CSE 4360 / 5364 - *Autonomous Robots*

Homework 3- Fall 2023

Due Date: Nov. 12 2023, 5:30pm

Problems marked with * are mandatory only for students of CSE 5364 but will be graded for extra credit for students of CSE 4360.

Edge Detection, Template Matching, and Blob Coloring

Basic feature extraction and region growing techniques are important steps in the first stages of image processing. In this homework assignment you are going to implement edge detection, template matching, and blob coloring as basic techniques.
Programming will follow the same procedure as in the last assignment. You are provided with a C library containing facilities to display images which will link with the file *process_image.c*. Your code for the assignment should be written in this file in the function *process_image(image, size, proc_img)*. To start, you have download the appropriate code repository for your machine. This directory contains the following files:

Imakefile This file is used to create a machine specific Makefile by typing *xmkmf*.

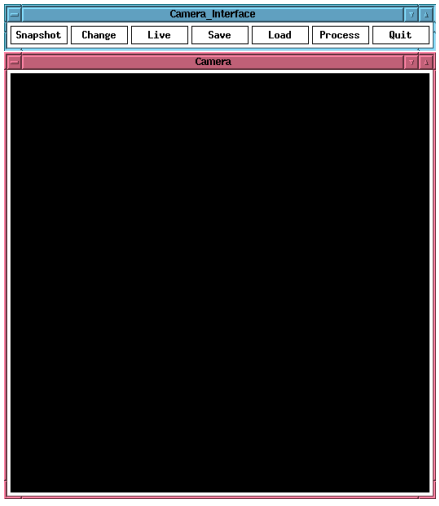
process_image.c This is the file you have to edit in order to implement your visual routines.

lib/libCamera.a This library contains the graphical interface.

pictures A directory containing a number of example images.

The Image Processing Environment

To generate the vision program you have to type *make*. This will create the graphical image interface *Vision*. This interface should look as follows:



The graphical interface consists of two separate windows containing a set of buttons and the actual image display window. The most important features for this assignment are the *Load*, *Save*, and *Process* buttons. *Load* will open a small window to enter the name of an image file for processing. *Save* permits to save a processed image to a file. There are some peculiarities to this interface. Do not hit *return* after entering the name of a file. This will simply insert a new line into the filename. To load (or save) the file you have to press the *ok* button in the filename window.
The image files used here consist of 2 integers indicating the width and height of the image in pixels followed by the array of pixel values (each pixel is an *unsigned char*, i.e 0 – 255). A number of image files can be found in the *pictures* directory. If you want you can also add your own pictures (a description of how to convert images into this format is given below).
The *Process* button will call the function *process_image* (your routine) and the resulting processed image will be displayed. The original image is overwritten and the processed image can be saved using the *Save* button.

Generating your own Images (Only in Linux Version)

A number of test images are provided in the pictures subdirectory. However, you can also add your own images. For this purpose, a conversion program that runs on Linux (including Omega.uta.edu) is provided. The program *xv_cam* allows you to transform a large number of image formats into the format used here (*LPR format*). To do this, just load your favorite image and save it in the *LPR* format. The only fact to consider is that the image interface will not use anything beyond a 512x512 image resolution (the rest of the image will be truncated when loading it into the *Vision* program.
In the same way a processed image can be converted into your favorite image format (such as *GIF* or *JPEG*).

The Assignment

In this assignment you are to implement three basic vision processing techniques. To implement these you have to write the required routines as the function *process_image()*. This function has the following structure:

```
void process_image(image, size, proc_img)
unsigned char image[DIM][DIM];
int size[2];
unsigned char proc_img[DIM][DIM];
{
...
}
```

The parameters of this function are *image* which is the original image, *size* which is the width and height of this image, and *proc_img* which is the image resulting from your processing. If this resulting image does not have the same dimensions as the original image, then you have to update *size* to the dimensions of the new image.

1. Implement Edge Detection Using Sobel Templates

   Edge detection is one of the most basic feature extraction techniques. One of the most common techniques is using convolution with edge templates representing different directional edges with Sobel templates among the most frequently used.

   $3x3$ Sobel templates for vertical, horizontal, and the two diagonal edges are represented as:

   vert.: $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$, hor.: $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$, maj. diag.: $\begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}$, min. diag.: $\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$

   For this part you are to implement edge detection using convolution with the four Sobel templates above[1]. Your code, after computing the convolution, should display the resulting feature map in the result image (you will have to normalize the convolution results to fall into the range between 0 and 255). Edge detection with each of the templates should be performed separately and results for each image should thus be shown in four separate pictures (one per template. If you want, you can also include a fifth picture showing the combined edge map for all features together (but this is not required).

   You should hand in a short description, the code (submitted electronically), and printouts of the images resulting from your processing (one for each of the four edge orientation templates). The printouts are best included into the same document as the short description. applied to the Nedderman Hall and the chess board images, *nedderman.lpr* and *chess.lpr*.

2. Implement Template Matching Using Normalized Convolution

   Implement normalized convolution for template matching. To do this you should rename the *process_image.c* file from the previous part of the assignment and write a new *proc_img* function. Your routine has to permit specifying a region of the image which will subsequently be used as the template (thus finding all instances of similar objects in the image). Using this template you are to implement template matching using normalized cross correlation (convolution which adjusts for the image and template means as well as for the image and template variances - i.e. contrast)[2]. You can either use global or local image normalization. As in the previous part, you should write the result of the convolution back into the result image, normalizing the values to be between 0 and 255 (and of the correct data type).

   You can select a portion of the image using the mouse. Pressing the left mouse button will set one corner of a rectangle and releasing the button will set the opposite corner. The image coordinates of this rectangle can be read from the variable *roi*, where *roi.x* and *roi.y* are the left upper corner and *roi.width* and *roi.height* are the width and height of the selected rectangle.

   Again, you should hand in a short description, the code (submitted electronically), and printouts of the images resulting from your processing applied to the Nedderman Hall and the chess board images, *nedderman.lpr* and *chess.lpr*.

[1] Note: Be careful with your data types. The image is represented as unsigned characters (i.e. numbers between 0 and 255). The convolution, however can generate positive and negative numbers as well as numbers much larger than 255 so it needs to be performed using a different datatype and can thus not be done directly in the result image array.

[2] Note: Be again careful with your data types. Normalization will generate floating pint numbers and thus normalized convolution can not be performed in terms of integers or unsigned characters.

3.* Implement Segmentation Using Blob Coloring

Here you have to implement blob coloring to identify regions with a common intensity in the image. To do this you should again rename the *process_image.c* file from the previous part of the assignment and write a new *proc_img* function which performs blob coloring. The result of this operation should consist of a number of regions, each with its own, unique color (intensity).

Again you should hand in a short description, the code (submitted electronically), and the result of your algorithm on the image *blocks2.lpr* from the *pictures* directory.