Colorado School of Mines

Computer Vision

Professor William Hoff
Dept of Electrical Engineering & Computer Science
http://inside.mines.edu/~whoff/
Pose Estimation using OpenCV
Pose Estimation of a Known Model

• Assume we have a known object model, with five fiducial targets
• Each target is a “concentric contrasting circle”; it can easily be recognized by thresholding the image and finding connected components

From the 2D-3D point correspondences, we can determine the pose of the model with respect to the camera
Solving for Pose

• The OpenCV function “solvePnP” finds the pose of an object from 3D-2D point correspondences.
  – “PnP” stands for “Perspective n-Point” problem.

• Inputs:
  – A “vector” of 3D object points
  – A “vector” of corresponding 2D image points
  – Intrinsic camera parameters

• Outputs:
  – Rotation vector (axis of rotation, multiplied by angle)
  – Translation vector

This is a C++ vector (a dynamic array)
Camera matrix $K$

See “axis-angle” rotation convention, in Lecture 6
Angle-Axis Representation of Rotation

• A general rotation can be expressed as a rotation $\theta$ about an axis $\mathbf{k}$

\[
R_\mathbf{k}(\theta) = \begin{pmatrix}
k_x k_x v \theta + c \theta & k_x k_y v \theta - k_z s \theta & k_x k_z v \theta + k_y s \theta \\
k_x k_y v \theta + k_z s \theta & k_y k_y v \theta + c \theta & k_y k_z v \theta - k_x s \theta \\
k_x k_z v \theta - k_y s \theta & k_y k_z v \theta + k_x s \theta & k_z k_z v \theta + c \theta
\end{pmatrix}
\]

where

\[
c \theta = \cos \theta, \ s \theta = \sin \theta, \ v \theta = 1 - \cos \theta
\]

\[
\hat{\mathbf{k}} = (k_x, k_y, k_z)^T
\]

• The inverse solution (i.e., given a rotation matrix, find $\mathbf{k}$ and $\theta$):

\[
\theta = \cos \left( \frac{r_{11} + r_{22} + r_{33} - 1}{2} \right)
\]

\[
\hat{\mathbf{k}} = \frac{1}{2 \sin \theta} \begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix}
\]

Note that $(\mathbf{-k}, -\theta)$ is also a solution.
bool cv::solvePnP ( InputArray  objectPoints, 
InputArray  imagePoints, 
InputArray  cameraMatrix, 
InputArray  distCoeffs, 
OutputArray rvec, 
OutputArray tvec, 

    useExtrinsicGuess = false, 
    int flags = SOLVEPNP_ITERATIVE 
)

Finds an object pose from 3D-2D point correspondences.

Parameters

- **objectPoints**: Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3f> can be also passed here.
- **imagePoints**: Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2f> can be also passed here.
- **cameraMatrix**: Input camera matrix $A = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$.
- **distCoeffs**: Input vector of distortion coefficients $(k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, k_7, k_8, k_9[, k_{10}, k_{11}, k_{12}]]]]])$ of 4, 5, 8, 12 or 14 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**: Parameter used for SOLVEPNP_ITERATIVE. 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.
Example

- This code creates a “vector” of 3D points lying in the Z=0 plane

```cpp
std::vector<cv::Point3d> pointsObject;
pointsModel.push_back(cv::Point3d(0.0, 0.0, 0.0));
pointsModel.push_back(cv::Point3d(1.0, 0.0, 0.0));
pointsModel.push_back(cv::Point3d(1.0, 1.0, 0.0));
pointsModel.push_back(cv::Point3d(0.0, 1.0, 0.0));
```

- This “vector” holds the corresponding detected image points

```cpp
std::vector<cv::Point2d> pointsImage;
```

- This computes the pose

```cpp
bool foundPose = cv::solvePnP(
    pointsObject, // object points
    pointsImage,  // image points
    K,            // intrinsic camera parameter matrix
    dist,         // distortion coefficients
    rotVec,transVec); // output rotation and translation
```
Projecting Points

• For visualization, we can project a 3D point onto an image using the OpenCV function "projectPoints"

• Example: this backprojects the original 3D points onto an image using the found pose

\[
\text{cv::projectPoints(pointsObject, rotVec, transVec, K, dist, pointsImage);}
\]