Capstone Project

26.4.2020

Machine Vision for Brain Analysis

Project Owner: Francisco Lopez

Student Tutor: Oliva David

Group Members: Doncenco Dorin, Faßbender David, Nissi Jonatan,

Jose Roig Sara, Schuurman Joonas, Laanaya Salim

Abstract

This project was conducted in order to limit a time-consuming process that currently must be done manually by researchers. The comparing of hundreds of mice brain images from the lab to their original (healthy) counterparts individually, takes too much time. In order to prevent this, the idea was to create a program that could automatically detect and single out individual brain slices from a laboratory brain scan consisting of over 140 mouse brain slices. The program would then match the corresponding brain area from the laboratory images to the normal mice brain images. During this process also eliminating unusable images such as ones with smudges or imperfections. Then it would analyze and output each brain region with readable data regarding the brain activity.

The current status of the program allows the laboratory images to be blurred, contoured, cropped, and outputted, which by itself could be used as a stand-alone program. The use of a graphical user interface (GUI) has also been implemented to add simplicity to the program. Some other functions that would still need adding are for example: image matching, region recognition, and data analysis. All these concepts will be covered in further detail in this document.

Contents

1. What has been accomplished

1.1 UI (User interface) – **succeeded**

- Create a UI that is simple enough for an average user such as a researcher.
- The idea is to have an option of uploading the image consisting of the brain slices and with minimal options to change some parameters. When the user presses start the program should run.
- Depending on the time taken to run, the program can either be left alone overnight or while the researcher does something else.
- The user should be able to remove pictures
- The user should be able to flag interesting pictures
- The user should be able to rename pictures
- If the data from the pictures is displayed in the UI it should be easily accessible and understandable

1.2 Blurring the image – **succeeded**

- Otsu Thresholding
- Done to remove background noise

1.3 Contouring – **succeeded**

- Canny edge detection
- Remove shapes that are too small to be a brain

1.4 Output – **doable** - cancelled; slices are not similar enough to average them out

To do:

- Sort output
- Arrange the images in order from left to right? Numbering them?

3. Software requirements:

- Distribution: Python 3.7
- Libraries: PyQt5, cv2, numpy
    - pip install –r requirements.txt

    OR
    - pip3 install –r requirements.txt
- Input image format: .tif

2. What still needs to be done

2.1 Image registration

- SimpleElastix
- Tried using matrices
- Tried using numpy.fft
- Tried using imreg_dft
- Idea suggestion: get enough sample data, average out the values of the slices to get the matrix of one slice

2.1.2 Matching images with a homography matrix

Using the python libraries opencv, matplotlib and numpy, an image matching between atlas images and lab images has been tried. The procedure used has the following steps:

1. Upload both images

2. Resize the reference image and rotate the lab image: the bigger size of the Atlas images in comparison with the other one makes the matching really difficult otherwise.

3. Initiate and ORB (Oriented FAST and Rotated BRIEF).

An ORB is an open source algorithm that consists of FAST (features from accelerated and segments test) key points detector and BRIEF (binary robust independent elementary feature) descriptors.

In short, FAST is an intensity-based registration method that looks for the edges of the objects inside the picture. Given a pixel checks the pixels around in a radius specified and classifies them in lighter, darker or similar. If there are more than 8 pixels darker than the center it is chosen as a key point. After that, it assigns a new orientation followed by a couple more steps to make it scale and rotation invariant.

Once all key points are found, the rBrief algorithm is implemented (it is the same as BRIEF algorithm, but it is also rotation-aware). This converts the previous algorithm in an object composed of binary vectors where each key point is described by a vector. A few more mathematical steps are followed, and the descriptors are created. These steps do not need to be specified as they are already implemented in opencv library so they can be created in a few commands.

4. Create key points and descriptors for both images. It is accomplished running twice the points generator function.

5. Match descriptors in both sets with the key points and descriptors as a parameters and sort them. Afterwards we generate two empty nonnegative integers arrays of the length of the matches and proceed to fill them up using a loop with our points to compare.

6. Create an homography matrix.

A homography matrix is the relation between two images in a same plane. In other words, it matches the corresponding 2D points of one image with the points of the other one. The product of the homography matrix with one set of points equals to the corresponding set of points in the other picture. This algorithm works nicely when both images are from the same object in different angles, however it presents some issues when comparing different pictures with similarities, as it is our case.

Even though is mathematically complex it is done quite straight forward with opencv.

6. Finally, proceed to use this matrix to compare both images with a specified number of matchings as a parameter, such as 100 for example (more matchings may be needed).

The following figure is the output accomplished. In this case the number of matches is set to 50 to be able to see them separately. As shown in the image, the program finds some points, but it is unable to match it with the reference with an acceptable accuracy.

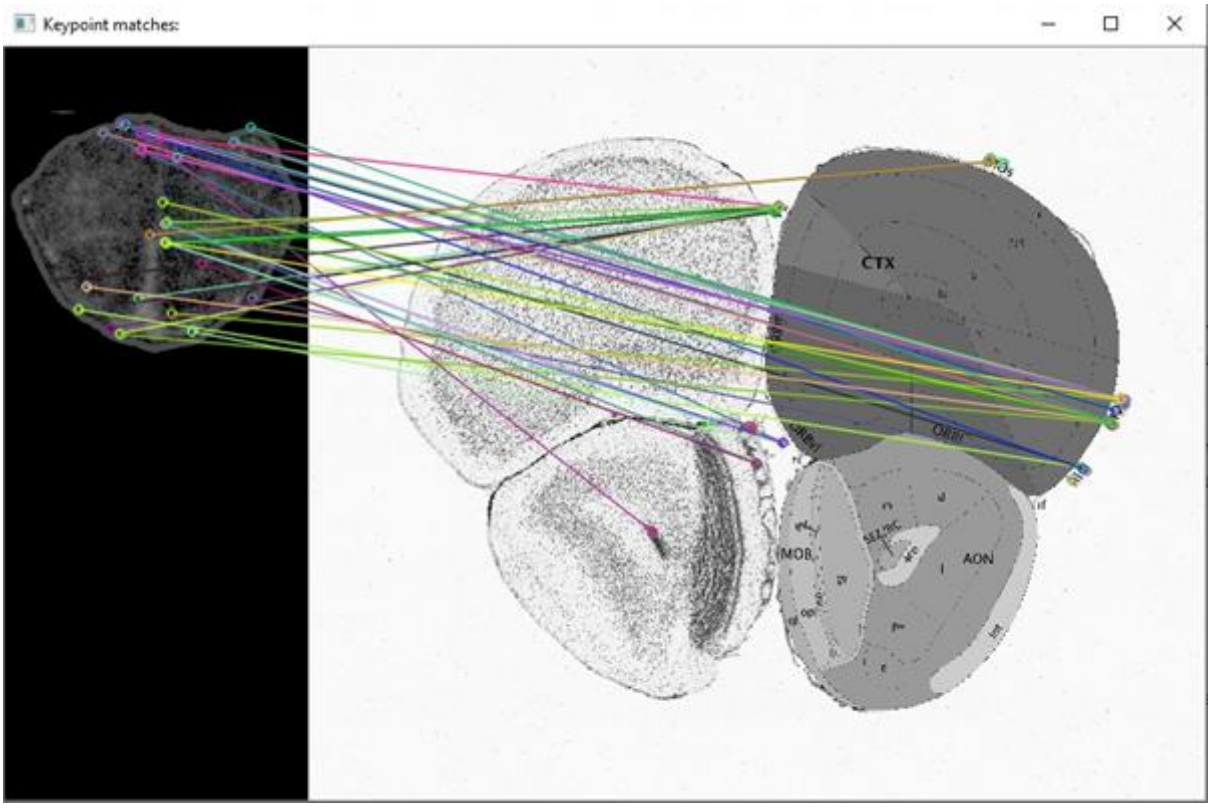

Figure 1: Output of the image registration by homography matrix

The implemented code for this procedure can be found in Annex I.

2.2 MATLAB – Registration Estimator

In the following method the idea was to use MATLAB's Registration Estimator tool to find/detect points from two brain images. This could then be further developed to possible match the raw brain images from the lab to the Atlas. Everything presented is only pure speculation as to what could be done.

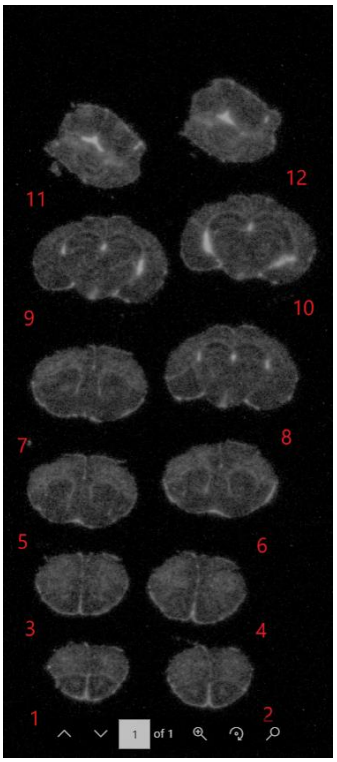

Figure 2.1

By using the registration estimator in MATLAB some form of detection is determined when using two of the exact same lab images.

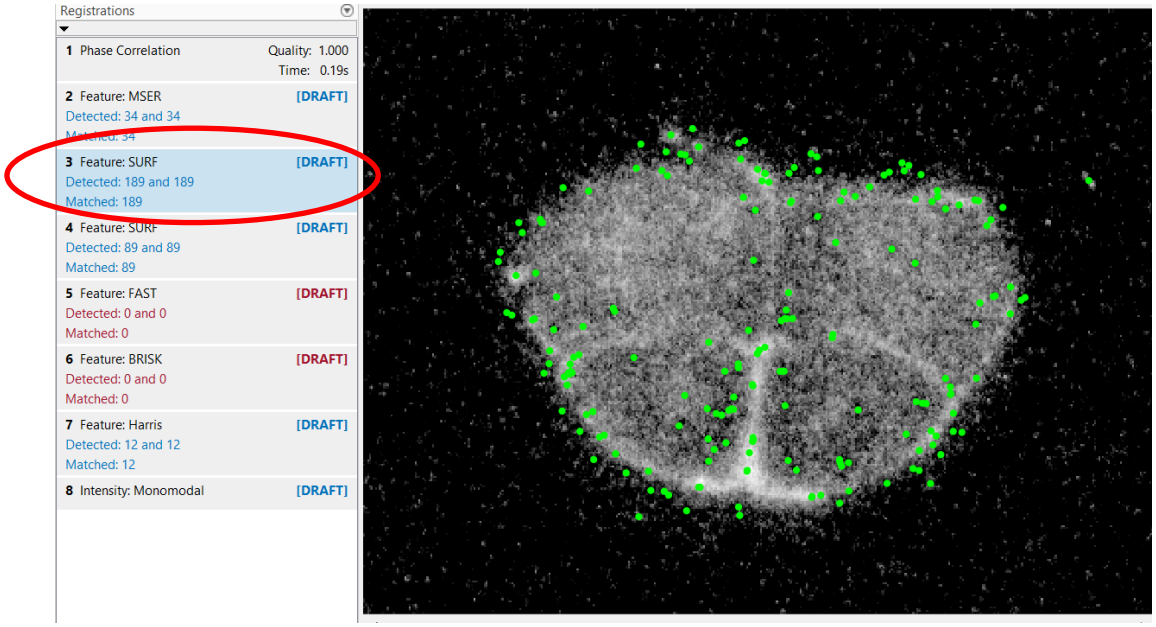

Figure 2.2 Comparing image 1 with itself from figure 2.1 using MATLAB.

When using the exact same brain images as seen in figure 2.2 the program has detected 189 points where all 189 have been matched. Showing that the images are possible to match when using another image that is alike from the original. Having tried this, the use of unlike images will result in too many unmatched points. If this method were to be used one must have images that are quite similar.

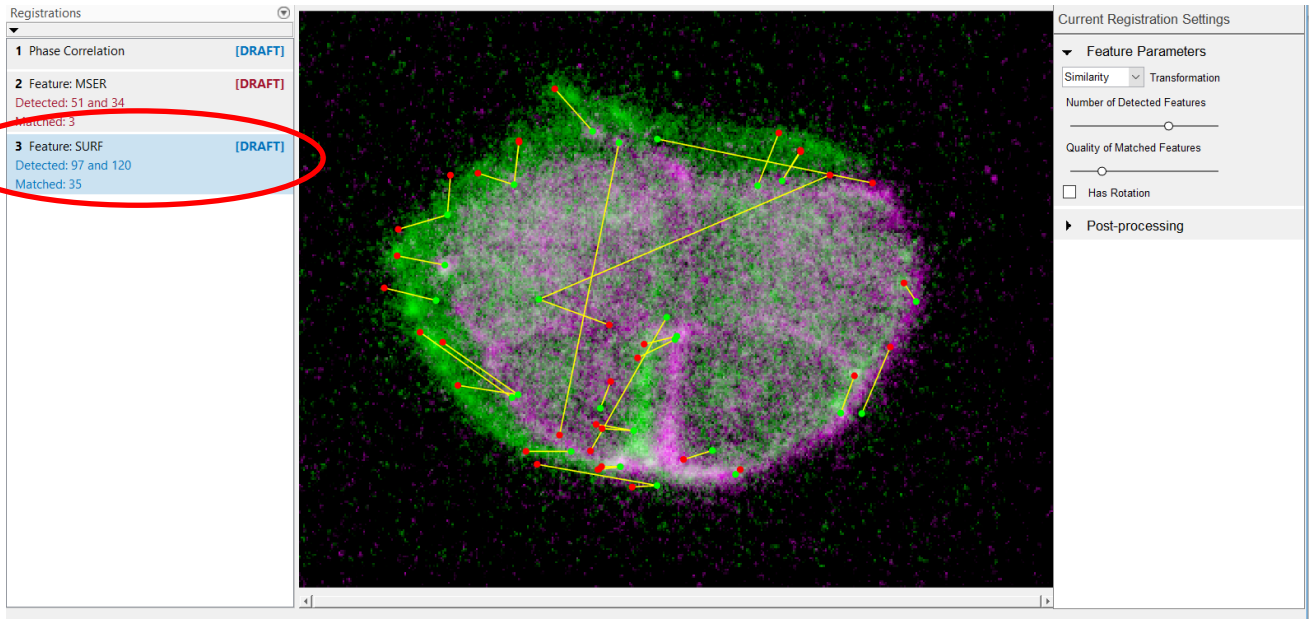

Figure 2.3 Comparing images 1 and 2 from figure 2.1 using MATLAB.

In this scenario images 1 and 2 from *figure 2.1* where compared with one another. Depending on the adjustments made in the registration settings, more matches were found. By adjusting higher, the 'Number of Detected Features' one can find more matches. Although, with the current settings there were 97 and 120 detected where 35 points were matched. It can also be seen that the highlighted green area is the difference in the two images, which means there is detection occurring. Some points were a bit off, with more time and dedication, the matching of the brain images could be done.

Some things to try:

- MATLAB – Some form of object detection
  For spotting defects in the laboratory images, one could use object detection on Matlab. For example: https://www.mathworks.com/discovery/object-detection.html
- MATLAB – Image processing toolbox
  https://www.mathworks.com/help/images/getting-started-with-image-processing-toolbox.html?category=getting-started-with-image-processing-toolbox&s_tid=CRUX_gn_documentation_getting-started-with-image-processing-toolbox

2.3 Region "recognition"

To do:

- Write code to recognize the main areas of the brain
- Match the areas with the Atlas(?)

2.4 Data analysis

To do:

- Figure out the method for getting the data
- How to output the data in an understandable way
- Where to output the data? UI? Excel

Sources:

https://de.mathworks.com/help/vision/examples/object-detection-in-a-cluttered-scene-using-point-feature-matching.html

https://se.mathworks.com/help/vision/ref/matchfeatures.html

https://www.mathworks.com/discovery/image-registration.html

https://mouse.brain-map.org/static/atlas Allen brain atlas Data portal

Notes: build GUI

From initial image to .tif

=> rest

Notes: get full-shape image from atlas

Link from PO: https://www.frontiersin.org/articles/10.3389/fninf.2018.00093/full

Annex I: Code for image registration using ORB and homography matrix

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
im1 = cv2.imread('image1.tif') #Image to be registered
im2 = cv2.imread('image2.tif') #Reference image

img1 = cv2.cvtColor(im1, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(im2, cv2.COLOR_BGR2GRAY)

#Resize reference image
resizedImg2 = cv2.resize(img2,None,fx=0.1,fy=0.1,interpolation=cv2.INTER_CUBIC)

#Rotate lab image
(h1, w1) = img1.shape[:2]
# calculate the center of the image
center = (w1 / 2, h1 / 2)
#Arguments are centre, degree and scale
M = cv2.getRotationMatrix2D(center, 90, 0.8)
rotateImg1 = cv2.warpAffine(img1, M, (w1, h1))

#Initiate ORB
orb = cv2.ORB_create(50)

#Create keypoints and descriptors for both images
kp1, des1 = orb.detectAndCompute(rotateImg1, None)
kp2, des2 = orb.detectAndCompute(resizedImg2, None)

matcher = cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_BRUTEFORCE_HAMMING)

#Match descriptors
matches = matcher.match(des1, des2, None)
matches = sorted(matches, key=lambda x: x.distance)

points1 = np.zeros((len(matches), 2), dtype=np.float32)
points2 = np.zeros((len(matches), 2), dtype=np.float32)

for i, match in enumerate(matches):
    points1[i, :] = kp1[match.queryIdx].pt
    points2[i, :] = kp2[match.trainIdx].pt

#Create homography matrix h
h, mask = cv2.findHomography(points1, points2, cv2.RANSAC)
print(h)

#Use homography
height, width = resizedImg2.shape
print("size image 2: ", resizedImg2.shape)
print("size image 1: ", img1.shape)

im1Reg = cv2.warpPerspective(im1, h, (width, height))
img3 = cv2.drawMatches(rotateImg1, kp1, resizedImg2, kp2, matches[:50], None)

cv2.imshow("Keypoint matches: ", img3)
cv2.waitKey(0)
```