Task #3 Computer Vision

GitHub Repo Link:

Team Information

Name	Sec	BN
Ahmed Khaled	1	4
Alaa Gamal	1	15
Bassam Mostafa	1	22
Mustafa Yehia	2	33

How to Run our Application

For VS code users

1. Open VS Code
2. Download extensions named (CMake, CMake Tools)
3. Open the Project Directory on VS Code
4. Press CTRL + SHIFT + P
5. Choose "CMake: Configure" → If it asks you to choose a kit, Choose GCC 9.3.0 → If it is not in the options then download GCC on your device
6. Press CTRL + SHIFT + P
7. Choose "CMake: Build"
8. Press CTRL + SHIFT + P
9. Choose "CMake: Run Without Debugging"
10. Enter the full "absolute" path of the image you want to detect the edges on
11. Go check out the results in the outputs folder

For UNIX terminal users

1. Create "build" directory if it does not exist
2. Run a terminal in the build folder
3. Run the command ```cmake ..```
4. Run the command ```make```
5. Run the project ```./CVProject```
6. Enter the full "absolute" path of the image you want to detect the edges on
7. Go check out the results in the outputs folder

Demo Link

- Demo Link

GUI Design and Implementation

Framework

- The used framework is GTK and is implemented in C++ using gtkmm wrapper which supports all OOP features of C++ that would be impossible to use without it.

Reasons for this framework

- GTK is chosen over other C++ frameworks such as Qt and wxWidgets for multiple reasons:
    - Seamless integration with GNOME desktop environment (the GNOME environment itself is built using GTK)
    - Is free to use for proprietary software (as opposed to Qt)
    - Uses C++ std classes (as opposed to Qt which uses its own classes)

Design

- The application is designed using the “Glade” app, which produces an XML file that contains the properties of the widgets (components) of the app.

Extract Unique Features

Harris Operator (Harris Corner Detector)

Main Concept

The Harris Corner Detector finds the interest points by calculating the variations of pixel intensities over x and y directions (assuming a simple 2D image plane) and then thresholding the Corner Response which is function of eigen values of the covariance matrix formed from the Intensity derivatives.

The pixels with Corner Response over a threshold value are our interest points because in a sense there is higher chance of finding a corner at that pixel.

It is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image. It is since a corner has twoish gradients. Harris depended on the intensity difference law,

$$E(u,v) = \sum_{c,y} w(x,y)[I(x+u,y+v)^2 - I(x,y)^2)]$$

Where:

- $w(x,y)$: Gaussian Window
- I: Intensity of Pixel.

Using Taylor expansion and vectorization, we can deduce that:

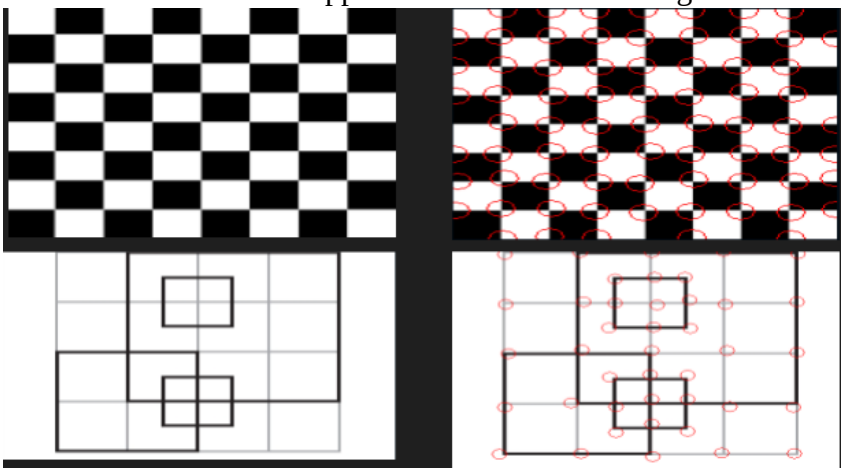
$$E(u,v) \cong [u\ v]\ M\ [u\ v]^T$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Where:

- M: Calculate Harris-Eigen operator R
- $R = det(M) - k * trace(M)^2 = \lambda_1\lambda_2 - k * (\lambda_1 + \lambda_2)^2$

For pixels that have R larger than a threshold - that may differ for each image -, we consider this pixel a corner taking into consideration the non-max suppression for a block of neighbors.



Algorithm

1. Calculate variations along x and y (i.e., the x-derivative and the y-derivative (I_x, I_y)). We can do this using the Sobel Operator.
2. For every pixel in the image compute Covariance matrix from the values of (I_x, I_y) at neighboring pixels (i.e., Pixels within a window).
3. Calculate eigen values of Covariance Matrix at each pixel using SVD Decomposition.
4. Compute Corner Response $= \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$ where k is a constant $(k = 0.04)$.
5. Get rid of pixels with Corner Response lower than their surrounding pixels (By applying a window).
6. The final Interest points are the pixels with Corner Response greater than a Threshold value.

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

$$C = U \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} V^T$$

Computation Time Comment

$$Differ\ for\ Different\ Machines, Result\ on\ Video = 120\ ms$$

Result

Feature Descriptors

Scale Invariant Feature Transform (SIFT)

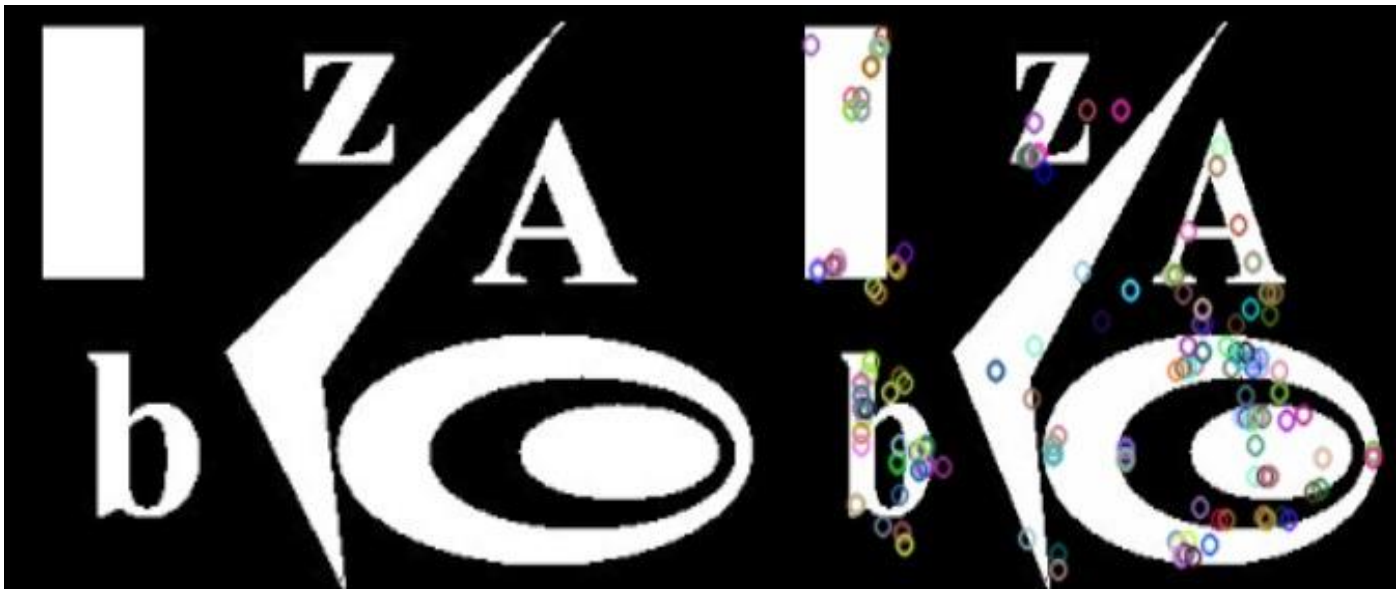
It is a feature detection algorithm in computer vision to detect and describe local features in images. SIFT key points of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on matching algorithms. This algorithm consists of sequential steps:

1. Generate Space: first, we generate new images in 2 new dimensions: octaves which is the image resized in a new ratio each new octave is half the size of the previous octave ($400 * 400 \rightarrow 200 * 200 \rightarrow 100 * 100 \rightarrow 50 * 50$), and scales, where a scale is the original image applied to a Gaussian filter and each scale image, is filtered with $n = kn * 0$ where $0 = 1.6$, $k = 1.414$ and $n = 1, 2, 3, 4$ is the order of the scale.
2. Scale-Space Extrema Detection: after creating the 5 scales, we interpret 4 DoGs (Difference of Gaussians) and we search for every pixel in the space-scale dimension, if the pixel is a local extrema (compared to the 8 surrounding neighbors and the equivalent 9 pixels in the other DoGs), we consider it as a key point.
3. Contrast thresholding: to have better accuracy of the interpreted key points in the last step, we use contrast thresholding by taking the Taylor expansion. If the new, more accurate, located key point has an intensity less than a threshold (0.03), it is rejected.
4. For the key points that satisfy the thresholding, we find an orientation by taking the phase of the gradient of a window around that key point. An orientation 36-bins histogram is used where we take the peak of this histogram as our orientation in addition to any angle that has an amplitude in the histogram of 0.8 of the peaks. A key point descriptor is then created and used by matching algorithms.

Computation Time Comment

$$Differ\ for\ Different\ Machines, Result\ on\ Video = 630\ ms$$

Results



Matching Features

Sum of Square Difference (SSD)

Main Concept

In this method, we compute the score by summing up the square of the difference between each neighboring pixel value. Note that the neighboring pixel values are stored in the feature descriptor.

Algorithm

$$SSD_{score} = \frac{1}{m^2} \sum_{i=1}^{n} (IP_1(i) - IP_2(i))^2$$

Where:

- $IP_1(i)$: Intensity at Interest point (corner) I of image1.
- n: Number of interest points.
- m: Size of the neighbor window used.

The lower the value of SSD_{score} the more likely that the interest points are I. It can be observed that some features may not be unique in a picture which means when you try to make correspondence you find multiple interest points giving, I score. To get rid of such points we put a threshold on the value of the ratio of minimum score and second minimum score (SSD_{ratio}).

Normalized Cross Correlation (NCC)

Main Concept

In this method, we compute the cross correlation of difference between pixel value and mean of feature descriptor using the following formula.

Algorithm

$$NCC_{score} = \frac{\sum_{i=1}^{n} (IP_1(i) - mean_1)(IP_2(i) - mean_2)}{\sqrt[2]{\sum_{i=1}^{n} (IP_1(i) - mean_1)^2 \sum_{i=1}^{n} (IP_2(i) - mean_2)^2}}$$

Where:

- $IP_1(i)$: Intensity at Interest point (corner) I of image1.
- n: Number of interest points.

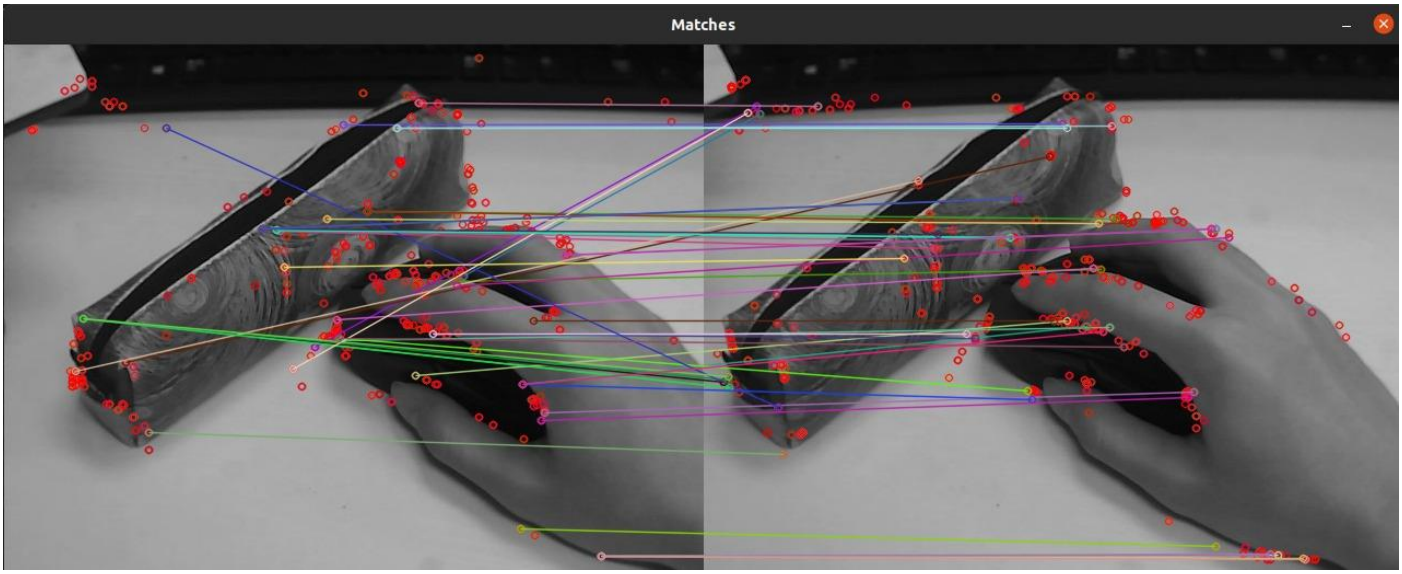
The larger the value of correlation the more likely that the interest points are similar. We perform a threshold on the ratio of the maximum score and second maximum score to get rid of the non-unique interest points.

Computation Time Comment

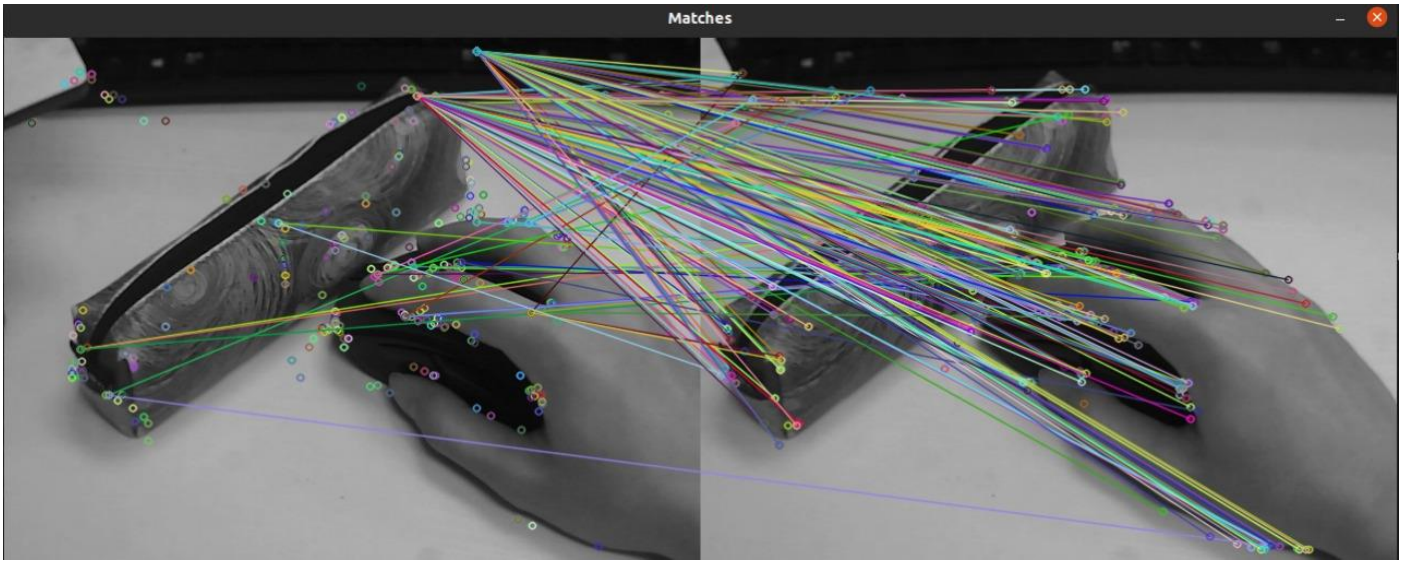
Sum of Square Difference (SSD)	Normalized Cross Correlation (NCC)
$Differ\ for\ Different\ Machines, Result\ on\ Vide = 250\ ms$	$Differ\ for\ Different\ Machines, Result\ on\ Vide = 659\ ms$

Results

Sum of Square Difference (SSD)



Normalized Cross Correlation (NCC)



Comment on NCC Result

Output differs from windows to ubuntu, the previous result is obtained while using g++ compiler in ubuntu OS and the below is from windows OS (Same Code for Both)