Feature Extraction and Matching

Sharif Anani
South Dakota School of Mines & Technology
Rapid City, South Dakota, SD
sharif.anani@mines.sdsmt.edu

Abstract

In this project paper, an attempt is made at extracting corner features from an image and matching them with others in an image showing a similar scene. The project relies on Harris corners for the features and operates on the feature patches to make them invariant to rotation and increase their invariance to illumination. Section 1 introduces the general project, section 2 shows the steps taken to execute the algorithm, section 3 shows results and discusses their success, sections 4, 5, and 6 analyse, conclude, and discuss future improvement.

1. Introduction

Finding good features in an image is an ongoing problem in the field of Computer Vision. It is very important to find the good features that are repeatable in multiple types of scenes, and at the same easy enough to distinguish from other features in an image. The features chosen for this project are Harris corners, found using the Harris response (discussed later in the paper).

1.1. Background

Before discussing how the corners are found, need to know how corners are respresented in a digital image in order to be able to identify them. In a gray scale image, edges are transitions from light values to dark values (i.e. 255 to 0 for a uint8 image). We can think about corners the same way; corners are generally a transition from dark to light(or vice versa). Knowing this, we know that edges will show on the gradient of the image (the vector containing its directional derivatives).
The problem with stopping at the gradient is that edges also show up as high values in the gradient image, so we need a method to differentiate between edges and corners in a gradient image, and this is where the Harris response comes into play.

1.2. Edges Vs. Corners

The harris corner detector can be considered as a "combined corner and edge detector". The basic idea of operation is simple: taking a small window over the image, we can distinguish corners and edges from flat surfaces by looking at the changes in intensity as we move the window in all the directions. If the intensity changes significantly when moving in one direction, but not the other, then we have an edge. If intensity changes significantly in all directions, then we have a corner.

1.3. The Mathematics

As mentioned before, the Harris detector works with directions and changes in intensity. To quantify these changes, Harris and Stephens looked at this change using the response operator:

$$R = det(M) - k * trace(M) \tag{1}$$

Where

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$$

and $w(x,y)$ is a window function - in our case, Gaussian - and k is empirically measured to be between 0.04 and 0.06.

2. Algorithm Development

The solution developed for this project is a simple algorithmic program running through simple stages.

2.1. Finding the Features

The first step is preconditioning the image: the image is read into MATLAB, converted to grayscale, and then run through histogram equalization; histogram equalization will create better repeatable illumination conditions across different shots of the same scene, and this helps the detector be invariant to more factors. The image is then normalized

by subtracting the mean and dividing by the standard deviation.
The second step is detecting the features: the result of the first few operations is passed into `cornerDetection.m`, a function created to calculate and threshold the harris response. The function is available on `https://github.com/sharifanani/ComputerVisionF15`.
in `cornerDetection.m`, the gradients are calculated along with the Harris response. The Harris response is then subjected to a threshold and finally non-maximum suppression.

2.1.1 Non-Maximum Suppression

Non-maximum suppression is applied in one of two ways:
The first method of non-maximum suppression goes through every pixel that is seen as a corner and then loops through the whole image checking for pixels that are corners and are within a 10 pixel radius. If the pixel response value is higher than that of the pivot pixel, it is left alone, otherwise, it is set as zero.
The other method is faster and uses MATLAB's `blockproc` function, and finds the maximum in every block. The second method does not work as good because the blocks do not overlap, so pixels at the boundaries can be local maxima in other blocks, but they are being brought down to zero earlier. However, it is much faster than the first method and works well enough to use for the purpose of the project.

The following block of code shows non-maximum suppression for the `blockproc` approach:

```
supp =@(block_struct) suppressNonMax(
    double(block_struct.data));

J2 = blockproc(R,[45 45],supp);

[rows,cols]=find(J2 == 1);%return
    locations of corners
```

Where `suppressNonMax.m` is a simple function that returns a matrix where the maximum element of a black is set to 1 and the rest are set to 0:

```
function [B] = suppressNonMax( I )
Z = zeros(size(I));
[row,col] = find(I == max(I(:)));
Z(row,col) = 1/max(I(:));
B = I.*Z;
end
```

A comparison of the two methods for non-maximum suppression is available in the results section of this paper.

2.2. Creating a Feature Descriptor

To be able to match the features we find, we need to be able to compare them to one another. Since the corner is pointed to by its pixel, it is rather difficult to be able to describe the feature using one pixel value. To better describe the feature, we start by taking a block of 10x10 around the pixel.
The 10x10 block is rotated with the angle of the average gradient of the bock - the dominant angle. This aligns all the features horizontally such that all features have the same angle when compared to other feature, which provides the descriptor with rotation invariance. After rotating the 10x10 block, another smaller patch is taken from the block, this way we don't have the black (or zero) filled regions that are an artefact of the rotation.
Histogram equalization and normalization of the image before detecting features makes the features more invariant to luminance/brightness. The descriptors are all normalized to have a length of 1.

2.3. Matching the Features

The first part of the algorithm is considered to be `makeFeatures.m`, which is a function that returns two variables: `FEATURES` and `squares`. `FEATURES` holds the center pixels, patches, rotated patches, and descriptors, `squares` holds the bounding boxes of each feature that point toward the direction of the gradients.
To match the features, two sets of features and squares are passed into `featureMatcher2.m` along with the images and a threshold number. `featureMatcher2.m` takes the descriptors and calculates the Sum of Squared Distances (SSD).
A matrix `similarities` with a size of *features1(2),features2(2)* is generated. Every row of `similarities` is a set of SSDs for a feature in the first image with all the features in the second image.

2.3.1 Ratio Test

Every row is sorted in an ascending order, and the first element is devided by the second. If the ratio satisfies the threshold, the first value is considered to be a match, and the indices of the two matching features are stored in the matrix *matches*. After *matches* is generated, any duplicates are deleted, preventing two features from matching to the same place in the second image.

3. Test Results and Verification

To test the the success of the project, we were provided a set of test images of 6 images each. Matching the first image gets more difficult to match as we move forward in the

images.
The images each test a different aspect of the quality of matching. For example, the `leuven` set tests luminance invariance, the `bikes` set varies focus, the `wall` set tests affinity, and the `graf` set tests combinations of affinity and rotation. The results are all placed in a pdf file that can be viewed at: