CSEE 4840 Embedded Systems Design Doc

qSIFT

Name	UNI
Madhav Bhat	mb4989
Khushi Gupta	kg3023
Prathamesh Sahasrabudhe	ps3320
Jeffrey Wolberg	jnw2138
Daniel Seligson	ds3824

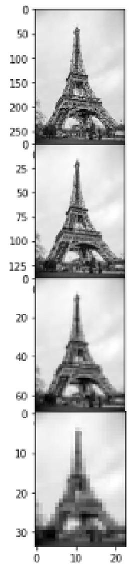
Table of Contents

1. Introduction
2. System Overview
    2.1 Software
    2.2 Hardware
    2.3 Hardware Memory Constraints
3. Milestones
4. References

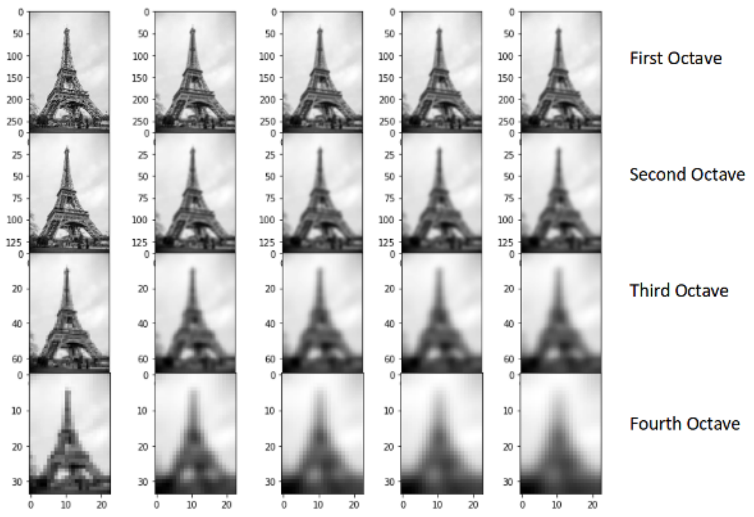
1. Introduction

Scale Invariant Feature Transform (SIFT) is an algorithm that is widely used in image processing. It detects and describes image features in scale and rotation invariant encoding, and therefore can be used to register images taken from different distances and angles. Below is an example of SIFT features from two images of the same object, taken at different scales and angles.
This feature allows the SIFT algorithm to perform better than other image processing techniques since it is less dependent on the orientation and size of the object in the image.
Broadly, the SIFT algorithm can be divided into the following steps:

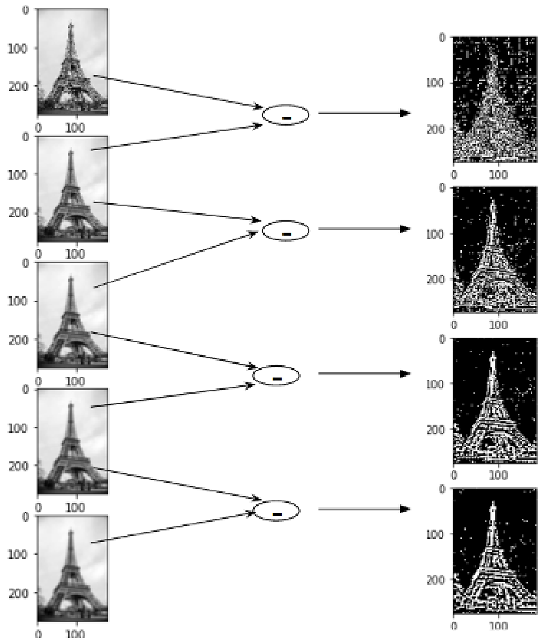
1. Scale-space peak selection: Finding the potential location for features. This is done by
    - Converting the image into grayscale.
    - Creating multiple scaled down versions of this image, say four images.



- Applying different sized gaussian blurs to each of the scaled images, say five images each.



- Obtaining the Difference of Gaussian (DoG) for each set of the scaled images by subtracting every image from the previous image in the same scale.



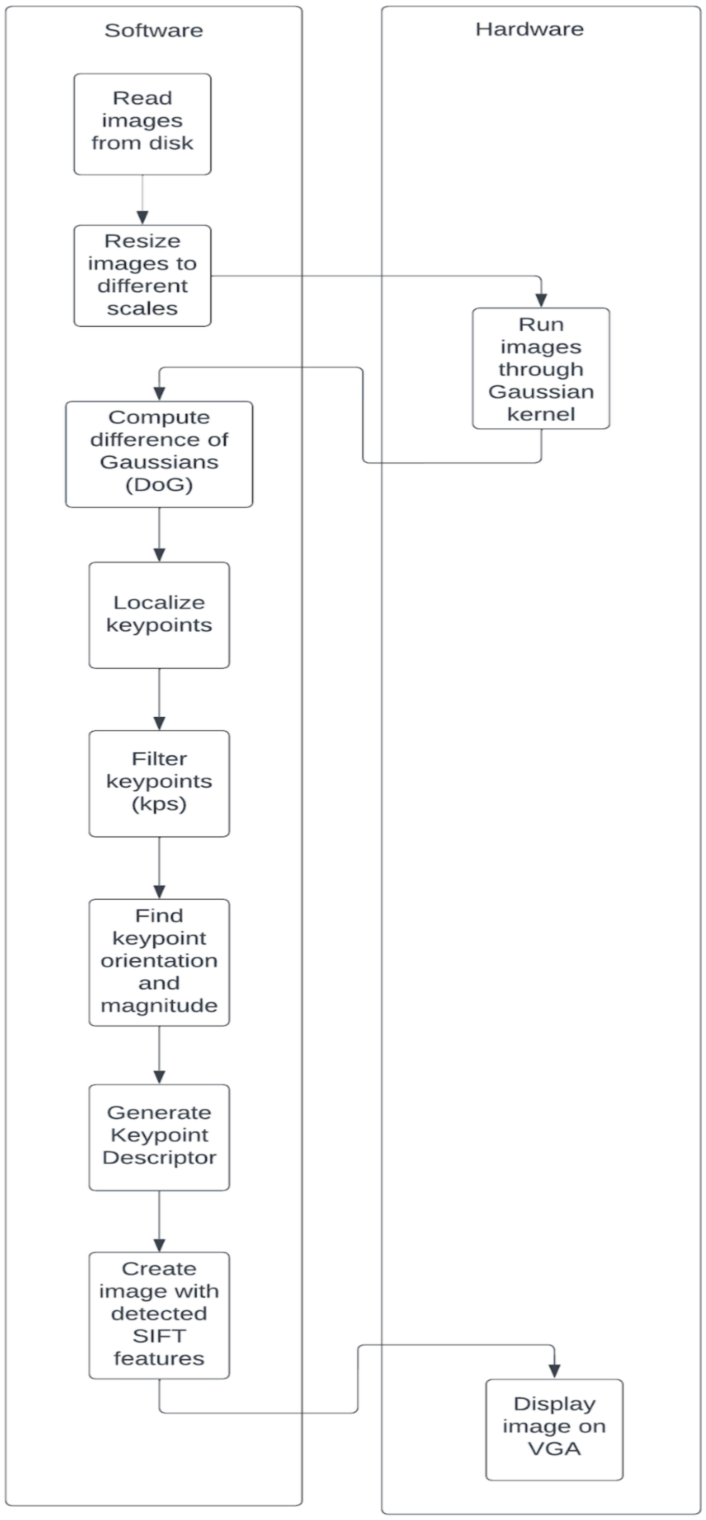
- Extracting these images to further find important keypoints.

2. Keypoint Localization: Accurately locating the feature keypoints.

3. Orientation Assignment: Assigning orientation to keypoints.
4. Keypoint descriptor: Describing the key points as a high dimensional vector.

The gaussian blur mentioned in step 1 is planned to be accelerated by the FPGA. The rest of the steps will be implemented in C/C++ and are planned to be processed in the HPS.

2. System Overview


Software
Hardware
Read images from disk
Resize images to different scales
Run images through Gaussian kernel
Compute difference of Gaussians (DoG)
Localize keypoints
Filter keypoints (kps)
Find keypoint orientation and magnitude
Generate Keypoint Descriptor
Create image with detected SIFT features
Display image on VGA

Software:

Software will be responsible for most of the algorithmic steps in the SIFT algorithm except for the most computationally expensive step. Software will resize the image before passing it to hardware in order to compute the Gaussian blur. After receiving the result from hardware it will proceed with the remaining steps of the SIFT algorithm and calculate the final SIFT keypoints.

Hardware:

The hardware will be responsible for doing the Gaussian kernel convolutions that blur the image and are computationally expensive. Depending on how blurred the image must be, there will be different standard deviations used in the equation $G(x, y, \sigma) = \frac{1}{2\pi\sigma} e^{-\frac{(x^2+y^2)}{\lambda\sigma^2}}$ used to create the Gaussian kernels. These kernels are matrices of different sizes, representing a weighted average of the surrounding pixels for each pixel in the image. A kernel is convolved with the photo, creating a new blurred image. Since these kernels will be reused for all images passed through the algorithm, we will calculate the values for these and hardcode them into the hardware memory.

A sample code written in python that performs the Gaussian blur over different kernel sizes is as follows:

```
def DoG_fixed_kernel_size(image, num_blurs=5):
    assert num_blurs > 3
    out_images = []
    kernels = [5 + 4*i for i in range(num_blurs)]
    sigma = math.sqrt(2)
    for i in range(num_blurs):
        sigma = math.sqrt(2)**(i+1)
        img = cv2.GaussianBlur(image, ksize=(kernels[i], kernels[i]), sigmaX=sigma, sigmaY=sigma)
        out_images.append(img)

    diffs = []
    for i in range(len(out_images)-1):
        diff = out_images[i+1] - out_images[i] # DoG is done here
        diffs.append(diff)

    return out_images, diffs
```

Before performing any convolutions, the image data will be passed through the Avalon bus via the writedata and address fields. Reading the image data from hardware memory will be done in a similar manner. This simplifies the passing of image data without having to use more complicated memory protocols like Direct Memory Access or SDRAM Controllers. The software will also specify which Gaussian kernel to use for the convolution, since they will be stored in the hardware memory.

The final image with the overlaid SIFT keypoints will be displayed on the VGA screen and will be implemented in hardware based on lab-3.

Hardware Memory Constraints: There are only 4,450 Kilobits (~570,000 bytes) of on-chip embedded memory, and we plan to use most of it. We plan on using

1. Two 600*400 grayscale (8 bpp) images, one for both input and output. A total of 480,000 bytes

2. Gaussian kernels can be hard coded directly in memory (they are always the same). This requires 8 * (21*21 + 17*17 + 13*13 + 9*9 + 5*5) bytes, a total of 8,000.

We must test if two images and the Gaussian kernels can all fit in chip memory without anything running out of memory. We can always lower the resolution of the input image if this is the case.

3. Milestones

1. Loading image data to and from the board from software
2. Implementing Gaussian convolution in systemVerilog
3. Writing the other steps of the SIFT algorithm in software
4. Constructing the output image overlayed with detected SIFT keypoints
5. Displaying the image to the VGA board.

4. References

1. https://medium.com/data-breach/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40
2. https://www.sciencedirect.com/science/article/pii/S0141933115001921?via%3Dihub
3. https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf
4. FPGA Based Parallel Hardware Architecture for SIFT Algorithm
5. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7784039&tag=1