1. Implement Edge Detection Using Sobel Templates

* Sobel Vertical
* Description:

Vertical template:A number with black text

Description automatically generated with medium confidence

By comparing a 3x3 pixel area in the image with each template, the system assesses the similarity of this region to the provided template. In the context of vertical edge detection, a high value indicates that the template has identified a vertical edge in the image.

Nedderman Hall

A screenshot of a computer

Description automatically generated

Chess board

A computer screen shot of a computer screen

Description automatically generated

* Sobel Horizontal
* Description:

Horizontal template: A number with numbers on it

Description automatically generated with medium confidence

By comparing a 3x3 pixel area in the image with each template, the system assesses the similarity of this region to the provided template. In the context of horizontal edge detection, a high value indicates that the template has identified a horizontal edge in the image.

Nedderman Hall

A screenshot of a computer

Description automatically generated

Chess board

A screenshot of a computer

Description automatically generated

* Sobel Major Diagonal
* Description:

Major Diagonal template: A number with black numbers

Description automatically generated with medium confidence

By comparing a 3x3 pixel area in the image with each template, the system assesses the similarity of this region to the provided template. In the context of major diagonal edge detection, a high value indicates that the template has identified a major diagonal edge in the image.

Nedderman Hall

A screenshot of a building

Description automatically generated

Chess board

A screenshot of a computer

Description automatically generated

* Sobel Minor Diagonal
* Description:

Minor Diagonal template: A number with numbers on it

Description automatically generated with medium confidence

By comparing a 3x3 pixel area in the image with each template, the system assesses the similarity of this region to the provided template. In the context of minor diagonal edge detection, a high value indicates that the template has identified a minor diagonal edge in the image.

Nedderman Hall

A screenshot of a computer

Description automatically generated

Chess board

A computer screen shot of a computer screen

Description automatically generated

Code:  
#include <stdio.h>

#include <math.h>

#include <X11/Xlib.h>

#define DIM 512

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* This structure contains the coordinates of a box drawn with \*/

/\* the left mouse button on the image window. \*/

/\* roi.x , roi.y - left upper corner's coordinates \*/

/\* roi.width , roi.height - width and height of the box \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

extern XRectangle roi;

unsigned char convolution(unsigned char image[DIM][DIM],int template[3][3],int row,int col);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Main processing routine. This is called upon pressing the \*/

/\* Process button of the interface. \*/

/\* image - the original greyscale image \*/

/\* size - the actual size of the image \*/

/\* proc\_image - the image representation resulting from the \*/

/\* processing. This will be displayed upon return \*/

/\* from this function. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void process\_image(image, size, proc\_img)

unsigned char image[DIM][DIM];

int size[2];

unsigned char proc\_img[DIM][DIM];

{

// Sobel templates

// Please uncomment the desired template as a parameter in the convolution function

// int sobelVertical[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};

// int sobelHorizontal[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};

// int sobelMajDiag[3][3] = {{0, -1, -2}, {1, 0, -1}, {2, 1, 0}};

int sobelMinDiag[3][3] = {{-2, -1, 0}, {-1, 0, 1}, {0, 1, 2}};

for (int x = 0; x < size[0] - 2; x++)

{

for (int y = 0; y < size[1] - 2; y++)

{

proc\_img[x][y] = convolution(image,sobelMinDiag,x,y); // switch the second parameter, if needed.

}

}

}

unsigned char convolution(unsigned char image[DIM][DIM],int template[3][3],int row,int col)

{

int sum = 0;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

sum += (int)image[row + i][col + j] \* template[i][j];

}

}

// check if sum is out of range [0:255]

if (sum < 0)

{

sum = 0;

}

else if (sum > 255)

{

sum = 255;

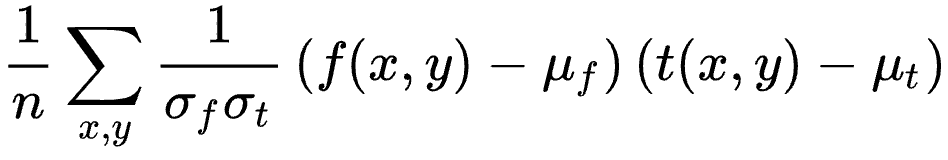
}

return (unsigned char)sum;

}

2. Implement Template Matching Using Normalized Convolution

* Description:

Formula used: 

Normalized Convolution calculates the cross-correlation between a portion of an image and a selected template, providing a measure of similarity. Both the image and the template need to undergo normalization before the cross-correlation process. The brighter the area, the more similar it is to the template.

* Nedderman Hall

Template: A black and white image of a square object

Description automatically generated

Result:

A screenshot of a computer

Description automatically generated

* Chess board

Template: 

Result:

A screenshot of a computer

Description automatically generated

Code:

#include <stdio.h>

#include <math.h>

#include <X11/Xlib.h>

#define DIM 512

extern XRectangle roi;

float convolution(unsigned char image[DIM][DIM], float template[roi.width][roi.height], int row, int col, float template\_mean, float template\_stddev);

void normalize(float data[roi.width][roi.height], float mean, float stddev);

void get\_mean\_stddev(float data[roi.width][roi.height], float \*mean, float \*stddev);

void process\_image(unsigned char image[DIM][DIM], int size[2], unsigned char proc\_img[DIM][DIM])

{

// selected templates

float template[roi.width][roi.height];

float template\_mean, template\_stddev;

// extracting template from image

for (int x = 0; x < roi.width; x++)

{

for (int y = 0; y < roi.height; y++)

{

template[x][y] = (float)image[roi.x + x][roi.y + y];

}

}

get\_mean\_stddev(template, &template\_mean, &template\_stddev);

normalize(template, template\_mean, template\_stddev);

for (int x = 0; x < size[0]-roi.width; x++)

{

for (int y = 0; y < size[1]-roi.height; y++)

{

float conv\_result = convolution(image, template, x, y, template\_mean, template\_stddev);

// Normalize the result of the convolution to be between 0 and 255

proc\_img[x + roi.width / 2][y + roi.height / 2] = (unsigned char)(conv\_result \* 255.0f);

}

}

}

float convolution(unsigned char image[DIM][DIM], float template[roi.width][roi.height], int row, int col, float template\_mean, float template\_stddev)

{

float sum = 0.0;

float n = roi.height \* roi.width;

float subimage\_mean, subimage\_stddev;

float subimage[roi.width][roi.height];

// Extract subimage and calculate its mean and standard deviation

for (int x = 0; x < roi.width-1; x++)

{

for (int y = 0; y < roi.height-1; y++)

{

subimage[x][y] = (float)image[row+x][col+y];

}

}

get\_mean\_stddev(subimage, &subimage\_mean, &subimage\_stddev);

normalize(subimage, subimage\_mean, subimage\_stddev);

// Perform normalized cross-correlation

for (int i = 0; i < roi.width; i++)

{

for (int j = 0; j < roi.height; j++)

{

sum += subimage[i][j] \* (template[i][j] - template\_mean) / template\_stddev;

}

}

sum = fmax(0.0, fmin(sum / n, 1.0)) \* 255.0;

return sum;

}

void normalize(float data[roi.width][roi.height], float mean, float stddev)

{

// Normalize each pixel value using the mean and standard deviation

for (int i = 0; i < roi.width; i++)

{

for (int j = 0; j < roi.height; j++)

{

data[i][j] = (data[i][j] - mean) / stddev;

}

}

}

void get\_mean\_stddev(float data[roi.width][roi.height], float \*mean, float \*stddev)

{

\*mean = 0.0;

\*stddev = 0.0;

// Compute mean

for (int i = 0; i < roi.width; i++)

{

for (int j = 0; j < roi.height; j++)

{

\*mean += data[i][j];

}

}

\*mean /= (roi.width \* roi.height);

// Compute standard deviation

for (int i = 0; i < roi.width; i++)

{

for (int j = 0; j < roi.height; j++)

{

\*stddev += pow(data[i][j] - \*mean, 2);

}

}

\*stddev /= (roi.width \* roi.height);

\*stddev = sqrt(\*stddev);

}