

Head Pose Estimation using OpenCV and Dlib

SEPTEMBER 26, 2016 BY **SATYA MALLICK** — 84 COMMENTS



In this tutorial we will learn how to estimate the pose of a human head in a photo using OpenCV and Dlib.

In many applications, we need to know how the head is tilted with respect to a camera. In a virtual reality application, for example, one can use the pose of the head to render the right view of the scene. In a driver assistance system, a camera looking at a driver's face in a vehicle can use head pose estimation to see if the driver is

paying attention to the road. And of course one can use head pose based gestures to control a hands-free application / game. For example, yawing your head left to right can signify a NO. But if you are from southern India, it can signify a YES! To understand the full repertoire of head pose based gestures used by my fellow Indians, please partake in the hilarious video below.

Indian Headshakes | What do they mean?



My point is that estimating the head pose is useful. Sometimes.

If you want to quickly jump into code go to the the [code section](#). To access code in all the tutorials in this blog, please [subscribe](#) to our newsletter.

Before proceeding with the tutorial, I want to point out that this post belongs to a series I have written on face processing. Some of the articles below are useful in understanding this post and others complement it.

1. Facial Landmark Detection
2. Face Swap
3. Face Averaging

4. Face Morphing

What is pose estimation ?

In computer vision the pose of an object refers to its relative orientation and position with respect to a camera. You can change the pose by either moving the object with respect to the camera, or the camera with respect to the object.

The pose estimation problem described in this tutorial is often referred to as **Perspective-n-Point** problem or PNP in computer vision jargon. As we shall see in the following sections in more detail, in this problem the goal is to find the pose of an object when we have a calibrated camera, and we know the locations of **n** 3D points on the object and the corresponding 2D projections in the image.

How to mathematically represent camera motion ?

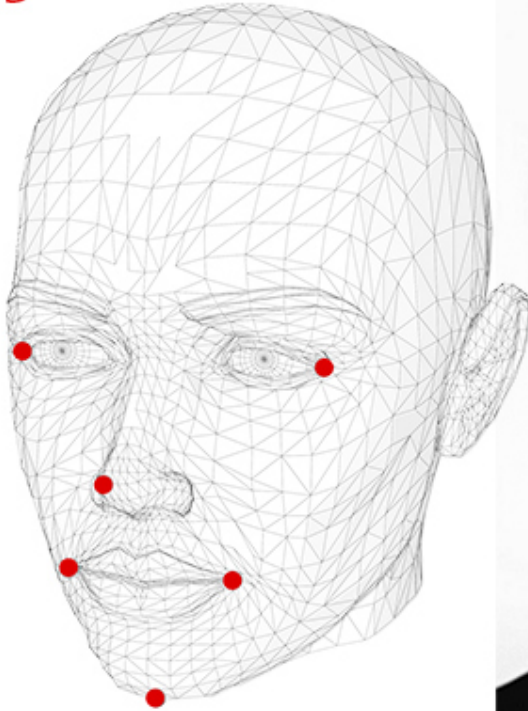
A 3D rigid object has only two kinds of motions with respect to a camera.

1. **Translation** : Moving the camera from its current 3D location (X, Y, Z) to a new 3D location (X', Y', Z') is called translation. As you can see translation has 3 degrees of freedom — you can move in the X, Y or Z direction. Translation is represented by a vector **t** which is equal to $(X' - X, Y' - Y, Z' - Z)$.
2. **Rotation** : You can also rotate the camera about the **X**, **Y** and **Z** axes. A rotation, therefore, also has three degrees of freedom. There are many ways of representing rotation. You can represent it using **Euler angles** (roll, pitch and yaw), a **3 × 3 rotation matrix**, or a **direction of rotation (i.e. axis) and angle**.

So, estimating the pose of a 3D object means finding 6 numbers — three for translation and three for rotation.

What do you need for pose estimation ?

3D



2D



To calculate the 3D pose of an object in an image you need the following information

1. **2D coordinates of a few points** : You need the 2D (x,y) locations of a few points in the image. In the case of a face, you could choose the corners of the eyes, the tip of the nose, corners of the mouth etc. Dlib's **facial landmark detector** provides us with many points to choose from. In this tutorial, we will use the tip of the nose, the chin, the left corner of the left eye, the right corner of the right eye, the left corner of the mouth, and the right corner of the mouth.
2. **3D locations of the same points** : You also need the 3D location of the 2D feature points. You might be thinking that you need a 3D model of the person in the photo to get the 3D locations. Ideally yes, but in practice, you don't. A generic 3D model will suffice. Where do you get a 3D model of a head from ? Well, you really don't need a full 3D model. You just need the 3D locations of a few points in some arbitrary reference frame. In this tutorial, we are going to use the following 3D points.
 1. Tip of the nose : (0.0, 0.0, 0.0)
 2. Chin : (0.0, -330.0, -65.0)
 3. Left corner of the left eye : (-225.0f, 170.0f, -135.0)
 4. Right corner of the right eye : (225.0, 170.0, -135.0)
 5. Left corner of the mouth : (-150.0, -150.0, -125.0)

6. Right corner of the mouth : (150.0, -150.0, -125.0)

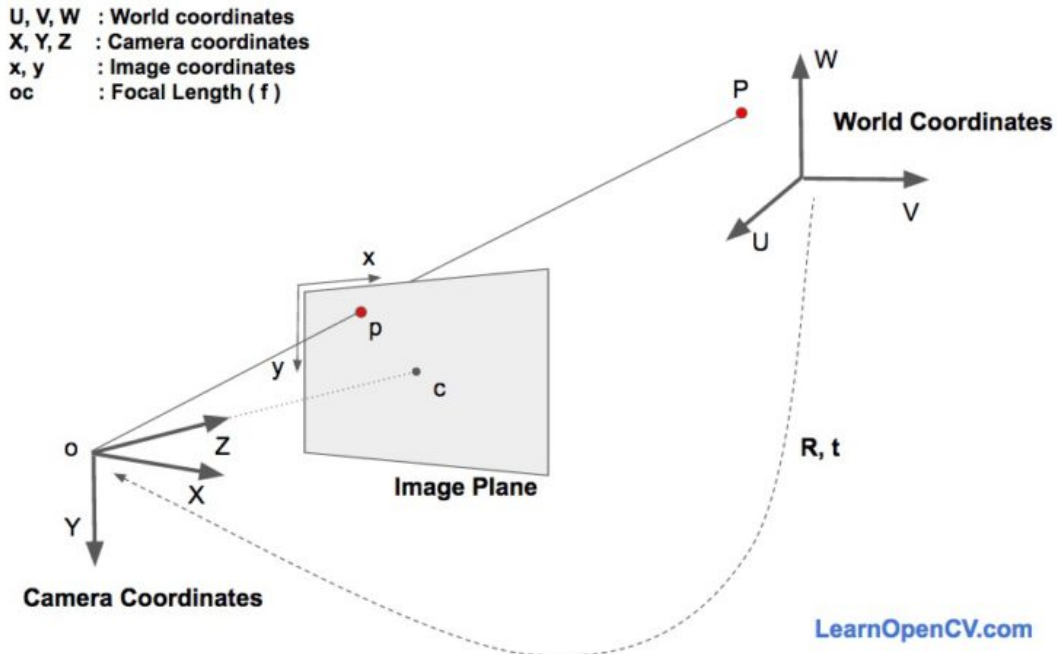
Note that the above points are in some arbitrary reference frame / coordinate system. This is called the **World Coordinates** (a.k.a Model Coordinates in OpenCV docs) .

3. **Intrinsic parameters of the camera.** As mentioned before, in this problem the camera is assumed to be calibrated. In other words, you need to know the focal length of the camera, the optical center in the image and the radial distortion parameters. So you need to **calibrate your camera**. Of course, for the lazy dudes and dudettes among us, this is too much work. Can I supply a hack ? Of course, I can! We are already in approximation land by not using an accurate 3D model. We can approximate the optical center by the center of the image, **approximate the focal length** by the width of the image in pixels and assume that radial distortion does not exist. Boom! you did not even have to get up from your couch!

How do pose estimation algorithms work ?

There are several algorithms for pose estimation. The first known algorithm dates back to 1841. It is beyond the scope of this post to explain the details of these algorithms but here is a general idea.

There are three coordinate systems in play here. The 3D coordinates of the various facial features shown above are in **world coordinates**. If we knew the rotation and translation (i.e. pose), we could transform the 3D points in world coordinates to 3D points in **camera coordinates**. The 3D points in camera coordinates can be projected onto the image plane (i.e. **image coordinate system**) using the intrinsic parameters of the camera (focal length, optical center etc.).



Let's dive into the image formation equation to understand how these above coordinate systems work. In the figure above, o is the center of the camera and plane shown in the figure is the image plane. We are interested in finding out what equations govern the projection p of the 3D point P onto the image plane.

Let's assume we know the location (U, V, W) of a 3D point P in World Coordinates. If we know the rotation \mathbf{R} (a 3×3 matrix) and translation \mathbf{t} (a 3×1 vector), of the world coordinates with respect to the camera coordinates, we can calculate the location (X, Y, Z) of the point P in the camera coordinate system using the following equation.

$$\begin{aligned}
 \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} &= \mathbf{R} \begin{bmatrix} U \\ V \\ W \end{bmatrix} + \mathbf{t} \\
 \Rightarrow \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} &= [\mathbf{R} \mid \mathbf{t}] \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}
 \end{aligned} \tag{1}$$

In expanded form, the above equation looks like this

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \tag{2}$$

If you have ever taken a Linear Algebra class, you will recognize that if we knew sufficient number of point correspondences (i.e. (X, Y, Z) and (U, V, W)), the above is a linear system of equations where the r_{ij} and (t_x, t_y, t_z) are unknowns and you can trivially solve for the unknowns.

As you will see in the next section, we know (X, Y, Z) only up to an unknown scale, and so we do not have a simple linear system.

Direct Linear Transform

We do know many points on the 3D model (i.e. (U, V, W)), but we do not know (X, Y, Z) . We only know the location of the 2D points (i.e. (x, y)). In the absence of radial distortion, the coordinates (x, y) of point p in the image coordinates is given by

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = s \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (3)$$

where, f_x and f_y are the focal lengths in the x and y directions, and (c_x, c_y) is the optical center. Things get slightly more complicated when radial distortion is involved and for the purpose of simplicity I am leaving it out.

What about that s in the equation ? It is an unknown scale factor. It exists in the equation due to the fact that in any image we do not know the depth. If you join any point P in 3D to the center o of the camera, the point p , where the ray intersects the image plane is the image of P . Note that all the points along the ray joining the center of the camera and point P produce the same image. In other words, using the above equation, you can only obtain (X, Y, Z) up to a scale s .

Now this messes up equation 2 because it is no longer the nice linear equation we know how to solve. Our equation looks more like

$$s \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \quad (4)$$

Fortunately, the equation of the above form can be solved using some algebraic wizardry using a method called **Direct Linear Transform (DLT)**. You can use DLT any time you find a problem where the equation is almost linear but is off by an unknown scale.

Levenberg-Marquardt Optimization

The DLT solution mentioned above is not very accurate because of the following reasons . First, rotation \mathbf{R} has three degrees of freedom but the matrix representation used in the DLT solution has 9 numbers. There is nothing in the DLT solution that forces the estimated 3×3 matrix to be a rotation matrix. More importantly, the DLT solution does not minimize the correct objective function. Ideally, we want to minimize the **reprojection error** that is described below.

As shown in the equations **2** and **3**, if we knew the right pose (\mathbf{R} and \mathbf{t}), we could predict the 2D locations of the 3D facial points on the image by projecting the 3D points onto the 2D image. In other words, if we knew \mathbf{R} and \mathbf{t} we could find the point \mathbf{p} in the image for every 3D point \mathbf{P} .

We also know the 2D facial feature points (using Dlib or manual clicks). We can look at the distance between projected 3D points and 2D facial features. When the estimated pose is perfect, the 3D points projected onto the image plane will line up almost perfectly with the 2D facial features. When the pose estimate is incorrect, we can calculate a **re-projection error** measure — the sum of squared distances between the projected 3D points and 2D facial feature points.

As mentioned earlier, an approximate estimate of the pose (\mathbf{R} and \mathbf{t}) can be found using the DLT solution. A naive way to improve the DLT solution would be to randomly change the pose (\mathbf{R} and \mathbf{t}) slightly and check if the reprojection error decreases. If it does, we can accept the new estimate of the pose. We can keep perturbing \mathbf{R} and \mathbf{t} again and again to find better estimates. While this procedure will work, it will be very slow. Turns out there are principled ways to iteratively change the values of \mathbf{R} and \mathbf{t} so that the reprojection error decreases. One such method is called **Levenberg-Marquardt** optimization. Check out more details on [Wikipedia](#).

OpenCV solvePnP

In OpenCV the function **solvePnP** and **solvePnP Ransac** can be used to estimate pose.

solvePnP implements several algorithms for pose estimation which can be selected using the parameter **flag**. By default it uses the flag **SOLVEPNP_ITERATIVE** which is essentially the DLT solution followed by Levenberg-Marquardt optimization.

SOLVEPNP_P3P uses only 3 points for calculating the pose and it should be used only when using **solvePnP Ransac**.

In OpenCV 3, two new methods have been introduced — **SOLVEPNP_DLS** and **SOLVEPNP_UPNP**. The interesting thing about **SOLVEPNP_UPNP** is that it tries to estimate camera internal parameters also.

C++: `bool solvePnP(InputArray objectPoints, InputArray imagePoints, InputArray cameraMatrix, InputArray distCoeffs, OutputArray rvec, OutputArray tvec, bool useExtrinsicGuess=false, int flags=SOLVEPNP_ITERATIVE)`

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

Parameters:

objectPoints – Array of object points in the world coordinate space. I usually pass vector of N 3D points. You can also pass Mat of size Nx3 (or 3xN) single channel matrix, or Nx1 (or 1xN) 3 channel matrix. I would highly recommend using a vector instead.

imagePoints – Array of corresponding image points. You should pass a vector of N 2D points. But you may also pass 2xN (or Nx2) 1-channel or 1xN (or Nx1) 2-channel Mat, where N is the number of points.

cameraMatrix – Input camera matrix $A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$. Note that f_x, f_y can be approximated by the image width in pixels under certain circumstances, and the c_x and c_y can be the coordinates of the image center.

distCoeffs – Input vector of distortion coefficients ($k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6, s_1, s_2, s_3, s_4$) of 4, 5, 8 or 12 elements. If the vector is NULL/empty, the zero distortion coefficients are assumed. Unless you are working with a Go-Pro like camera where the distortion is huge, we can simply set this to NULL. If you are working with a lens with high distortion, I recommend doing a full camera calibration.

rvec – Output rotation vector.

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.

flags –

Method for solving a PnP problem:

SOLVEPNP_ITERATIVE Iterative method is based on Levenberg-Marquardt optimization. In this case, the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections imagePoints and the projected (using projectPoints()) objectPoints .

SOLVEPNP_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang “Complete Solution Classification for the Perspective-Three-Point Problem”. In this case, the function requires exactly four object and image points.

SOLVEPNP_EPNP Method has been introduced by F.Moreno-Noguer, V.Lepetit and P.Fua in the paper “EPnP: Efficient Perspective-n-Point Camera Pose Estimation”.

The flags below are only available for **OpenCV 3**

SOLVEPNP_DLS Method is based on the paper of Joel A. Hesch and Stergios I. Roumeliotis. "A Direct Least-Squares (DLS) Method for PnP".

SOLVEPNP_UPNP Method is based on the paper of A.Penate-Sanchez, J.Andrade-Cetto, F.Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation". In this case the function also estimates the parameters f_x and f_y assuming that both have the same value. Then the cameraMatrix is updated with the estimated focal length.

OpenCV solvePnPRansac

solvePnPRansac is very similar to **solvePnP** except that it uses **Random Sample Consensus (RANSAC)** for robustly estimating the pose.

Using RANSAC is useful when you suspect that a few data points are extremely noisy. For example, consider the problem of fitting a line to 2D points. This problem can be solved using linear least squares where the distance of all points from the fitted line is minimized. Now consider one bad data point that is wildly off. This one data point can dominate the least squares solution and our estimate of the line would be very wrong. In RANSAC, the parameters are estimated by randomly selecting the minimum number of points required. In a line fitting problem, we randomly select two points from all data and find the line passing through them. Other data points that are close enough to the line are called inliers. Several estimates of the line are obtained by randomly selecting two points, and the line with the maximum number of inliers is chosen as the correct estimate.

The usage of **solvePnPRansac** is shown below and parameters specific to **solvePnPRansac** are explained.

```
C++: void solvePnPRansac(InputArray objectPoints, InputArray imagePoints,  
InputArray cameraMatrix, InputArray distCoeffs, OutputArray rvec, OutputArray
```

```
tvec, bool useExtrinsicGuess=false, int iterationsCount=100, float  
reprojectionError=8.0, int minInliersCount=100, OutputArray inliers=noArray(),  
int flags=ITERATIVE )
```

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

iterationsCount – The number of times the minimum number of points are picked and the parameters estimated.

reprojectionError – As mentioned earlier in RANSAC the points for which the predictions are close enough are called “inliers”. This 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 .

OpenCV POSIT

OpenCV used to a pose estimation algorithm called POSIT. It is still present in the C API (`cvPosit`), but is not part of the C++ API. POSIT assumes a scaled orthographic camera model and therefore you do not need to supply a focal length estimate. This function is now obsolete and I would recommend using one of the algorithms implemented in `solvePnp`.

OpenCV Pose Estimation Code : C++ / Python

In this section, I have shared example code in C++ and Python for head pose estimation in a single image. You can download the image [headPose.jpg](#) [here](#).

The locations of facial feature points are hard coded and if you want to use your own image, you will need to change the vector `image_points`

C++

```
1  #include <opencv2/opencv.hpp>
2
3  using namespace std;
4  using namespace cv;
5
6  int main(int argc, char **argv)
7  {
8
9      // Read input image
10     cv::Mat im = cv::imread("headPose.jpg");
11
12     // 2D image points. If you change the image, you need to change vector
13     std::vector<cv::Point2d> image_points;
14     image_points.push_back( cv::Point2d(359, 391) );    // Nose tip
15     image_points.push_back( cv::Point2d(399, 561) );    // Chin
16     image_points.push_back( cv::Point2d(337, 297) );    // Left eye left corner
17     image_points.push_back( cv::Point2d(513, 301) );    // Right eye right corner
18     image_points.push_back( cv::Point2d(345, 465) );    // Left Mouth corner
19     image_points.push_back( cv::Point2d(453, 469) );    // Right mouth corner
20
21     // 3D model points.
22     std::vector<cv::Point3d> model_points;
23     model_points.push_back(cv::Point3d(0.0f, 0.0f, 0.0f));    // Nose tip
24     model_points.push_back(cv::Point3d(0.0f, -330.0f, -65.0f));    // Chin
25     model_points.push_back(cv::Point3d(-225.0f, 170.0f, -135.0f));    // Left eye
26     model_points.push_back(cv::Point3d(225.0f, 170.0f, -135.0f));    // Right eye
27     model_points.push_back(cv::Point3d(-150.0f, -150.0f, -125.0f));    // Left Mouth corner
28     model_points.push_back(cv::Point3d(150.0f, -150.0f, -125.0f));    // Right Mouth corner
29
30     // Camera internals
31     double focal_length = im.cols; // Approximate focal length.
32     Point2d center = cv::Point2d(im.cols/2,im.rows/2);
33     cv::Mat camera_matrix = (cv::Mat_<double>(3,3) << focal_length, 0, center.x, 0,
34     cv::Mat dist_coeffs = cv::Mat::zeros(4,1,cv::DataType<double>::type); // Assuming no lens distortion
35
36     cout << "Camera Matrix " << endl << camera_matrix << endl ;
37     // Output rotation and translation
38     cv::Mat rotation_vector; // Rotation in axis-angle form
39     cv::Mat translation_vector;
40
41     // Solve for pose
42     cv::solvePnP(model_points, image_points, camera_matrix, dist_coeffs, rotation_vector, translation_vector);
43
44
45     // Project a 3D point (0, 0, 1000.0) onto the image plane.
46     // We use this to draw a line sticking out of the nose
47
48     vector<Point3d> nose_end_point3D;
49     vector<Point2d> nose_end_point2D;
50     nose_end_point3D.push_back(Point3d(0,0,1000.0));
51
52     projectPoints(nose_end_point3D, rotation_vector, translation_vector, camera_matrix, dist_coeffs, nose_end_point2D);
53
54
55     for(int i=0; i < image_points.size(); i++)
56     {
57         circle(im, image_points[i], 3, Scalar(0,0,255), -1);
58     }
59
60     cv::line(im,image_points[0], nose_end_point2D[0], cv::Scalar(255,0,0), 2);
61
62     cout << "Rotation Vector " << endl << rotation_vector << endl;
63     cout << "Translation Vector" << endl << translation_vector << endl;
64 }
```

```

65     cout << nose_end_point2D << endl;
66
67     // Display image.
68     cv::imshow("Output", im);
69     cv::waitKey(0);
70
71 }

```

Python

```

1  #!/usr/bin/env python
2
3  import cv2
4  import numpy as np
5
6  # Read Image
7  im = cv2.imread("headPose.jpg");
8  size = im.shape
9
10 #2D image points. If you change the image, you need to change vector
11 image_points = np.array([
12     (359, 391),      # Nose tip
13     (399, 561),      # Chin
14     (337, 297),      # Left eye left corner
15     (513, 301),      # Right eye right corne
16     (345, 465),      # Left Mouth corner
17     (453, 469),      # Right mouth corner
18 ], dtype="double")
19
20 # 3D model points.
21 model_points = np.array([
22     (0.0, 0.0, 0.0),      # Nose tip
23     (0.0, -330.0, -65.0), # Chin
24     (-225.0, 170.0, -135.0), # Left eye left corner
25     (225.0, 170.0, -135.0), # Right eye right corne
26     (-150.0, -150.0, -125.0), # Left Mouth corner
27     (150.0, -150.0, -125.0), # Right mouth corner
28
29 ])
30
31
32 # Camera internals
33
34 focal_length = size[1]
35 center = (size[1]/2, size[0]/2)
36 camera_matrix = np.array(
37     [[focal_length, 0, center[0]],
38      [0, focal_length, center[1]],
39      [0, 0, 1]], dtype = "double"
40 )
41
42 print "Camera Matrix :\n {0}".format(camera_matrix)
43
44 dist_coeffs = np.zeros((4,1)) # Assuming no lens distortion
45 (success, rotation_vector, translation_vector) = cv2.solvePnP(model_points, image_p
46
47 print "Rotation Vector:\n {0}".format(rotation_vector)
48 print "Translation Vector:\n {0}".format(translation_vector)
49
50
51 # Project a 3D point (0, 0, 1000.0) onto the image plane.
52 # We use this to draw a line sticking out of the nose
53
54
55 (nose_end_point2D, jacobian) = cv2.projectPoints(np.array([(0.0, 0.0, 1000.0)]), ro

```

```
56
57 for p in image_points:
58     cv2.circle(im, (int(p[0]), int(p[1])), 3, (0,0,255), -1)
59
60
61 p1 = ( int(image_points[0][0]), int(image_points[0][1]))
62 p2 = ( int(nose_end_point2D[0][0][0]), int(nose_end_point2D[0][0][1]))
63
64 cv2.line(im, p1, p2, (255,0,0), 2)
65
66 # Display image
67 cv2.imshow("Output", im)
68 cv2.waitKey(0)
```

Real time pose estimation using Dlib

The video included in this post was made using my fork of dlib which is freely available for subscribers of this blog. If you have already subscribed, please check the welcome email for link to my dlib fork and check out this file

dlib/examples/webcam_head_pose.cpp

If you have not subscribed yet, please do so in the section below

Head Pose Estimation



Subscribe & Download Code

If you liked this article and would like to download code (C++ and Python) and example images used in this post, please **subscribe** to our newsletter. You will also receive a free **Computer Vision Resource** guide. In our newsletter we share OpenCV tutorials and examples written in C++/Python, and Computer Vision and Machine Learning algorithms and news.

Filed Under: **Application, Face, Tutorial**

Tagged With: **dlib, Head Pose, OpenCV, Pose, solvepnp, solvepnpransac**

Download C++ and Python Code

Get FREE access to all code (C++ / Python) and example images used in this blog by subscribing to our newsletter.

[Click Here to Subscribe](#)

84 Comments

Learn OpenCV

 **Zhilei Liu** ▾

 Recommend 7

 Share

Sort by Best ▾



Join the discussion...



Anonymous • 2 months ago

Hi. nice tutorial ..But its running slow on my system i.e. 30fps only. Also it detects only within a limited range. Outside that it simply doesn't detect at all. Is this problem in actual system also or only my problem? How to increase speed further? i have set AVX instruction flag still no effect.

^ | ▾ • Reply • Share ›



Satya Mallick Mod ➔ **Anonymous** • 2 months ago

Thanks.

You can try some suggestions here

...can try some suggestions here

[http://www.learnopencv.com/...](http://www.learnopencv.com/)

^ | v • Reply • Share ›



andres • 2 months ago

nice tutorial!! but is there a way to process using gpu

^ | v • Reply • Share ›



Angad Nayyar • 4 months ago

Hi Satya,

Can we use the information determined from this, to get the location of a real world object from it's pixel co-ordinates?

For example, I use an A4 paper to do the mentioned steps. Can I then use the translational vector, rotation vector, and my knowledge of the dimensions of the paper to get the real world location of a coin next to it?

^ | v • Reply • Share ›



CO • 4 months ago

Hi Satya, Thank you for very good tutorial about dlib and opencv. I am beginner at c++ and I have some question to ask about webcam_head_pose.cpp as in code. My goal is to draw laser from eyes like Superman so I need to get eyes position from face. Is there anyway to get eyes position from it ? Thank you very much.

^ | v • Reply • Share ›



Satya Mallick Mod ➔ **CO** • 4 months ago

Thanks!

You will have to train your own dlib model that contains the center of the eyes. You can also use the points around the eyes to come up with a heuristic for the the location of the center of the pupil, but it won't be very good.

^ | v • Reply • Share ›



Damian Allen • 4 months ago

Satya,

Great post. Do you have a suggestion as to how to derive 3D coordinate locations for the landmark features themselves, rather than the entire head?

The approaches I can think of, using a simple mesh of a generic head:

1. Raycast from a given 2D landmark position to the head mesh model and calculate the point position where the ray intersects.
2. Render a position map of the head mesh from the camera POV (i.e. render the XYZ coordinates of the face model into the RG and B channels respectively), then retrieve the pixel value at the location of the landmark.
3. Render a depth matte of the head mesh and use its value paired with the XY screen coordinates of the landmark to derive the world XYZ from these

the landmark to derive the world XYZ from these.

Not sure if there's a simpler approach, or which one of these would be the most efficient, since I haven't dealt too much with rendering 3D objects via OpenCV.

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Damian Allen • 4 months ago

Thanks Damian,

If you have a 3D triangulated mesh and you have found the head pose using the method I have described, you can use the project any point on the mesh to the image plane. Conversely, if you want to estimate the 3D location of a 2D point, you can transform the mesh into camera coordinates (see figure in the article), and shoot a ray from the camera center through the pixel location and see where it intersects the mesh (in camera coordinates). Obviously, it is possible the ray will intersect the mesh multiple times and so you need to choose the point closest to the camera.

^ | v • Reply • Share ›



Damian Allen ➔ Satya Mallick • 4 months ago

Thanks,

YEs, I'm actually already putting a workflow together based on using your pose prediction to inform a more detailed mesh.

One other quick question: if you're only concerned with landmark detection for a single actor, would it be better to train a model with multiple photos of their head in various orientations and lighting, rather than a variety of faces? If so, should they also be of various facial expressions? Forgive my ignorance of the training model; I'm a few levels of encapsulation away from wanting to understand the fine details neural network implementation...

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Damian Allen • 4 months ago

Yes it would be better. In fact I have done something similar for a project. It is very difficult to find the same person under different lighting conditions. You also need to label all those images. So the best trick is to run the standard landmark detector on the person's face, fix the points that are not accurate, and put these new images in the training set as well. 50% images of this person and 50% of random people will still bias the results toward this person's face and also have sufficient variety in lighting etc. Hope that helps.

^ | v • Reply • Share ›



Damian Allen ➔ Satya Mallick • 4 months ago

Yes indeed, thank you. I actually plan to add custom markers to the face and train those (i.e. dots at landmark positions like cheek bones, corner of mouth and above eyebrows). I imagine at that point using other faces would just confuse the results. And if you don't mind me asking one more question: in the case of adding custom markers would the shape of the marks need to be unique, or would their proximity to facial features (e.g. a marker just to the side of a mouth corner) be sufficient for the training to see them as unique?

of a mouth corner, be confident for the training to see them as unique.

^ | v • Reply • Share ›



Jon Watte • 4 months ago

This tutorial shows how to unproject 2D points to 3D points, which is a somewhat interesting optimization/fitting problem, but to have a working solution, the important bit is finding where the feature points are in the faces in the input images -- corners of eyes, nose tip, mouth, etc. I can't find any code in your github that actually calls the opencv face detect functions --- there are just files with hard-coded point locations as input. How did you generate these input files?

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Jon Watte • 4 months ago

That is done using dlib. This is the file you need.

<https://github.com/spmallic...>

and here is the compilation instruction

<http://dlib.net/compile.html>

^ | v • Reply • Share ›



Clement Ng • 5 months ago

Hi Satya and thank you for your tutorial. It is very useful for me. I would have one question to ask about swap face. I would like to do an android application about putting model's face to a user's face so that they can see the result for applying the cosmetic in our application. I have watched your tutorial (face swap and face morph). Which one do you think is more suitable and can I swap their face feature and without change their face size and hair style? Because I saw that the face shape would be changed in the face swap tutorial. Thank you very much.

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Clement Ng • 4 months ago

Thanks. Both of those are not actually good for applying makeup. For makeup the technique is very different and each makeup element is rendered differently. You can try to look at something I did at my previous company (<http://www.taaz.com>).

^ | v • Reply • Share ›



Clement Ng ➔ Satya Mallick • 4 months ago

If I am doing a college assignment, which one do you think would be more suitable? Thank you.

^ | v • Reply • Share ›



Siddhant Mehta • 5 months ago

Thank you so much Satya Sir for your wonderful tutorials. They helped me alot to learn OpenCV and creating my projects. Until now I have implemeted pose estimation with SolvePnP as you explained above. But as faar as I understood, camera is fix in this scenario. If both camera and target are moving then how it will be possible to detect the pose of the camera w.r.t. target? My camera has 2 degrees of freedom (pitch, yaw). Do you have any suggestions? I am thiking to estimate homography matrix from point matching between change of pose and somehow add that to the rvec and tvec? Any suggestions? Thank you, Siddhant Mehta

^ | v • Reply • Share ›



Satya Mallick Mod → Siddhant Mehta • 5 months ago

Hi Siddhant. If the camera moves you get the relative orientation of the object w.r.t the camera. But I guess you are asking how do you recover camera motion. For that you have to look at static parts of the scene, find point correspondences. If the point correspondences come from a plane (e.g. the floor or one wall) you can estimate Homography and decompose it into R and t . Otherwise, you need to estimate the Essential Matrix / Fundamental Matrix. BTW if you are doing this to learn, go ahead and implement these yourself. But if you are using it in a real world project check out VisualSFM, Theia, and OpenMVG.

^ | v • Reply • Share ›



Alexey Ledovskiy • 5 months ago

Hey Satya, I was trying to do just a face recognition using dlib and standard face landmark from their site, it seems like the features and matching are not rotation invariant, I was wondering if you have any ideas how to make the face recognition rotation invariant with dlib?

^ | v • Reply • Share ›



Satya Mallick Mod → Alexey Ledovskiy • 5 months ago

Hi Alexey, Face Recognition usually means identifying who the person is. Landmark detection can be used as a preprocessing step in face recognition for alignment. Does that make sense ?

^ | v • Reply • Share ›



Alexey Ledovskiy → Satya Mallick • 5 months ago

Thanks for replying. I mean face detection phase inside dlib, seems like the landmark detection is not rotation invariant, so when rotate the camera like 90 degrees it doesn't detect a face. Maybe you have some thoughts where to look to fix that. Thank you.

^ | v • Reply • Share ›



Chomskyite → Alexey Ledovskiy • 4 months ago

If you're using the ".dat" file that came with dlib then it's limited to detecting the 68 facial landmarks that it was trained on. To get it to detect a profile view you'll have to create a new ".dat" file using several photographs of people with their faces turned 90 degrees to the camera. In the /tools directory you should find imglab which helps you do this.

^ | v • Reply • Share ›



Zongchang Chen • 6 months ago

Hi Satya! Is there any functions in OpenCV or any other libraries that I can use to find the rotation 3×3 matrix R and the translation matrix t when given the intrinsic camera matrix, the 2D image points and their corresponding 3D model points? Or I have to implement the wheel to find the extrinsic camera matrix in this scenario? I'm actually looking for a way to find the camera pose in world coordinates, so the extrinsic matrix might be the key in the case.

^ | v • Reply • Share ›



Satya Mallick Mod → Zongchang Chen • 6 months ago



Hi Zongchang,

Yes, solvePnP does precisely that :).

Satya

^ | v • Reply • Share ›



Zongchang Chen → Satya Mallick • 6 months ago

Hi Satya, thank you for your quick reply again! But when I run this code, the rotation vector rvec returned is actually a 3x1 column vector. I don't think that is the 3x3 rotation vector that I actually want here.

^ | v • Reply • Share ›



Satya Mallick Mod → Zongchang Chen • 6 months ago

They are both the same rotation expressed differently. Look for openCV documentation on Rodrigues to convert one form to other

^ | v • Reply • Share ›



Zongchang Chen → Satya Mallick • 6 months ago

I see. It really really helps! Thank you so much!

^ | v • Reply • Share ›



Mohammed ElBalkini • 6 months ago

Hi Satya, does the higher number of model points affect the precision of the estimated pose matrix?

^ | v • Reply • Share ›



Satya Mallick Mod → Mohammed ElBalkini • 5 months ago

Yes, the pose estimate can be made better with more points. Also, if you could have some points on the ears etc. , the pose estimate will be more stable.

^ | v • Reply • Share ›



Zongchang Chen • 6 months ago

Hi! This is really a fantastic blog. I'm wondering what is the measure of the image coordinate and the world coordinate respectively? Are they pixel and millimeter?

^ | v • Reply • Share ›



Satya Mallick Mod → Zongchang Chen • 6 months ago

Thank you!

The image coordinates are in pixels, but the world coordinates in are arbitrary units. You can produce the world coordinates using real measurements in millimeter or inches etc, or it could be just the coordinates in some arbitrary 3D model.

^ | v • Reply • Share ›



Zongchang Chen → Satya Mallick • 6 months ago

Wow that was a very fast reply! Thank you for your answer. Can I interpret your answer as the units of the world coordinates actually does not matter in computation as long as we keep the consistency of the measure of each point in 3D model?

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Zongchang Chen • 6 months ago

Yes that's right.

^ | v • Reply • Share ›



Mohammed ElBalkini • 6 months ago

Thanks Satya for this amazing tutorial. I would like to get you advice on how to reduce jitter resulted from pose matrix when used in augmented reality.

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Mohammed ElBalkini • 6 months ago

Thanks for the kind words Mohammed.

One option is to smooth out jitter by calculating the moving average of the points over multiple frames (say plus and minus 2 frames).

You can do average the rotation / translation directly. Be careful while averaging rotation matrices -- it is not straightforward. You may find this discussion helpful

<http://stackoverflow.com/qu...>

Satya

^ | v • Reply • Share ›



Mohammed ElBalkini ➔ Satya Mallick • 6 months ago

Thanks for your prompt reply.

i was thinking of converting the rotation matrix to quaternion, average it and then back to rotation matrix. will this work?

^ | v • Reply • Share ›



keizou • 6 months ago

I really thank this article.

I'm so sorry but is there an example of webcam_head_pose in python?

I watched this and tried to code it in python but I couldn't do it
`dlib/examples/webcam_head_pose.cpp`

^ | v • Reply • Share ›



Satya Mallick Mod ➔ keizou • 5 months ago

Sorry, I don't have a python version currently. But if you follow the logic in the C++ code, you will be able to write your own. There are not many lines of code.

^ | v • Reply • Share ›



Tolga Durak • 6 months ago

Dear Satya, thanks for sharing this post and explaining it. I am interested in developing gaze estimation program. It can estimate the center of pupil. In other words, I have the point of the center of pupil. How can I estimate gaze on computer like head pose estimation? Thanks a lot

pupil. How can I estimate gaze on computer like head pose estimation ? Thanks a lot.

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Tolga Durak • 6 months ago

Hi Tolga,

You will have to detect the center of the pupils first. Dlibs landmark detector does not detect it, but it is possible to do so by retraining a landmark detector with your own data that contains the center of the eyes. In fact, in a few weeks I plan to release a model with the pupil center.

Satya

^ | v • Reply • Share ›



Hank White • 7 months ago

I run the program in xcode, but it's too slow than compiled webcam_head_pose.

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Hank White • 6 months ago

Are you sure you are compiling release mode ? Check this out

<http://dlib.net/faq.html#Wh...>

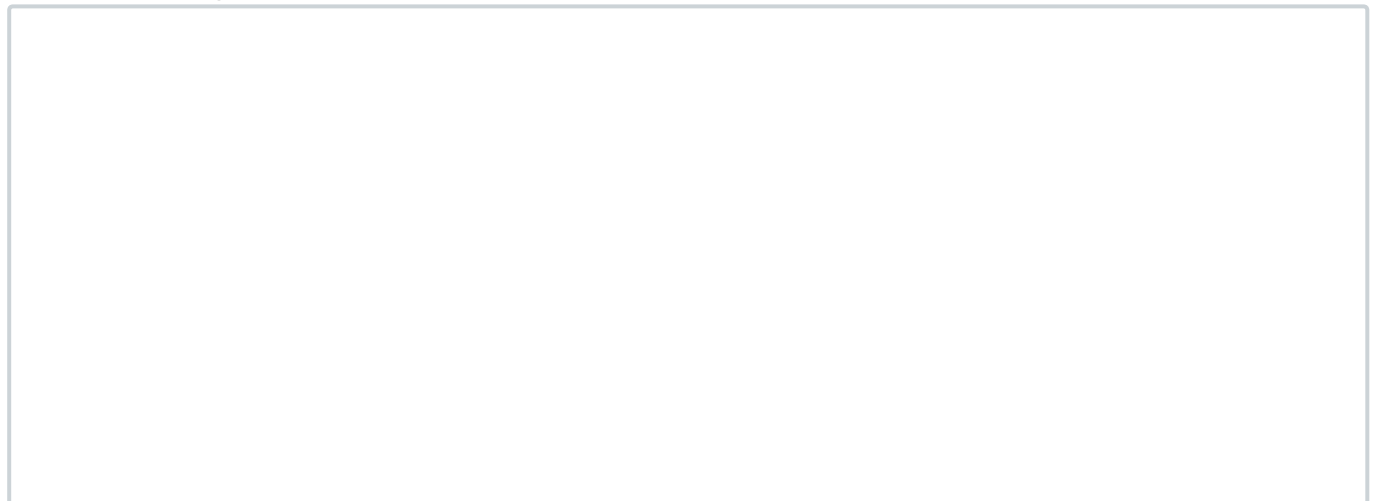
^ | v • Reply • Share ›



Minu • 7 months ago

Hey there Mister Satya!

Great job with all the tutorials and explanation. I wanna do the pose calculation by myself from scratch. I understand the method, the only thing that keeps me away is that i dont know how to extract only 6 landmarks, instead of 68. I've checked the code so many times, the dlib/opencv indexes too. I really need some help, im stucked... I uploaded the code too. Maybe u can give me a fast advice, i know ur time is precious! Thanks a lot!



[see more](#)

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Minu • 5 months ago

If you look at my version of dlib, you will see the indices of 6 points. I have shared the C++ code below.

```
std::vector<cv::point2d> get_2d_image_points(full_object_detection &d)
{
    std::vector<cv::point2d> image_points;
    image_points.push_back( cv::Point2d( d.part(30).x(), d.part(30).y() ) );    // No
    image_points.push_back( cv::Point2d( d.part(8).x(), d.part(8).y() ) );      // Ch
    image_points.push_back( cv::Point2d( d.part(36).x(), d.part(36).y() ) );    // Le
    image_points.push_back( cv::Point2d( d.part(45).x(), d.part(45).y() ) );    // Ri
    image_points.push_back( cv::Point2d( d.part(48).x(), d.part(48).y() ) );    // Le
    image_points.push_back( cv::Point2d( d.part(54).x(), d.part(54).y() ) );    // Ri
    return image_points;
}
```

^ | v • Reply • Share ›



Liron • 7 months ago

Hello Satya,

I was wondering how i can get the 3D model points in real time (like i can see in your video with the vector that comes from your nose).

Thanks

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Liron • 6 months ago

If you look at the code, I have put a 3D point some distance from the nose in the 3D model. I simply project this point onto this image plane using the estimated rotation and translation.

^ | v • Reply • Share ›



Max Kutny • 7 months ago

Hi Satya,

There is a mistake in left eye 3d coords in the text ("Left corner of the left eye : (0.0, 0.0, 0.0)"). In the code they are (-225.0, 170.0, -135.0) which seem to be correct.

^ | v • Reply • Share ›



Charles Zheng ➔ Max Kutny • 2 months ago

hi, Max, I try to use some other points to calculate pose, could you please tell me where I can get other landmarks 3d coords?

^ | v • Reply • Share ›



Satya Mallick Mod ➔ Max Kutny • 6 months ago

Thank you so much. I have fixed the mistake.

^ | v • Reply • Share ›



Charles Zheng ➔ Satya Mallick • 2 months ago

hi, Satya, I try to use some other points to calculate pose, could you please tell me where you get these 3d coords?

^ | v • Reply • Share ›

[Load more comments](#)

Install OpenCV 3 on Windows

18 comments • 7 days ago•

Miguel Medina — I tested on VS 2017 and VS 2013 and it was the same, So I changed cmake: from 3.8.0 version to 3.8.2 and the problem was ...

Image Recognition and Object Detection : Part 1

49 comments • 7 months ago•

Rahmaniansyah D Putri — Good Morning Sir, I am college student from Indonesia University of Education. Can you make an article about How ...

Training a better Haar and LBP cascade based Eye Detector using OpenCV



22 comments • 4 months ago•

joe el khoury — I reinstalled opencv and everything is ok now! just in case someone faces the same problem!

Configuring Qt for OpenCV on OSX

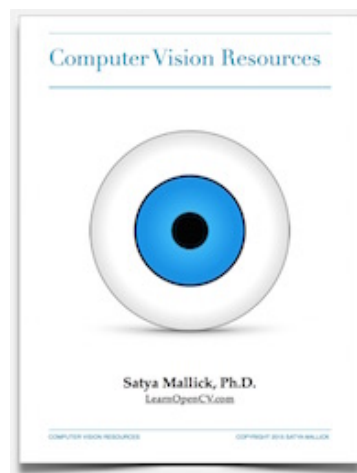
15 comments • a year ago•

mvh — I solved the problem. See my other post.

✉ Subscribe •  Add Disqus to your siteAdd DisqusAdd  Privacy

SEARCH

COMPUTER VISION RESOURCES



To learn more about OpenCV books, libraries, and web APIs download our free resource guide.

Download

FOLLOW

SATYA MALLICK



I am an entrepreneur who loves Computer Vision and Machine Learning. I have a dozen years of experience (and a Ph.D.) in the field.

I am a co-founder of TAAZ Inc where the scalability, and robustness of our computer vision and machine learning algorithms have been put to rigorous test by more than 100M users who have tried our products. [Read More...](#)

DISCLAIMER

This site is not affiliated with or endorsed by OpenCV Foundation (opencv.org).

RECENT POSTS

[Install OpenCV 3 on Windows](#)

[Get OpenCV Build Information \(getBuildInformation\)](#)

[Color spaces in OpenCV \(C++ / Python\)](#)

[Neural Networks : A 30,000 Feet View for Beginners](#)

[Alpha Blending using OpenCV \(C++ / Python\)](#)
