**1.**

Forward Affine Transformation Backward Affine Transformation

For P1(0,0)&P1\*(10,0)

For P2(5,15)&P2\*(0,10)

For P3(25,15)&P3\*(10,10)

Solving the equations above, we get:

We can conclude that & matrices are inverses of each other.

**2.**

If we consider {x} as a grayscale image and **ng** as the number of occurrences of gray level **g**, the probability of an occurrence of a pixel of level **g** in the image is given by:

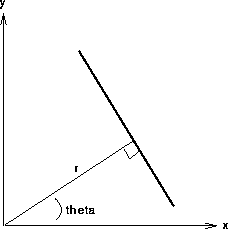
(g is between 0-255 and n is the total number of pixels).

After histogram equalization, the gray level histogram of the image {x} becomes a straight line which means that the probability of occurrences of all gray levels are equalized (i.e. gray levels are uniformly distributed). So, a second pass of histogram equalization changes nothing.

**3.**

The Hough Transform is a global method for finding straight lines (functions) hidden in larger amounts of other data.

Hough Space: Each point (r,θ) in Hough space corresponds to a line at angle T and distance d from the origin in the original data space.

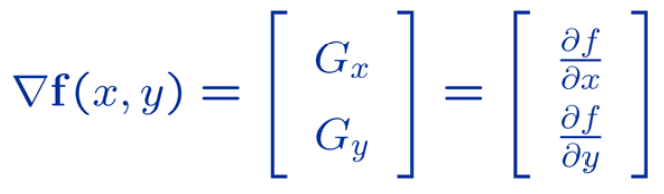


If we plot the possible values defined by each , points in cartesian image space map to curves in the polar Hough space. This point-to-curve transformation is the Hough transformation for straight lines. When we look at Hough space, points which are collinear in the Cartesian image space become readily apparent as they yield curves that intersect at a common point.

The transform is implemented by quantizing the Hough parameter space into finite intervals called accumulator cells. As the algorithm runs, each is transformed into a discretizedcurve and the accumulator cells which lie along this curve are incremented. Resulting peaks in the accumulator array represent strong evidence that a corresponding straight line exists in the image.

Using the Gradient:

The gradient is a measure of how the function f(x,y) changes as a function of changes in the arguments x and y. The gradient vector points in the direction of maximum change. The length of this vector indicates the size of the gradient:



If the gradient components gx&gy are given for an image g(x,y) containing a line, we can compute the gradient direction φ(x,y) = tan-1(gy/gx).

(Note: A(r,) in the algorithm represents the accumulator cells described above)

Algorithm:

for all g(xi,yi) > threshold

for all

r=xi.cos(φg(x,y)) + yi.sin(φg(x,y))

find index m corresponding to r and increment A(m, φg(x,y))

**Median Filter Code**

|  |
| --- |
| #include "img\_pro.h"  #include "my\_header.h"  int compare(const void \*a, const void \*b) {  return (\*(unsigned char \*)a - \*(unsigned char \*)b);  }  // applies median filter to image  // does not modify outer values outside the filter  void medianMask(unsigned char \*\*img, int i, int j, int size) {  int rows = 0, elements = 0, cols = 0;  unsigned char array[size \* size];  int limit = (size - 1) / 2; // to calculate limit addresses  // add the numbers inside filter to array  for (rows = i - limit; rows <= (i + limit); ++rows)  for (cols = j - limit; cols <= (j + limit); ++cols, ++elements)  array[elements] = img[rows][cols];  // apply quicksort algorithm from stdlib to find median  qsort(array, size \* size, sizeof(unsigned char), compare);  // change the img pointer to median value  img[i][j] = \*(array + ((size \* size - 1) / 2));  }  int main(int argc, char \*\*argv) {  unsigned char \*\*img, \*\*img2;  char \*pgm\_file;  int i, j, c, NC2, NR2, count, size, limit;  if (argc != 4) {  printf("\n Usage: median [filter strength] [times] [.pgm file] \n");  exit(-1);  }  pgm\_file = argv[3];  size = atoi(argv[1]);  count = atoi(argv[2]);  img = pgm\_file\_to\_img(pgm\_file, &NC2, &NR2);  show\_pgm\_file(pgm\_file);  limit = (size - 1) / 2;  for (c = 0; c < count; ++c) {  for (i = limit; i <= NR2 - limit; ++i)  for (j = limit; j <= NC2 - limit; ++j)  medianMask(img, i, j, size);  }  img\_to\_pgm\_file(img, "test2.pgm", NC2, NR2);  show\_pgm\_file("test2.pgm");  free\_img(img);  return (0);  } |