Real-time perspective correction in video stream

Vladislav Glagolev^{1,*}, and Alexander Ladonkin¹

¹Control Devices department, Tula State University, 300600, pr. Lenina, 92, Tula, Russia

Abstract. The paper describes an algorithm used for software perspective correction. The algorithm uses the camera's orientation angles and transforms the coordinates of pixels on a source image to coordinates on a virtual image from the camera whose focal plane is perpendicular to the gravity vector. This algorithm can be used as a low-cost replacement of a gyrostabilizer in specific applications that restrict using movable parts or heavy and pricey equipment.

1 Introduction

Various computer vision tasks may require a video camera images to satisfy a number of conditions. One of such conditions is a constantly horizontal orientation of the camera, which is especially relevant for video-based measurements. This means the camera should be always pointed down even if the object that holds it is leaned. The mentioned requirement can be satisfied using a gyrostabilizer, however, this solution implies fixing a camera on a movable platform, which greatly increases the system complexity and reduces its reliability.

Solving this problem in software seems to be a good option for wide-angle cameras. The general idea is to acquire the camera's orientation relative to the earth and wrap the perspective according to it.

2 Description

Consider an object that holds a video camera. Its orientation is defined by three angles: roll (γ), pitch (ϑ) and yaw (ψ) as shown on figure 1.

In the simplest case, the γ and ϑ angles can be determined using a micromechanical accelerometer, although the accelerometer-only measurements are the subject to significant errors due to linear acceleration. A complex attitude control system is a better option.

The γ and ϑ Euler angles determine the position of an object relative to the surface, which allows for transformation that compensate the perspective wrap.

The required transformation is performed with the matrix operation (1), which transforms raw pixel coordinates (x_0, y_0) into the (x, y) coordinates and compensates the perspective wrap.

---

*Corresponding author: tgupu@yandex.ru

Fig. 1. Euler angles

$$\begin{pmatrix} y_1 \\ z_1 \\ x_1 \end{pmatrix} = \begin{pmatrix} \cos \gamma & \sin \gamma & 0 \\ -\sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \vartheta & \sin \vartheta \\ 0 & -\sin \vartheta & \cos \vartheta \end{pmatrix} \times \begin{pmatrix} y_0 \\ z_c \\ x_0 \end{pmatrix}; \quad (1)$$

$$\begin{cases} x = x_1 \frac{f}{z_1}; \\ y = y_1 \frac{f}{z_1}, \end{cases}$$

where: z_c depends on camera's viewing angle and should be treated as a **response** of the perspective wrapper on angle values;

f —**scaling coefficient** that normally equals to the lens' focal length, but may be changed for scaling purposes.

The formula assumes that the origin of coordinates is located in the optical center (c_x, c_y) and the image is not distorted. These requirements can be achieved with the camera calibration.

The exact values of z_c and f parameters are different for each camera and can be determined experimentally, because the analytic expression $z_c = \frac{c_x}{\tan(\omega_{c_x})}$ includes the camera's viewing angle ω_{c_x} which is also an experimental value (in general case).

The rotation for a yaw angle is missing because the yaw stabilization is not a common task. If you need to stabilize the ψ angle, multiply the source vector $(y_0, z_c, x_0)^T$ by the matrix (2) before applying the formula (1).

$$\begin{pmatrix} \cos \psi & 0 & \sin \psi \\ 0 & 1 & 0 \\ -\sin \psi & 0 & \cos \psi \end{pmatrix} \quad (2)$$

The described method enables you to recalculate the points of interest coordinates or change the image by wrapping the perspective based on the four angle points' coordinates.

3 Implementation

In order to test the described method, a smartphone application was developed. Any modern smartphone has accelerometer and camera, which makes them a great platform for testing such methods. The application was created for an Android smartphone using the Kotlin language.

The OpenCV library was used to simplify image transformations and matrix operations. This library also implements camera calibration method [1], which was used to find the *Camera Matrix* (A) and *Distortion Coefficients* (D) (3) according to the Brown–Conrady’s camera model [2, 3]:

$$A = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}; \quad D = \begin{pmatrix} k_1 \\ k_2 \\ p_1 \\ p_2 \\ k_3 \end{pmatrix}. \quad (3)$$

The Kotlin implementation is an **UnTilter** class whose main methods are **remapPoint**, **remapFrame** and **unTilt**:

```
fun remapPoint(point: Point, pitch: Double, roll: Double): Point{
    val (sinGamma, cosGamma) = Pair(Math.sin(roll), Math.cos(roll)) // x
    val (sinTheta, cosTheta) = Pair(Math.sin(pitch), Math.cos(pitch)) // y

    val G = Mat(3, 3, CvType.CV_32F) // gamma transformation matrix
    G.put(0, 0, cosGamma); G.put(0, 1, sinGamma); G.put(0, 2, 0.0)
    G.put(1, 0, -sinGamma); G.put(1, 1, cosGamma); G.put(1, 2, 0.0)
    G.put(2, 0, 0.0); G.put(2, 1, 0.0); G.put(2, 2, 1.0)

    val T = Mat(3, 3, CvType.CV_32F) // theta transformation matrix
    T.put(0, 0, 1.0); T.put(0, 1, 0.0); T.put(0, 2, 0.0)
    T.put(1, 0, 0.0); T.put(1, 1, cosTheta); T.put(1, 2, -sinTheta)
    T.put(2, 0, 0.0); T.put(2, 1, sinTheta); T.put(2, 2, cosTheta)

    val A = Mat(3, 1, CvType.CV_32F) // source vector
    A.put(0, 0, point.y); A.put(1, 0, zc); A.put(2, 0, point.x)

    // 1.psi 2.theta 3.gamma: A = G . (T . (P . A))
    Core.gemm(G, T, 1.0, Mat(), 0.0, G) // G = G . T
    Core.gemm(G, A, 1.0, Mat(), 0.0, A) // A = G . A

    val z1 = A.get(1, 0)[0]
    return Point(A.get(2, 0)[0] * f / z1, A.get(0, 0)[0] * f / z1)
}

fun remapFrame(pitch: Double, roll: Double) {
    val c = Point(cameraMatrix.get(0, 2)[0], cameraMatrix.get(1, 2)[0])
    var p = Point(0.0, 0.0)
    for (i in 0..3) { // Calculating in four corners
        when (i) {
            0 -> { p.x = -c.x; p.y = -c.y } //top left
            1 -> { p.x = -c.x; p.y = imgHeight - c.y } //bottom left
            2 -> { p.x = imgWidth - c.x; p.y = imgHeight - c.y } //bottom right
            3 -> { p.x = imgWidth - c.x; p.y = -c.y } //top right
        }
        p = remapPoint(p, pitch, roll)
        p.x += c.x; p.y += c.y
        outputQuad.put(i, 0, p.x, p.y)
    }
}
```

```

fun unTilt(img: Mat, undistort: Boolean = false){
    if (undistort) Imgproc.remap(img, img, undistortMap1, undistortMap2, 1)
    Imgproc.warpPerspective(img, img,
        Imgproc.getPerspectiveTransform(inputQuad, outputQuad),
        Size(img.width().toDouble(), img.height().toDouble()))
}

```

Note that the sinuses in matrix transformation for the ϑ angle are inverted. The reason for that is the difference between the positive direction of the axis assumed in the **Sensor.TYPE_ROTATION_VECTOR** Android’s built-in virtual sensor and the direction assumed on figure 1.

The **inputQuad** and **outputQuad** matrices initialized as **Mat(4, 1, CvType.CV_32FC2)** are filled with the following values in the constructor:

```

inputQuad.put(0, 0,      0.0,      0.0) //top left
inputQuad.put(1, 0,      0.0, imgHeight) //bottom left
inputQuad.put(2, 0, imgWidth, imgHeight) //bottom right
inputQuad.put(3, 0, imgWidth,      0.0) //top right
inputQuad.copyTo(outputQuad) // No wrapping by default

```

4 Testing

The presented code was tested on a **Sony XPERIA mini ST15i** smartphone. The figure 2 shows the screenshots taken during the testing with different angles. To make the stabilization clearer, three images from figure 2 were overlapped, the result is shown on figure 3. Sample shots were taken by hands, thus there were linear movements of the device between shots. These movements were compensated with the operations of translation, scaling, and rotation.

Fig. 2. Screenshots

Fig. 3. The combination of three images acquired by multiplication of the images with 80% opacity

Objects remain approximately at the same place on the image regardless of the smart-phone's tilt. The accuracy of tilt compensation depends on camera calibration quality, the orientation angles' precision and the precision of the z_c parameter.

5 Conclusion

The described method is applicable when there is no need in the "untilted" image. For instance, some applications [4] require only the points of interest coordinates without the full image. In this case, the resulting coordinates can be calculated only for the points of interest and avoid the resource-demanding image transformations.

This also works for the undistortion procedure: it is possible to calculate the undistorted coordinates only for the points of interest without processing the entire image [5].

References

1. Z. Zhang, IEEE Trans. Pattern Anal. Mach. Intell. **22**, 1330 (2000)
2. D.C. Brown, Photometric Engineering **32**, 444 (1966)
3. A.E. Conrady, Monthly Notices of the Royal Astronomical Society **79**, 384 (1919)
4. V.M. Glagolev, A.V. Ladonkin, News of the Tula state university. Technical sciences **10**, 186 (2016)
5. V.M. Glagolev, News of the Tula state university. Technical sciences **9.2**, 188 (2017)