cs512 Assignment 2: Report

Amit Nikam
Department of Computer Science
Illinois Institute of Technology

October 22, 2020.

Abstract

In this programming assignment 2, I have implemented an application that performs ellipse fitting on objects in image like hand, mice, etc. Concepts related to computer vision like model fitting, edge detection, simple image transformations were put to test. To achieve the outcome OpenCV and NumPy were used. In this report I cover my application's complete implementation, testing and the outcomes.

1. Problem Statement - Ellipse Fitting

The goal of this application was to perform ellipse fitting onto world objects like hand, face, etc. The particularly challenging part about this was to configure the parameters such that the ellipse fits well onto the objects. Since fitting is also dependent on how contours are retrieved, these parameters also required configuring. Some of these parameters are made interactive and can be changed on the go. The application had to be made capable of capturing and fitting the ellipse onto the image continuously.

2. Solution

The solution happens to be two part (1) Retrieving contour from image (2) Fitting Ellipse, and thus we will looking into it part by part.

(1) Retrieving Contour:

To retrieve the contours, I first converted the image to gray scale and applied canny edge detection to get edges. By default the values I use for canny edge detection thresholds are 130 and 195, but these parameters are interactive. Higher threshold happens to be x1.5 of lower threshold. The multiplier has been picked after multiple testing. The lower threshold can be raised to max 170 which puts the higher threshold at 255, max cap. The default values of 130 and 195 were picked based on results I got in my development environments, they can be configured to get best results.

Next I find the difference between the current and lagging frame. Always a current frame is stored to be used as lagging frame in the next loop sequence. The difference really separates the objects like hand from the background by removing anything that doesn't move at all.

I then take the intersection mask of these images as bitwise_or and perform binary threshold on the mask. This gives me a contour drawing with points in the image forming the object in binary. Finally I take the pixel coordinates of the non zero values, this gives me a contour set to which I can fit my ellipse model to.

```
# Retrieve Edge With Canny
l_canny = cv2.getTrackbarPos('Canny Edge Threshold','Image')
edge = cv2.Canny(image_instance,l_canny,1.5*l_canny)

# Find Frame Difference & Update Lagging Frame
diff = cv2.subtract(image_instance, prev_instance)
prev_instance = image_instance.copy()

# Intersection Points b/w Edge and Difference For Fitting
inter = cv2.bitwise_or(edge,diff)
_, inter = cv2.threshold(inter, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
inter = cv2.findNonZero(inter)
```

(2) Fitting Model:

The fitting algorithm takes three parameters as input. The set of contours, the scaling factor and the rotation factor. The scaling and rotation factor are interactive and can be used to adjust the ellipse to fit onto the object. It can be scaled up to 20 times and can be rotated by up-to 180 degrees. Any adjustment to scaling or rotation are carried on for the next frames too and so the fitting over the object can be improved.

For fitting ellipse, I have implemented the implicit equator(conic curve) to find the ellipse objective. For that, I take the X and Y coordinates from the contours and form the objective function $S^{-1}C$(from S,C and D). Since the solution to this would be the eigenvector of $S^{-1}C$ belong to it's -negative eigenvalues, I maximize the negative value to get the solution vector V. This gives me the answer vector which can now be used to get unknown (a,b,c,d,f,g) find center coordinates of the ellipse, axes and rotation angle.

```
# Prepare Objective Function
cdef np.ndarray[long, ndim=2] D = np.hstack([x*x,x*y,y*y,x,y,np.ones([sx,sy], dtype=int)])
cdef np.ndarray[long, ndim=2] S = np.dot(D.T,D)
cdef np.ndarray[long, ndim=2] C = np.zeros([6,6], dtype=int)
C[0,2]=C[2,0]=2
C[1,1]=-1

# Solve Objective for Answer
cdef np.ndarray[double, ndim=1] E
cdef np.ndarray[double, ndim=2] V
E , V = np.linalg.eig(np.dot(np.linalg.inv(S),C))
cdef int n = np.argmax(E)

# Answer
cdef np.ndarray[double, ndim=1] an = V[:,n]
```

After we received the ellipse from the algorithm, we draw the ellipse.

3. Implementation Details

The following files are included in src folder:
- Main.py
- self_func.pyx
- self_func.c
- self_func.so
- setup.py

Before running the main program, make sure to build the cython dependencies. To build the cython modules, simply execute the following command in the terminal in src direct:

```
$ python3 setup.py build_ext --inplace
```

After the cython files have been built, make sure that .so extension module(might be different for windows/mac) is named ‘self_func’.ext

Now the program is ready to run.

To simply start the program with camera input, run the main program. It will automatically try to get camera response from front camera (0)

```
$ python3 main.py
```

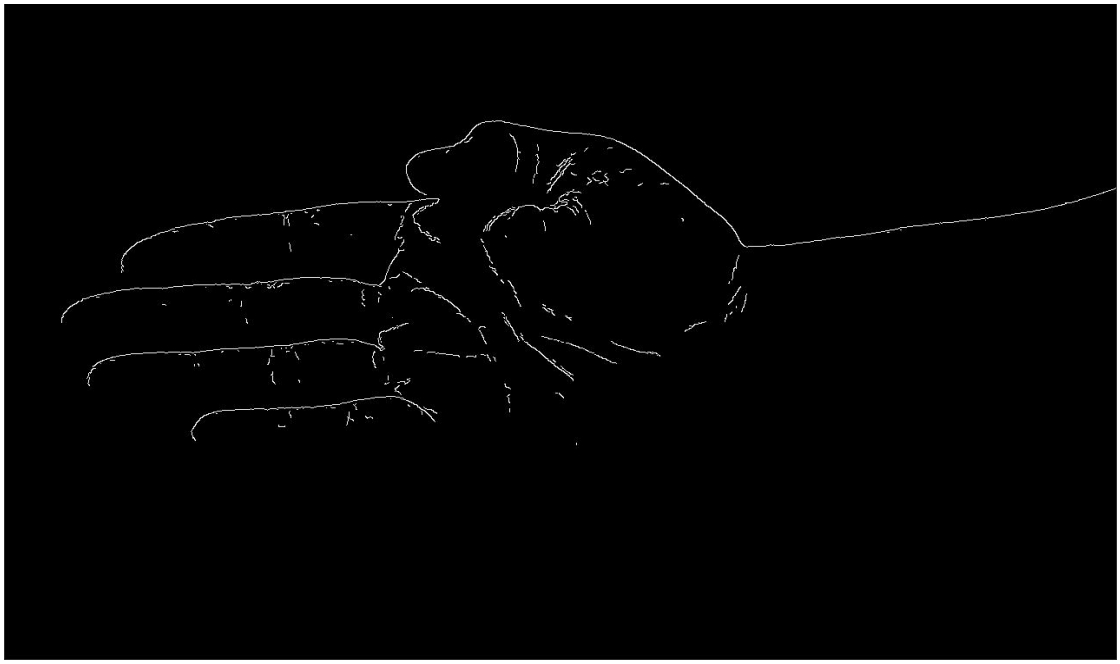
To use an image ‘img_name’ located at ‘loc’ to process in the program, simply run the command as such in the terminal:

```
$ python3 main.py --img loc/img_name
```

Passing the image name as argument --img during initialization runs the application in non camera mode.

4. Results and Discussion

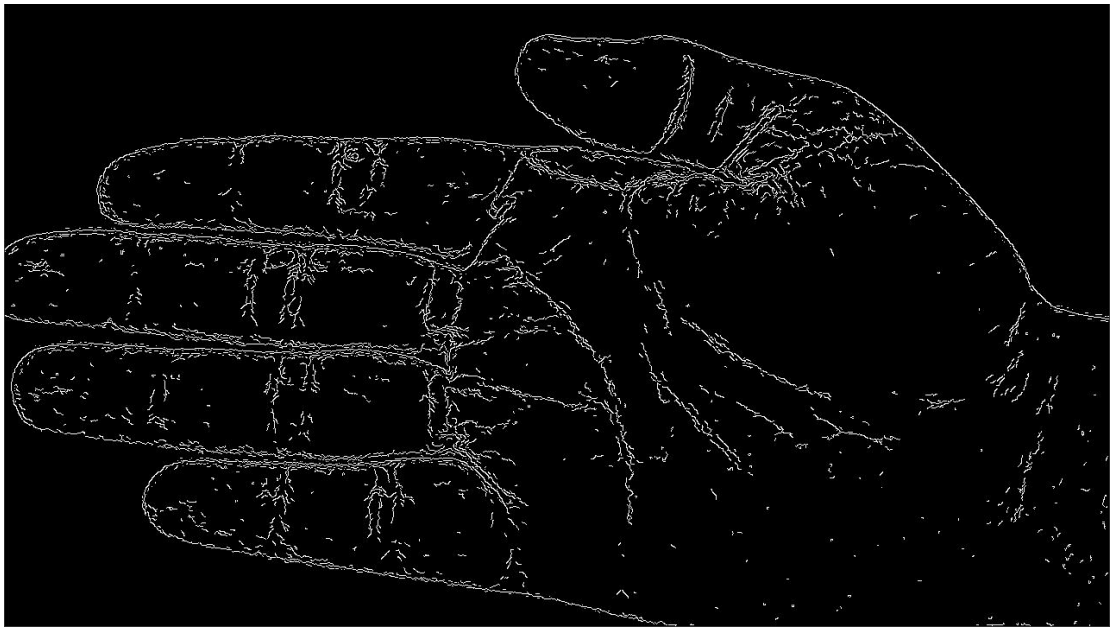
Contour Mask: The following image shows the binary contour mask that is formed after bitwise_or (intersection) of edge detection and frame difference. Pixel Positions of the non zero pixels from this are used to fit the ellipse.



The corresponding ellipse drawn: Since shadow makes the lower area of the palm invisible to the model, the upper area is fitted. The rotation and scaling parameters can be used to adjust such errors caused by loss of data.

Issues with Edge Detection:
In the above example, the thresholds for Canny edge detection were 130 and 195, which is a good value for my environment. But after reducing or increasing the Canny Edge Thresholds, it is observed that the fitting changes drastically. This is due to the introduction of unwanted **noise** when the threshold is low, and the loss of data points when the threshold is high.

The following contour is an example of **noise in image**: In this case a lot of extra points are present which will negatively affect the model fitting. In opposite case, a lot more data-points are lost, leaving the model to be a bad fit.



Adjustments:
Although the model is adjusted, it might not give good results in different environment, but it can be adjusted using interactive Canny Edge Threshold, scaling and rotation parameters which helps to get better outcome. This works as a fault tolerance so some degrees where shadows results in loss of data points or in general if the contour is not perfect.

Conclusion: In the end fitting ellipse onto world objects was accomplished. As model fitting depends entirely on the data that is used as input, a poorly prepared or retrieved contour can make even a good model unusable. Some of the other factors contributing for failure could be noise and illumination.