Face Detection Concepts Overview

Face detection is the process of automatically locating human faces in visual media (digital images or video). A face that is detected is reported at a position with an associated size and orientation. Once a face is detected, it can be searched for landmarks such as the eyes and nose.

- A **landmark** is a point of interest within a face. The left eye, right eye, and nose base are all examples of landmarks. The Face API provides the ability to find landmarks on a detected face.
- **Classification** is determining whether a certain facial characteristic is present. For example, a face can be classified with regards to whether its eyes are open or closed. Another example is whether the face is smiling or not.

Face Orientation

The face API detects faces at a range of different angles, as illustrated below:

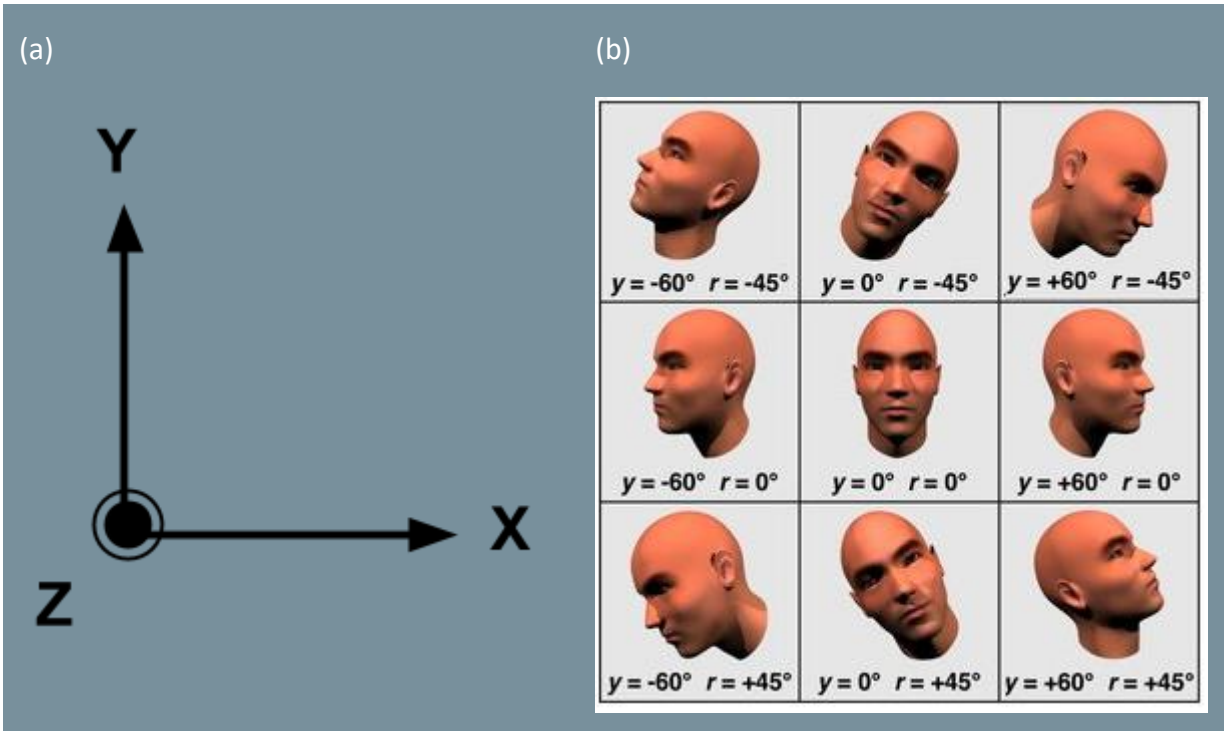

Fig. 1. Pose angle estimation. (a) The coordinate system with the image in the XY plane and the Z axis coming out of the figure. (b) Pose angle examples where y==Euler Y, r==Euler Z.

The **Euler X**, **Euler Y**, and **Euler Z** angles characterize a face's orientation as shown in Fig. 1. The Face API provides measurement of Euler Y and Euler Z (but not Euler X) for detected faces.

The Euler Z angle of the face is always reported. The Euler Y angle is available only when using the “accurate” mode setting of the face detector (as opposed to the “fast” mode setting, which takes some shortcuts to make detection faster). The Euler X angle is currently not supported.

Landmarks

A landmark is a point of interest within a face. The left eye, right eye, and nose base are all examples of landmarks. The figure below shows some examples of landmarks:

Rather than first detecting landmarks and using the landmarks as a basis of detecting the whole face, the Face API detects the whole face independently of detailed landmark information. For this reason, landmark detection is an optional step that could be done after the face is detected. Landmark detection is not done by default, since it takes additional time to run. You can optionally specify that landmark detection should be done.

The following table summarizes all of the landmarks that can be detected, for an associated face Euler Y angle: Each detected landmark includes its associated position in the image

Euler Y angle	**detectable landmarks**
< -36 degrees	left eye, left mouth, left ear, nose base, left cheek
-36 degrees to -12 degrees	left mouth, nose base, bottom mouth, right eye, left eye, left cheek, left ear tip
-12 degrees to 12 degrees	right eye, left eye, nose base, left cheek, right cheek, left mouth, right mouth, bottom mouth
12 degrees to 36 degrees	right mouth, nose base, bottom mouth, left eye, right eye, right cheek, right ear tip
> 36 degrees	right eye, right mouth, right ear, nose base, right cheek

Creating the face detector

In this example, the face detector is created in the onCreate method of the app's main activity. It is initialized with options for detecting faces with landmarks in a photo:

```
FaceDetector detector = new FaceDetector.Builder(context)
    .setTrackingEnabled(false)
    .setLandmarkType(FaceDetector.ALL_LANDMARKS)
    .build();
```

Setting "tracking enabled" to false is recommended for detection for unrelated individual images (as opposed to video or a series of consecutively captured still images), since this will give a more accurate result. But for detection on consecutive images (e.g., live video), having tracking enabled gives a more accurate and faster result.

Note that facial landmarks are not required in detecting the face, and landmark detection is a separate (optional) step. By default, landmark detection is not enabled since it increases detection time. We enable it here in order to visualize detected landmarks.

Detecting faces and facial landmarks

Given a bitmap, we can create Frame instance from the bitmap to supply to the detector:

```
Frame frame = new Frame.Builder().setBitmap(bitmap).build();
```

The detector can be called synchronously with a frame to detect faces:

```
SparseArray<Face> faces = detector.detect(frame);
```

The result returned includes a collection of Face instances. We can iterate over the collection of faces, the collection of landmarks for each face, and draw the result based on each landmark position:

```
for (int i = 0; i < faces.size(); ++i) {
    Face face = faces.valueAt(i);
    for (Landmark landmark : face.getLandmarks()) {
        int cx = (int) (landmark.getPosition().x * scale);
        int cy = (int) (landmark.getPosition().y * scale);
        canvas.drawCircle(cx, cy, 10, paint);
    }}
```

To detect the probability of smiling face

```
val face = mFaces!!.valueAt(i) // Take face i from the container of faces
var sp = face.isSmilingProbability
```

see (link) for full documentation

***public float** getIsSmilingProbability ()*

Returns a value between 0.0 and 1.0 giving a probability that the face is smiling.

This returns `UNCOMPUTED_PROBABILITY` if the probability was not computed. The probability is not computed if smile classification is not enabled via `setClassificationType(int)` or the required landmarks are not found. The`LEFT_MOUTH`, `RIGHT_MOUTH`, and `NOSE_BASE` landmarks are required to compute a smile probability.

Returns

- the probability that the face is smiling

Querying the face detector's operational status

The first time that an app using the Face API is installed on a device, GMS will download a native library to the device in order to do face detection. Usually this is done by the installer before the app is run for the first time. But if that download has not yet completed, then the above “detect” method will not detect any faces. This could happen if the user is not online, if the user lacks sufficient storage space on their device, or if the download is otherwise delayed (e.g., due to a slow network).

The detector will automatically become operational once the library download has been completed on device.

A detector’s isOperational method can be used to check if the required native library is currently available:

```
if (!detector.isOperational()) {
   // ...
}
```

Your app can take action based upon the operational state of a detector (e.g., temporarily disabling certain features, or displaying a notification to the user).

Releasing the face detector

The face detector uses native resources in order to do detection. For this reason, it is necessary to release the detector instance once it is no longer needed:

```
detector.release();
```

But note that you can reuse the same face detector instance for detection with multiple photos, if you choose