Visual Pattern Matching Using Resolution Pyramids

April 11, 1997

Abstract

I have created a new algoithm for visual pattern matching which can go faster that previous methods and largely retain accuracy. It works by using a small version of the image to guide its eventual finding the answer in the full image. The image is reduced using a resolution pyramid which uses the whole original image creating hi-fidelity miniatures. Because the computational cost of search falls off exponentially the miniature image can be searched very quickly and give a general vicinity that the original image should be searched in. All this has been implemented, tested, and works quite well.

Contents

Chapter 1

Background

This project was spawned from the frustration with the status quo and the inspiration from a demonstration of what was possible.

The frustration arose from reading several papers while researching for this project [BRG96, Loa95, BGL95, CC95]. These papers made extensive use of higher mathematics to describe their work when upon reading the text the concepts were really much simpler. Unfortunately some of these papers made no attempt to explain their equations in simpler terms. Another personal gripe was the lack of attention to actual implementation, coming from an engineering background I would have liked to see more about algorithms and 'how to' type information.

The inspiring papers were much more down to earth, and from my perspective much more informative [Wes95, Hor94, FG96]. These papers were also a departure from the traditional paradigms of how to do computer vision. One achieved useful vision running on a 2 MHz 8 bit microprocessor with less than 64 kilobytes of RAM [FG96]. Another made a completely autonomous robot that navigated by sight in real time [Hor94]. My personal favorite which is somewhat the basis for this project tracked objects at full video rate of 30 frames per second [Wes95].

1.1 Resolution Pyramid

The resolution pyramid is a well known data structure for reducing the amount of image data to be processed [BB82]. In a resolution pyramid successive levels are smaller versions of the image with fewer pixels to compute.

There are three basic ways of calculating a resolution pyramid. Subsampling is the fastest but has the least fidelity. If an image were to be subsampled to one quarter of original size, every other pixel of every other row would be copied to adjacent positions in the new image. Because of the number of pixels which are ignored subsampling can cause aliasing leading to harsh edges where previously a soft gradient existed.

There exist complex methods by which pyramids of any multiplication step can be achieved, not just easy rational fractions ($\frac{1}{2}, \frac{1}{4}, \cdots$). These employ a variation on subsampling with some blurring between each step to reduce the aliasing effect of plain subsampling. There have been studies of how much blurring to use to strike a balance between preserving information and reducing aliasing [CC95].

The third method of neighbor averaging is somewhere between the other two. Neighbor averaging works best when the image is to be reduced by an integer which divides evenly into the number of pixels in the height and width of the image. To make a new image one quarter size of the original, the original is grouped into squares of four pixels which are averaged into one pixel in the new image (See figures 1.1 and 2.1). Neighbor averaging is the pyramid method used in this project.

1.2 Key Matching

Key matching is a simple comparison, pixel by pixel, of a key and an equally sized area of an image. A key is a small image expected to be found in a larger image. For example, a key image of the ace of spades might be searched for in a picture of a card table.

Key matching has some restrictions due to it's simplicity. There are no simple methods of dealing with rotation or scaling of the image. If a subject to be matched rotated excessively relative to the camera key matching will fail unless a library of all possible rotations of a key is checked against each image searched. A similar situation arises if an object changes it size in the view of the camera (changes distance to camera). Neither of these things make it a bad method, it simply must be applied to the right application.

The focus of this paper is key matching's shortcomings due to the horrendous rate at which the number of comparisons done grows in relation to the size of the key and area to be searched. This

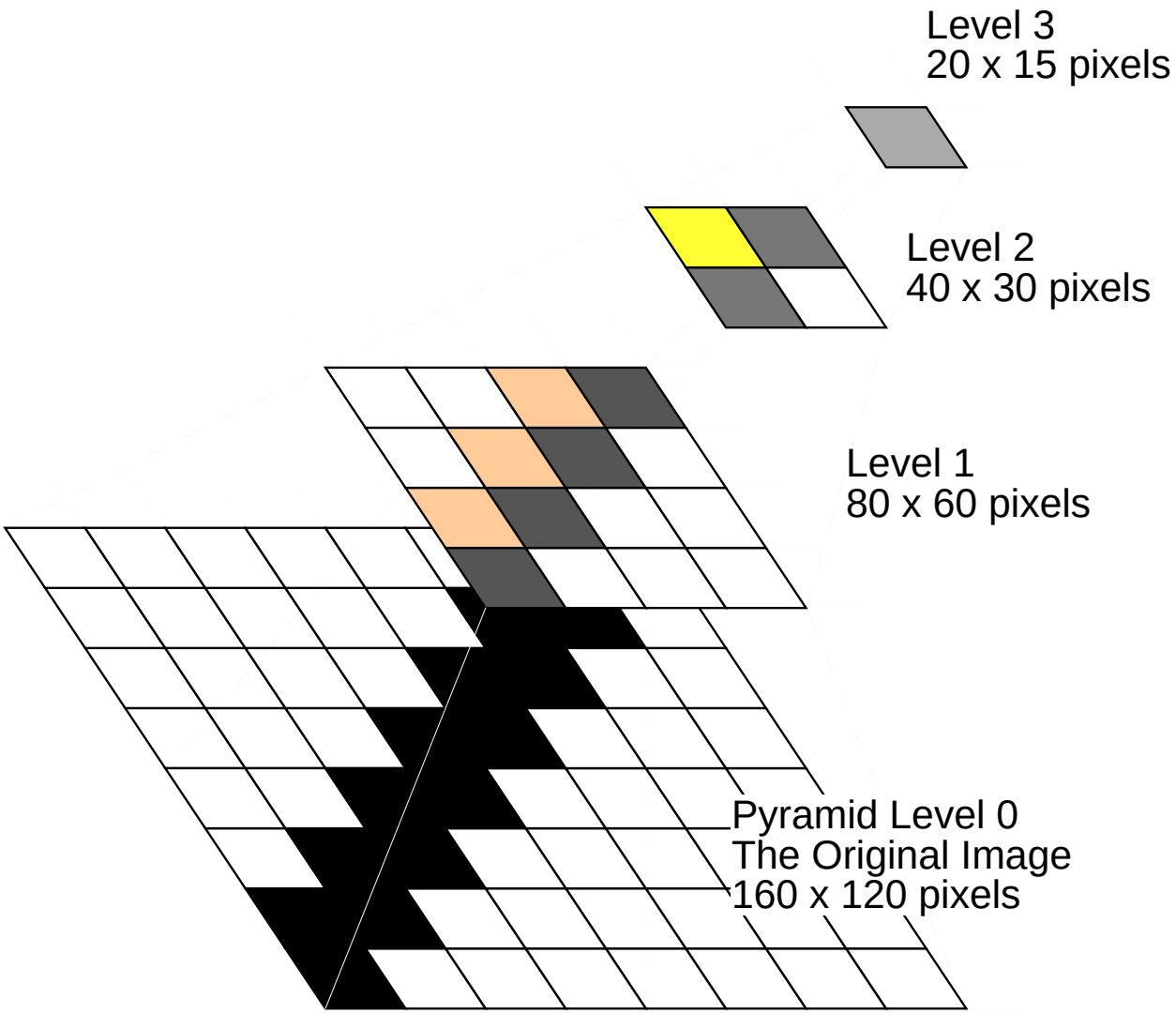


Figure 1.1: A section of an image going through resolution pyramid calculations.

is shown in equations 1.1 and 1.2 where x_i, y_i is the image size and x_k, y_k is the key size.

$$keycompares = (x_i - x_k + 1)(y_i - y_k + 1) \tag{1.1}$$

$$pixelcompares = (x_i - x_k + 1)(y_i - y_k + 1)(x_k y_k) \tag{1.2}$$

To simplify, we assume both image and key are square with sizes of i^2 and k^2 respectively. The equations become:

$$keycompares = i^2 + 2i - 2ki - 2k + k^2 \tag{1.3}$$

$$pixelcompares = (i^2 + 2i - 2ki - 2k + k^2)(k^2) \tag{1.4}$$

For example: a standard TV signal of 640 by 480 pixels searched by a key of 72 pixels (one inch) square would have 232,721 key compares and 1,206,425,664 pixel compares. This project attempts only a 160 by 120 pixel image and 32 pixel square key ($keycompares = 11,481, pixelcompares = 11,756,544$).

As recently as two years ago specialized dedicated equipment costing $2000 or a workstation of $10000 was needed for enough power to achieve 30 frames per second on a small image[Wes95][1].

[1] This project was carried out on a three year old $3000 computer.

Chapter 2

This Project

This project explores an area of computer vision that has been impractical until recently due to the speed of computers. The principles are very simple but a large amount of those simple pieces is required. In this decade computers have advanced to the point that the necessary equipment is easily accessible.

2.1 Pyramid and Match

The match function takes to areas of image of equal size and compares them. The difference between pairs of corresponding pixels is squared and all the squared differences are summed. The sum is divided by the number of pixels compared to give a final match score in the range of zero to one. A lower score is a closer match.

$$\left(\sum_{x=0,y=0}^{keysize} (key_{xy} - image_{xy})^2 \right) \div pixels \tag{2.1}$$

The resolution pyramids in this project are of the third type described in section 1.1. Pyramids are calculated to predefined depths with each new image having one quarter of the pixels of the old.

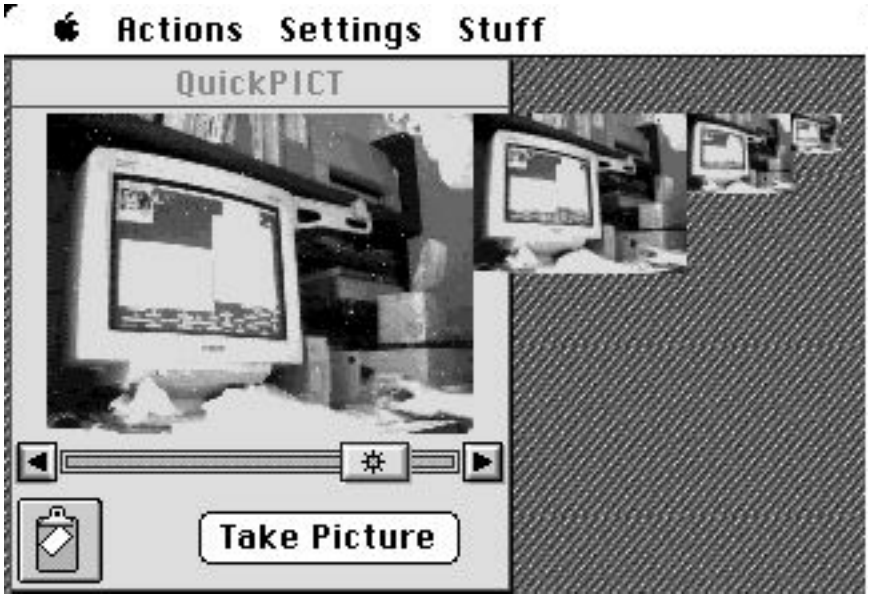


Figure 2.1: Screen shot of a resolution pyramid of the main image.

2.2 Exhaustive Search

An exhaustive search checks every possible variation of key position against a source image. Because of its' completeness, an exhaustive search will always find the best possible solution. Also because of its completeness, it will take the longest possible time to analyze an image.

2.3 Pyramid Search

The pyramid search, which is the focus of this project, is my solution to going faster than exhaustive searches. The search begins by reducing both the image and the key using resolution pyramids. An exhaustive search is performed at the bottom level of the pyramid. To regain accuracy the coordinates of the match are translated to higher levels of the pyramid where a small region is compared against a key of the same pyramid level. Expansion continues back to the start of the pyramid to find the best possible solution. In the case that the loss of detail caused a false optimum at a small stage of the pyramid, it is possible to follow several leads backwards through the expansion. Pyramid searches have Depth and Breadth. Depth refers to levels in the pyramid. In this paper "Deeper" and "Lower" levels of a pyramid refer to levels that have been further processed. Thus level 1 has 80 by 60 pixels and is deeper than level 0 which has 160 by 120 pixels.

Breadth refers to the number of best options that are expanded upon while backtracking up the pyramid. During the exhaustive search at the bottom of the pyramid the best n coordinates are kept in a sorted list. Those are then translated to the next higher level in the pyramid and their immediate vicinity (± 2 pixels) are searched, again keeping the top n between all that is searched in that level. A search with depth and breadth of 0 is considered to be the exhaustive search.

2.4 Testing

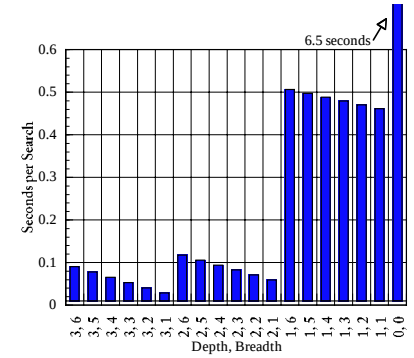
The test process was to capture an image from the camera, randomly chose 100 keys from that image, recapture the scene from the camera, and search for the keys. The second capturing of the same scene introduced camera noise to make sure that the searches would be able to handle it.

My tests were run on twenty two images, 19 synthetic and 3 real-world. I created five sets of test pattern for my camera to look at. One set was a checker board pattern of white and black rectangles; this was tested at three sizes, each had rectangles half the size of the previous. The second set was tesselated black and white isolateral triangles, tested at four sizes. The third and fourth sets were tested at five sizes. The third was a single white square in a black back ground, the fourth was a white triangle and circle on a black background. The last test patterns I created were black to white gradations.

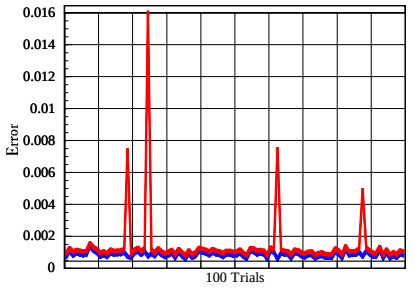
The reasoning behind these patterns was to test the searches under very controlled conditions of feature size, clutter, and sharpness. In addition to the synthesised images I ran the tests on three scenes around the computer: a bookcase (cluttered), the computer area (harder edges, semi cluttered), and a bed (softer edges).

2.5 Results

The following results are from 22 images tested over 19 search variations with 100 keys in each test for a grand total of over 40,000 searches.



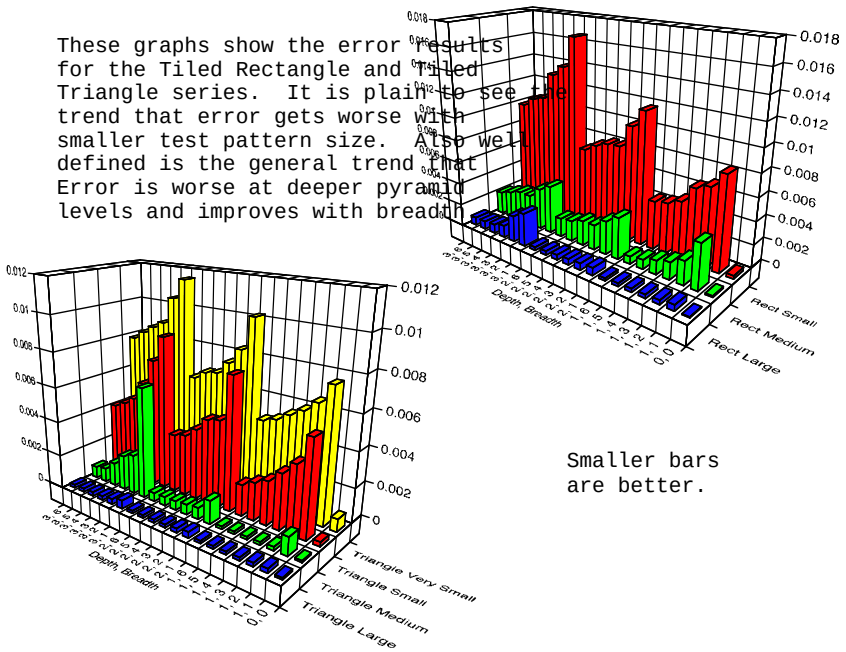
This graph shows the time it takes execute a single search of the var configurations of Depth and Breadt These values closely follow predic relative execution times based on number of operations performed.



Standard Deviation of Error can be measure of reliability because of this graph. The lines are the err individual key tested in a set of exhaustive search (blue bottom li Pyramid Search at Depth 2 Breadth line) ran on that set of keys. On did the Pyramid Search not get the

Figure 2.2:

These graphs show the error results for the Tiled Rectangle and Tiled Triangle series. It is plain to see the trend that error gets worse with smaller test pattern size. Also well defined is the general trend that Error is worse at deeper pyramid levels and improves with breadth.



Smaller bars are better.

Figure 2.3:

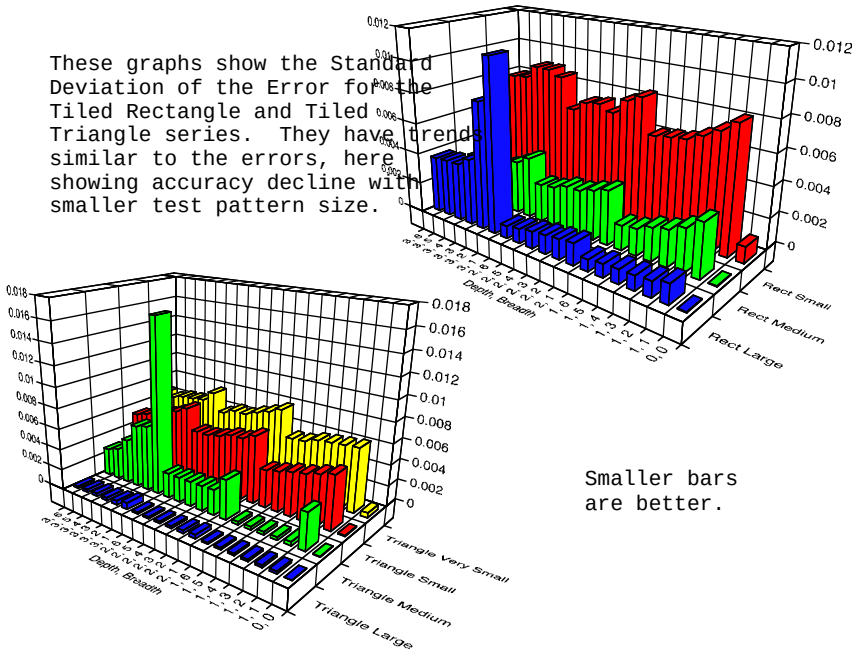


Figure 2.4:

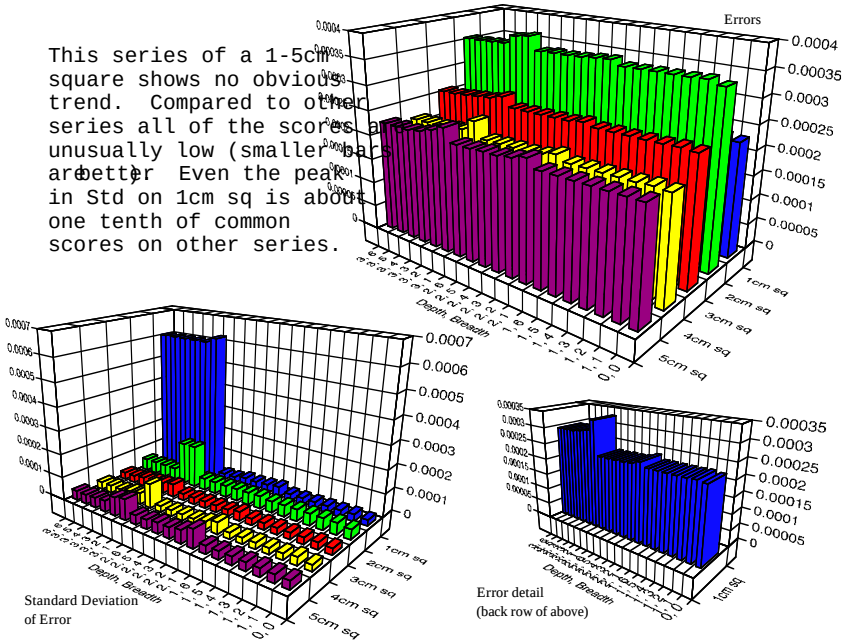


Figure 2.5:

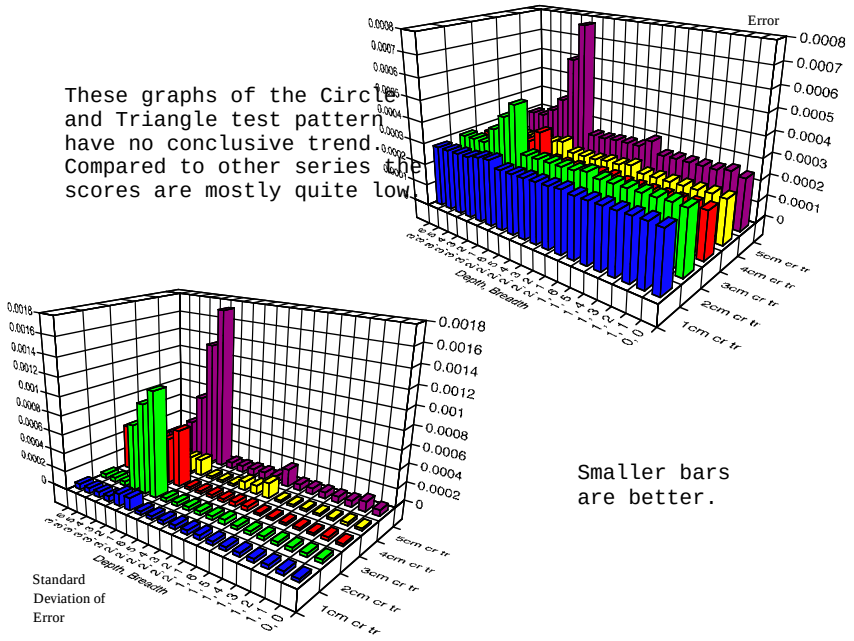


Figure 2.6:

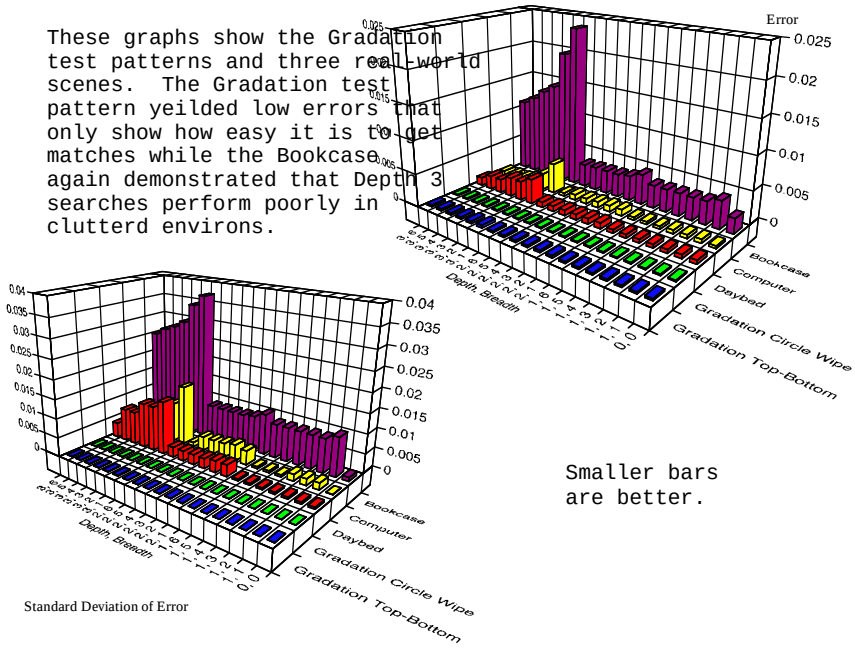


Figure 2.7:

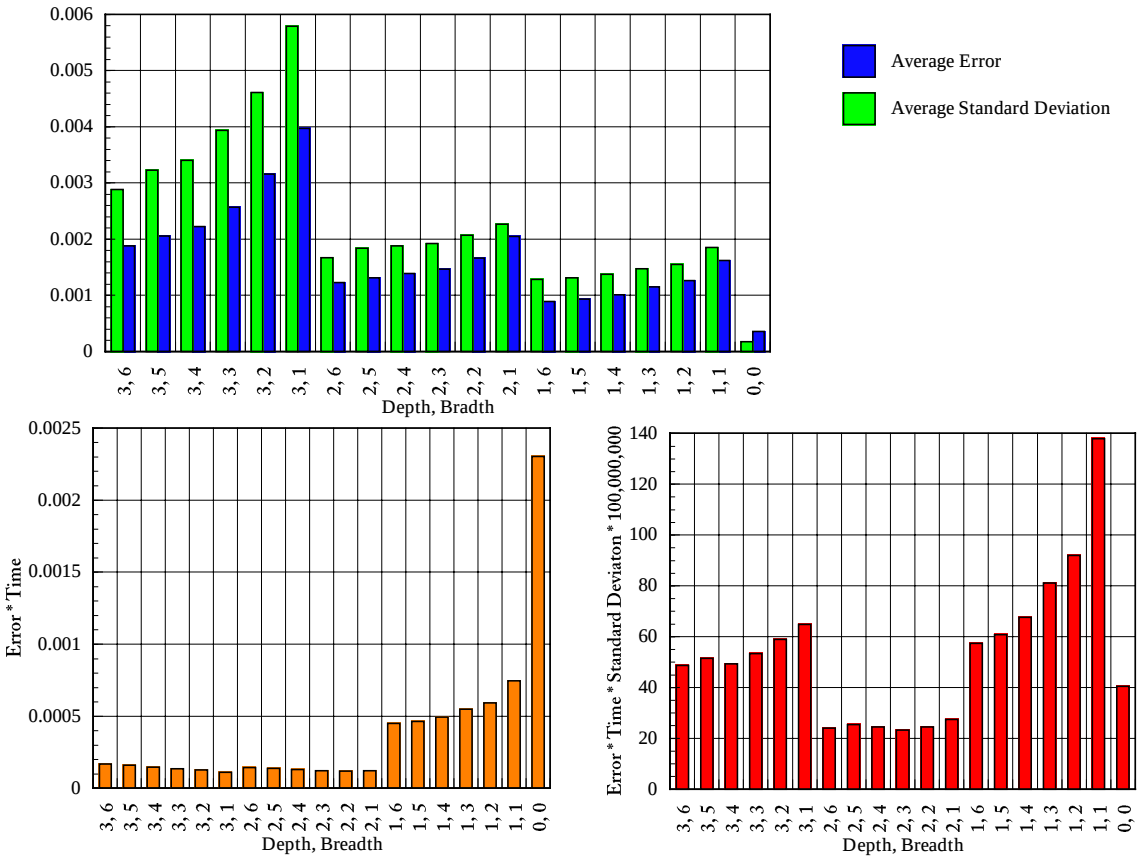


Figure 2.8: These graphts are the summation of all 22 test sets. I use Error multiplied by Time as a measure of worth for searches. It is proportionalbetween searches to show the trade off of speed and accuracy. $Error * Time * StandardDeviationofError$ add the element of reliability to that measure.

2.6 Analysis

I wanted a single number with which to rank the searches so I multiplied the error and time together. I feel this contrived measure to be valid because it maintains the proportions of the changes in both times and errors.

Figure 2.8 shows the errors multiplied by the times for all the searches in all of the data sets. Depth two searches are the clear winner.

2.7 Conclusions

Pyramid Searching works. At depth two, which got the best combined scores, loss of accuracy is small enough that it would reliably find the best match as seen by my human eye. At depth three, the program was frequently confused and not as accurate. Depth one had better accuracy, but in most applications I imagine, that trade-off for accuracy would not be desirable.

2.8 Applications

This approach has two obvious limitations: it cannot handle rotation or changes in distance. This could be overcome with a large set of keys for every reasonable combination of orientation and distance from the camera, but then the computational problem is beyond most present technology.

With those limitations noted, there are still plenty of situations where orientation and distance will be known in advance. Assembly lines, highways and text recognition are a few examples.

Key matching also works well for tracking. After an initial key is captured, the key is slowly averaged with new images of the tracked object. Thus changes in orientation and distance will be blurred so that the key remains similar to the target in new frames. This is fully worked out in [Wes95]. I have implemented a tracking routine that works fairly well but needs some optimization in a few variables.

2.9 Future Work

While programming, I thought up a way to make searching for multiple keys more efficient. It is simply to check for each one at the smallest level in the search and still follow up with a set breadth so that the best candidates of key and position in image are traced up the pyramid. If the breadth were five, the five keys checked at a higher level could be any combination of different keys or different positions of one or two keys. This could be advantageous over doing complete searches for each key and then comparing their errors, which was to me the more immediately obvious method.

If the captured image could be distributed quickly enough this might parallelize well when searching for many keys, with the keys split up among the processors.

2.10 Materials Used

The computer used in this project was a Apple Power Macintosh 7100/66AV with 32 Megabytes of RAM and a 256 Kilobyte L2 Cache. The video input device was a Connectix Color QuickCam.

The program was developed under the Symantec Project Manager version 8 using both Symantec's C Compiler and Apple's MrC compiler.

Illustrations were made in Claris Draw and GraphicConverter. Graphs were made in Delta Graph Pro 4. This report was edited in BBEdit and Alpha and was typeset using the Macintosh LaTeX 2_ε implementation OzTeX. Typeset on April 11, 1997.

Acknowledgments

Thanks to Mom, Dad, Tom Laeser, and many others for reading and critiquing the paper.

Thanks to Connectix, for making computer video affordable.

Thanks to Apple for creating Macintosh, without which I may have never learned to enjoy computing.

Appendix A

Data

This appendix contains the numbers behind the graphs in the Results section.

Appendix B

Source Code

This is only the crucial parts of the program. The complete source code is 1619 lines long.

Shown here are functions for getting keys, matching keys to images, calculating pyramids, search exhaustively or by pyramid and run performance tests. Also included is the main() block showing initialization and the main event loop.

Bibliography

[BB82] Dana H. Ballard and Chistopher M. Brown. *Computer Vision*. Prentice Hall Inc, Englewood Cliffs, NJ 07632, 1982.

[BGL95] J. Ross Beveridge, Christopher R. Graves, and Christopher E. Lesher. Local search as a tool for horizon line mathcing. Technical report, Colorado State U., December 1995.

[BRG96] J. Ross Beveridge, Edward M. Riseman, and Christopher R. Graves. How easy is matching 2d line models using local search? Technical report, Colorado State U., 1996.

[CC95] J. L. Crowley and H. I. Christensen, editors. *Vision as Process: Basic Research on Computer Vision Systems*. Springer Verlag, Berlin, 1995.

[FG96] John Fischer and Maria Gini. Vision-based mini-robots. *The Robotics Practitioner*, 2(2):40–46, Spring 1996.

[Hor94] Ian Horswill. Polly: A vision-based artificial agent. Technical report, MIT AI Lab, 1994.

[Loa95] Chris Loader. Local search algorithms for 2d geometric object recognition. Honors paper, University of Western Australia, 1995.

[Wes95] Mike Wessler. A modular visual tracking system. Master's thesis, Massachusetts Institute of Technology, June 1995.