Assignment #6 Sample Solutions

Question 1: Filtering (see also file filtering.c on the course homepage)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "netpbm.h"

// Linearly convolve the intensity information of an image with a convolution filter (both
// stored in Matrices). The center of the filter serves as its anchor for storing the result.
// For even number of rows or columns, the anchor will be rounded to the left or upward, resp.
// The output is a matrix of the same size as the original image, with zeroes filling the
// border regions that are unreachable by the filter.
Matrix convolve(Matrix img, Matrix filter)
{
    int m, n, k, l, mOffset = filter.height / 2, nOffset = filter.width / 2;
    double sum;
    Matrix result = createMatrix(img.height, img.width);
    for (m = filter.height - 1; m < img.height; m++)
        for (n = filter.width - 1; n < img.width; n++)
        {
            sum = 0.0;
            for (k = 0; k < filter.width; k++)
                for (l = 0; l < filter.height; l++)
                    sum += img.map[m - k][n - l] * filter.map[k][l];
            result.map[m - mOffset][n - nOffset] = sum;
        }
    return result;
}

Image sobel(Image img)
{
    double hFilter[3][3] = { { 1.0, 0.0, -1.0 }, { 2.0, 0.0, -2.0 }, { 1.0, 0.0, -1.0 } };
    double vFilter[3][3] = { { 1.0, 2.0, 1.0 }, { 0.0, 0.0, 0.0 }, { -1.0, -2.0, -1.0 } };
    Matrix hSobel = createMatrixFromArray(&hFilter[0][0], 3, 3);
    Matrix vSobel = createMatrixFromArray(&vFilter[0][0], 3, 3);
    Matrix inputMx = image2Matrix(img);
    Matrix hResult = convolve(inputMx, hSobel);
    Matrix vResult = convolve(inputMx, vSobel);
    Matrix result = createMatrix(inputMx.height, inputMx.width);
    Image output;
    int i, j;
    for (i = 0; i < inputMx.height; i++)
        for (j = 0; j < inputMx.width; j++)
            result.map[i][j] = sqrt(SQR(hResult.map[i][j]) + SQR(vResult.map[i][j]));
    output = matrix2Image(result, 1, 1.0);
    deleteMatrix(hSobel);
    deleteMatrix(vSobel);
    deleteMatrix(inputMx);
    deleteMatrix(hResult);
    deleteMatrix(vResult);
    deleteMatrix(result);
    return output;
}
```
// Create and return a Gaussian filter of size (vSize, hSize) with standard deviation sigma.
// The entries in the filter always add up to 1.
Matrix makeGaussianFilter(int vSize, int hSize, double sigma)
{
    double sum = 0.0;
    int i, j;
    Matrix gauss = createMatrix{vSize, hSize};

    for (i = 0; i < vSize; i++)
        for (j = 0; j < hSize; j++)
            {
            gauss.map[i][j] = exp(-(SQR((double)i - (double)(vSize - 1) / 2.0) + SQR((double)j -
            (double)(hSize - 1) / 2.0)) / SQR(sigma) / 2.0);
            sum += gauss.map[i][j];
            }
    for (i = 0; i < vSize; i++)
        for (j = 0; j < hSize; j++)
            gauss.map[i][j] /= sum;
    return gauss;
}

Image gauss(Image img, int size)
{
    Matrix gFilter = makeGaussianFilter(size, size, (double)size / 4.0);
    Matrix inputMx = image2Matrix(img);
    Matrix result = convolve(inputMx, gFilter);
    Image output = matrix2Image(result, 0, 1.0);
    deleteMatrix(gFilter);
    deleteMatrix(inputMx);
    deleteMatrix(result);
    return output;
}

void main()
{
    Image inputImg = readImage("sample.ppm");
    Image sobelImg = sobel(inputImg);
    Image gauss3Img = gauss(inputImg, 3);
    Image gauss9Img = gauss(inputImg, 9);
    writeImage(sobelImg, "sobel.pgm");
    writeImage(gauss3Img, "gauss3.pgm");
    writeImage(gauss9Img, "gauss9.pgm");
    deleteImage(inputImg);
    deleteImage(sobelImg);
    deleteImage(gauss3Img);
    deleteImage(gauss9Img);
}

Question 2: A Maximum Filter?

We have seen that a median filter is good at removing salt-and-pepper noise. What effect would a maximum filter have, i.e., a filter whose output, instead of the median, is the maximum intensity within its \(n \times n\) pixel area? What kind of noise could this filter remove best? What would be its side effect?

This filter would expand light areas in the image and increase the average intensity of the image. It would be good at removing “dark” noise, but substantial “light” noise would result in much of the image turning bright. This filter would also affect the contours in the image.
**Question 3: FAST and BRIEF**

(a) What is the effect of increasing the threshold in the FAST algorithm?

The algorithm will become more selective and pick fewer corner points.

(b) What would happen if instead of 12-pixel runs we would only require 8-pixel runs for detecting a corner with FAST? Would we still detect corners?

We would still fund corners but also points on almost straight lines. The algorithm would be much less selective and also indicate points that are not well-suited for object recognition.

(c) In the BRIEF method, we compare intensity at point pairs \((a, b)\), and if it is greater for \(a\) than for \(b\), we add a “1” to our descriptor and otherwise a “0”. What would be the advantages and disadvantages of using a vector of real numbers as the descriptor instead, each indicating the intensity difference between \(a\) and \(b\) for one of the pairs \((a, b)\)?

The algorithm would become less efficient because instead of one bit per pair we would now have to store a number between -255 and 255. The descriptor would become more selective which would be an advantage if there is little difference in lighting and orientation between the two views of an object that we are comparing. If there are changes in lighting between the images, then the binary vectors might work better, because they only consider which of the two pixels in a pair is brighter, which usually does not change even if the lighting varies.

(d) We found that the original BRIEF method is slightly rotation invariant, i.e., if it sees the same keypoint at a slightly different orientation, it will still recognize it. What is the reason for it? It always looks at exactly the same points relative to the keypoint, so if the local information is rotated, it will compare completely different pixels, so should it not fail for even the slightest rotation?

If we consider two points that are close to each other in a real-world image, then we will typically find that they have very similar intensity. The further they are apart from each other, the less likely we are to find it. This is a property of our world; it does not look like a random distribution of pixels but contains local objects and surfaces within which the intensity does not vary much. Consequently, if there is only a slight rotation of a few degrees and a corresponding shift of only a few pixels in the points to be compared, for most of the pairs \((a, b)\) the comparison will not flip from a 0 to a 1 or vice versa. Therefore, the matching of these areas will still work.

**Question 4 (required for CS670, bonus for CS470): A Hough Transform Variant**

For some mysterious application, you need to build an application that detects horizontal lines in a picture. These lines are precisely horizontal, but your client does not only want to know in which rows of the image they are located but also where they start and end, i.e., what their leftmost and rightmost pixels are.

How would you set up a Hough transform to accomplish this task? What would be the dimensions of the output space, and what would be the equation for placing votes?

In order to describe such a line, we need to provide its column \(u\) and its leftmost and rightmost columns \(v_L\) and \(v_R\). These are the 3 dimensions of our output space.

For an edge at position \((i, j)\) we cast a vote at \((u, v_L, v_R)\) if \(i = u \land v_L \leq j \leq v_R\)