Pose Estimation

Project: Marker-based Localization

Niharika Jayanthi, Dheeraj Kamath
Mentor: Sanam Shakya

Goals

- To determine the pose of the camera
- Using **cv2.solvePnP()**, **cv2.solvePnPRansac()** and **cv2.Rodrigues()** functions

Prerequisites

- Camera calibration

Theory

Pose of an object refers to its position and orientation with respect to the camera. To determine the pose, we need to solve the perspective-n-point problem. In perspective-n-point problem, we are given a set of object points and their corresponding image points and we should determine the pose of the image. Trigonometric operations are applied to find the solution. Refer to resources to learn in detail.

The pose of an object is represented in the form of its rotational vector and translational vector. We have seen in camera calibration that using cv2.calibrateCamera() function, we already obtain these two vectors. So why do we need an alternative method to get this information? cv2.calibrateCamera() has a lot of processing, so it consumes more time. Instead, we save the camera matrix and distortion coefficients obtained from cv2.calibrateCamera() and use them in cv2.solvePnP() or cv2.solvePnPRansac() functions.

cv2.solvePnPRansac is better than cv2.solvePnP because this function finds such a pose that minimizes reprojection error, that is, the sum of squared distances between the observed projections imagePoints and the projected objectPoints. The use of RANSAC makes the function resistant to outliers.

Now, after finding the translational vector (tvec) and rotational vector (rvec), we need to extract the required information from them. The distance from camera is acquired from the Z-coordinate in the tvec. This is because we assume the object is always on Z=0 plane and the camera is put at a distance from it. The x-coordinate in tvec denotes the displacement between the origin object and camera axis in x-direction. The angle is obtained from rvec. But first, we need to convert it into a rotation matrix(R). This is done using cv2.Rodrigues() function. After obtaining the matrix, we compare it to the rotational matrix of y-axis.

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Figure 1: Rotation matrix in y-axis

We then take the sine inverse of R[0][2]. This gives us the angle of inclination. The following image depicts this-

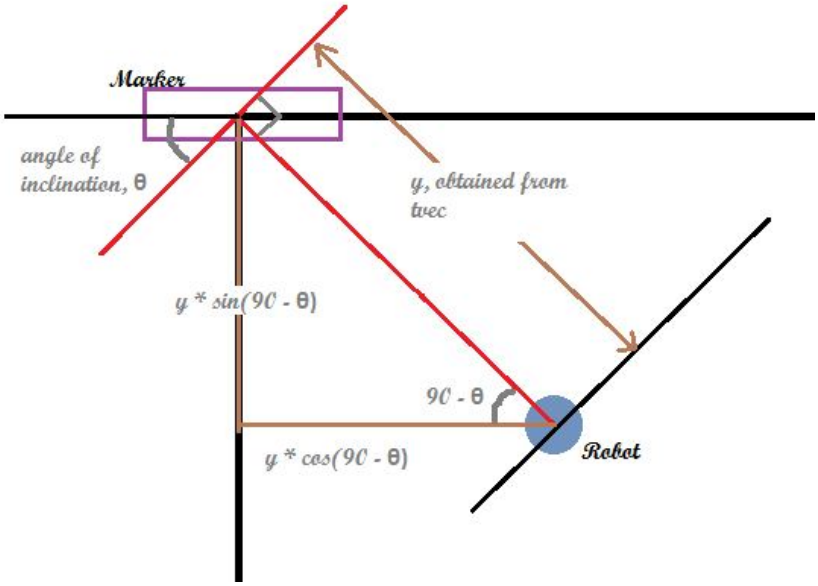

Figure 2: Pose estimation

Code

The main functions and their parameters are-

cv2.solvePnP(objectPoints, imagePoints, cameraMatrix, distCoeffs[, rvec[, tvec[, useExtrinsicGuess[, flags]]]]) → retval, rvec, tvec

where

- **objectPoints** – Array of object points in the object coordinate space, 3xN/Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points.
- **imagePoints** – Array of corresponding image points, 2xN/Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points.
- **cameraMatrix** – Input camera matrix
- **distCoeffs** – Input vector of distortion coefficients (k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6]]) of 4, 5, or 8 elements. If the vector is NULL/empty, the zero distortion coefficients are assumed.
- **rvec** – Output rotation vector (see Rodrigues()) that, together with tvec , brings points from the model coordinate system to the camera coordinate system.
- **tvec** – Output translation vector
- **useExtrinsicGuess** – If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.
- **flags** – Method for solving a PnP problem:

CV_ITERATIVE Iterative

CV_P3P

CV_EPNP

---

cv2.solvePnPRansac(objectPoints, imagePoints, cameraMatrix, distCoeffs[, rvec[, tvec[, useExtrinsicGuess[, iterationsCount[, reprojectionError[, minInliersCount[, inliers[, flags]]]]]]]]) → rvec, tvec, inliers

where -

- **objectPoints** – Array of object points in the object coordinate space, 3xN/Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector can be also passed here.
- **imagePoints** – Array of corresponding image points, 2xN/Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector can be also passed here.
- **cameraMatrix** – Input camera matrix
- **distCoeffs** – Input vector of distortion coefficients (k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6]]) of 4, 5, or 8 elements. If the vector is NULL/empty, the zero distortion coefficients are assumed.
- **rvec** – Output rotation vector (see Rodrigues()) that, together with tvec , brings points from the model coordinate system to the camera coordinate system.
- **tvec** – Output translation vector.
- **useExtrinsicGuess** – If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.
- **iterationsCount** – Number of iterations. reprojectionError – Inlier threshold value used by the RANSAC procedure. The parameter value is the maximum allowed distance between the observed and computed point projections to consider it an inlier.
- **minInliersCount** – Number of inliers. If the algorithm at some stage finds more inliers than minInliersCount , it finishes.
- **inliers** – Output vector that contains indices of inliers in objectPoints and imagePoints .
- **flags** – Method for solving a PnP problem

---

cv2.Rodrigues(src[, dst[, jacobian]]) → dst, jacobian

where -

- **src** – Input rotation vector (3x1 or 1x3) or rotation matrix (3x3).
- **dst** – Output rotation matrix (3x3) or rotation vector (3x1 or 1x3), respectively.
- **jacobian** – Optional output Jacobian matrix, 3x9 or 9x3, which is a matrix of partial derivatives of the output array components with respect to the input array components.

The following code snippet depicts how to find pose-

```
mtx = np.load('matrix.npy')
dist = np.load('distortion.npy')

rvec, tvec, inliers = cv2.solvePnPRansac(objp, imgp, mtx, dist)
print "Rvec\n", rvec
print "\nTvec", tvec


dst, jacobian = cv2.Rodrigues(rvec)
x = tvec[0][0]
y = tvec[2][0]
t = (math.asin(-dst[0][2]))

print "X", x, "Y", y, "Angle", t
print "90-t", (math.pi/2) - t

Rx = y * (math.cos((math.pi/2) - t))
Ry = y * (math.sin((math.pi/2) - t))

print "rx", Rx, "ry", Ry
```

Resources

- http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
- http://projekter.aau.dk/projekter/files/14427578/A_Comparison_of_2D-3D_Pose_Estimation_Methods.pdf
- http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/high/pia/solving.htm