Tutorial on Client-Server Communications

EE368/CS232 Digital Image Processing, Winter 2019

Introduction

In this tutorial, we will learn how to set up client-server communication for running an image processing application on a server from an Android device. Since many image processing algorithms require high complexity cost, running these algorithms on a mobile client with responsive interactions is often infeasible. One remedy to this problem is to offload the work to a high-performance server over the network.

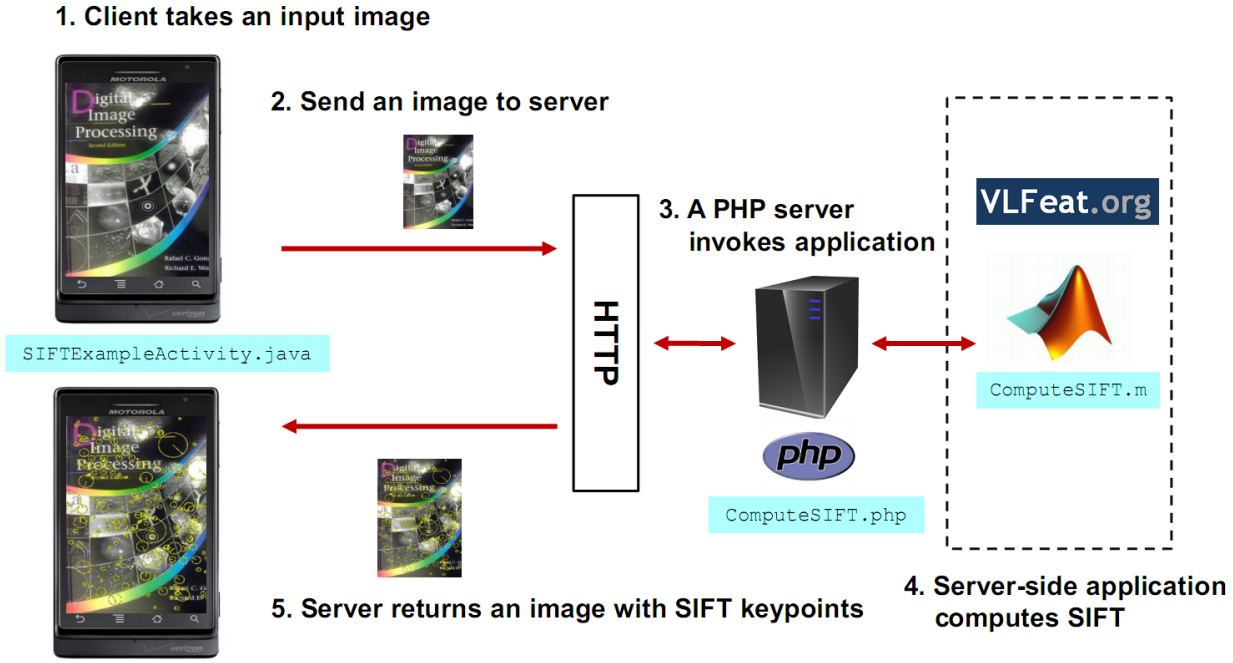

Figure 1 An example of a client-server image processing system

As an illustrating example, suppose you would like to offload the computation of SIFT[1] from an Android device to a server in your class project. Figure 1 depicts a possible scheme for a client-server image processing system. The user will first capture an input image using an Android client and send the image to a server via HTTP. A PHP script on the server then invokes the server-side application to compute SIFT on the image. After the computation is completed, the server will send the result back to the Android device for display.

[1] SIFT: Scale-invariant feature transform. For reference, see EE368 *"Scale-Space Image Processing"* and *"Feature-based Methods for Image Matching"* lecture slides or Lowe, David G. (2004) *"Distinctive Image Features from Scale-Invariant Keypoints"*.

We will be using the above example throughout our tutorial. The tutorial is divided into two parts. The first part of the tutorial will focus on how to set up the server application. We will implement a PHP script to facilitate client-server communication and build a server-side application to compute SIFT using Matlab and VLFeat library. In the second part, we will explain how to handle client-server communication on an Android client.

Disclaimer: This tutorial is for educational purpose only. It is intended for students who would like to build a fast-working prototype for their class projects. Please be aware that the tutorial does not consider practical system issues, such as security and reliability, in a real-world setting.

Part I: Server

Server Setup

This tutorial assumes you have a web server with HTTP software (Apache, IIS, etc) and PHP installed. Otherwise, please refer to the following links on how to setup a web server:

http://www.wampserver.com/en/	(Windows)
https://www.mamp.info/en/	(Mac)
https://help.ubuntu.com/community/ApacheMySQLPHP	(Linux/Ubuntu)

Download Server Source Code

Download the server source code of the tutorial from the EE368 git repository:
https://github.com/ee368/EE368-Android-Samples

Copy the directory `Tutorial3/ServerCode` to your web hosting directory. Example: `/var/www/` or `/home/<username>/public_html` in Linux/Ubuntu.

VLFeat Library

In this tutorial, we will use VLFeat library (http://www.vlfeat.org) to compute SIFT of an image on the server. VLFeat is an open-source software written in C with interface to Matlab for ease of use and experimentation. The library implements many common computer vision algorithms including SIFT, MSER and k-means. Some of you may find this library very useful for your class project.

To use VLFeat, download and unpack the binary package from:
http://www.vlfeat.org/download.html

Alternatively, you can use VLFeat package (`./vlfeat-0.9.14`) that is already provided in the repository.

To use and interface VLFeat with Matlab, add this line to your `startup.m` file:

```
run('<path_to_your_VLFeat_Library>/toolbox/vl_setup')
```

Please refer to this link for creating or modifying `startup.m`:
http://www.mathworks.com/help/matlab/ref/startup.html

Matlab Script : computeSIFT.m

Inside the tutorial files, we have already written a Matlab script, `computeSIFT.m`, to compute SIFT of an image for you. It takes an input image from the `upload` folder and converts it to a grayscale image with single precision:

```
InputImg = imread(input_img_path) ;
GrayImg = single(rgb2gray(InputImg)) ;
```

After obtaining a grayscale image, we simply use `vl_sift` from VLFeat to compute SIFT,

```
[f,d] = vl_sift(GrayImg) ;
```

- Each column of `f` is a SIFT keypoint with format [X; Y; S; TH], X,Y is the center of the keypoint, S is the scale and TH is the orientation (in radians).
- Each column of `d` is the 128-dimensional SIFT descriptor.

After SIFT is computed, we plot the relevant keypoints using `vl_plotframe` on the colored image and save the result to the `output` folder.

Matlab Script : computeSIFTLoop.m

Please run `computeSIFTLoop.m` in Matlab. You can run it without UI from the command line by using the following command:

```
matlab -nodesktop -nodisplay -r "run('computeSIFTLoop.m')"
```

This Matlab script runs a persistent while-loop. Twice a second, it looks for the existence of an indicator file called `image_ready` (generated by the PHP file that we will describe subsequently) that signals a new query image has been uploaded to the server. If this indicator file is found, then Matlab calls `computeSIFT` on the newly uploaded query image.

After `computeSIFT` is finished, this Matlab script saves another indicator file called `result_ready`, which signals to the PHP file that the processed image has been generated. The PHP file will wait until result_ready appears before it tries to send the processed image to the mobile device.

Effectively, Matlab and PHP give handshakes to each other via the two indicator files `image_ready` and `result_ready` during every new query. PHP never has to call Matlab explicitly. Since Matlab is always running, although sleeping most of the time when a query is not present, the 2-3 second delay of starting a new Matlab instance for every new query can also be avoided by this approach.

PHP script : computeSIFTLoop.php

A PHP script called `computeSIFTLoop.php` is also included in the tutorial files. Please have a quick look. This PHP script allows a client to upload a captured image and returns the difference-of-Gaussian (DoG) keypoints of the image from the computation of SIFT. Below is a code snippet from `computeSIFTLoop.php` for taking an input image uploaded from a client:

```
#declare target path for storing photo uploads on the server
$photo_upload_path = "./upload/";
```

```
$photo_upload_path = $photo_upload_path. basename(
$_FILES['uploadedfile']['name']);

# copy temporary upload file to target path that stores the photo upload
if(copy($_FILES['uploadedfile']['tmp_name'], $photo_upload_path)) {
 #perform something on the image ….
}
```

On the client side, we will post an image upload to the server by connecting to the PHP script `computeSIFTLoop.php`. The image file is uploaded to a temporary storage area on the server. To save our uploaded file, we need to refer to the associative array `$_FILES`, which stores all the information about the file posted. There are three elements of this array you should know for our purpose:

- `uploadedfile` - the reference we will assign when we post a file from a client. We will need this to tell the `$_FILES` array which file we want to process.
- `$_FILES['uploadedfile']['name']` - `name` contains the original path of the client's uploaded file.
- `$_FILES['uploadedfile']['tmp_name']` - `tmp_name` contains the path to the temporary file that resides on the server.

After the temporary file path is known, we can copy the uploaded file to our `upload` folder for processing. To invoke our Matlab script `computeSIFT.m` in PHP, we signal that the image is ready to processing via an indicator file:

```
$handle = fopen($photo_upload_indicator_path, 'w');
fprintf($handle, '%s', $photo_upload_path);
fclose($handle);
```

After the image file has been processed by Matlab, the PHP file will receive a signal via another indicator file.

```
while (!file_exists($processed_photo_output_indicator_path))
{
      usleep(1000000);
}
usleep(1000000);
unlink($processed_photo_output_indicator_path);
```

After the Matlab script is executed, `computeSIFTLoop.php` needs to automatically push the result back to the Android client. We have implemented a PHP function called `streamFile` (see line #9) to do this for you easily:

```
streamFile( $location, $filename, $mimeType)
```

- `$location` is the file path of the source file.
- `$filename` is the filename that the client will use to save file.
- `$mimeType` is the character set type. You may use `'application/octet-stream'` as the MIME type in this example.

Testing Server Code

To test your server code, we provide `test.html` in the tutorial files. You can load `test.html` using an Internet browser and upload a test image in JPEG format. In the default configuration, you can test the script `computeSIFT.php` but you can change `computeSIFT.php` to `computeSIFTLoop.php` in `test.html` if you want to test the latter. If your server scripts are setup successfully, an output image with SIFT keypoints annotation will be downloaded to your browser within a few seconds. Figure 2 shows an example of a test image and its corresponding result.

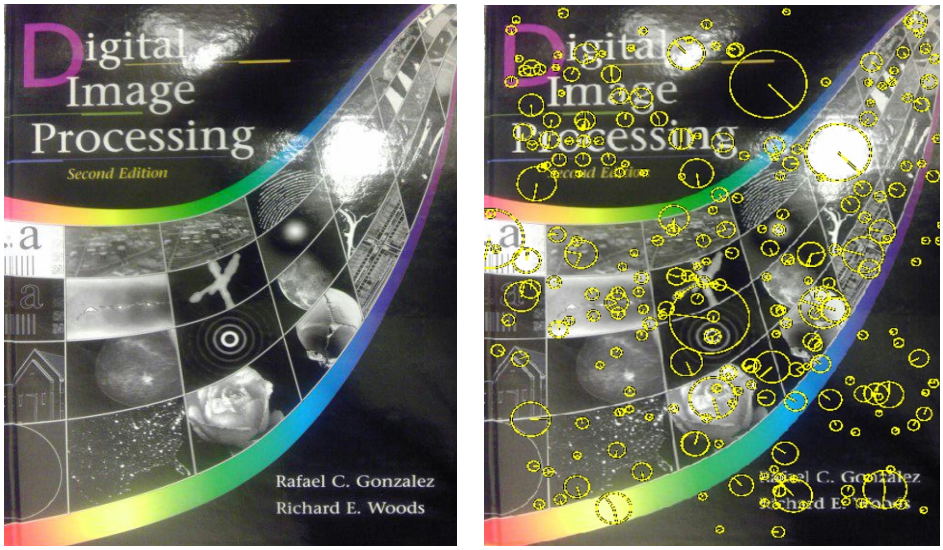

Figure 2 (Left) Input test image (Right) Output result with SIFT keypoints

When Matlab is not available

If you do not have Matlab available on your server, you can easily refer to the VLFeat library and write your own C program for computing SIFT. Instead of invoking Matlab command, you can replace the `$command` (line #58) with your binary execution command in `computeSIFT.php`:

```
$command = "<your_program_path>/<your_program_name> <arguments>"
```

Part II: Android Client

Download Client Source Code

The client source code is located in `Tutorial3/ClientCode` on the git repository.

Setting up Android Client Application

Open the project in Android Studio.

Go to `SIFTExampleActivity.java` and enter your server URL of the relevant PHP script (`computeSIFT.php` or `computeSIFTLoop.php` depending if you use the version with the loop or not) by editing line #51:

```
private final String SERVERURL = "http://<server-path>/computeSIFT.php";
```

Run the application on your Android device. Please make sure your phone is already connected to the Internet. A camera preview will be displayed. Take an image snapshot by pressing the volume up button. The image should then be automatically uploaded to your server. If everything is setup correctly, an output image with SIFT keypoints should be displayed on the screen after a few seconds. To take another image, you can press the same button again.

Congratulations! You have successfully built your own client-server image processing system.

Handling of Client-Sever Communication on Android Client

In the Android project, we have three main source files:

- `SIFTExampleActivity.java`: Main activity file. Handles UI and client-server communication.
- `ResultView.java`: A surface view to draw your output image result.
- `Preveiew.java`: A surface view that displays camera preview. Same as what you have seen in Tutorial #1.

Often times, it is a good programing practice to have server-client communication not interfering with the flow of our main program thread. This helps keep our application responsive at all time. Therefore, one solution to this problem is to have an asynchronous task to manage our server-client communication. To implement an asynchronous task, please refer to the `AsyncTask` API from the Android SDK: https://developer.android.com/reference/android/os/AsyncTask.html. You can also refer to `SIFTExampleActivity.java`, line #180 to #340, for the actual implementation in the tutorial.

To offload our image processing to the server, we implemented a function called `processImage` (see `SIFTExampleActivity.java`, line #277 to #299) that abstracts all handling of server-client communication for you. Below is a code snippet of `processImage`:

```
void processImage (String inputImageFilePath){

        File inputFile = new File(inputImageFilePath);
        try {
                //<1> create file stream for input image
                FileInputStream fileInputStream  = new FileInputStream(inputFile);

                //<2> upload photo
                final HttpURLConnection  conn = uploadPhoto(fileInputStream);

                //<3> get processed photo from server
                if (conn != null){
                        getResultImage(conn);
                }
                fileInputStream.close();

        }
        catch (FileNotFoundException ex){Log.e(TAG, ex.toString()); }
        catch (IOException ex){Log.e(TAG, ex.toString());
        }
}
```

In `processImage`, `uploadPhoto` (see `SIFTExampleActivity.java`, line #191- #257) will help you establish a HTTP connection to your URL and upload your photo. When the HTTP connection is successful, the server will reply your HTTP request with the result data. Using `getResultImage(conn)` (see `SIFTExampleActivity.java`, line #260-273), we can download and display the data. Please take a look at these functions for their implementations. You may want to modify them for your project.

Reference

1. PHP - File Upload. http://www.tizag.com/phpT/fileupload.php
2. Post a File from the Phone to a PHP Server. http://getablogger.blogspot.com/2008/01/android-how-to-post-file-to-php-server.html
3. VLFeat. http://www.vlfeat.org/
4. Android AsyncTask. https://developer.android.com/reference/android/os/AsyncTask.html
5. Android HttpURLConnection. https://developer.android.com/reference/java/net/HttpURLConnection.html
6. EE368 *"Scale-Space Features"* and *"Image Matching"* lecture slides.
7. Lowe, David G. (2004) *"Distinctive Image Features from Scale-Invariant Keypoints"*.