays believes it is the best fit, unaware of the other competing matches, so it does not go off to find another ray for which it alone is the best match.

To correct this, we have to make two sweeps through the data. In the first sweep, we will compute a score for each target ray compared combinatorially to each of the candidate rays. Then in the second pass, we select the highest of all of the pairwise matches. The score for that highest match is incorporated into the overall pattern match score and then the target and candidate rays that contributed that score are eliminated from further consideration. We then proceed to the next highest ray-ray score and repeat, continuing this elimination process until all of the target rays have been matched. The accumulated pattern match score is then returned as the score for matching this particular target and candidate.

This altered comparison process ensures that a ray can only match to a single partner and that any other close matches for a ray will be forced to match to a much less satisfying partner, thus reducing the score. This appears to work very well in practice, but it should be noted that the best-to-worst score allocation scheme used here does not necessarily ensure the highest overall score.

To improve the matching algorithm further, this problem should be treated as an optimization problem and the standard solutions should be easily applicable.

3.4.2 Cleaving and Marginal Rejection

Consider the case illustrated in Figure 3.12. The six yellow lines (both bright and dim) depict the rays that were used to construct the pattern for the central star when this entire image was registered in the database. However, at query time, a different image is used. The query image only contains the portion of the sky to the right of the blue line.

Since the evaluation of the target image does not contain those three companions on the left side, the dim yellow rays will not be constructed. Instead, the three red rays are selected, as they represent the next three closest neighbors of suitable magnitude[6]. I refer to this condition as *cleaving* of the pattern.

If we allow the search algorithm to search for the cleaved pattern, it is unlikely that the original pattern will

[6]This example assumes that we are working on a system in which only six rays are computed for each pattern.

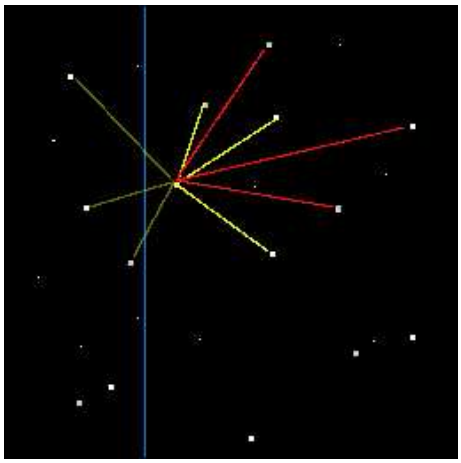

Figure 3.12: Pattern Cleaving

yield a sufficiently high match, even though three of the rays will be match perfectly.

Given this impending failure, it would be better if we didn't attempt to match the cleaved pattern at all. Fortunately, this is an easy condition to test. Note that some of the rays in the erroneous pattern are further away (radially) from the target star than is the edge of the image (represented by the blue line).

Using this as a guideline, a target star's pattern will not be processed if any of its component rays is longer than the distance to the nearest image edge. In this way we ensure that we have the greatest possible chance of finding the right match when we go to the expense of matching a pattern. I call this the rule of *marginal rejection* since it rejects from consideration any patterns near the margins of the image.

Analysis: Shrimpy Little Brother

One problem I observed during performance assessment was the issue of extremely short little brother rays. When the little brother is less than 5% of the length of some of the other rays in the pattern, we get scaling-induced inaccuracies. Since the lengths of rays are expressed relative to the length of the little brother, minor discrepancies in the accuracy of the little brother length translate into larger inaccuracies in the length of the pattern rays. This increases the likelihood of making match mistakes.

To counter this, I've entertained the idea of rejecting patterns for comparison if they have significantly undersized little brothers.

Chapter 4

Results

4.1 Project Code

The code was implemented in ANSI C, using the OpenGL and MySQL libraries. Both the GUI client and the MySQL database server are running on Linux Redhat 9.

The project was managed using CVS and at final count, comprised 3010 lines of code.

4.2 Early Tests

During early trials of the SkyD system, success rates of approximately 70% were common. However, these results were based on a training set of three images and a test set composed of manually altered versions of the training data. (The test images were rotated, scaled and cropped with some additional noise added by hand.)

When the training set was expanded to seven images, and the test set was replaced with independently captured images of sky regions similar to those in the training set, the individual pattern matching success dropped to approximately 30%. But these results were still based on the comparison engine that permitted promiscuous rays.

Fortunately, satisfactory results were regained once the comparison engine was redesigned.

4.3 Test Data

The point of this algorithm is to be able to find the queried star from among a number of stars from a number of images. It is often difficult to be sure, when looking at these images, if they overlap in any area. This makes it difficult to judge whether an unexpected result from a target star query might in fact have matched the correct star appearing in a different image. To avoid that issue, we have selected a training set of images specifically so that they do not overlap. The constellations chosen were: Orion, The Corona, Cassiopeia, Aquarius, Hercules, The Phoenix, Pavo, Carina, Columba and Perseus.

(and Canis Major and ???)

The training images were all taken from the Deep Sky Survey archive. Of the different catalogs available for downloading, all the images were taken from the DSS1 catalog, in GIF format at a height and width of 60 arcminutes. These images were all 2119x2119 pixels. This proved to be too large for the algorithm to run fast enough on my development machine, so I downsampled all the training images by 50%.

Images in DB	12
Stars in DB	29147
Min. Stars (Corona)	83 of 212
Max. Stars (Canis Major)	4790 of 5562

Figure 4.1: Test Data Details

The test images were all taken from the same site, but from an archive captured separately: the HST-Phase 2 survey. These images were downloaded with a width of 45 arcminutes in GIF format and then preprocessed in the same way the training images were managed.

It is interesting to note that, as shown in Figure 4.3, the Pictor image was the most successful of the test images. This is particularly curious because the Pictor image was the only one modified from the HST archival version downloaded. It was rotated by 41 degrees prior to testing, as a way of testing rotational invariance.

Figure 4.2 shows the distribution of these images in a map of the night sky. Each image is 60 arcminutes (1 degree) square.

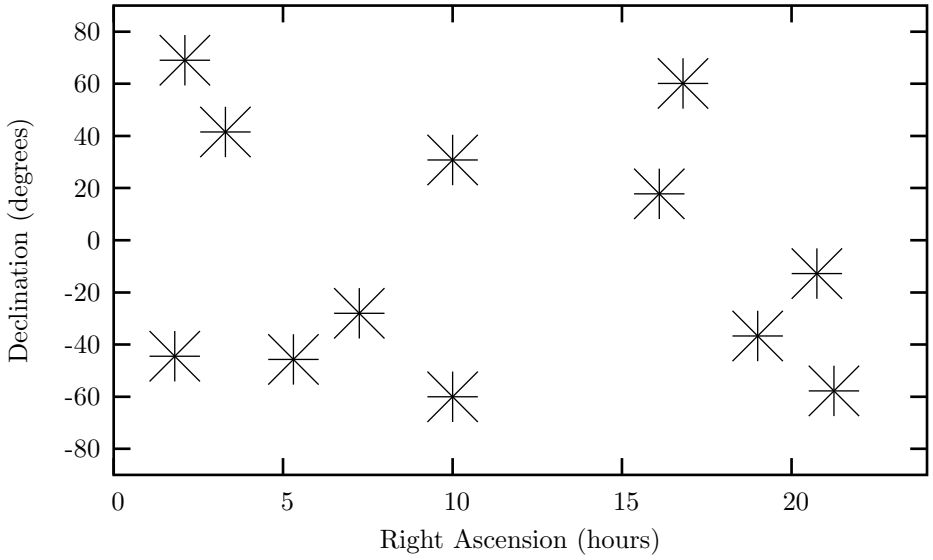

Figure 4.2: Placement of Training Images In the Night Sky

4.4 Verifying Success

Given the nature of the project, it is not possible to ensure analytically that the results of a query indeed match the target star. Instead, we must rely on human judgement for this determination.

With each of the test images reported in Figure **??**, I loaded the image and clicked on as many different stars as seemed practical, testing a minimum of ten stars in each. Judging the successful correspondence of target and result was done by eye.

Test Image	Trials	Successes	Score%
Aquarius	11	9	82%
Pictor	13	12	92%
Phoenix	12	10	83%
Draco	11	7	64%
Total	47	38	81%

Figure 4.3: Trial Successes

I have yet to devise a reliable algorithm for automating this analysis. It occurs to me that I could process all the stars in a test image and count the number of results that come from clearly wrong images. Assuming that the number of mis-matches occuring within the target image itself is approximately equal to the average number of mis-matches occuring in other images, we could simply count the number of results that were in the correct image and adjust for this mismatch rate to get an approximate number of successful matches,

computed over a much larger number of stars.

This approach strikes me as equally faulty and would require a larger number of test images to ensure that image-specific variations had been averaged out. For this reason (and also because I'm out of time), I have elected to stick with the manual judgement process.

4.5 Analysis: Flaws in the Ointment

4.5.1 Confirmation Confusion

A more robust test would involve overlapping images in the training data. In a perfect world, SkyD should be able to identify **all** of the images that contain the target star, not just the best match. And it should be able to do so while also rejecting superficially similar stars.

The biggest problem with this is not the algorithm, but the test procedures. It is extremely difficult to register two different images of the night sky when they contain any more than superficial details. This makes it nearly impossible to determine whether a test result is wrong, or merely a different image of the target star found in an unexpected image.

To this end, add combinatorial leverage.

The original star images chosen for this experiment were not selected to represent disparate regions of the sky. So it was likely that two mutually proximate target stars from a single image might yield correspondences with stars from two different images. In fact the local region being queried might be represented in several training images.

It is often difficult to determine from visual inspection whether two different images contain overlapping regions of the sky, especially when those two images have different resolutions and orientations. So it became very difficult to judge whether a particular query result was correct. It was very difficult to register the two images with each other visually.

Instead of continuing with this uncertainty, I restructured the training data to ensure that no two images contained the same regions of sky. Furthermore, the test images were chosen from identical regions of sky covered by some of the training images (although captured separately). This way, if a test star in the Canis Major constellation produced a result from the Phoenix asterism, we could be certain it was a mistake. The determination of correctness between the target and result can now be done visually. The two images involved are of similar extent and orientation (although not necessarily similar resolution or sensitivities)

which makes the process of visually registering the two images much simpler.

To visualize the distribution of the new training data, I've constructed the map of the night sky shown in Figure 4.2. In it, one can see the centers of each of the training images. Each image is one degree in width and hight, so it should be clear from the diagram that the images cannot contain overlapping data.

Chapter 5

Image Matching

Although matching images to images is the long-range goal, it is beyond the stated scope of this project. However, there are a few points worth mentioning about possible implementations at that level.

Finding uniquely matching patterns from an arbitrary input image against a database of perhaps thousands of images

5.1 Adaptive Source Weighting

As we accumulate information from successive queries on a single image, we should be able to make use of this growing body of evidence to weight the likelihoods of corresponding images. If we have 3 stars in a close grouping who all returned the same query result image, chances are high that a subsequent query taken from within the bounding hull of those previous 3 images will also yield the same corresponding image.

5.2 Iterated Pruning

When the database gets large, searching each pattern for a 10-ray match can be quite time consuming. Fortunately we can take advantage of a scalability aspect of the pattern system. For an initial pass, it should be possible to simply compare a subset of the target pattern, searching for correspondence with the rays defined by the closest two neighbors.

This will yield a large set of candidates, but a significantly small subset of the entire database. This candidate

list could then be searched with the more exhaustive 10-ray comparison.

5.3 Aggregation Leverage

As we will see in the image matching section to follow, we can use a number of aggregation techniques to milk significantly better results in matching images to images, despite the weaker performance of the pattern-level matching.

Chapter 6

Conclusions and Future Work

A promising approach to sky image recognition. Potential application to spacecraft orientation systems? Potential application to color palette matching.

6.1 Combinatorial Leverage

If we can achieve 30% success by comparing individual stars, we should be able to borrow from combinatorial logic to significantly increase the success rate by comparing star neighborhoods.

Bibliography

[1] K. Abazajian, et a. . The first data release of the sloan digital sky survey. *The Astronomical Journal*, 126:2081–2086, October 2003.

[2] B. Aschenbach. Observational capabilities of EXOSAT and ROSAT. In *Accreting Neutron Stars*, pages 309–313, October 1982.

[3] Paul Fieseler. The galileo star scanner as an instrument for measuring energetic electrons in the jovian environment. Master's thesis, University of Southern California, December 2000.

[4] Laurence Boxer. Point set pattern matching in 3-d. *Pattern Recogn. Lett.*, 17:1293–1297, 1996.