3D Reconstruction with Stereo Images -Part 1: Camera Calibration

AI/HUB
Oct 24, 2018 · 9 min read

Introduction

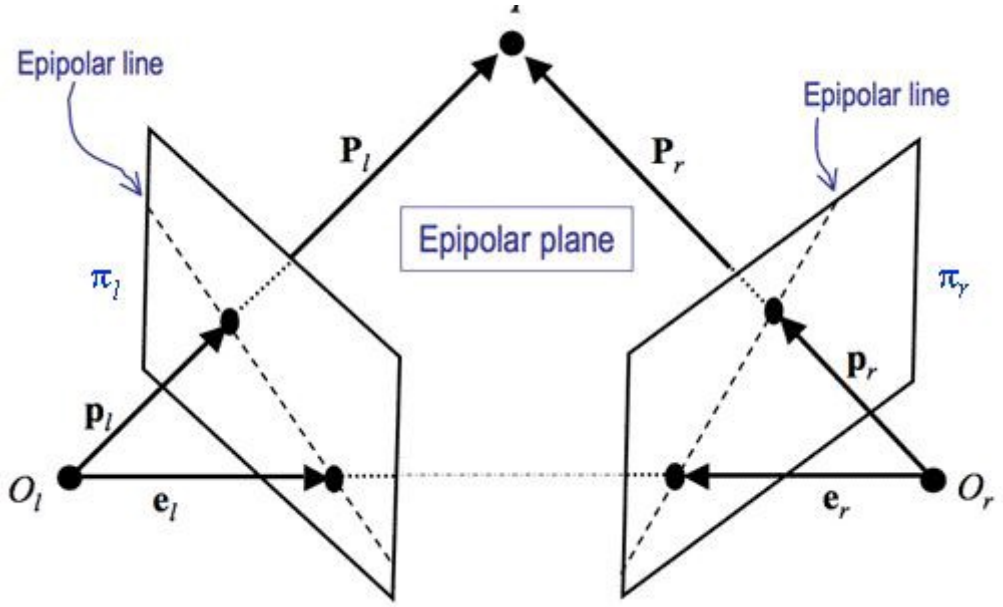
The following series of posts will attempt to explain the essential tools and techniques needed to extract 3D information from a set of 2D images. The process known as 3D reconstruction is a powerful tool with many applications. It can build 3D models of faces, landscapes or other objects by calculating depth information based on pixels from 2D images.

3D Reconstruction

Since modern pinhole digital cameras are relatively cheap and accessible, we can see how this solution offers a far more practical approach to 3D scanning when compared with other more expensive technologies; Lidar to name one. Furthermore, standard 2D camera equipment is already pervasive in our day to day lives, with the rise of smart-phones, surveillance tech, and the Internet of Things, it lends the possibility for 3D

reconstruction to be derived and utilized on preexisting equipment; greatly reducing the financial barrier to entry.

In our specific case we will investigate the strategy of using stereo images to perform 3D reconstruction. Stereo images mean that there are two cameras and 2 images required to calculate a point in 3D space. Essentially the pixels from one image are matched with pixels of the second and epipolar geometry is used to calculate that same point in 3D space. The result, after processing all the relevant pixels in each image is a 3D point map of the pictured object including depth information.



In practice however 3D reconstruction is not that easy and more steps are required in order to see an accurate 3D point map. There will be 4 parts to this series of posts each covering essentials techniques which need to be understood in order to effectively perform 3D reconstruction. The series directly follows tutorials found in Open CV documentation that can be viewed at the link below.

Camera Calibration and 3D Reconstruction - OpenCV 3.0.0-dev documentation

docs.opencv.org

The 4 sections are as follow:

- Camera Calibration

- Pose Estimation
- Epipolar Geometry
- Depth Map from Stereo Images

By the end of the series I am hoping you will have reasonable understanding how to approach a 3D reconstruction or 3D scanning problem using this tool-set. For more specifics please review the Open CV documentation.

Camera Calibration

The Goal for this first post will be to help you learn about camera distortions that are typically present in photos taken with common pinhole cameras. We will also learn the definition and differences between intrinsic vs extrinsic parameters of the camera and why they are needed in our code. Once these parameters are found, we can use Open CV to undistort the image. This is the first step towards full 3D reconstruction.

The common pinhole camera introduces distortion to an image via two major factors. Radial distortion; that essentially makes straight lines appear as slightly curved within an image. The effect of radial distortion exacerbates the further the line is away from the center of an image. Below we can see an example of a chess board whose curved features are highlighted by the straight line in red draw over top of the image.

In order to correct this issue and calibrate the camera accordingly the following equation is used.

Another factor of distortion created by pinhole cameras is that of tangential distortion. This form of distortion occurs when the lens of the camera being utilized is not perfectly aligned with the image plane. Imagine a camera facing one specific direction, in order for tangential distortion to not appear in the image the lens that covers the pinhole light sensor must be perfectly perpendicular to the direction that the camera is facing. However this is not this case as most lenses are slightly askew, this leads objects to appearing slightly closer to the camera on one side of the image as opposed to the other. Although this distortion may be near impossible to see with a quick glance of the human eye, in order to perform 3D reconstruction we will need to employ an equation to correct the image.

What is intended by these equations is to identify five parameters that will be needed by our code and referred to as distortion coefficients. This information will be used to execute methods found within our Open CV package specific to 3D reconstruction.

In addition to the distortion coefficients we will need to identify the intrinsic and extrinsic parameters of our camera.

Intrinsic parameters refers to camera specific information such as focal length and optical centers.

Focal length is used as a basic description for a photographic lens typically represented in (mm). Focal length has nothing to do with the size of a given lens but refers to the point at optical distance in which light rays converge to produce a sharp and detailed image for the underlying digital sensor. For our case it is represented with the below notation.

An optical center of a lens refers to the single point where light rays are undeflected upon passing through its curvature. As a lens holds a convex or concave shape, any other point of the lens will deflect the light rays toward or away from the optical center. In our case it is represented with the below notation.

For the purpose of our code the intrinsic parameters are contained in a 3 x 3 matrix as shown below.

Extrinsic parameters refer to the information that describes the relative position of a camera in 3D space; such as the rotation and translation vectors.

For our application specific to stereo images, the distortions need to be corrected first using the parameters as described above before further attempting to perform 3D reconstruction. To discover these parameters we will need to provide a sample image of a well defined object whose general dimensions are already known. By using the

coordinates of the object in the 2D plane matched with the known dimension of the real object in 3D space, we can calculate the distortion coefficients needed by the methods in the Open CV module. Fortunately we will be able to use the chessboard shown above as our calibration object, with pictures already supplied in Open CV.

Code

Although it is good practice to supply at least 10 images to effectively calibrate our camera, for the sake of simplicity we will just focus on one as an example.

We can use the image of the know object to extract information that can will calibrate our camera. What we will need is both a set of 2D coordinates of the object pictured within the 2D plane of the image in addition to its 3D coordinates in the real world space. The 2D coordinates are referred to as image points while the 3D coordinates are referred to as object points.

Image points are easy to determine as it is simply a matter of measuring a single point on an image in relation to the rest denoted with X,Y coordinates. Object points however are harder to calculate. What we need to know are the X,Y,Z coordinates of the object in real world space. Again for simplicity's sake let's assume that the object stays fixed on the X,Y plane therefore value Z will always equal 0. With this consideration we can now find the X and Y value as the location of a point in 3D space. With this method we can effectively describe the size and location of the object being pictured.

In our case we want to describe the size of a single square on the chessboard with an X and Y value. We should note that since we are using a picture supplied to us that was not taken by ourselves we do not know the exact size of the chessboard that is pictured. If we did we would be able to pass specific values (metric or other) to our coordinate system. Since we do not have this information we can still proceed by using the size of a single square on our chessboard as the standard of measurement. For example the point of the bottom left corner of a square can be denoted 0,0 while the top right hand corner would be 1,1. Assuming that all squares are the same size on the chessboard we can then extrapolate its position in the 3D real world space by referencing this information against the 2D coordinates of the same point in the image itself.

Setup

Next we can proceed with code that will find the pattern in our chessboard. We can start by employing the function cv2.findChessBoardCorners(). This function will need specific information regarding the grid we are trying to find such as 8 x 8 or 4 x 4. In our case we will find a 7 x 6 grid. What returns from this function is the image point coordinates of each corner of the chess board and a boolean value denoting if a complete chess board was found or not.

Once the corners of the chessboard are discovered it is recommended to further increase their accuracy by using the method cv2.cornerSubPix() and then draw the pattern of the chessboard overlaid on the image using cv2.drawChessboardCorners().

The code that covers what we have accomplished so far is shown below.

```
import numpy as np
import cv2
import glob

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

# Arrays to store object points and image points from all the
images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob('*.jpg')

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (7,6),None)

    # If found, add object points, image points (after refining
them)
    if ret == True:
        objpoints.append(objp)

        cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners)

        # Draw and display the corners
        cv2.drawChessboardCorners(img, (7,6), corners2,ret)
        cv2.imshow('img',img)
```

```
            cv2.waitKey(500)

    cv2.destroyAllWindows()
```

The code above will produce the following image with a 7 x 6 grid highlighted on the pictured chessboard.

We can finally proceed with the calibration of our camera and correction of our images. In order to do this we will use the function cv2.calibrateCamera(). It returns the camera matrix and distortion coefficients including rotation and translation vectors for our extrinsic values.

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
imgpoints, gray.shape[::-1],None,None)
```

Undistortion

At this point we are now able to take an image and undistort it using other methods of the Open CV package. First step however is to refine the camera matrix that hold our intrinsic values with cv2.getOptimalNewCameraMatrix(). The code for this is shown below.

```
img = cv2.imread('left12.jpg')
h,  w = img.shape[:2]
newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),1,
(w,h))
```

Next we can proceed with undistortion. The next code block shows the simplest implementation for the methods found within Open CV that will perform this task. We will use the ROI returned from the previous function in order to crop the resulting image.

```
# undistort
dst = cv2.undistort(img, mtx, dist, None, newcameramtx)

# crop the image
x,y,w,h = roi
dst = dst[y:y+h, x:x+w]
cv2.imwrite('calibresult.png',dst)
```

The result is the image shown below that pictures an undistorted chessboard.

. . .

The goal of this post was to progress my own understanding of computer vision applications with hopes it can help others as well.

Written by: Keenan James under supervision of Professor Amit Maraj

Virtual Reality Computer Vision AI Computer Science