CS 512 HW5 Report
SUDIPTA SWARNAKAR

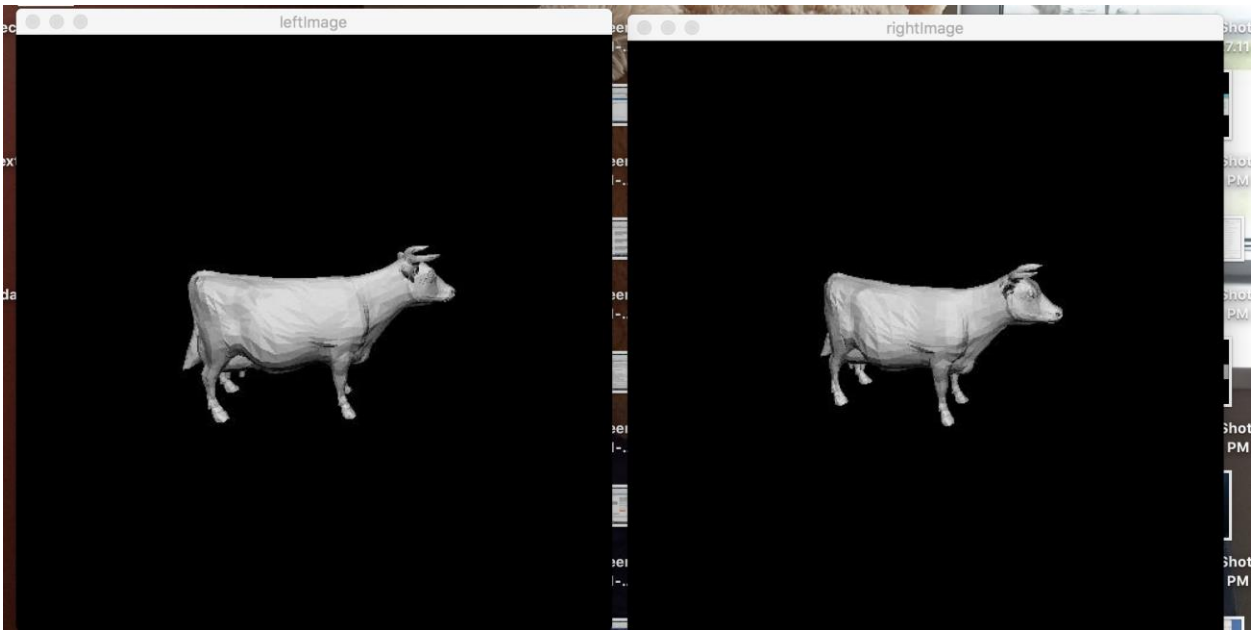
Problem Statement 1:
Get corresponding points from two images on mouse click

Sol:
I have used 'EVENT_LBUTTONDOWN' and 'EVENT_LBUTTONUP' events from OpenCV to capture user's input from left and right images which are shown on side by side. getMouseInput(leftImage,rightImage) is taking two images and calling the above events to capture coordinates given by user on mouse click.

Source Code:

```
def click_and_crop(event, x, y, flags, param):
    # grab references to the global variables
    global refPt, cropping

    # if the left mouse button was clicked, record the starting
    # (x, y) coordinates and indicate that cropping is being
    # performed
    if event == cv2.EVENT_LBUTTONDOWN:
        refPt = [(x, y)]
        print refPt
        cropping = True


    # check to see if the left mouse button was released
    elif event == cv2.EVENT_LBUTTONUP:
        # record the ending (x, y) coordinates and indicate that
        # the cropping operation is finished
        refPt.append((x, y))
        print x,y
        cropping = False
        inputArray.append((x,y))
        print "Input Array Length",len(inputArray)

        # draw a rectangle around the region of interest

        #cv2.rectangle(leftImage, refPt[0], refPt[1], (0, 255, 0), 2)
        #cv2.rectangle(rightImage, refPt[0], refPt[1], (0, 255, 0), 2)
        cv2.imshow("leftImage", leftImage)
        #cv2.imshow("rightImage", rightImage)

def getMouseInput(leftImage,rightImage):

    clone1 = leftImage.copy()
    clone2 = rightImage.copy()

    cv2.namedWindow('leftImage')
    cv2.setMouseCallback('leftImage', click_and_crop)
```

```
cv2.namedWindow('rightImage')
cv2.setMouseCallback('rightImage', click_and_crop)

while True:

    # display the image and wait for ca keypress

    cv2.imshow("leftImage", leftImage)
    cv2.imshow("rightImage", rightImage)

    key = cv2.waitKey(1) & 0xFF

    # if the 'r' key is pressed, reset the cropping region
    if key == ord("r"):
        img1 = clone1.copy()
        img2 = clone2.copy()

    # if the 'c' key is pressed, break from the loop
    elif key == ord("c"):
        break

# if there are two reference points, then crop the region of interest

# close all open windows
cv2.destroyAllWindows()
```



Images placed side by side to take input coordinates based on mouse click

Problem Statement 2:

Calculate Fundamental Matrix from given corresponding coordinate points.

Sol:
To calculate Fundamental matrix, I have used 8-points algorithm which takes given corresponding points and uses below steps to calculate Fundamental matrix.

8-point algorithm
Given a number of corresponding points, we could use the epipolar constraints to try to recover camera pose (R, T).
–Least squares solution using SVD on equations from 8 pairs of correspondences
–Enforce det(F)=0 constraint using SVD on F
1.Solve a system of homogeneous linear equations
a. Write down the system of equations
b. Solve f from Af=0 using SVD
2.Resolve det(F) = 0 constraint by SVD

Computes the fundamental matrix from corresponding points (x1,x2 3*n arrays) using the 8 point algorithm. Each row in the A matrix below is constructed as
[x'*x, x'*y, x', y'*x, y'*y, y', x, y, 1]

```
A = zeros((n, 9))
for i in range(n):
    A[i] = [x1[0, i] * x2[0, i], x1[0, i] * x2[1, i], x1[0, i] * x2[2, i],
            x1[1, i] * x2[0, i], x1[1, i] * x2[1, i], x1[1, i] * x2[2, i],
            x1[2, i] * x2[0, i], x1[2, i] * x2[1, i], x1[2, i] * x2[2, i]]
```

compute linear least square solution

```
U, S, V = linalg.svd(A)
F = V[-1].reshape(3, 3)
```

make rank 2 by zeroing out last singular value

```
U, S, V = linalg.svd(F)
S[2] = 0
F = dot(U, dot(diag(S), V)
```

Output:
Fundanental Matrix: [[-0.54622827 0.4112342 -0.07571255]
 [-0.09986165 0.10424375 -0.08340719]
 [-0.33691098 -0.18362366 1.   ]]

Problem Statement 3:
Computes the fundamental matrix from corresponding points x1, x2 using the normalized 8-points algorithm

Sol:
Solve in normalized coordinates

–mean=0
–RMS distance = (1,1,1)

normalize.

```
x1 = x1 / x1[2]
mean_1 = np.mean(x1[:2], axis=1)
S1 = np.sqrt(2) / np.std(x1[:2])

T1 = np.array([[S1, 0, -S1 * mean_1[0]],
               [0, S1, -S1 * mean_1[1]],
               [0, 0, 1]])

x1 = np.dot(T1, x1)

x2 = x2 / x2[2]
mean_2 = np.mean(x2[:2], axis=1)
S2 = np.sqrt(2) / np.std(x2[:2])

T2 = np.array([[S2, 0, -S2 * mean_2[0]],
               [0, S2, -S2 * mean_2[1]],
               [0, 0, 1]])
x2 = np.dot(T2, x2)

F = compute_fundamental(x1, x2)
```

denormalize.

```
F = np.dot(T1.T, np.dot(F, T2))
return F / F[2, 2]
```

Problem Statement 4:
Compute Left and Right Epipoles

Sol:
Computes the (left) epipole from a fundamental matrix F

```
def compute_left_epipole(F.T):

    # return null space of F (Fx=0)
    U, S, V = linalg.svd(F)
    e = V[-1]
    return e / e[2]
```

Computes the (right) epipole from a fundamental matrix F

```
def compute_right_epipole(F):

    # return null space of F (Fx=0)
    U, S, V = linalg.svd(F)
    e = V[-1]
    return e / e[2]
```

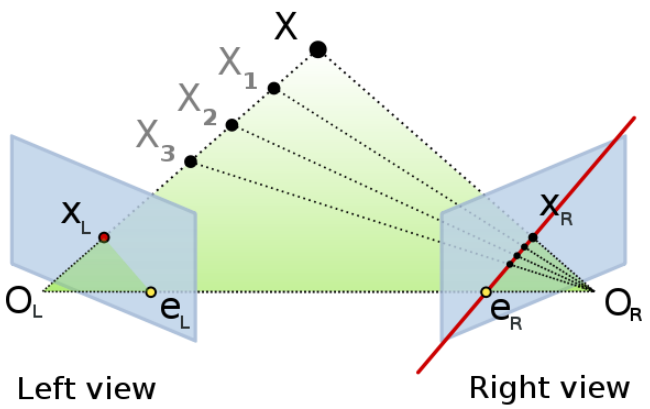
Right Epipole: [1.66352101 2.39370841 1.   ]

Left Epipole: [-3.36755827 15.04626215   1.   ]

Problem Statement 5:

Plot the epipole and epipolar line F*x=0 in an image. F is the fundamental matrix and x a point in the other image given by user through mouse click event.

Sol:

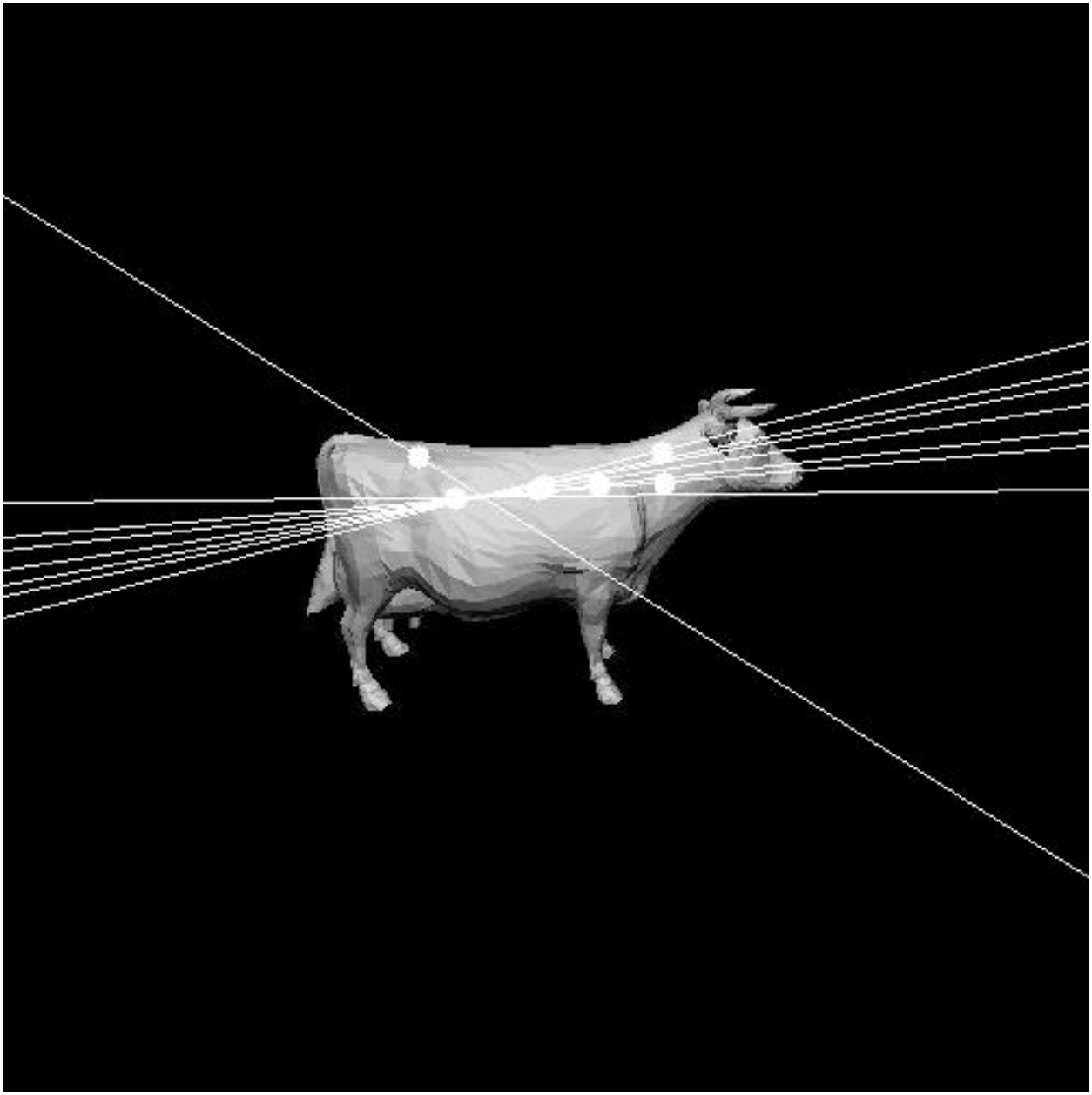


The projection of the different points on OX form a line on right plane, We call it epiline corresponding to the point. It means, to find the point on the right image, search along this epiline. It should be somewhere on this line. This is called Epipolar Constraint. Similarly all points will have its corresponding epilines in the other image. This plane XOlOR is called Epipolar Plane.

Source Code:

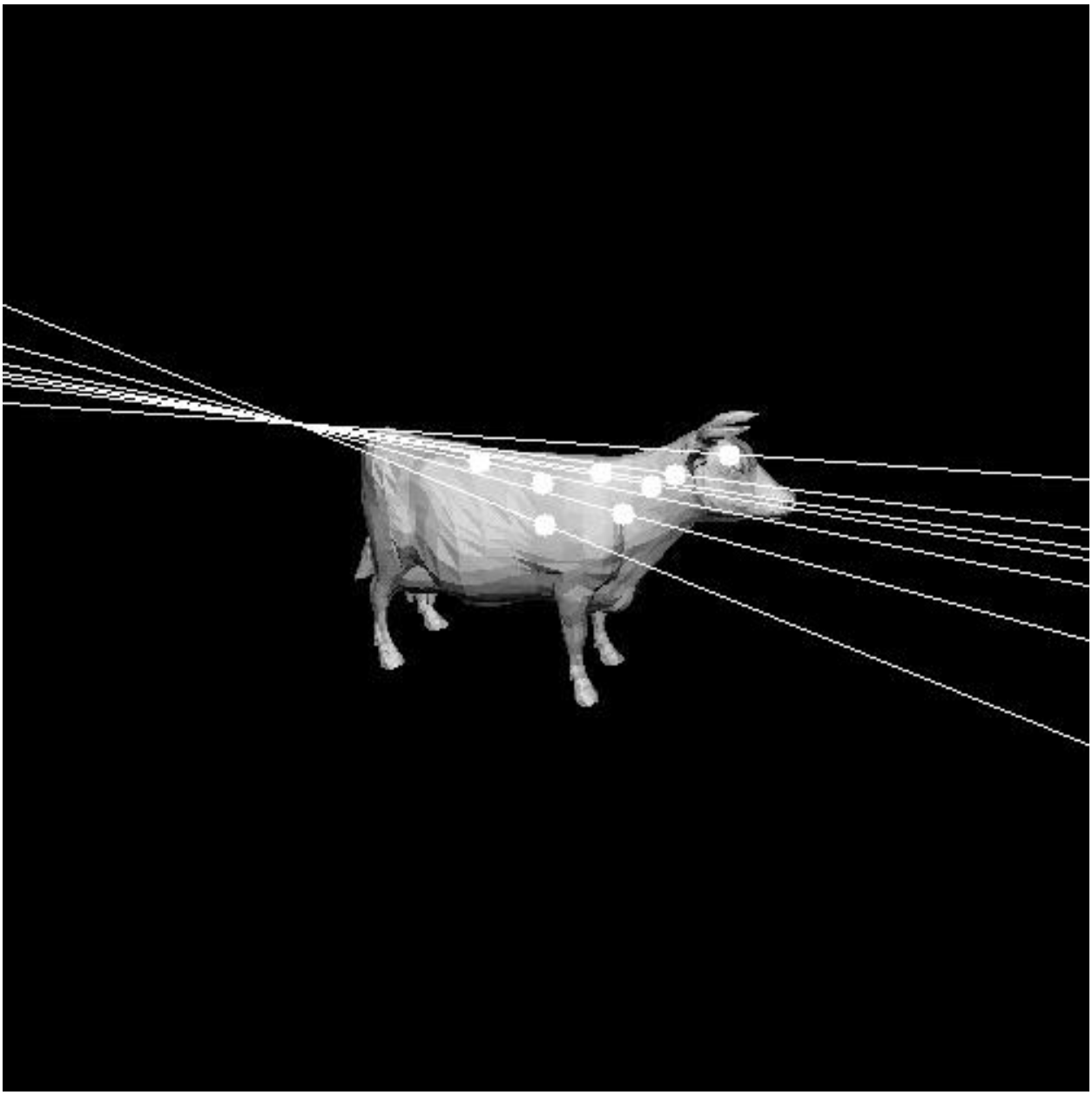
```
def plot_epipolar_line(im, F,x, epipole=None, show_epipole=True):

    """ Plot the epipole and epipolar line F*x=0
       in an image. F is the fundamental matrix
       and x a point in the other image."""
    m, n = im.shape[:2]
    line = dot(F, x)

    # epipolar line parameter and values
```

```
t = linspace(0, n, 100)
lt = array([(line[2] + line[0] * tt) / (-line[1]) for tt in t])

# take only line points inside the image
ndx = (lt >= 0) & (lt < m)
plot(t[ndx], lt[ndx], linewidth=2)

if show_epipole:
    if epipole is None:
        epipole = compute_right_epipole(F)
    plot(epipole[0] / epipole[2], epipole[1] / epipole[2], 'r*')

return line
```



epilines corresponding to points in right image

[[1.51339337e-01 9.88481879e-01 -2.57660919e+02]
[1.83059752e-01 9.83101785e-01 -2.63451599e+02]
[2.03806490e-01 9.79011178e-01 -2.67109283e+02]
[1.09276123e-01 9.94011462e-01 -2.49618179e+02]
[8.19624439e-02 9.96635437e-01 -2.44176376e+02]
[2.46205419e-01 9.69217658e-01 -2.74259094e+02]
[1.35236979e-02 9.99908566e-01 -2.29792175e+02]
[-5.31187594e-01 8.47254217e-01 -7.47720413e+01]]



epilines corresponding to points in left image

```
[[ -1.99628845e-01   9.79871571e-01  -1.62054321e+02]
 [ -2.63874561e-01   9.64556992e-01  -1.50502090e+02]
 [ -2.63874561e-01   9.64556992e-01  -1.50502090e+02]
 [ -1.70167178e-01   9.85415220e-01  -1.67066940e+02]
 [ -1.56365201e-01   9.87699330e-01  -1.69354919e+02]
 [ -3.73438627e-01   9.27654862e-01  -1.28723297e+02]
 [  1.32116213e-01  -9.91234243e-01   1.73282684e+02]
 [  7.14756772e-02  -9.97442365e-01   1.82597687e+02]]
```