Location-Guided Scanning of Visual Codes from Long Distances: Supplementary Technical Report

Anonymous Author(s)

1 Code Localization through Computer Vision

In this report, we introduce approach to the code localization using computer vision techniques. For readers who want to learn more about the underlying concepts and related techniques, please refer to computer vision textbooks such as [3].

Recover camera pose in code space. Since the scale information cannot be determined from a single perspective camera, let us start our discussion in a code-centered coordinate system where the size of the code is normalized to 1 (called normalized code space). Let the four corners of the code in the code space be $\mathbf{x}_1 = (-0.5, -0.5, 0), \mathbf{x}_2 = (0.5, -0.5, 0), \mathbf{x}_3 = (-0.5, 0.5, 0), \mathbf{x}_4 = (0.5, 0.5, 0)$. The rigid transform from code space to the camera space is formulated using homogeneous coordinates as

$$\mathbf{x}^{\text{Cam}} = \mathbf{T}_{xy}\mathbf{T}_z\mathbf{R}_z\mathbf{R}_y\mathbf{R}_x\mathbf{x}\,, \tag{1}$$

where a point $\mathbf{x} = (x, y, z, 1)$ is first rotated and then translated. The translation is separated into the xy-component and z-component. The rotation is parameterized by extrinsic Euler angles in the order of x, y, z. Following are the expressions for these matrices.

$$\mathbf{T}_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{T}_{xy} = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2}$$

$$\begin{aligned} \mathbf{R}_x &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \\ \mathbf{R}_y &= \begin{pmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \\ \mathbf{R}_z &= \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \end{aligned} \tag{3}$$

Since only the front face of the code is visible, we add another constraint that $-\frac{\pi}{2} < \alpha < \frac{\pi}{2}, -\frac{\pi}{2} < \beta < \frac{\pi}{2}$.

The point is then projected onto the image plane:

$$\mathbf{u} = \mathbf{K}\mathbf{x}^{\text{Cam}} = \mathbf{K}\mathbf{T}_{xy}\mathbf{T}_z\mathbf{R}_z\mathbf{R}_y\mathbf{R}_x\mathbf{x} = (u, v, 1)^{\text{T}}\,, \tag{4}$$

where $\mathbf{K}$, in general, is the camera intrinsic matrix that follows a perspective camera model (assuming no nonlinear lens distortion):

$$\mathbf{K}_{\text{persp}} = \begin{pmatrix} f_u & 0 & u_0 & 0 \\ 0 & f_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{5}$$

where f_u, f_v are the horizontal and vertical focal lengths terms in pixels, (u_0, v_0) is the principal point. In many latest smartphone models, the camera intrinsic matrix is pre-calibrated and is available through API. Here we focus on the more general, challenging case where such information is not available. First, we make the assumption that the camera intrinsics can be parameterized by a single parameter: focal length f. This assumption holds when the following conditions are met:

- Aspect ratio of the pixels is one ($f_u = f_v$).
- Principal point is at the center of the image ($u_0 = v_0 = 0$).

It is known that, under the assumptions above, when correspondences of four corners of a planar target is known, it is possible to estimate the camera pose and focal length simultaneously [1]. However, the solution becomes numerically unstable when the depth variation of the target is small compared to its absolute depth to the camera. In the extreme case, when the code is parallel to the image plane, there is an ambiguity between depth and focal length. Such scenarios happen frequently when people pull out their phone to scan a code. Therefore, we do not recover the focal length and instead make the following two additional assumptions:

- The depth variation of the target is small compared to its absolute depth to the camera.
- The object is close to the optical axis, *i.e.*, the center of the code is close to the center of the image.

Notice these assumptions are easy to verify, which means that we can pick the user images that meet the assumptions and only use them to localize the code. When these assumptions are met,the camera model can be approximated by the weak-perspective camera model:

$$\mathbf{K}_{\text{weak}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & d_z/f \end{pmatrix}. \tag{6}$$

A weak-perspective model preserves parallelism. In other words, the image of the code is always a parallelogram, and only three corner correspondences provide useful information (as the fourth can be derived from the three). Here we only consider $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ because these are the corners where the fiducials are located, which are more accurately detected.

Now let us get back to Eq. 4. Note that the order of $\mathbf{T}_z$ and $\mathbf{R}_z$ can be swapped without changing the result:

$$\mathbf{u} = \mathbf{K}\mathbf{T}_{xy}\mathbf{T}_z\mathbf{R}_z\mathbf{R}_y\mathbf{R}_x\mathbf{x} = \mathbf{K}\mathbf{T}_{xy}\mathbf{R}_z\mathbf{T}_z\mathbf{R}_y\mathbf{R}_x\mathbf{x}\,. \tag{7}$$

To reduce the number of unknown variables to estimate, we pre-process the captured image by

- Translate the upper-left corner ($\mathbf{x}_1$)) to the center of the image. Denote the translation vector as $\mathbf{q} = (q_u, q_v)$.
- Rotate the image so that the upper-right corner ($\mathbf{x}_2$) is on the same horizontal line as the upper-left corner and on the right of it. Denote the rotation angle as ξ.

This is equivalent to compensate for $\mathbf{T}_{xy}$ and $\mathbf{R}_z$ in the image space. Notice that

$$\begin{aligned} \mathbf{K}\mathbf{T}_{xy} &= \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 0 & d_z/f \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & d_x f/d_z \\ 0 & 1 & d_y f/d_z \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & d_z/f \end{pmatrix} \\ &= \mathbf{C}_1\mathbf{K}\,. \end{aligned} \tag{8}$$

$$
\begin{aligned}
\mathbf{KR}_z &= \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 0 & d_z/f \end{pmatrix} \\
&= \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & d_z/f \end{pmatrix} \\
&= \mathbf{C}_2\mathbf{K}\,.
\end{aligned} \tag{9}
$$

Therefore, the pre-warping process can be described mathematically as:

$$\mathbf{u} = \mathbf{K}\mathbf{T}_{xy}\mathbf{R}_z\mathbf{T}_z\mathbf{R}_y\mathbf{R}_x\mathbf{x}\,, \tag{10}$$

$$\mathbf{u} = \mathbf{C}_1\mathbf{C}_2\mathbf{K}\mathbf{T}_z\mathbf{R}_y\mathbf{R}_x\mathbf{x}\,, \tag{11}$$

$$\mathbf{C}_2^{-1}\mathbf{C}_1^{-1}\mathbf{u} = \mathbf{K}\mathbf{T}_z\mathbf{R}_y\mathbf{R}_x\mathbf{x}\,. \tag{12}$$

Let $\tilde{u}$ be the image coordinates after warping:

$$\tilde{\mathbf{u}} = (\tilde{u}, \tilde{v}, 1) = \mathbf{C}_2^{-1}\mathbf{C}_1^{-1}\mathbf{u}\,. \tag{13}$$

The warp parameters are then related to the pose parameters by:

$$q_u = -d_x f/d_z + 0.5, \quad q_v = -d_y f/d_z + 0.5, \quad \xi = -\gamma. \tag{14}$$

Now we plug the coordinates of the warped corners into Eq. 12, we get three independent equations,

$$\tilde{u}_2 = \cos\beta\frac{f}{d_z}, \quad \tilde{u}_3 = \sin\alpha\sin\beta\frac{f}{d_z}, \quad \tilde{v}_3 = \cos\alpha\frac{f}{d_z}. \tag{15}$$

Eliminate β, f and d_z, we get a quadratic equation in terms of $r = \cos^2\alpha$,

$$ar^2 + br + c = 0 \tag{16}$$

$$a = \tilde{u}_2^2,\ b = \tilde{u}_2^2 + \tilde{u}_3^2 + \tilde{v}_3^2,\ c = \tilde{v}_3^2 \tag{17}$$

The root can then be found using the quadratic formula:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{18}$$

It is easy to show that $r_1 > 1$. Therefore, the only possible solution is (remember $-\frac{\pi}{2} < \alpha < \frac{\pi}{2}$)

$$\cos\alpha = \sqrt{r_2} \tag{19}$$

The remaining pose parameters can then be computed,

$$
\begin{aligned}
\cos\beta = \frac{\tilde{u}_2}{\tilde{v}_3}\sqrt{r_2}\,, \quad \gamma = -\xi\,, \quad \frac{d_z}{f} = \frac{\sqrt{r_2}}{\tilde{v}_3}\,, \\
d_x = -\frac{\sqrt{r_2}}{\tilde{v}_3}q_u\,, \quad d_y = -\frac{\sqrt{r_2}}{\tilde{v}_3}q_v\,.
\end{aligned} \tag{20}
$$

Depending on the sign of $\tilde{u}_3$, there are two possibilities for α and β. If $\tilde{u}_3 > 0$,

$$\begin{cases} \alpha = \arccos\sqrt{r_2} \\ \beta = \arccos\frac{\tilde{u}_2}{\tilde{v}_3}\sqrt{r_2} \end{cases}, \text{ or } \begin{cases} \alpha = -\arccos\sqrt{r_2} \\ \beta = -\arccos\frac{\tilde{u}_2}{\tilde{v}_3}\sqrt{r_2} \end{cases}. \tag{21}$$

Otherwise,

$$\begin{cases} \alpha = \arccos \sqrt{r_2} \\ \beta = -\arccos \frac{\tilde{u}_2}{\tilde{v}_3} \sqrt{r_2} \end{cases} \text{, or } \begin{cases} \alpha = -\arccos \sqrt{r_2} \\ \beta = \arccos \frac{\tilde{u}_2}{\tilde{v}_3} \sqrt{r_2} \end{cases} . \tag{22}$$

Finally, the camera location in the code space can be found by applying the inverse of the rigid transform:

$$\begin{aligned} \mathbf{x} &= \mathbf{R}_x^{-1} \mathbf{R}_y^{-1} \mathbf{R}_z^{-1} \mathbf{T}_z^{-1} \mathbf{T}_{xy}^{-1} (0,0,0,1)^{\mathrm{T}} \\ &= \mathbf{R}_x^{-1} \mathbf{R}_y^{-1} \mathbf{R}_z^{-1} ((-d_x, -d_y, 0, 1) - d_z(0,0,1,0))^{\mathrm{T}} . \end{aligned} \tag{23}$$

Notice that we assume the code is close to the optical axis, *i.e.*, $d_x \approx 0$, $d_y \approx 0$.

$$\mathbf{x} \approx -d_z \mathbf{R}_x^{-1} \mathbf{R}_y^{-1} \mathbf{R}_z^{-1} \cdot (0,0,1,0)^{\mathrm{T}} . \tag{24}$$

In other words, although we do not know the depth of the code, we know in which direction it lies with respect to the code, which gives a line constraint passing through the center of the code. Notice that since there are two possibilities for α and β, the camera can lie on either of the two rays.

Recover code pose in world space. As mentioned in Sec. 1, once we get a bundle of rays in the code space, the next step is to find the 3D translation and rotation to transform the code to world space such that the rays pass through the GPS locations of user cameras. One important observation here is that this problem can be converted to the Perspective-n-Points (PnP) problem [2], which is a well-studied problem in computer vision for which many open source algorithms have been developed. Such a conversion is done by treating the code as a "virtual camera", whose focal length can be determined arbitrarily, *e.g.*, chosen to be 1:

$$\mathbf{K}_{\text{code}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{25}$$

The rays are then mapped to 2D coordinates on a "virtual image plane" defined by $\mathbf{K}_{\text{code}}$. For a code scanned by n users, we get n correspondences between 3D objects (user cameras) and their 2D projections. The code pose is then solved using OpenCV's SOLVEPNP_ITERATIVE method, which forms the problem as a nonlinear optimization by minimizing the reprojection errors.

Notice that there is an ambiguity in the estimated camera pose: the camera can lie on either of two rays in the code space. In some applications, it is reasonable to assume that the code lies always higher than the code so only one ray is possible. In general, the code pose can be found using RANSAC:

- Randomly choose n cameras. For each camera, randomly choose either of the two rays.
- Compute the code pose from the chosen n rays.
- For each camera, choose the ray that gives lower reprojection error. Count the number of outliers (cameras).
- Repeat m times. Return the code pose with lowest number of outliers.
- Refine the code pose from the set of inliers.

References

[1] Abidi, M., and Chandra, T. A new efficient and direct solution for pose estimation using quadrangular targets: Algorithm and evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 17*, 5 (May 1995), 534–538.

[2] Fischler, M. A., and Bolles, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6 (June 1981), 381–395.

[3] Hartley, R., and Zisserman, A. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.