Cairo University
Faculty of Engineering
Department of Computer Engineering

Code Explanation

Corner Detection Module

```python
class ShiTomasi:

    minDistance = 10
    numPeaks = 6
    thresholdAbs = 0.10
    excludeBorder = 2


    def __int__(self):
      pass

    def getCorners(self,img):
        self.xx, self.xy, self.yy = self.correlationMatrix(img) #get the derivitives of dxdx,dydy and dxdy

        corners = self.minEigenValue(self.xx,self.yy,self.xy) #calc minimum eigenvalue

        coordinates = self.peakLocalMax(corners) #get the coordinates
        return coordinates

    def correlationMatrix(self,image):
        image = toFloat(image)
        # self.yy = nImg.sobel(image, axis=0, mode='constant', cval=0)
        # self.xx = nImg.sobel(image, axis=1, mode='constant', cval=0)
        self.xx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
        self.yy = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)


        x2x = self.xx * self.xx
        y2y = self.yy * self.yy
        x2y = self.xx * self.yy

        kernel = np.ones((9, 9), np.float32) / 81
        self.Dxx = cv2.filter2D(x2x, -1, kernel)
        self.Dxy = cv2.filter2D(x2y, -1, kernel)
        self.Dyy = cv2.filter2D(y2y, -1, kernel)

        # self.Dxx = cv2.GaussianBlur(x2x,(9,9),0)
        # self.Dxy = cv2.GaussianBlur(x2y,(9,9),0)
        # self.Dyy = cv2.GaussianBlur(y2y,(9,9),0)

        # self.Dxx = nImg.gaussian_filter(x2x, 1, mode='constant', cval=0)
        # self.Dxy = nImg.gaussian_filter(x2y, 1, mode='constant', cval=0)
        # self.Dyy = nImg.gaussian_filter(y2y, 1, mode='constant', cval=0)

        return self.Dxx, self.Dxy, self.Dyy
```

```
53      # return minimum eigenvalue of A
54      def minEigenValue(self,xx,yy,xy):
55          return ((xx + yy) - np.sqrt((xx - yy) ** 2 + 4 * xy ** 2)) / 2
56
57      #Find peaks in img
58      #minDistance is the distance between one pixel and another making it the loca
59      #return coordinates of the peaks
60      def peakLocalMax(self,image):
61
62          # Non maximum filter
63          size = 2 * self.minDistance + 1
64          image_max = nImg.maximum_filter(image, size=size, mode='constant')
65          mask = image == image_max
66
67          # zero out the image borders
68          for i in range(mask.ndim):
69              mask = mask.swapaxes(0, i)
70              remove =  self.excludeBorder
71              mask[:remove] = 0
72              mask[-remove:] = 0
73              mask = mask.swapaxes(0, i)
74
75          # find top peak candidates above a threshold
76          aboveThresholdCorners = image > self.thresholdAbs
77          mask &= aboveThresholdCorners
78
79          # get coordinates of peaks
80          coordinates = np.nonzero(mask)
81          # select num_peaks peaks
82          if len(coordinates[0]) > self.numPeaks:
83              intensities = image[coordinates]
84              idx_maxsort = np.argsort(intensities)
85              coTp = np.transpose(coordinates)
86              coordinates = coTp[idx_maxsort][-self.numPeaks:]
87          else:
88              coordinates = np.column_stack(coordinates)
89          # Highest peak first
90          coordinates =  coordinates[::-1]
91
92          return coordinates
```

```python
    def getFeatures(self,img,xmin=0,ymin=0,opencv=False):
        if opencv:
            return self.getFeaturesOpencv(img,xmin,ymin)
        else:
            return self.getFeaturesMine(img,xmin,ymin)

    def getFeaturesMine(self,img,xmin =0,ymin =0):

        corners = self.getCorners(img)
        corners[:,1] += xmin
        corners[:,0] += ymin

        return corners[:,1],corners[:,0]

    def getFeaturesOpencv(self,img,xmin = 0,ymin = 0):
        maxCorners = 6
        qualityLevel = 0.17
        minDistance = 10
        blockSize = 10

        corners = cv2.goodFeaturesToTrack(img, mask=None, maxCorners=maxCorners,
                                qualityLevel=qualityLevel,
                                minDistance=minDistance,
                                blockSize=blockSize)
        corners = np.int0(corners)

        x,y = [],[]
        ind = []
        for i in corners:
            xx, yy = i.ravel()
            yy +=ymin
            xx +=xmin
            x.append(xx)
            y.append(yy)
            # ind.append((xx,yy))
        return x,y
        # return ind
```

Lines	Explanation	Input/Output
8	Minimum distance to the neighbouring pixels in the frame.	-
9	Num of corners we want.	-
10	The min threshold for the corner.	-
11	Don’t get near the border becauses they have heights change and that will affect on the results of the heightest corners peak.	-
17 → 23	The main flow of the class to get the corners. “applying shitomasi algorithm” First: get the variables of the correlattion matrix Sec: calculate min eigenvalues Third: choose the best corners	Input: cut frame with car. Output: 6 corners coordinates.
25 →50	Changes the values of the pixels form integer to float so when you make divisions it will be more accurate,	Input: cut frame with car. Ouptut: IxxM,IyyM,IxyM

	Then get the Ix and Iy Using sobel edge filter with window size of 5*5, Then calculate IxIx, IyIy, IxIy then make window of 9*9 (find it the best after several trials with other dimensions) and window is median filter (tried gaussian but the results weren’t as good as the median filter) then iterate across all the image with the median filter as the algorithm said in its paper And return the results.	
54→55	Get the minimum eigenvalues by solving a second order equation	Input: IxxM,IyyM,IxyM. Output: Whole matrix with values of the minimum eigenvalues.
60→92	Here comes the part to choose best corners according to our criteria, First: applying non maximum filter to choose the heightest values in a given window size (2* minDistance + 1), Sec: make a mask to the image then exlude the borders from the results by excluding by exludeBorder parameter we specificed above, Third: as we work in 1 and 0 now we could make anding mask with above threshold corners with the parameter we specified above, Now we have a mask has the following attributes 1) got the heightes peaks in every reigon 2) no corners on the borders 3) the peaks higher than the threashold Now only one parameter left which is the numofPeaks So we want to sort these values in the mask to get the highest peaks so we mask the cornerImage and start sorting them from lowest to heightest peak then reverse array.	Input: corners Image. Output: array of corners.
95→99	If you want to get features (corners) you call the get feature method and specifiy if you want to use opencv or my method.	Input: cut frame with car. Outptut: corners in the correct places on the big frame.
101→107	My method first call the getcorners function passing the image and incremenet the xmin and ymin of the big frame on the corners values and return the corners.	Input: cut frame with car. Outptut: corners in the correct places on the big frame.
109→130	Opencv method same approach as above.	Input: cut frame with car. Outptut: corners in the correct places on the big frame.

The next photo is just code I wrote to make demo and test my shitomasi implementation:-
1- read video.
2- loop until to reach your frame index you want.
3- convert BGR to gray level.
4- open window to take a box on the car and close the window.
5- call shitomasi implementation.
6- show corners on image.
7- call opencv implementation.
8- show corners on image.
9- print some statistics.
10- wait until you hit a key on your keyboard and close a window.

```python
def show(image,x,y,windowName):
    for i in range(len(x)):
        xx, yy = x[i],y[i]
        cv2.circle(image, (xx, yy), 3, (0,0,255), -1)
    cv2.imshow(windowName, image)


if __name__ == "__main__":
    cap = cv2.VideoCapture("Easy.mp4")
    i =0
    while(i < 1):
        ret, frame = cap.read()  # get first frame
        i+=1

    frame_gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    bbox = cv2.selectROI("select window",frame_gray)
    cv2.destroyAllWindows()
    xmin,ymin = bbox[0],bbox[1]
    img = frame_gray[bbox[1]:bbox[1]+bbox[3] , bbox[0]:bbox[0] + bbox[2]]

    shiTomasi = ShiTomasi()

    tMine = time()
    x,y = shiTomasi.getFeatures(img,xmin,ymin,False)
    tMine = time() -tMine
    show(frame.copy(),x,y,"Mine")

    tOpencv = time()
    x,y = shiTomasi.getFeatures(img,xmin,ymin,True)
    tOpencv = time() - tOpencv
    show(frame.copy(),x,y,"Opencv")


    print("Mine: "+str(tMine))
    print("Opencv: "+str(tOpencv))
    print("Opencv/Mine: "+str(tOpencv/tMine))
    print("Percentage of Speedup: " + str(((tMine - tOpencv) / tMine)*100)+" %")


    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

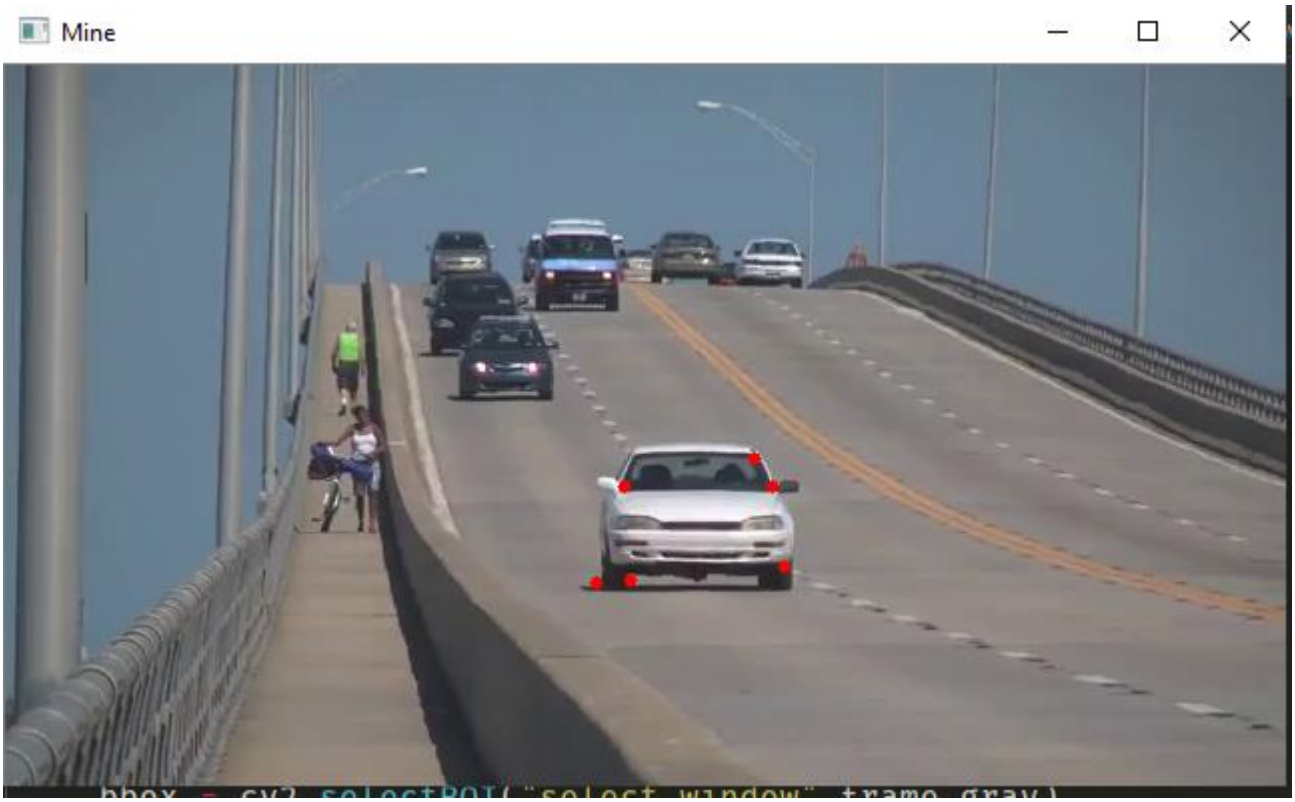
Statistics

```
"C:\Users\KhALeD SaBrY\AppData\Local\Programs\Python\Python37-32\p
Mine: 0.0069804191589355547
Opencv: 0.002032756805419922
Opencv/Mine: 0.2912084158754013
Percentage of Speedup: 70.87915841245986 %
```

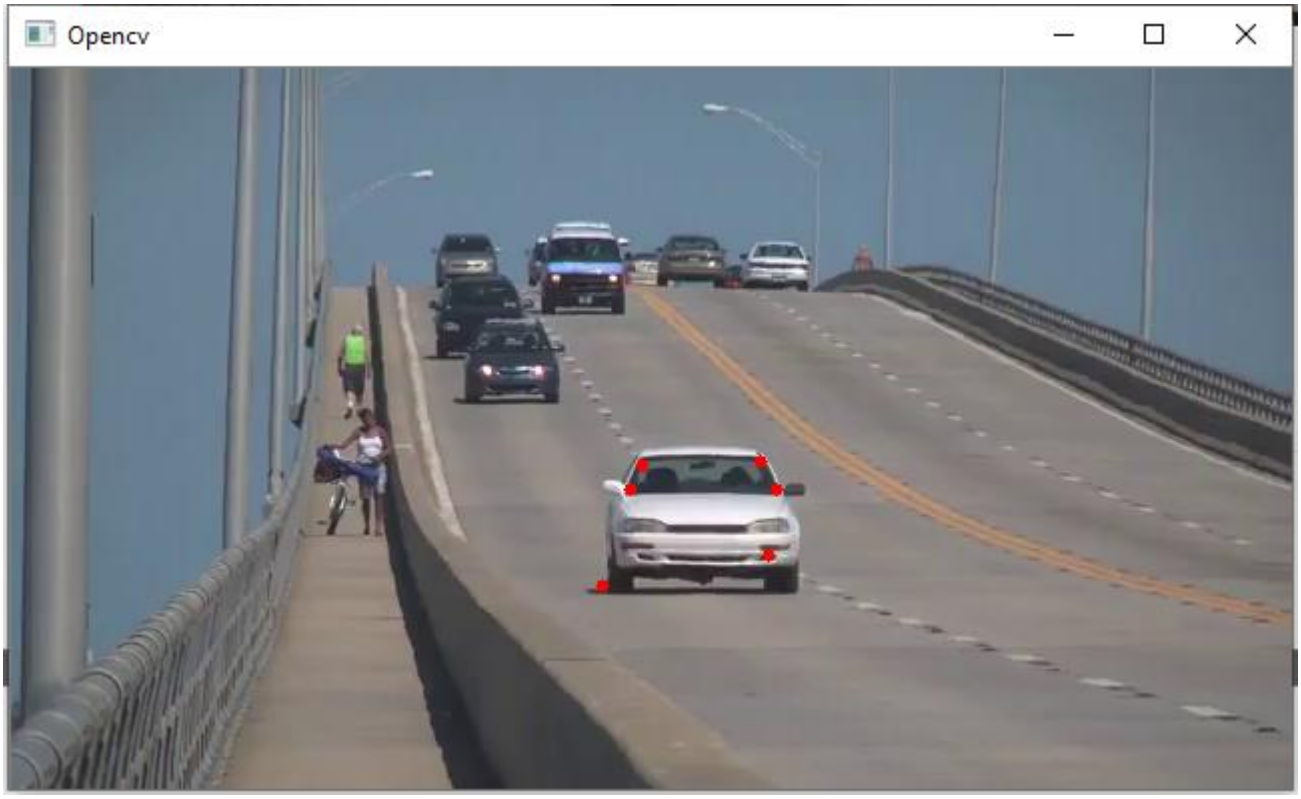
Capturing Image

Final Results

Mine



Opencv



J. Shi and C. Tomasi Algorithm

To find a corner, find the difference in intensity for a displacement of (u,v) in all directions. This is expressed as below:

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \underbrace{[I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

Window function is either a rectangular window or gaussian window which gives weights to pixels underneath.

We have to maximize this function E(u,v) for corner detection. That means, we have to maximize the second term. Applying Taylor Expansion to above equation and using some mathematical steps, we get the final equation as:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

Where:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Ix and Iy are image derivatives in x and y directions respectively, and then apply the score function:

$$R = min(\lambda_1, \lambda_2)$$

And that’s it now you have all the corners!

Implementation

Here are the steps that we implemented in the program for feature extraction module “corner detection”:-

1. Get the derivatives of the picture for the rows [yy] and cols [xx] and both [xy].
2. Average with a gaussian filter window the [xx*xx] , [yy*yy] and [xx*yy] so you can get the correlation matrix.
3. The corner measure is then defined as the smaller eigenvalue of correlation matrix Which we can get by solving an equation of 2nd order and get the minimum lamda.

Now, you have the corners, but you need the best n corners:

4. Apply maximum filter to extract the best in a region which we specified by:
   2 * minimumDistance +1
5. Exclude the borders which could cause in a lot of unwanted corners.
6. Sort the remaining points to its highest intensities and select the n corners you want.
7. Select the n corners and return them.