TRIUMF
ATLAS ITk

Applications of Computer Vision to the ATLAS Inner Tracker Upgrade using OpenCV

TRIUMF
Vancouver, BC

Prepared by:
Marko Miletic

April 28, 2017

Contents

1 Introduction

This report outlines the functions and workings of various programs using OpenCV regarding feature description and feature matching for object detection. The programs are written in python 2.7 and use OpenCV 3.0 beta . The idea behind feature matching is that we can use keypoints found by the SIFT algorithm to describe a given object, then use the positions and other information from these keypoints to find the same object in a scene, among other objects.

2 Useful Resources and Locations

All code and resources used are located in

`/Users/student_mac1/Desktop/marko_miletic`

Due to recent relocation of files, all file paths to images are broken and must be updated.

Links to resources can be found in google chrome bookmarks but here are the most relevant ones:

1. OpenCV 3.0 beta documentation: http://docs.opencv.org/3.0-beta/
2. OpenCV Python blog: http://www.pyimagesearch.com/
3. SIFT paper by David Lowe: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf
4. Harris Corner Detection Algorithm presentation: https://indico.cern.ch/event/628185/

3 SIFT Algorithm

This section describes an implementation of the Scale-Invariant Feature Transform (SIFT) algorithm to find features, referred to as keypoints, of a given image and display them. The exact definition of "feature" in this case, as well as an explanation of the algorithm itself, can be found in David Lowe's paper found here: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

```
/Users/student_mac1/Desktop/marko_miletic/cvcode/sift.py
#!usr/bin/python

import cv2
import numpy

#standard image loading commands to first import a sample image and
#to create a grayscale from that image
image = cv2.imread("/Users/student_mac1/Desktop/sample.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#creates a sift object and then finds keypoints according to the sift algorithm
sift = cv2.xfeatures2d.SIFT_create(75)
keypoints = sift.detect(gray, None)

#draws the keypoints onto the image with one of the options
#designated by the flags parameter
cv2.drawKeypoints(image, keypoints, image, flags = cv2.DRAW_MATCHES_FLAGS_DRA
W_RICH_KEYPOINTS)

#standard display commands to first create a window for
#both images then display them
cv2.namedWindow("gray", cv2.WINDOW_NORMAL)
cv2.namedWindow("corners", cv2.WINDOW_NORMAL)
cv2.imshow("gray", gray)
cv2.imshow("corners", image)

cv2.waitKey(0)
```

To start, functions that load the desired sample image.

```
image = cv2.imread(filepath)
```

Creates an image object from an image file specified by "filepath". Since the SIFT algorithm operates on a grayscale image, we next create another image object that contains a grayscale of "image" with

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

We can now initialize the sift detector by creating a sift object with

```
sift = cv2.xfeatures2d.SIFT_create(75)
```

This also specifies 75 keypoints to be drawn. Keypoints are then calculated from the grayscale image, where None refers to using no mask.

```
keypoints = sift.detect(gray, None)
```

Keypoints are then drawn onto "image" in a style denoted by the flags parameter.

```
cv2.drawKeypoints(image, keypoints, image, flags = cv2.DRAW_MATCHES_FLAGS_DRA
W_RICH_KEYPOINTS)
```

In this case, "image" is the input image for the keypoints to be drawn on, "keypoints" is the list of keypoints that need to be drawn, and "image" is the resultant image object with keypoints drawn on it. The chosen flag parameter will produce coloured circles around every feature with their respective orientation. Finally, some functions to display the images.

```
cv2.namedWindow("gray", cv2.WINDOW_NORMAL)
cv2.namedWindow("corners", cv2.WINDOW_NORMAL)
```

These functions initialize windows for the images to be displayed, with name and window type specified, While

```
cv2.imshow("gray", gray)
cv2.imshow("corners", image)
```

will display the images.

As an aside, we can use the list of SIFT keypoints to create a descriptor for the image using

```
sift.compute(image, keypoints)
```

4 Feature Matching and FLANN

We can use the SIFT algorithm to find keypoints and create a descriptor for any given image, then compare features with another image and try to find similarities. The idea is that we can locate an image of an object within a larger image of a group of objects regardless of rotation, scale, or occlusion.

This code uses Fast Library for Approximate Nearest Neighbours (FLANN), a collection of algorithms created for matching, to find the best keypoint matches for a given query image within a cluttered sample image.

```
#file: /Users/student_mac1/Desktop/marko_miletic/cvcode/flannmatch.py
#!usr/bin/python

import cv2
import numpy as np

#Load images for matching; a query image (source) and a sample image (train)
source = cv2.imread("/Users/student_mac1/Desktop/hexagon.png")
source = cv2.cvtColor(source, cv2.COLOR_BGR2GRAY)
train = cv2.imread("/Users/student_mac1/Desktop/shapes.jpeg", 0)
final = None


#SIFT initialization and descriptor calculation
sift = cv2.xfeatures2d.SIFT_create(0, 3, 0.006, 10)

kp1, des1 = sift.detectAndCompute(source, None)
kp2, des2 = sift.detectAndCompute(train, None)


#FLANN parameter setup and initialization
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 500)
flann = cv2.FlannBasedMatcher(index_params, search_params)


#creates a list of best 2 matches for each keypoint by using the FLANN algorithm
matches = flann.knnMatch(des1, des2, k=2)


#to show only best matches, a mask is created
matchesMask = [[0,0] for i in xrange(len(matches))]

for i, (m,n) in enumerate(matches):
    if m.distance < 0.9 * n.distance:
        matchesMask[i] = [1,0]
```

```
#dictionary that describes drawing details when the result is displayed
draw_params = dict(matchColor = (0, 0, 255), singlePointColor = (255,0,0),
matchesMask = matchesMask, flags = 0)

#creates final image
final = cv2.drawMatchesKnn(source, kp1, train, kp2, matches, final, **draw_params)


#displays result
cv2.namedWindow("final", cv2.WINDOW_NORMAL)
cv2.imshow("final", final)

cv2.waitKey(0)
```

Lines 1-15 are standard loading procedures and SIFT initialization, as in the previous code. Notice this time that we find keypoints and calculate descriptors in a single line for each image with

```
kp1, des1 = sift.detectAndCompute(source, None)
kp2, des2 = sift.detectAndCompute(train, None)
```

And subsequently assign two variables to hold the list of keypoints and descriptor respectively. Next we initialize the relevant FLANN details in lines 22-25. These FLANN functions have specific parameters and should generally not be changed.

Next we use the FLANN matcher to match keypoints by passing the descriptor of each image on line 29. In this case we find the best 2 matches so that we can apply the ratio test described in David Lowe's paper. Briefly, if we compare the 2 best matches and find that they are extremely similar then we can conclude it's likely due to noise, and we reject it. This is done in lines 33-37 by creating a mask that, when passed to the drawing function, will only display useful matches.

In line 41 we describe the drawing specifications and then use them to draw the final image in line 45. The rest of the program contains standard display functions.

5 Glossary of Terms

Image: In the context of OpenCV, a nested list that contains intensity values for every pixel in an image file. An image can have 3 colour channels (RGB) or 1 colour channel (Grayscale).
Mask: An object that contains information about what to display and what not to display. In most cases, a list.
SIFT Keypoint: A point of interest in an image defined by the SIFT algorithm. Aside from a unique location, it has a "goodness score" and an assigned orientation.
SIFT Descriptor: A set of vectors based off of the keypoints that can be used for feature matching.

6 Conclusions and Recommendations

After successfully implementing and testing feature matching to a high enough standard, we can pinpoint objects by analyzing their keypoints. For example we can describe a transformation matrix by comparing the relative positions of keypoints in the query image and sample image. This is known as perspective transformation, closely related to geometric homography.