Head Pose Estimation from Face Landmark Coordinates in Static Images

Xiaohai Zhang
Senior Software Engineer - Machine Learning
https://xiaohaionline.com

***Abstract*—In this article, we first presented a mean 3D face model from [1], [2], with 21 facial landmark coordinates, in a easy-to-use CSV file format. We reviewed the popular *POSIT* algorithm for head pose estimation. We then presented our simplified derivation of the *POSIT* algorithm. At the end, we provided a Python implementation of the modernPosit algorithm, and demonstrated the computation of face bounding rectangle, based on computed head pose. The novelty lies in the simplified interpretation of the *POSIT* algorithm.**

1. Introduction

For face detection and localization tasks, we often need to draw bounding rectangles around faces in images, either to extract face images as intermediate image files for model training, or to measure face detection accuracy of algorithms under development.

Manually marking face boundaries, in the form of rectangles or ellipses, is error prone and lacks objective criteria. One of the most popular approaches, is to mark prominent facial landmarks manually, and then use pose estimation algorithms to generate the bounding rectangles or ellipses. Head pose estimation has other important applications as well. One such example is to understand viewing directions of persons in gaming, virtual reality and automotive industries.

Determining head pose requires knowledge about coordinates of some prominent facial landmarks, such as corners and center of one's eyes, nose and mouse, both in the image pixel space and the wold space. The mapping between the world space coordinates and the image pixel space coordinates of the same facial landmarks provides clue for us to compute the head pose. The head pose estimation problem can be generalized to be the problem of finding out the transformation matrix, from the world space to the image space, given n feature points in both the world space and the image space. This is also coined as *Perspective n Point problem* (or PnP problem). For example, in OpenCV [3], the routine is called *solvePnP*.

However it is practically very difficult to have real facial landmarks' world coordinates corresponding to face pixels in images, as it is hard to obtain a 3D scan of one's face. But thanks to limited variation among human faces, we are able to use a mean 3D face model in [2] to provide approximate world coordinates of facial landmarks, without incurring significant errors. The facial landmark coordinates in image pixels are usually manually marked. In the next section we will first present the mean 3D face model.

There is an immense body of literature on head pose estimation. One of the prominent algorithms is called *POSIT* [4]. This algorithm is computationally very efficient and stable. We will give an interpretation of the *POSIT* algorithm different from that described in [4], with the hope of making it even easier to understand and implement. We finally present the Python codes for the algorithm.

2. The Mean 3D Face Model

In [2], a mean 3D face model is built out of 350 facial laser scans, consisting of different subjects exhibiting a wide variability in race, gender and age. The landmark coordinates are stored into a SQLite table in [1]. While SQLite is convenient for scripts, it is not friendly for human to read in a text editor. And thus it was converted into a CSV file that can be easily downloaded and processed with scripts. The download URL is given in [5].

The data consists of 21 featured landmarks, with nose center as the origin of the world coordinate. The full list of facial landmarks whose world coordinates are provided is listed in Table 1. The actual world coordinates can be found from the CSV file in [5]. The coordinate frame of reference is depicted in Figure 1.

3. Review of the POSIT Algorithm

The *POSIT* algorithm was proposed by D. F. Dementhon and L. S. Davis [4] in 1995. In this section, we are going to review the algorithm.

The set of known world feature points are denoted by M_i, for $i = 1, 2, ..., N$, with their coordinates in world space as (X_i, Y_i, Z_i), and with their coordinates in image space as (x_i, y_i, z_i). The coordinates in world space and image space are associated by a transformation as expressed in equation 1.

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \quad (1)$$

Each of the world feature point has a corresponding image pixel, and the image pixel coordinates are denoted as (u_i, v_i). The world feature point coordinates in the image

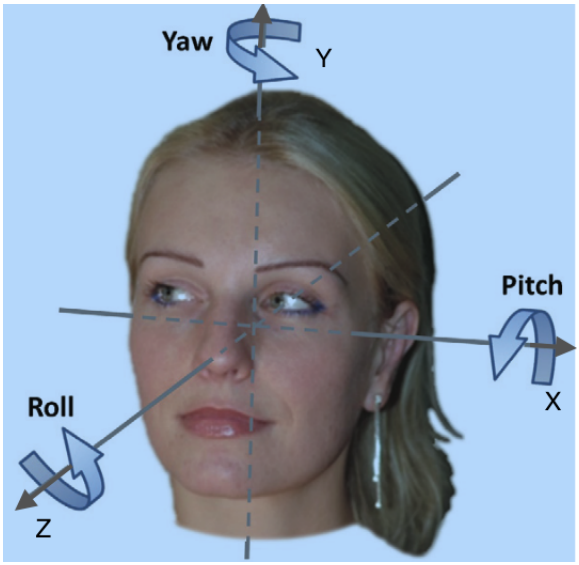


Figure 1. The 3D Face Model Coordinate System

Id	Description	Code
1	LeftBrowLeftCorner	LBLC
2	LeftBrowCenter	LBC
3	LeftBrowRightCorner	LBRC
4	RightBrowLeftCorner	RBLC
5	RightBrowCenter	RBC
6	RightBrowRightCorner	RBRC
7	LeftEyeLeftCorner	LELC
8	LeftEyeCenter	LEC
9	LeftEyeRightCorner	LERC
10	RightEyeLeftCorner	RELC
11	RightEyeCenter	REC
12	RightEyeRightCorner	RERC
13	LeftEar	LE
14	NoseLeft	LN
15	NoseCenter	NC
16	NoseRight	RN
17	RightEar	RE
18	MouthLeftCorner	MLC
19	MouthCenter	MC
20	MouthRightCorner	MRC
21	ChinCenter	CC
28	HeadCenter	HC

TABLE 1. FEATURES OF THE 3D FACE MODEL

space, and the image pixel coordinates, are associated by a perspective projection described in equation 2.

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \frac{1}{z_i} \begin{bmatrix} f_x & o & c_x \\ 0 & f_y & c_y \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \tag{2}$$

The goal of head pose estimation, or *POSIT* in this section, is to determine the rotation matrix, given the two sets of coordinates (u_i, v_i) and (X_i, Y_i, Z_i). From the rotation matrix, one can determine the yaw, pitch and roll values, as illustrated in Figure 1.

The coordinate notations are depicted in figure 2. One can start with a simpler scenario, where all M_i lie in a plane parallel, denoted as K, to the image plane, denoted as G,

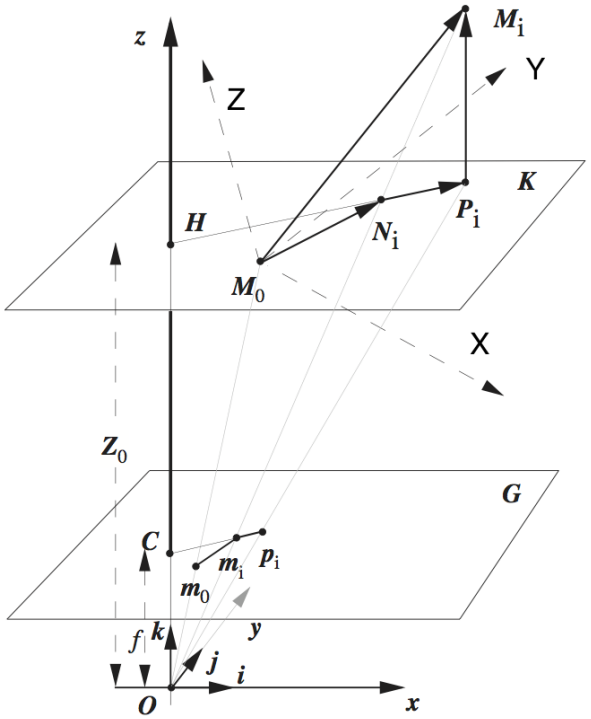


Figure 2. The coordinate system notation

with a distance of T_z from the origin of the camera frame of reference. In this simple scenario, the perspective projection transformation in Equation 2 becomes a simpler scaling as in equation 3.

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = T_z \begin{bmatrix} \frac{u_i - c_x}{f_x} \\ \frac{v_i - c_y}{f_y} \end{bmatrix} \tag{3}$$

Combining equation 1 and 3 leads to equation 4.

$$\begin{bmatrix} \frac{u_i - c_x}{f_x} \\ \frac{v_i - c_y}{f_y} \end{bmatrix} = \frac{1}{T_z} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \tag{4}$$

Combining equation 4 for all $i = 1 \cdots N$, we get equations 5, 6 and 7.

$$\mathcal{I} = \begin{bmatrix} \frac{u_1 - c_x}{f_x} & \frac{u_2 - c_x}{f_x} & \cdots & \frac{u_N - c_x}{f_x} \\ \frac{v_1 - c_y}{f_y} & \frac{v_2 - c_y}{f_y} & \cdots & \frac{v_N - c_y}{f_y} \end{bmatrix} \tag{5}$$

$$\mathcal{M} = \begin{bmatrix} X_1 & X_2 & \cdots & X_N \\ Y_1 & Y_2 & \cdots & Y_N \\ Z_1 & Z_2 & \cdots & Z_N \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{6}$$

$$\frac{1}{T_z} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \end{bmatrix} = \mathcal{I}\left(\mathcal{M}^T \mathcal{M}\right)^{-1} \mathcal{M}^T \tag{7}$$

Once the matrix from the right hand side of equation 7 is calculated, normalized row vectors of the first three columns of the matrix are the solution for (r_{11}, r_{12}, r_{13}) and (r_{21}, r_{22}, r_{23}), and the two 3-column row vectors' norms are

$\frac{1}{T_z}$. The fourth column of the matrix provides solution for T_x, T_y given T_z. Finally, (r_{31}, r_{32}, r_{33}) can be obtained as the cross product of the prior two row vectors computed. This is the *POS* algorithm, or *Pose from Orthography and Scaling*.

However, in practical situations, world feature points likely do not lie in the same plane parallel to the image plane. The idea then is to then first project M_i orthographically onto the plane K parallel to the image plane. Let the projected point be denoted as P_i, and P_i's coresponding image pixel be denoted as p_i.

Let P_i's coordinates in world space be denoted by (X_i', Y_i', Z_i'), P_i's coordinates in image space be denoted by (x_i', y_i', z_i'), and (u_i', v_i') denote p_i's image pixel coordinates. Observing that $x_i' = x_i$ and $y_i' = y_i$, and noting equation 1, we get equation 8.

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \tag{8}$$

Given that P_i are coplanar, applying equation 3 gives equation 9.

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = T_z \begin{bmatrix} \frac{u_i' - c_x}{f_x} \\ \frac{v_i' - c_y}{f_y} \end{bmatrix} \tag{9}$$

Combining equation 8 and 9, we can get an equation 10.

$$\begin{bmatrix} \frac{u_i' - c_x}{f_x} \\ \frac{v_i' - c_y}{f_y} \end{bmatrix} = \frac{1}{T_z} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \tag{10}$$

Equation 10 is very similar to equation 4, except that the left side is u_i' and v_i' instead of u_i and v_i. This means that, even if the world features points are not coplanar, if we can properly compute (u_i', v_i'), we can still apply the afore mentioned *POS* algorithm to compute the rotation matrix.

One important observation from figure 2 is equation 11 below.

$$\frac{Cp_i}{Cm_i} = \frac{HP_i}{HN_i} = \frac{z_i}{T_z} \tag{11}$$

Equation 11 leads to equation 12.

$$\begin{bmatrix} u_i' - c_x \\ v_i' - c_y \end{bmatrix} = \frac{z_i}{T_z} \begin{bmatrix} u_i - c_x \\ v_i - c_y \end{bmatrix} \tag{12}$$

The algorithm can be described with the following iterative steps.

1) Start by approximating (u_i', v_i') with (u_i, v_i), and use *POS* algorithm to compute an initial estimation of the transformation matrix consisting of the rotation matrix and the translation vector.
2) With an estimate of the transformation matrix, z_i can be computed from equation 1. With both the z_i and the T_z, we can use equation 12 to update the estimate of (u_i', v_i').
3) With updated (u_i', v_i'), we can compute an updated translation matrix with equation 10 and the *POS* algorithm. We then go back to the prior step to iterate until convergence criteria is met.

The iterative procedure of applying the *POS* algorithm is then the *POSIT* algorithm.

4. POSIT from a Different Perspective

In this section an explanation of *POSIT* algorithm from a different perspective is given, with the hope that it might be easier to understand, and more importantly, easier to generalize.

Using similar notations in [3], except replacing the symbol s with z, the z coordinate of the world point (X, Y, Z) in the image space. We have the transformation and perspective projection model described by Equation 13.

$$z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{13}$$

and equivalently by equation 14 below.

$$z \begin{bmatrix} \frac{u - c_x}{f_x} \\ \frac{v - c_y}{f_y} \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{14}$$

Equation 14 can be further rewritten into equation 15.

$$\begin{bmatrix} \frac{z}{T_z} \frac{u - c_x}{f_x} \\ \frac{z}{T_z} \frac{v - c_y}{f_y} \\ \frac{z}{T_z} \end{bmatrix} = \frac{1}{T_z} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{15}$$

With equation 15 the *POSIT* algorithm can be directly derived as the following:

1) For all feature points, we start by replacing $\frac{z}{T_z}$ in equation 15 with an approximation of value 1. We can then use the top two sub-equations of the equation 15 to compute the initial values of the rotation matrix as well as the translation vector. This is the *POS* algorithm.
2) Use the third bottom sub-equation of the equation 15 to obtain an updated value of $\frac{z}{T_z}$ (or ϵ_i in *POSIT* [4]) for each feature point. With the new values of $\frac{z}{T_z}$, we can utilize the *POS* algorithm again to obtain an updated rotation matrix and translation vector.
3) Iterate with the above step until convergence.

5. The Modern POSIT Implementation

The implementation of the *POSIT* algorithm in Python is attached below. It deviates from the afore described

algorithm only in the representation of the coordinate vectors. In the aforementioned algorithms, the coordinates are represented as column vectors, while in the implementation the coordinates are represented as row vectors. This change is to simplify the stacking of multiple coordinates as input parameters, which is also consistent with industry practice. The difference in notation only impacts the order of matrix concatenations and multiplications.

Listing 1. the *POSIT* implementation in Python

```
import numpy as np

#Usage:
#  [rot, trans] = modernPosit(
#     imagePts, worldPts,
#     focalLength, center)
#  Return rotation and translation
#  of world object given world
#  points and image points
#
#Inputs
#  imagePts is of nbPts x 2
#  worldPts is of nbPts x 3
#  focalLength is the focal length
#    of the camera in PIXELS,
#    usually one and a half of the
#    imag width in PIXELS
#  center is a row vector with the
#  two components of the image
#  center
#
#Outputs:
# rot: a 3 x 3 rotation matrix of
#    scene with respect to camera
# trans: 3 x 1 translation vector
#    from projection center of
#    camera to ORIGIN of
#  coordinate system of object/scene
#
#Example of input:
#
#cube = [ 0 0 0; 10 0 0; 10 10 0;
#      0 10 0; 0 0 10; 10 0 10;
#     10 10 10; 0 10 10]
#cubeImage = [0 0; 80 -93;
#      245 -77; 185 32;
#      32 135; 99 35; 247 62;
#      195 179]
#focalLength = 760
#[rot, trans] = modernPosit(
#    cubeImage, cube, focalLength)
#
#Outputs:
# rot = [ 0.4910 0.8493 0.1911;
#     -0.5699 0.1461 0.8093;
#     0.6594 -0.5063 0.5558];
#trans = [0.0033; 0.0046; 40.0728];
#
# when computation stops
#     after 4 iterations

def modernPosit(imagePts, worldPts,
                focalLength, center):
  nbPoints = np.shape(imagePts)[0]

  # centered & scaled pixel coordinates
  centeredImage = np.divide(
      np.subtract(imagePts, center),
      focalLength)
  ui = centeredImage[:, 0]
  vi = centeredImage[:, 1]

  # homogeneous world coordinates
  homogeneousWorldPts = np.append(
          worldPts, np.ones(
              (nbPoints, 1)), 1)

  # pseudoinverse
  objectMat = np.linalg.pinv(
          homogeneousWorldPts)

  converged = 0
  count = 0
  while converged == 0:
    # POS part of the algorithm
    # rotation vectors
    r1T= np.matmul(objectMat, ui)
    r2T = np.matmul(objectMat, vi)
    # 1/Tz1 is norm of r1T
    Tz1 = 1/np.linalg.norm(r1T[0:3])
    # 1/Tz2 is norm of r2T
    Tz2 = 1/np.linalg.norm(r2T[0:3])
    # geometric average
    Tz = np.sqrt(Tz1*Tz2)
    r1N = np.multiply(r1T, Tz)
    r2N = np.multiply(r2T, Tz)
    r1 = r1N[0:3]
    r2 = r2N[0:3]
    r3 = np.cross(r1,r2)
    r3T= np.append(r3, Tz)
    Tx = r1N[3]
    Ty = r2N[3]

    # Now update the z/T_z or epsilon
    # then ui, vi
    epsiloni= np.matmul(
            homogeneousWorldPts,
            np.divide(r3T, Tz))
    oldUi = ui
    oldVi = vi
    ui = np.multiply(epsiloni,
            centeredImage[:,0])
    vi = np.multiply(epsiloni,
            centeredImage[:,1])

    # check for convergence
```

```
    deltaUi = ui - oldUi
    deltaVi = vi - oldVi
    delta = np.square(
      focalLength) * \
      (
        np.square(
          np.linalg.norm(deltaUi)) +
        np.square(
            np.linalg.norm(deltaVi)))
    converged = 1 if count > 0 and \
        delta < 1 else 0
    count = count + 1

  trans = np.array([Tx, Ty, Tz],
    np.float64)
  rot = np.array([r1, r2, r3],
    np.float64)
  return rot, trans
```

6. The Head Pose Estimation

Given some marked facial landmarks in a static image, and, for each of the landmarks, the corresponding points in the mean 3D face model described in section 2 can be identified, both the worldPts from the 3D face data, and the imagePts from the manual marking are available. With additional meta information, such as the size of the image, the modernPosit algorithm in list 1 can be invoked to compute the head pose, the rotation matrix and the translation vector.

To further draw a bounding rectangle or ellipse around the face in the image, a 3D ellipsoid can be first parameterized in the world space, using the 3D mean face model, and then transformed into the image pixel space to be drawn on the image.

A sample image with the bounding rectangle and ellipse drawn is show in figure 3.

Landmark pixel coordinate data are show in table 2.

Description	u	v
LeftBrowLeftCorner	212.0	150.0
LeftBrowCenter	233.0	142.0
LeftBrowRightCorner	254.0	139.0
RightBrowLeftCorner	281.0	137.0
RightBrowCenter	291.0	132.0
LeftEyeLeftCorner	221.0	163.0
LeftEyeCenter	238.0	161.0
LeftEyeRightCorner	247.0	164.0
RightEyeCenter	281.0	156.0
LeftEar	156.0	220.0
NoseLeft	263.0	210.0
NoseCenter	296.0	203.0
MouthLeftCorner	249.0	240.0
MouthCenter	281.0	233.0
MouthRightCorner	283.0	235.0
ChinCenter	268.0	281.0

TABLE 2. FEATURE POINTS OF THE FACE DRAWN



Figure 3. Face bounding rectangle and bounding ellipse derived with transformation matrix

References

[1] P. M. R. Martin Koestinger, Paul Wohlhart and H. Bischof, "Annotated Facial Landmarks in the Wild: A Large-scale, Real-world Database for Facial Landmark Localization," in *Proc. First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*, 2011.

[2] M. Storer, M. Urschler, and H. Bischof, "3D-MAM: 3D morphable appearance model for efficient fine head pose estimation from still images," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, Sept 2009, pp. 192–199.

[3] OpenCV. Camera Calibration and 3D Reconstruction. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[4] D. F. Dementhon and L. S. Davis, "Model-based object pose in 25 lines of code," in *Int. Journal of Computer Vision*, no. 15, 1995, pp. 123–141.

[5] X. Zhang. Face 3D Model File in CSV. [Online]. Available: https://xiaohaionline.com/wp-content/uploads/2017/10/average-face3d.csv

[6] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," in *Int. Journal of Computer Vision*, 2008.