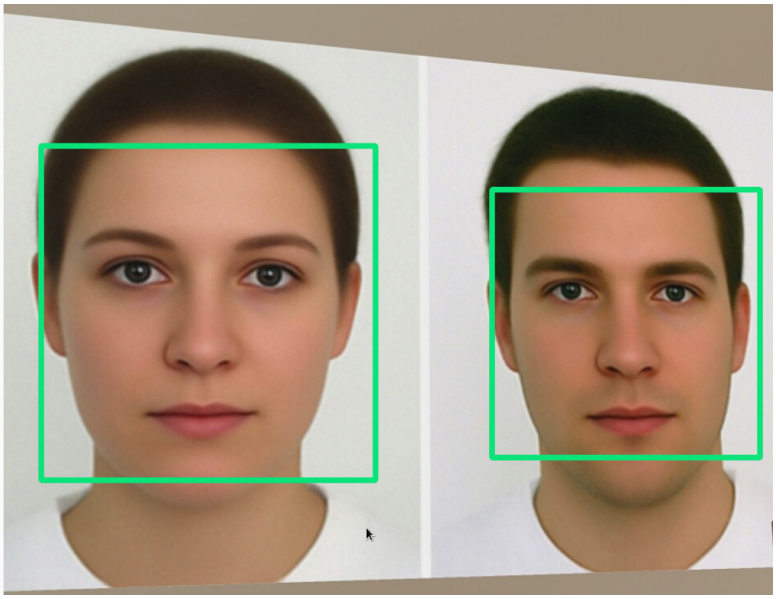
SIMPLE FACE SWAP

Náhradní tváře: http://mrl.cs.vsb.cz/data/vyuka/tamz2/opencv/faces.zip
Ukázka aplikace: http://mrl.cs.vsb.cz/data/vyuka/tamz2/opencv/opencv_swap.mp4
Ukázka živého videa: http://mrl.cs.vsb.cz/data/vyuka/tamz2/opencv/opencv_live_cam_01.mp4

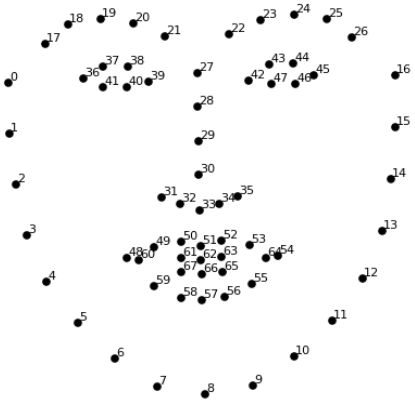
Pro nahrazení tváře je vhodné zjistit přesnější umístění tváře než získáme pouze z detektoru tváří. To je možné pomocí facial landmarks - zájmových bodů ve tváři (brada, ústa, nos, obočí). Opencv nabízí detektor Facemark, který tyto body ve tváři hledá. Použít se dá následujícím způsobem:

1. je potřeba lokalizovat tváře pomocí detektoru tváří (CascadeClassifier faceDetector)



2. tyto tváře dát na vstup funkci facemark.fit() například takto:
boolean success = facemark.fit(img, faces, landmarks);
Další ukázka použití zde:
https://github.com/bytedeco/javacv/blob/master/samples/KazemiFacemarkExample.java

Výsledkem jsou detekované zájmové body v tomto pořadí:



Pokud máme nalezené zájmové body tváře, můžeme tvář ořezat přesněji než pouze za použití detektoru tváří, protože nyní již máme k dispozi i přesnější pozice obočí a brady, takže můžeme získat přesnější obdélník reprezentující tvář - K souřadnicím landmarku s indexem 0 a 24 se dá přistupovat následujícím způsobem:

```
MatOfPoint2f lm = landmarks.get(i);
double[] dp0 = lm.get(0, 0);
double[] dp24 = lm.get(24, 0);
```

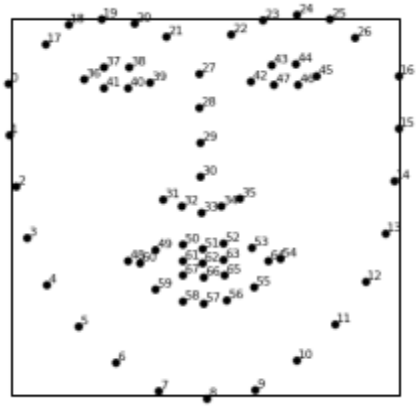
Ze souřadnic **x** (`dp0[0]`) a **y** (`dp24[1]`) těchto bodů lze vytvořit nový bod, který bude reprezentovat horní levý roh nového obdélníku tváře. Obdobně lze vytvořit i bod pt2, který bude reprezentovat pravý dolní bod.

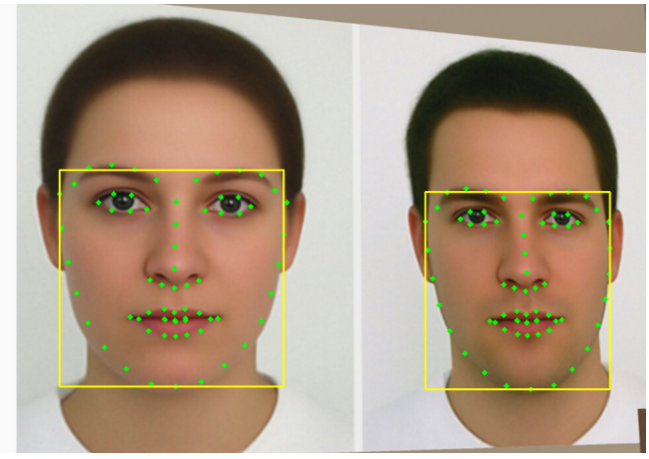
```
Point pt1 = new Point(dp0[0], dp24[1]);
```

Z těchto nových bodů lze vytvořit nový obdélník tváře:

```
Rect rect = new Rect(pt1, pt2);
ArrayList<Rect> facesArrayLandmark = new ArrayList<Rect>();
facesArrayLandmark.add(rect);
```

Tyto obdélníky je možné procházet a vykreslovat:





Pro záměnu tváří můžeme využít funkci z knihovny opencv **Photo.seamlessClone**, jen je pro ni nutné připravit následující vstupy:

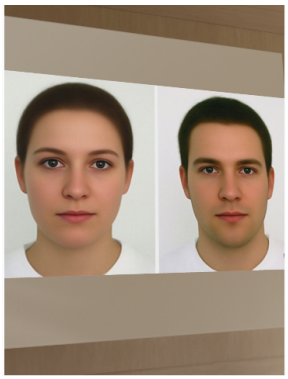
Photo.seamlessClone(src, dst, mask, center, out, Photo.NORMAL_CLONE);

src - v našem případě náhradní tvář - je vhodné změnit její velikost na velikost nalezených tváří (w, h, je šířka a výška žlutého obdélníku obdélníku kolem tváře - žlutý obdelník z předchozího obrázku získaný pomocí landmarků tváře):

```
Size sz = new Size(w,h);
Imgproc.resize( srcMat, resSource, sz );
cvtColor(srcMat, srcMat, COLOR_RGBA2RGB, 3);
```

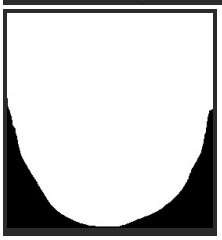
dst - v našem případě celá získaná fotka:

```
cvtColor(dstMat, dstMat, COLOR_RGBA2RGB, 3);
```

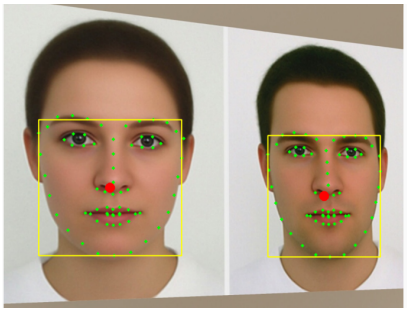


mask - maska, která musí být ve stejné velikosti src obraz - v našem případě může být maska bílý obraz- případně mask.jpg z adresáře drawable

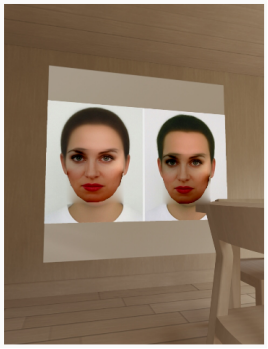
```
Mat mask = new Mat(src.size(),src.type(),new Scalar(255,255,255));
cvtColor(dstMat, dstMat, COLOR_RGBA2RGB, 3);
```



center - souřadnice středu tváře v **dst** obraze (červené body), pro výpočet je možné použít žluté obdélníky získané v jednom z předchozích bodů



out - výstupní obraz, do kterého se výsledek záměny vykreslí - případně může být i stejný jako dst



flag - způsob transformace, volit můžete mezi Photo.NORMAL_CLONE nebo Photo.MIXED_CLONE

Ještě pozor:
Vstupní obrazy do funkce **Photo.seamlessClone** musí být tří kanálové, takže je potřeba obrazy ve formátu Mat před vstupem do funkce **seamlessClone** konvertovat pomocí funkce **cvtColor**:
cvtColor(imgMat, imgMat, COLOR_RGBA2RGB, 3);

Dokumentace k funkci Photo.seamlessClone:
https://docs.opencv.org/3.4/df/da0/group__photo__clone.html#ga2bf426e4c93a6b1f21705513dfeca49d

Ukazka pouziti seamlessClone v jazyce c++/python
https://learnopencv.com/seamless-cloning-using-opencv-python-cpp/

LIVE CAMERA

https://github.com/opencv/opencv/tree/4.5.4/samples/android/tutorial-1-camerapreview