Computer Vision Emergency Response Toolkit

Paul Faugeras

Developers' guide

Contents

I
Introduction

1 Idea

The main idea of this software is to provide a good-looking user-friendly, yet quite powerful app that doesn't need any installed framework on the user's machine.

I wanted something that was good-looking, could work on ALL Operating Systems, and that the user could install and launch in a simple double-click.

Because some algorithms were already written in Python, I decided to keep that technology as a back-end, but chose a more web-looking framework for the front-end : Angular. That is because I find that there isn't much better-looking and diverse user interfaces than on the web, and I already know how to code in Typescript (and I know nothing of C#)...

2 Architecture

The architecture is quite simple :

On startup, the Angular app launches a Python server as a subprocess on port 5000, and sends it HTTP requests with all the infos needed every time it needs to use one of its algorithms. It then waits for the response from the server to update its UI.

II
Back-end

1 Python server

The Back-end is handled by a Python 3.6 server ran on Flask on port 5000.

The entry file is server.py, that launches the server. That server handles the incoming HTTP POST requests. All other requests are rejected. The POST request should contain a JSON data containing a 'targetPath' field.

If that field's value is 'response', then the sourcePath should be a base64 image, and the server should apply the algorithmto that base64 image data, and send a base64 image as an answer. This is typically the case for computer vision algorithms, as they need to be displayed as a URI on the <img> tag of the Angular front-end app (see below). For that, we call the 'handleFilter' method of the 'handlePost' module (see below).

If the 'targetPath' field, on the other hand, is not 'response', then we will be looking at a GPS fetch request, with GIS data in the JSON. It will not request base64 image data as a response, but asks for saving the processed files in the destination folder. In this case, we will use the 'handleGPSfetch' method of the 'handlePost' module (see below).

2 HandlePost

The HandlePost module has four functions.

First, 'handleFilter' will take as arguments the algorithm (in JSON shape, see assets.json), the sourcePath (as base64 string image), and the parameters (as JSON, from Angular Server parameters). It will dynamically import the right library from the algorithm JSON, then convert the base64 image into a cv2 image object using 'data_uri_to_cv2_img', apply the corresponding algorithm to it, with the input parameters, reconvert the resulting image into a base64 string, and return the results as JSON, to be sennt back by the server.

The second function, 'handleGPSfetch' will apply the GPSfetch algorithm to the sourcePath field of the JSON request (which is actually a list of paths to input files).

Once the algorithm finished, it will send back a dummy JSON just to warn that it has finished.

3 Algorithms

3.1 Image processing

Those are the algorithms (same as in the original project) that take into argument a cv2 image, and the server parameters, apply the image processing algorithm, then returns the result image, the process time and the success percentage, if relevant (0 otherwise).

Currently, there are three such algorithms : RXDetector, DXDetector and Dehaze (AODnet).

3.2 GPS fetch

This is a new function that I developed myself.
It takes as arguments a sourcePath (a list of image paths), a target path (a destination directory), and a gpsTarget JSON item (lat, lon, and fov) for the target GPS point.

Then, using XMP-exif data extraction in the source files, then vectorial computation, it finds all the images of which the field of view intersects with the GPS target point. If suc a point is found, it draws a red circle at its location on the image, and saves the image in the destination folder.

If no intersection is found, or if the XMP-exif data cannot be extracted from the image, it is simply ignored.

4 Developing

4.1 Install and manual run

First, you need to install Python 3.6 or higher on your computer.

`pip3` is normally installed with Python3 (otherwise, you can install it manually). After that, all you need to do is install the required libraries by running the following command, in the python-server directory :

```
pip3 install -r requirements.txt
```

Once that is done, you can launch the server simply by running the following command (in the same directory) :

```
python3 server.py
```

4.2 Adding an new algorithm

Fist, create a python file in the Algorithms directory.

In that file, you are free to put all the functions you like. Your main function, however (the one that `handlePost` will execute, needs to take as argument: (1) a cv2 image, (2) a parameters JSON. It can access the parameters using the notation `parameters['key']`, with 'key' being the name of the parameter. Then, after processing the image, it must return : (1) the cv2 resulting image, (2) the time elapsed (you can use the timer functions, see other algorithms for examples), (3) the accuracy percentage (0 if not relevant).

For example, in `Algorithms/newAlgorithm.py` :

```
import timer
import package1
...

def myAlgorithm(cv2_image, parameters):
        t = timer.Timer()
        t.start()
        myParameter = parameters['myParameterKey']
        ...
        result_image = do_something_with(cv2_image, myParameter)
        percentage = compare_images(cv2_image, result_image)
        ...
        t.stop()
        return(result_image, t.get-time(), percentage)
```

If you need a specfic package for your algorithm (above, for example, `package1`), add a line in `requirements.txt`, so that the next user to download the git repo will know what to install.

Now, we need to update `CVERT-ng/src/app/assets/assets.json` in the `serverFilters`

list : you need to add (1) the name that will be displayed on the front end, (2) the name of the library/module, (3) the name of the main Python function within the module, (4) the arguments (here, leave it as is).For the above example (please note the file name), you will add the following element :

```
{
    "name": "My Awesome Algorithm",
    "libName": "newAlgorithm",
    "pyFunction": "myAlgorithm",
    "args": [{"type": "button"}]
},
```

Then, in `CVERT-ng/src/app/assets/algorithmParameters.json`, if you need a custom parameter (in the above, `myParameterKey`, that we'd like to set at default value 8, for example), we should add it to the `algorithmParameters` array for the Angular app to prompt the user when he tunes the server parameters. For that, we need to set (1) the parameter name, (2) its default value. Leave the other fields as they are, they are used by the front end :

```
{
    "name": "myParameterKey",
    "useDefault": true,
    "defaultValue": 8,
    "userValue": 8
},
```

Finally, in order for the compiler (PyInstaller) to find your algorithm, and include its assets if your algorithm needs a specific file during its runtime (for example, the AOD-net algorithm needs the pretrained network (.npy) file located in the `Algorithm` directory. For that, we will need to add the algorithm and the asset in the `server.spec` file, like so :

```
imports = [
    "Algorithms.DXDetector",
    "Algorithms.RXDetector",
    "Algorithms.AODNet",
    "Algorithms.newAlgorithm"
]
```

```
extraFiles = [
    "pretrained_aod_net_numpy.npy",
    "your_resource_file.whatever"
]
```

(Make sure that your resource file is in the Algorithm directory : that's where the spec file will look for it...)

Now you should see your algorithm and settings appear in the software after compiling, yipee! :)

III
Front-end

The front-end is handled by an Angular App, encapsulated in an Electron instance, communicating with the server via HTTP.

1 Angular

Angular is a Typescript framework, made for designing good-looking, yet efficient user interfaces. It uses components and services to interact between objects, and uses Node Package Manager to import and manage external free-to-use libraries.

1.1 Libraries

As for any npm projects, all the used libraries can be found in `package.json`. You can find the main ones below.

1.1.1 deployed

In the deployed app, we use two UI libraries : Bootstrap (for nice-looking HTML elements, without messing too much with CSS), and Angular-Material (for theming and useful widgets).

For the histograms, we use Chart.js, which builds amazing-looking charts in no time, and for the GIS grid and markers, Three.js, which gives you great optimized browser tools for 3D-rendering on HTML canvases. We also use usng.js for converting GPS coordinates into USNG.

In order to extract XMP-exif, we use exifr and fast-xml-parser. We've tried multiple extraction tools, but found out that only the combination of those two gave us the desired results.

Finally, all of the front-end filters are handled with the (amazing) library : Jimp!

1.1.2 development

Since we are developing on Typescript using Electron, we need both the Typescript and the Electron dependencies.

As we would like to compile the app into a single executable file, we also use the

Electron-builder package, which we felt like it was the best suited for our needs, in terms of ease of use, and effectiveness (more on that below).

1.2 Components

For each of the UI tabs, we created a single component. Then, in the App component, we added a router outlet, so to be able to navigate to an other page for the settings page for the server (that's a little bit tricky, but needed when using Electron.).

only the images and the histograms use the same component for both the top and the bottom one, since they have the same behaviour. Yu just need to bind the right object to each of them in the main app component.

1.3 Services

There are four services, which are all provided in the root app component (for singleton generation).

- **Canvas :** this is the one used to display the grid on the bottom image. It has all the `Three.js` logic (generation, update, display) and mainly interacts with the GIS data, for visualisation on the overlayed canvas.
- **file :** This is the one that interacts with the Electron IPC (see below), to create a bridge with the file system (which you cannot access within Angular). It is mainly used when opening and saving files, selecting the input files and the output directory. It is also used when opening a new Electron window, for setting the server filter parameters, for example.
- **GIS :** This is the service that updates the marker position (GPS and USNG) when the user clicks on a location of the image.
- **Server :** The main service to communicate with the sever through HTTP. It handles the requests, the response, and plays with the Promises and observables.

2 Electron and IPC

Electron is a framework that enables you to embed your web apps into a desktop app. Here, the main entry to the app is the `main.ts` file, which is located in the Electron directory. This one creates a window, then loads the Angular main file (in the dist folder once converted into JS), and adds listeners (IPC for Inter-Process Communication) to interact with the file system (read, write files, extract XMP-Exif etc.), to

open new windows etc.

In the next section, you will also see that it is the piece of code that launches the Python server on startup (because it is the entry point of the app).

3 Front/Back communication

The Angular app, through the sevrer service and HTTP requests, communicates with the Python back-end on port 5000.
The address subsequently, is `http://127.0.0.1:5000`.

When the app launches (through Electron), the main.ts file first checks if the app is in its compiled form (with Electron-builder, see below), or still in development mode (through npm).

If in compiled form, it will look for the executable server file in the resources folder (decompressed in the temporary memory at runtime), and launch it as a child process with `child_process.execFile()`.

If it is development mode, it will, on the contrary, look for the python file : `server.py`, and execute it through a child process command line : `python3 server.py`. In this case, you will have to have installed Python 3.6 and the required libraries (in `requirements.txt`) needed to run the server and its image processing algorithms.

If you want to run your own server, but don't want to put Electron in development mode (use the single executable), you can modify the `server.py` file to serve on an other port (for example, 6000), and set the server port on the front-end to that port.

In this case, you will have 2 servers running (one, through child_process, on port 5000, and one that you launched manually on port 6000), but only your own server will be used. (You are basically redirecting the HTTP requests to your own server).

4 Developing

First, you need to install nodejs and npm (node Package Manager). Usually, if you install nodejs, npm is installed automatically.

Then, you will need to install the Angular-cli globally :

```
npm install -g @angular/cli
```

After you have cloned the git repo, navigate to `CVERT-ng` directory, and install all of the necessary libraries :

```
npm install
```

Then, you will normally be able to run the app with Electron through the following command (ran in the CVERT-ng directory) :

```
npm run electron
```

IV
Compilation

In order to compile your app into an executable file for your operating system, you will need to compile both the front-end, then the back-end with the following instructions.

1 PyInstaller

Once you have installed Python 3.6 and the required libraries (see above), once you have tested the server manually and it runs smoothly, if you have added an algorithm and followed the instructions in the corresponding section above, you are ready to compile the python server into an executable file !

Navigate to the `python-server` directory, and run the following command :

```
pyinstaller server.spec
```

That should create two directories : `build` and `dist`. Your executable file will be in `dist`. You can try it out by double-clicking on it, but be careful : since it doesn't have any window, you will not be able to close it (you might have to stop it through the task manager...).

Do not move the resulting file : it needs to stayy in the dist folder in order for Electron-builder to find it and include it in the final app.

2 Electron-Builder

Once the server executable has been generated for your OS, you will need to compile the Electron Angular app with Electron-builder. To do so, make sure you have installed all the dependencies, and that your app runs smoothly when launched manually with npm.

In `package.json`, you will see that there is a dictionnary called "build" : that is the one that Electron-builder will use as a reference for all the compilation settings. In there, you will see that the Python dist folder is used as an extraResource, and that we will ask for it to provide files for Linux, Windows and Mac, depending on the machine the command is executed on.

Before compiling your code, you will need to run one final time `npm run electron`, otherwise the last version of the files might not be taken ito account...

Then, you can run the following command (in the CVERT-ng directory) :

```
./node_modules/.bin/electron-builder
```

Your executable file will be located in the `releases` directory. You can test it by simply double-clicking on it.

3 Build scripts

We are working on platform-specific build scripts that will enable you to compile the python server, then the Angular app in a single command.

More on that later.