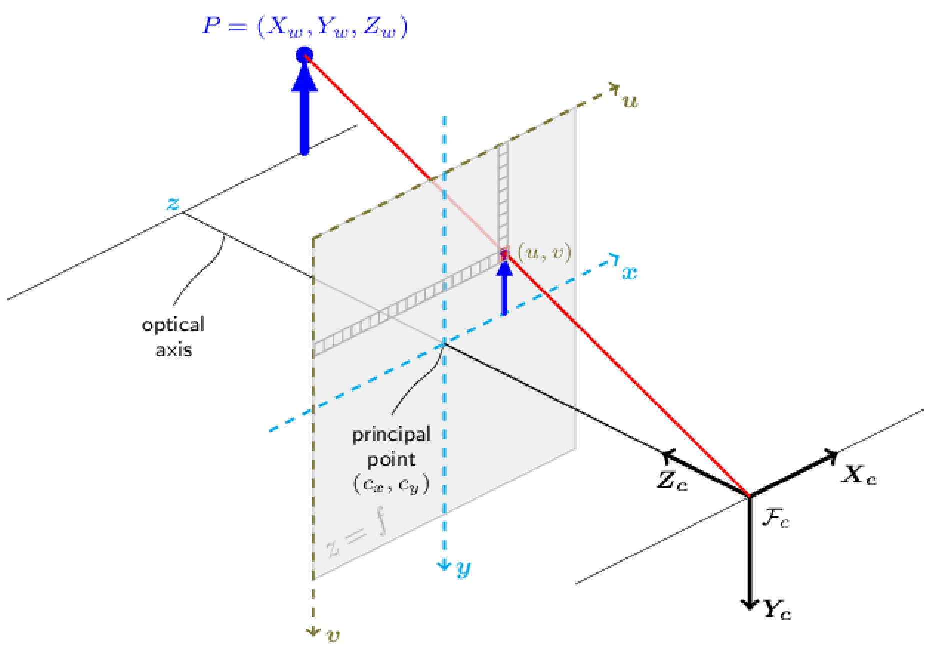
```
import numpy as np
from tqdm import tqdm
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error

get_world_path = lambda g,f,s,e: f"./gesture_{g}/finger_{f}/subject_{s}/essai_{e}/skeletons_
get_image_path = lambda g,f,s,e: f"./gesture_{g}/finger_{f}/subject_{s}/essai_{e}/skeletons_

train = np.loadtxt('./train_gestures.txt', dtype=np.uint16)
test  = np.loadtxt('./test_gestures.txt',  dtype=np.uint16)
```

Coordinate Systems

There are 4 coordinate systems.

- *Pixel coordinates* (u, v) - Origin at top left of image
- *Film coordinates* (x, y) - Origin at (c_x, c_y)
- *Camera coordinates* (X_c, Y_c, Z_c) - Transform Film coordinates using intrinsics matrix K
- *World coordinates* (X_w, Y_w, Z_w) - Arbitrary world system, transform camera by extrinsics matrix $[R|t]$



http://www.cse.psu.edu/~rtc12/CSE486/lecture12.pdf

https://docs.opencv.org/master/d9/d0c/group___calib3d.html

Matrices

Pixel to Film

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Pixel to Camera

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x \frac{X_c}{Z_c} + c_x \\ f_y \frac{Y_c}{Z_c} + c_y \end{bmatrix}$$

Pixel to World

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{1x3} & t_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

Equations for Regression

Assumptions Case 1

Let's assume `skeletons_world.txt` provides camera coordinates.

The relation between pixel coordinates (u, v) and camera coordinates (X_c, Y_c, Z_c) is given by:

$$u = f_x \frac{X_c}{Z_c} + c_x$$

$$v = f_y \frac{Y_c}{Z_c} + c_y$$

Assumptions Case 2

Let's assume `skeletons_world.txt` provides world coordinates.

If we consider $M = K \times [R|t]$, product of intrinsics K and extrinsics $[R|t]$ matrix. Also assume that there's no rotation i.e. $R_{3xr} = I_{3x3}$, so R is identity matrix. We want to find out the translation t on camera polar center (c_x, c_y) which is the origin of world coordinates.

Then,

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I_{1x3} & t_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

Therefore,

$$
\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x + t_x \\ 0 & f_y & c_y + t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}
$$

The equations are therefore,

$$
u = f_x \frac{X_w}{Z_w} + (c_x + t_x)
$$

$$
v = f_y \frac{Y_w}{Z_w} + (c_y + t_y)
$$

Training

```
train_skeletons_world = []
train_skeletons_image = []

for g, f, s, e, start, end, num in tqdm(train):
    xyz = np.loadtxt(get_world_path(g, f, s, e), dtype=np.float32)
    xyz = np.reshape(xyz, (-1,22,3))
    train_skeletons_world.append(xyz)

    uv  = np.loadtxt(get_image_path(g, f, s, e), dtype=np.float32)
    uv  = np.reshape(uv,  (-1,22,2))
    train_skeletons_image.append(uv)
```

```
100%|      | 1960/1960 [00:31<00:00, 61.93it/s]
```

```
train_skeletons_world = np.concatenate(train_skeletons_world).reshape((-1, 3))
train_skeletons_image = np.concatenate(train_skeletons_image).reshape((-1, 2))
train_skeletons_world.shape, train_skeletons_image.shape
```

```
((2535632, 3), (2535632, 2))
```

U and X

```
XbyZ = train_skeletons_world[:,0]/train_skeletons_world[:,2]
U = train_skeletons_image[:,0]
```

```
regx = LinearRegression().fit(XbyZ[:,None], U[:,None])
fx = regx.coef_[0][0]
cx = regx.intercept_[0]
print("fx = ", fx, "cx = ", cx)
```

```
fx =  440.44232 cx =  -0.00015258789
```

V and Y

```
YbyZ = train_skeletons_world[:,1]/train_skeletons_world[:,2]
V = train_skeletons_image[:,1]
```

```
regy = LinearRegression().fit(YbyZ[:,None], V[:,None])
fy = regy.coef_[0][0]
cy = regy.intercept_[0]
print("fy = ", fy, "cx = ", cy)
```

```
fy =  -461.0357 cx =  3.0517578e-05
```

Test

```
test_skeletons_world = []
test_skeletons_image = []

for g, f, s, e, start, end, num in tqdm(test):
    xyz = np.loadtxt(get_world_path(g, f, s, e), dtype=np.float32)
    xyz = np.reshape(xyz, (-1,22,3))
    test_skeletons_world.append(xyz)

    uv  = np.loadtxt(get_image_path(g, f, s, e), dtype=np.float32)
    uv  = np.reshape(uv,  (-1,22,2))
    test_skeletons_image.append(uv)
```

```
100%|     | 840/840 [00:13<00:00, 62.63it/s]
```

```
test_skeletons_world = np.concatenate(test_skeletons_world).reshape((-1, 3))
test_skeletons_image = np.concatenate(test_skeletons_image).reshape((-1, 2))
test_skeletons_world.shape, test_skeletons_image.shape
```

```
((1093686, 3), (1093686, 2))
```

U and X

```
XbyZ = test_skeletons_world[:,0]/test_skeletons_world[:,2]
U = test_skeletons_image[:,0]
```

```
pred_U = regx.predict(XbyZ[:,None])
```

```
mean_absolute_error(U, pred_U)
```

```
6.185125e-05
```

V and Y

```
YbyZ = test_skeletons_world[:,1]/test_skeletons_world[:,2]
V = test_skeletons_image[:,1]
```

```python
pred_V = regy.predict(YbyZ[:,None])
```

```python
mean_absolute_error(V, pred_V)
```

```
2.794273e-05
```

Final values

```python
fx, fy, cx, cy
```

```
(440.44232, -461.0357, -0.00015258789, 3.0517578e-05)
```

Note how (c_x, c_y) are so close to zero that we can just consider them as zero. However, camera principal point should be close to center of image (not exactly center of image). Hence, `skeletons_world.txt` cannot be camera coordinates. There's some translation done to shift the world origin back to $(0, 0)$ to align the world with pixel coordinate system's origin at top left of the image. This translation is $c + t = 0$, hence $t = -c$.

This implies `skeletons_world.txt` has origin aligned with that of pixel coordinate system of `skeletons_image.txt`, hence the values of (c_x, c_y) are not to be used for shifting the origin and we can directly find world coordinates from pixel coordinates using:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

Where

$$M = K \times [R|t] = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Let's test

World to image

$$image = f \frac{world}{Z_{world}}$$

neglecting c

```python
true_image_x = test_skeletons_image[:,0]
true_image_x
```

```
array([367.82114, 365.58813, 350.73312, ..., 363.6523 , 359.87857,
       356.39206], dtype=float32)
```

```python
calc_image_x = fx * test_skeletons_world[:,0] / test_skeletons_world[:,2]
calc_image_x
```

```
array([367.8212 , 365.58823, 350.73322, ..., 363.65237, 359.87863,
       356.39215], dtype=float32)
```

```
mean_absolute_error(true_image_x, calc_image_x)
```

```
9.117182e-05
```

```
true_image_y = test_skeletons_image[:,1]
true_image_y
```

```
array([319.84595, 286.9815 , 311.01834, ..., 246.4431 , 237.16376,
       228.7852 ], dtype=float32)
```

```
calc_image_y = fy * test_skeletons_world[:,1] / test_skeletons_world[:,2]
calc_image_y
```

```
array([319.84592, 286.9815 , 311.01834, ..., 246.44308, 237.16376,
       228.78519], dtype=float32)
```

```
mean_absolute_error(true_image_y, calc_image_y)
```

```
7.952551e-06
```

Image to World

$$world = \frac{image}{f} Z_{world}$$

neglecting c

```
true_world_x = test_skeletons_world[:,0]
true_world_x
```

```
array([0.49394462, 0.47372743, 0.45985433, ..., 0.39503437, 0.38705626,
       0.37991765], dtype=float32)
```

```
calc_world_x = test_skeletons_image[:,0] / fx * test_skeletons_world[:,2]
calc_world_x
```

```
array([0.4939445 , 0.4737273 , 0.45985422, ..., 0.39503428, 0.38705617,
       0.37991756], dtype=float32)
```

```
mean_absolute_error(true_world_x, calc_world_x)
```

```
1.0257276e-07
```

```
true_world_y = test_skeletons_image[:,1]
true_world_y
```

```
array([319.84595, 286.9815 , 311.01834, ..., 246.4431 , 237.16376,
       228.7852 ], dtype=float32)
```

```
calc_world_y = fy * test_skeletons_world[:,1] / test_skeletons_world[:,2]
calc_world_y
```

```
array([319.84592, 286.9815 , 311.01834, ..., 246.44308, 237.16376,
       228.78519], dtype=float32)
```

```
mean_absolute_error(true_world_y, calc_world_y)
```

```
7.952551e-06
```