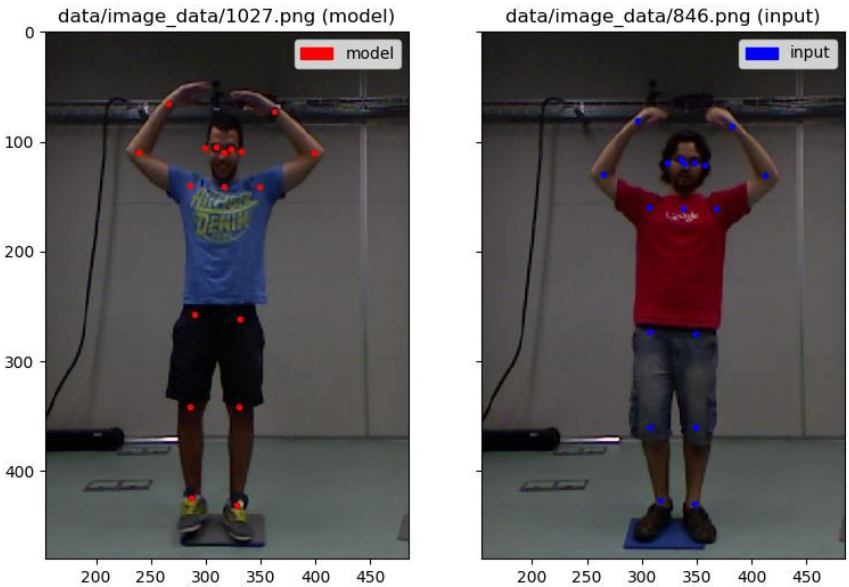
application using Human Pose Estimation (Part 2)

bilgeckers Follow
Dec 12, 2017 · 7 min read

This is the second part of a series. Part 1 (Intro)

Welcome back! In the previous blog we introduced a possible application of OpenPose (Human Pose Estimation). Because this algorithm doesn't require any special cameras, it works with a simple digital camera. WAIT! Before going any further, first try to realize this is a GREAT relaxation. This means you don't need some kind of Kinect or other depth camera, thus opens doors for mobile development. This just asks for a fun application …

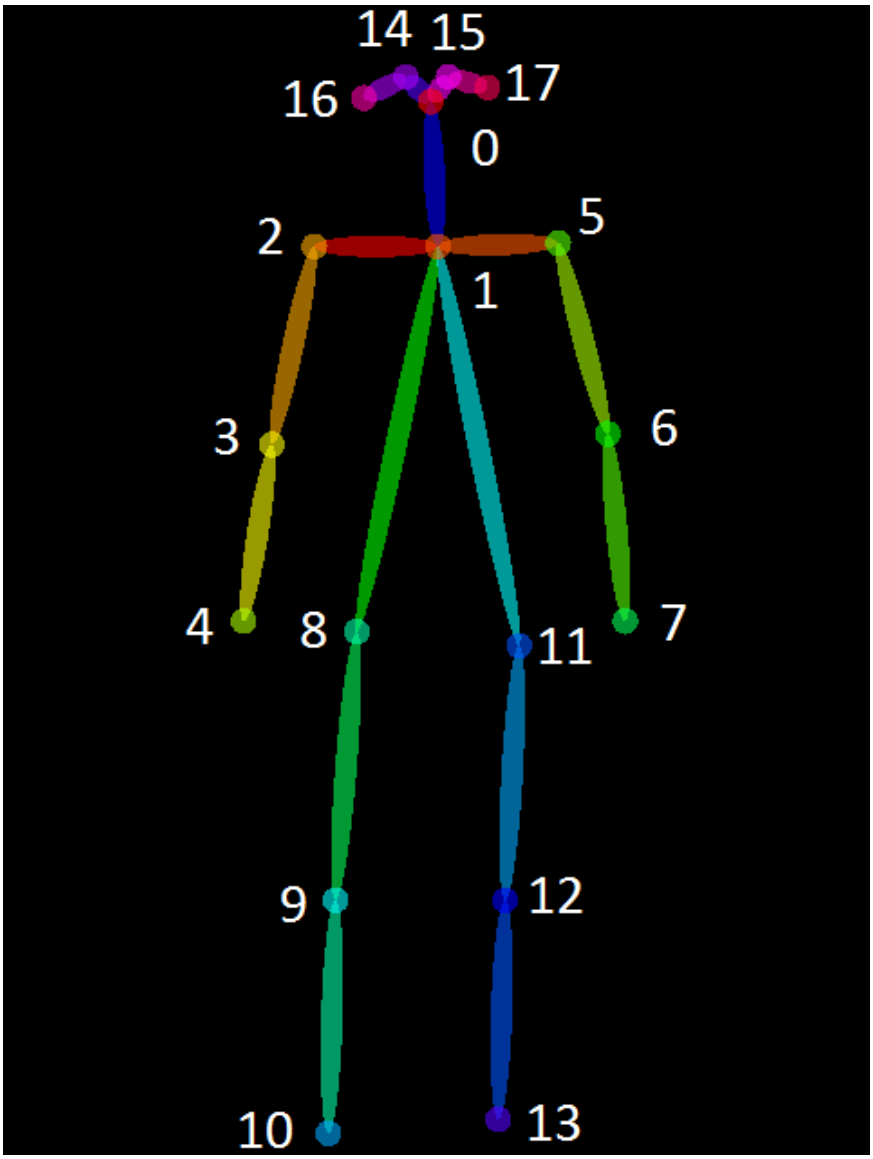
Basic idea: a person is instructed a model pose which ze must try to mimic. After taking a picture, it's decided if the input pose matches the model pose.
That's what this blog article is about: **matching human poses**
(An update on the mobile progress will follow shortly -hopefully-)



Smoooth posing

The Human Posture Description

The first step in this pipeline is extracting the body parts of every person appearing in the two images. The OpenPose library will do this for us. Using a neural network and some other cool functions, it returns a collection of sets. Each set represents a human skeleton and consists out of 18 2D points.



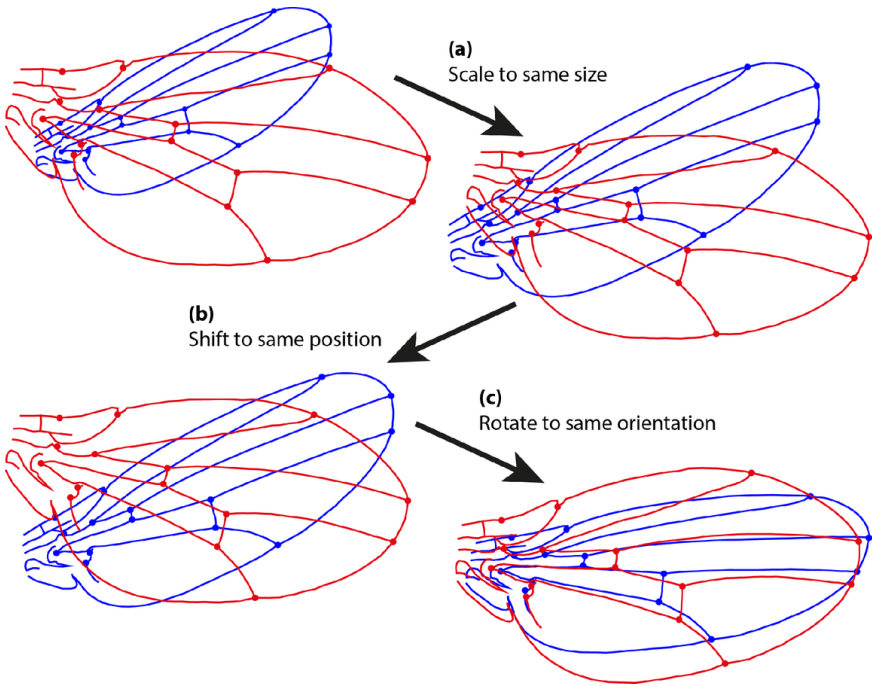
Human Pose Comparison

*We will consider the simple case with **only one person***

So, we have 2 sets of corresponding points
1. The model (which needs to be mimicked)
2. The input (needs to be checked on matching grade)

Now, concluding if the input pose matches the model pose essentially comes down to checking if the two poses have the same shape. This can be treated as a Procrustes Problem:

> *To compare the shapes of two objects, the objects must be first optimally "superimposed". **Procrustes superimposition** (PS) is performed by optimally translating, rotating and uniformly scaling the objects. In other words, both the placement in space and the size of the objects are freely adjusted. The aim is to obtain a similar placement and size, by minimizing a measure of shape difference called the Procrustes distance between the objects.*



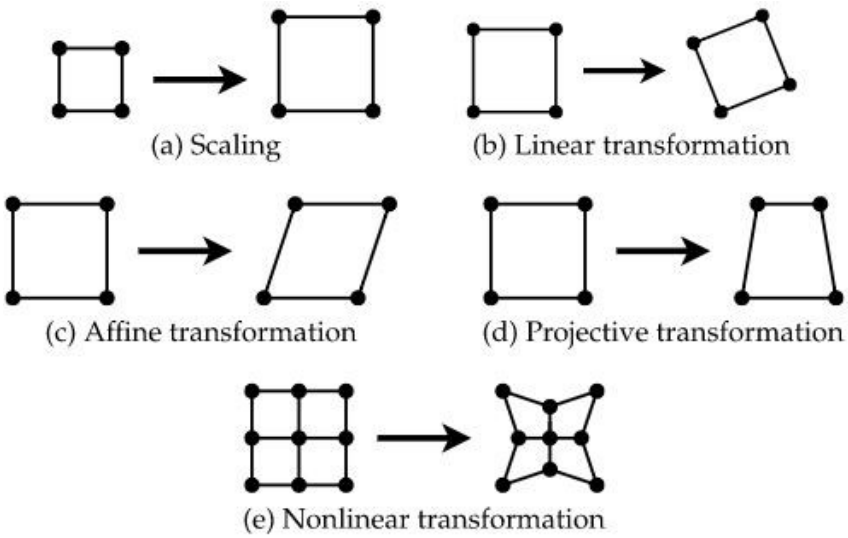
WIkipedia ftw

The difference between the shape of two objects can be evaluated only after "superimposing" the two objects by translating, scaling and optimally rotating them as explained above. Consider the perfect case (identical shapes), after PS, the objects will perfectly coincide. Of course, e.g. due to different body proportions, there is noway a person will succeed in perfectly copying the model pose. There is a need for some *thresholding;* If all corresponding points are approximately adjacent, we can conclude the two poses match. From the moment a distance exceeds a certain threshold, they're different.

The Transformation — Affine

We're looking for the combination of a translation, scaling and rotation

that best transforms the input pose onto the model pose. From Linear Algebra, we know this combination of operations is wrapped in a linear transformation, more precisely an **affine transformation (composition of linear map and a translation)**.
The properties:
- Lines maps to lines
- Parallel lines remain parallel
- Origin does not necessarily map to origin
- Ratios are preserved



Source: 'Accelerating nonlinear image transformations with OpenGL' -Vegard Øye

The linear map is represented as a multiplication by a matrix **A** and the translation as the addition of a vector **b**. An affine map **f()** acting on a vector **x** is represented as

$$\vec{y} = f(\vec{x}) = A\vec{x} + \vec{b}$$

Using an augmented matrix, it is possible to represent both the linear map and the translation using a single matrix multiplication. We need this augment matrix to solve our linear system in the next step.

This augmented matrix is created as follows:
1. Pad all vectors with a "1" at the end.
2. Augment the matrix with an one extra row of zeros at the bottom
3. Augment the matrix with an extra column (the translation vector) to the right and a "1" in the lower right corner.

$$\begin{bmatrix} \vec{y} \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & A & & \vec{b} \\ 0 & \dots & 0 & 1 \end{array}\right] \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}$$

In this particular case **x** -the input pose- and **y** -the model pose- are know and we want to find the augmented matrix. Using **least-squares algorithm** (which minimizes the sum of the squares), we can approximate the solution of this linear system and find the **affine transformation matrix**.

```
# Shout out to https://stackoverflow.com/a/20555267/87
# 2D array containing the 18 corresponding feature poi
model_features = [[x1,y2],[x2,y2],...]
input_features = [[x1,y2],[x2,y2],...]

# In order to solve the augmented matrix (incl transla
# it's required all vectors are augmented with a "1" a
# -> Pad the features with ones, so that our transform
pad = lambda x: np.hstack([x, np.ones((x.shape[0], 1))
unpad = lambda x: x[:, :-1]

# Pad to [[ x y 1] , [x y 1]]
Y = pad(model_features)
```

Using the transformation matrix, we find **x'**; the input pose displayed onto the model pose. This is the **best-fit** resulting from the least-squares algorithm, that tries to transform the input pose onto the model pose.

```
# Now we have found the augmented matrix A
# we can display the input on the model = X'
transform = lambda x: unpad(np.dot(pad(x), A))

#Image of input pose onto model pose
```

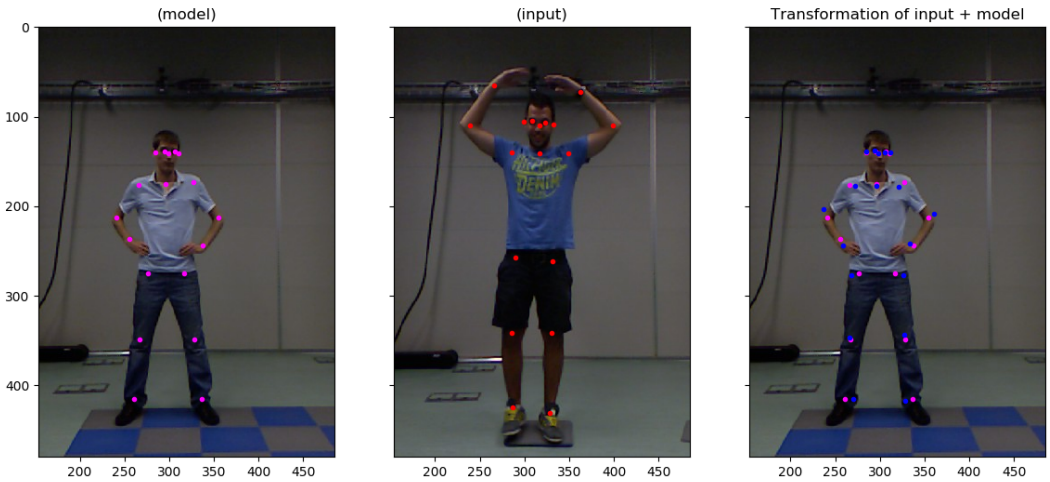
. . .

The Thresholds

Now that we have the transformed input **X' ,** we can compare it with the proposed model pose. We will base our decision (different or similar shape) on two parameters: the distance between two corresponding points and the rotation introduced by the affine transformation.

1. Rotation

Consider a person performing handstand. This is not the same pose as another person just standing straight. Although, in some cases, it's perfectly possible a descent affine transformation is found. When you examine the transformation matrix a bit closer, you will find rotation angle around 180°.



Rotation illustration [using split()]

The above example illustrates the case a decent transformation is found even though this is not what we humans understand under "the same pose".
Fortunately the rotation angle of the torso (arms) is around 175°, so well above the accepted value.

2. Distance

As distance metric the **Euclidean distance** is used (there are other options, as there are many distance metrics). We limit the maximum Euclidean distance between two corresponding points by a determined

threshold. This means, from the moment one distance exceeds a certain threshold, the whole pose does not match. Note we don't take the sum of all euclidean distances. This is because we want to focus on **outliers (maximum values)**. From the moment one body-part is not well transformed, we can conclude the global pose doesn't match.

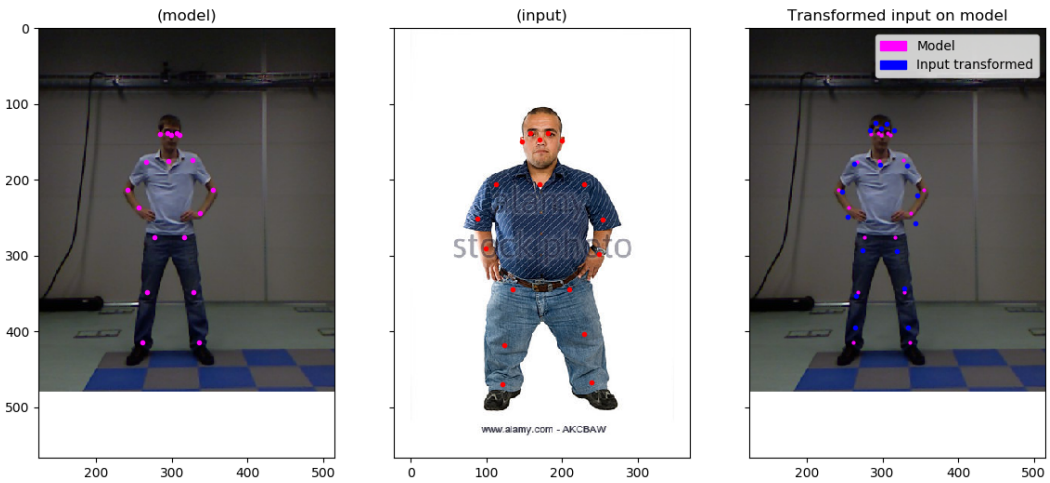
. . .

The Split() — Torso Legs and Face separately

Our distance thresholding starts to look quite good, but there is still room for some improvement. Our gut instinct tells us we need to make a distinction between different body regions when evaluating the distances. As it is more easy to control your arms than your legs, you can feel we need to treat those two different. Same applies to the eyes, ears and nose. We're looking for a way to evaluate these distinct regions with their **own threshold**.

In addition we want to introduce more freedom in the search for the optimal transformation. In particular, we want to ensure that **difference in individual body proportions** don't influence the matching result. For example; a person who has very small legs in comparison with his chest (distance between two shoulders) will result in unpleasant contradictions.

These findings are confirmed by the following illustration:

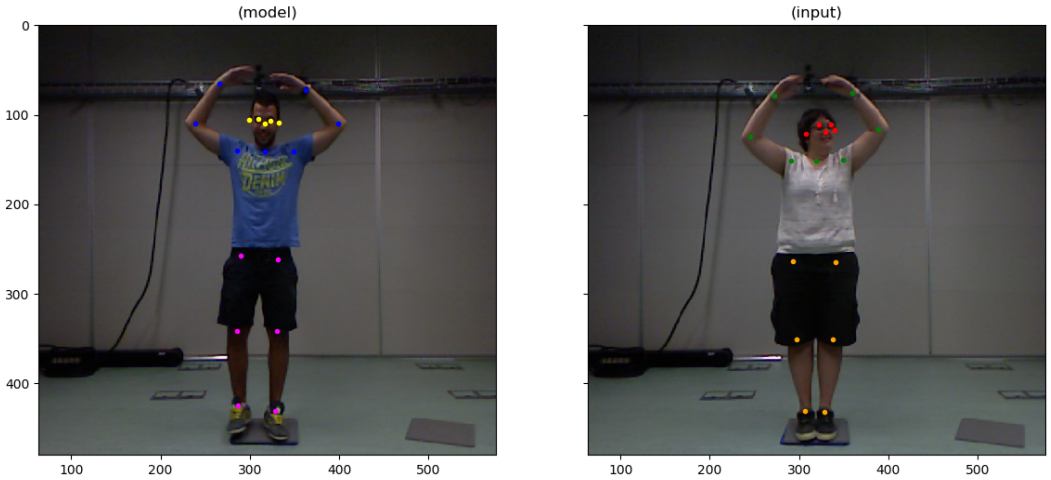


Transformation of all 18 points together, as one system

The result is not that good. Due to difference in body ratio, the transformed arms are to long and the transformed legs to small. It's trying to find a middle ground, but it's not acceptable.

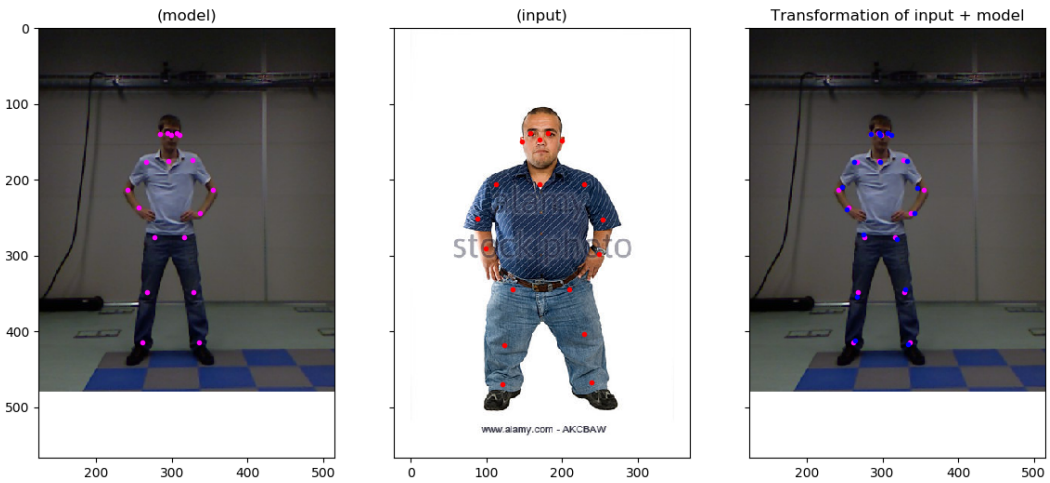
Instead of calculating the affine transformation for the whole body, we split it in three parts:

1. Face (eyes + ears + nose)
2. Torso (arms + neck)
3. Legs + Hips



This results in three transformation matrices and in turn three partially transformed inputs; a transformed torso, transformed face and a transformed legs. Again, comparison is done based on the euclidean distance and rotation. But now with their own specific thresholds.

```
# Split features in three parts
(model_face, model_torso, model_legs) = split_in_face_
(input_face, input_torso, input_legs) = split_in_face_

# 3x3 the same; not the best code design I know ...
#   But the illustrative factor prevails here

# Solve the least-sqaures problem and find the transfo
# and the corresponding image of input
(input_transformed_face,  A_face)   = affine_transform
(input_transformed_torso, A_torso)  = affine_transform
(input_transformed_legs,  A_legs)   = affine_transform

(max_distance_face, rotation_face)   = max_distance_an


(max_distance_torso, rotation_torso) = max_distance_an

```



This is **eye candy** — result of a split()

Looks good! We're almost there, only one step left. If we want to reuse our precious thresholds, we better **standardize** everything before comparing. In that way different image resolutions are no issue. We will rescale the features to the range in [0, 1].

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

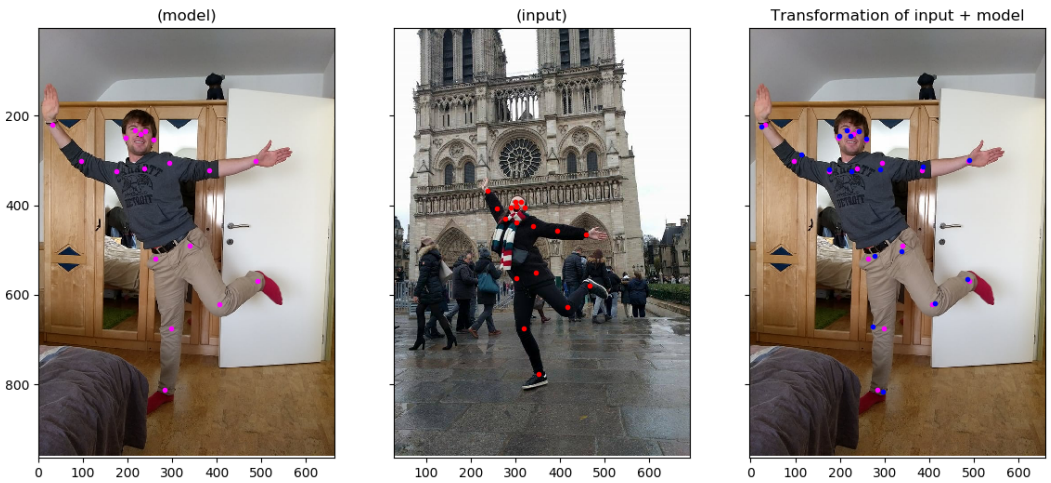
Feature scaling, x' is our new normalized value

. . .

Results

Voilà, concerning **single pose** matching, that's about it. Hope you liked it. However, to convince the skeptics among us I realize a more challenging example needs to be considered than the ones shown above.

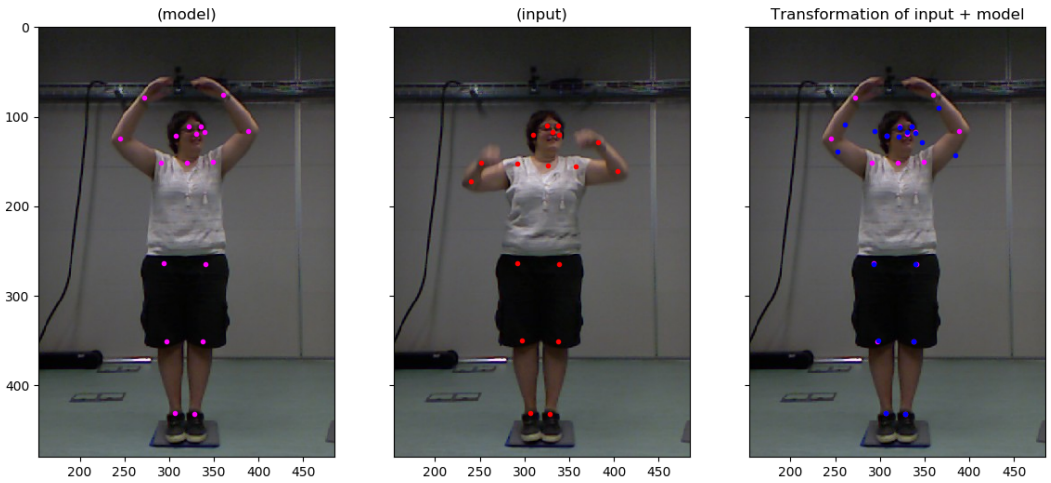
— Behold ;



Sick freaking pose

The input pose is almost the same as the model, except the left elbow, left hand and left knee differ a bit. Ergo, as we would expect, the biggest deviations are located in the left elbow and the left knee.

What happens in the other case? When the person fails at copying the proposed model pose. Well, no surprise I guess; the quest for the best transformation fails as well. This is reflected in the greater distance between some corresponding points.



Epic fail epic fail epic fail

Legs and face are OK, but torso is completely lost. Hence, a fail is the only right outcome.

Thanks for reading.

Ciao, bella