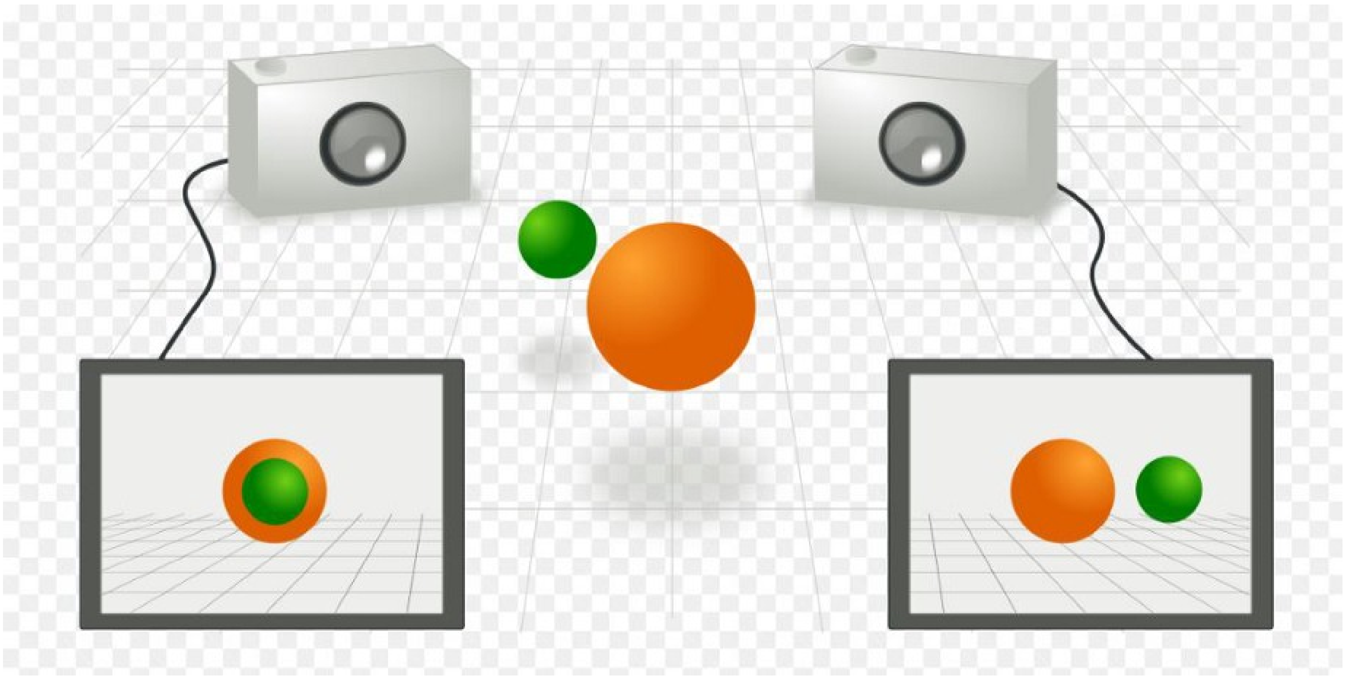
3D Reconstruction with Stereo Images — Part 3: Epipolar Geometry

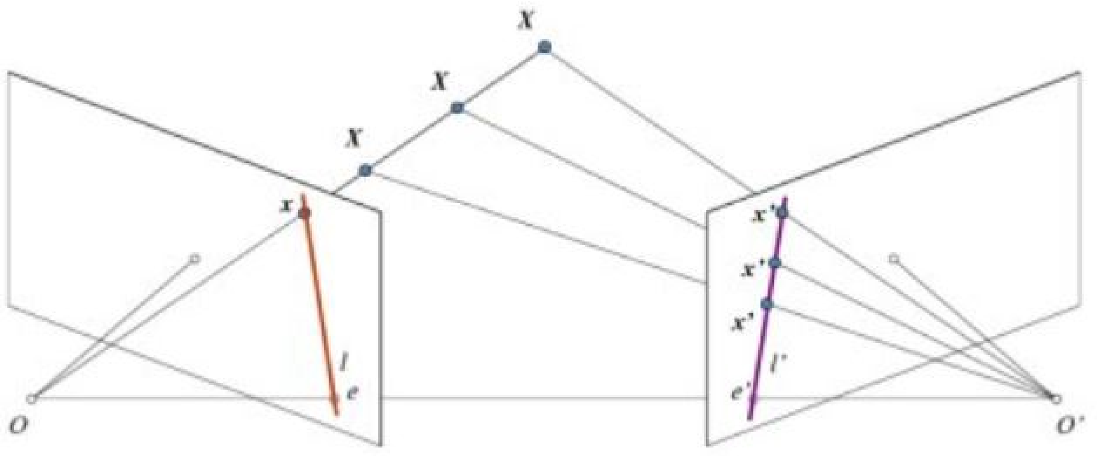
AI/HUB
Jan 10, 2019 · 7 min read

At this stage we are ready explore the core concepts that allow 3D reconstruction to work in practice. Specifically we will learn about the basics of multi-view or epipolar geometry.



Let's think about the functionality of a modern pinhole camera. When an image is taken its information is presented on a 2D plane where the distance of each point (or pixel) away form the camera lens is simply non-existent. But this is precisely the information we need in order to perform 3D reconstruction with 2D images. The key to our solution lies in using a second camera to take a picture of the same object and compare each image to extract depth information. Interesting to note is that our eyes act in a similar war to perform this very same task. Fortunately OpenCV contains tools that will help us see depth in our images produced by our non-human pinhole cameras.

Let's start by analyzing the figure below that illustrates some basic concepts of multi-view geometry in a scenario with two cameras taking a picture of the same scene.



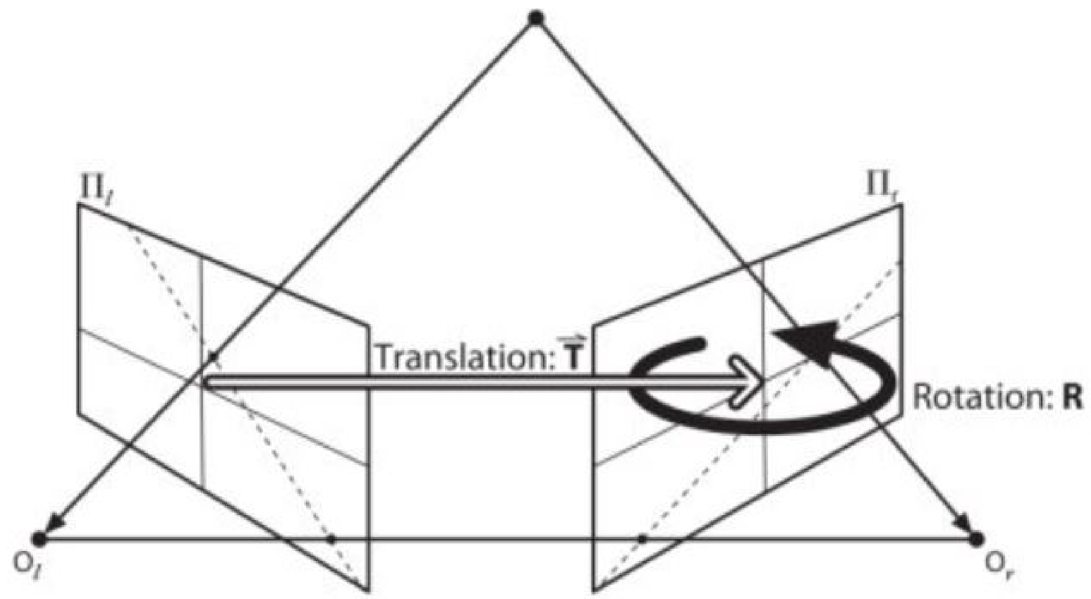
By examining the line OX pictured above that is emanating from the center-point of the left camera we can see that there are 3 points along that line in which we want to know their distance from the point O. As stated early we're not going to be able to extract this information from the left sided camera alone. With the right sided camera however we can see that the 3 points of X that are in question can be projected along a single line on the plane of its image (the line denoted with: l'). This line is referred to as the epiline; the line in which any point X along the line OX must appear. It's significance is that when we are searching for the matching point of X in the right sided camera's image we already know that it will appear along this epiline therefore greatly reducing our search effort. Furthermore we can assume every point that appears in the left handed image will always have an accompanying epiline found in the right sided image. This is known as the epipolar constraint.

Another important line to note is the epipole. The epipole is the line that connects the center-points of each camera. Above it pictured with 0 and 0 ' denoted at each end. The points e and e' are the points in which the center-point of the opposing camera will appear in its counterpart's image. By including the point X(the point whose depth value we are trying to extract) with the epipole line we can derive the plane denoted as X00'. This plane is referred to as the epipolar plane.

We can also see in the figure above that the epiline in the right sided image on which the points of X in question are projected intersects with the epipole point of the opposing camera(e'). This shows how all epilines will pass through the epipole point found in the image(the image point where the real would center point of the opposing camera will appear). The significance of this is that we are able to calculate exactly where the epipole point is in our image by finding the image point where multiple epilines intersect. We should also note that there are many cases where 1 or both camera center points may not be captured in the frame of the opposing image, meaning that the epipole point will fall outside of its 2D pixel matrix. Regardless of this we will still be able mathematically calculate where the fictional epipole pole exists through the analysis of multiple epilines and therefore will still be able to proceed towards reaching our end goal of extract depth information.

Since we have now covered the basics of epipolar geometry we can now address two more ingredients that are needed in order to find the epipole and epiline, that is; the fundamental and essentials matrices.

The essential matrix contains information about translation and rotation which describes the camera's position relative to its counterpart. The Essential Matrix is illustrated below for a better understanding.



But for our solution we want to make our calculations in terms of camera pixel coordinates. This is where the fundamental matrix comes into play as it contains the same information as the essential matrix but also includes intrinsic information about both cameras so that we can relate the two in such terms. While the information found in the essential matrix is derived from the real world positioning of our cameras, the fundamental matrix will have to be calculated ourselves. What the fundamental matrix allows us to do is map a single point in one image to its corresponding epiline in the other giving us a starting point to later find the epipole line between the two cameras center points and the epipolar plane.

But how do we calculate the fundamental matrix? Fortunately we can use OpenCV to derive the matrix from a set of known matching points in each image. This is essentially a calibration process whereby it is recommended that a minimum of 8 matching points from the images be found and used to achieve the needed accuracy. To do this we use OpenCV to analyze both images and extract their best matching pixel coordinates. In the code below we extract these points using SIFT descriptors and FLANN based matcher and ratio text.

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

img1 = cv2.imread('myleft.jpg',0)  #queryimage # left image
img2 = cv2.imread('myright.jpg',0) #trainimage # right image

sift = cv2.SIFT()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)

flann = cv2.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)

good = []
pts1 = []
pts2 = []

for i,(m,n) in enumerate(matches):
    if m.distance < 0.8*n.distance:
        good.append(m)
        pts2.append(kp2[m.trainIdx].pt)
        pts1.append(kp1[m.queryIdx].pt)
```

This will give us a list of the best pixel matches in both images which can then be sent to calculate the fundamental matrix as we will do so below with the abilities of OpenCV.

```python
pts1 = np.int32(pts1)
pts2 = np.int32(pts2)
F, mask = cv2.findFundamentalMat(pts1,pts2,cv2.FM_LMEDS)
```

```
# We select only inlier points
pts1 = pts1[mask.ravel()==1]
pts2 = pts2[mask.ravel()==1]
```

Let’s take a second to create a function that will draw lines over our images which will later be used to visualize the epilines.

```
def drawlines(img1,img2,lines,pts1,pts2):
    ''' img1 - image on which we draw the epilines for the points in
img2
        lines - corresponding epilines '''
    r,c = img1.shape
    img1 = cv2.cvtColor(img1,cv2.COLOR_GRAY2BGR)
    img2 = cv2.cvtColor(img2,cv2.COLOR_GRAY2BGR)
    for r,pt1,pt2 in zip(lines,pts1,pts2):
        color = tuple(np.random.randint(0,255,3).tolist())
        x0,y0 = map(int, [0, -r[2]/r[1] ])
        x1,y1 = map(int, [c, -(r[2]+r[0]*c)/r[1] ])
        img1 = cv2.line(img1, (x0,y0), (x1,y1), color,1)
        img1 = cv2.circle(img1,tuple(pt1),5,color,-1)
        img2 = cv2.circle(img2,tuple(pt2),5,color,-1)
    return img1,img2
```

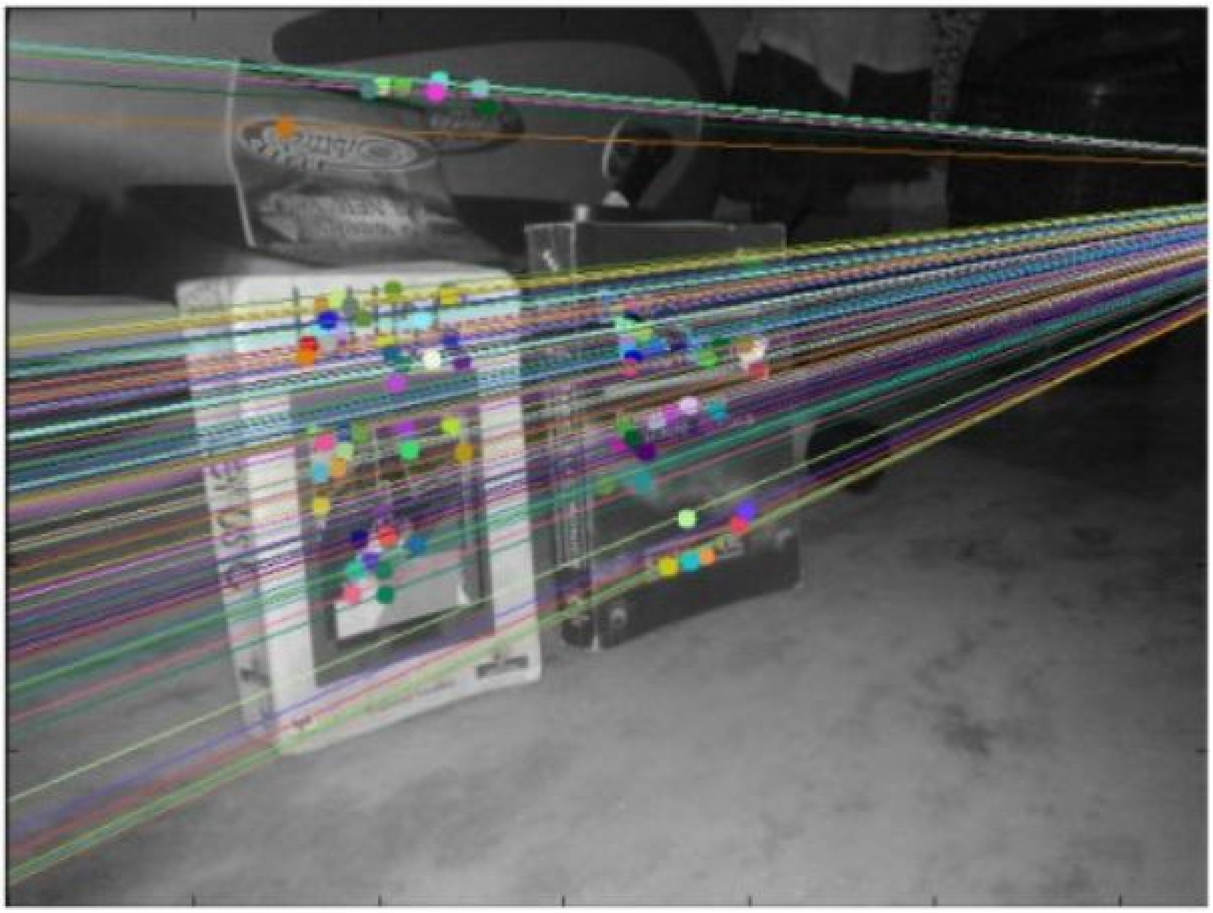
Now we can proceed to calculate the epilines which correspond to points in the opposing image. At this stage we also need to be mindful of which camera we are working with as we are going to represent the points found in one image with epilines drawn on its counterpart. The code below will produce an array of lines which we can subsequently send to our draw function.

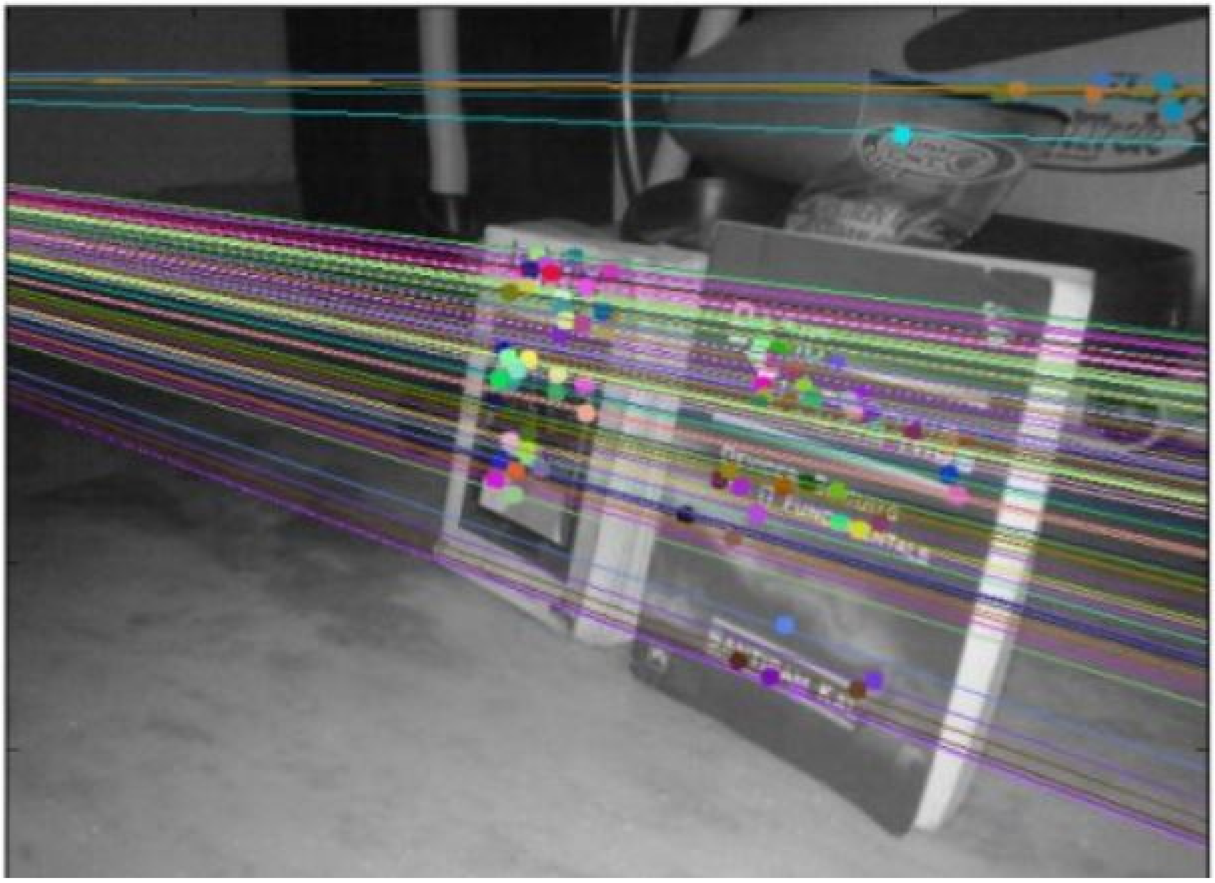
```
# Find epilines corresponding to points in right image (second
image) and
# drawing its lines on left image
lines1 = cv2.computeCorrespondEpilines(pts2.reshape(-1,1,2), 2,F)
lines1 = lines1.reshape(-1,3)
img5,img6 = drawlines(img1,img2,lines1,pts1,pts2)

# Find epilines corresponding to points in left image (first image)
and
# drawing its lines on right image
lines2 = cv2.computeCorrespondEpilines(pts1.reshape(-1,1,2), 1,F)
lines2 = lines2.reshape(-1,3)
img3,img4 = drawlines(img2,img1,lines2,pts2,pts1)

plt.subplot(121),plt.imshow(img5)
plt.subplot(122),plt.imshow(img3)
plt.show()
```

The result shows two images with epilines representing an array of points from the opposite image. We can take the time to note that the meeting point of each respective array of lines occurs outside of the cameras view. That fictional point is the epipole(or where the center-point of the opposing camera exists in 3D space).





Lastly we should acknowledge how important camera resolution comes into play when attempting to perform 3D reconstruction. This is due to the fact that as a first step in our process we must employ an algorithm that will identifying closely matching pixels in each image so that we can produce an accurate fundamental matrix. With lower resolution we simply have less information to work with and therefore the ability to maintain accuracy is greatly impeded. Nonetheless we are well on our way to full 3D reconstruction with stereo images as our last step is to calculate the depth of the points on our first image using the corresponding epilines of the second. We will explore this further in a forth and final post.

Written by: Keenan James under supervision of Professor Amit Maraj

Machine Learning Computer Science Computer Vision Artificial Intelligence Opencv