In order to solve for the four unknowns t_x, t_y, t_z, and r_y, one requires minimal four Plücker line correspondences. This gives rise to a system of four polynomials with the other three polynomials in similar form as equation (2.10) with the coefficients denoted by b_1 to b_8, c_1 to c_8 and d_1 to d_8. After stacking all four correspondences and separating r_y from t_x, t_y, t_z, we arrive at an equation system

$$\underbrace{\begin{pmatrix} a_2 + a_6 r_y & a_3 + a_7 r_y & a_4 + a_8 r_y & a_1 + a_5 r_y \\ b_2 + b_6 r_y & b_3 + b_7 r_y & b_4 + b_8 r_y & b_1 + b_5 r_y \\ c_2 + c_6 r_y & c_3 + c_7 r_y & c_4 + c_8 r_y & c_1 + c_5 r_y \\ d_2 + d_6 r_y & d_3 + d_7 r_y & d_4 + d_8 r_y & d_1 + d_5 r_y \end{pmatrix}}_{\mathbf{M}(r_y)} \begin{bmatrix} t_x \\ t_y \\ t_z \\ 1 \end{bmatrix} = \mathbf{0}. \tag{2.12}$$

Since $\mathbf{M}\left(r_y\right)$ is a square matrix and it has a non-trivial solution when the determinant of $\mathbf{M}\left(r_y\right)$ is zero, consequently:

$$A r_y^4 + B r_y^3 + C r_y^2 + D r_y + E = 0, \tag{2.13}$$

where A, B, C, D and E are formed from the coefficients a, b, c and d of the system of polynomials. Since the univariate polynomial (2.13) is 4-th order, it admits closed form solutions with maximum 4 real roots (its analytic solutions are given in PlanetMath). Once the rotation is found, we simply use QR decomposition of the above equation system to solve for the translation. Finally, the approximation $\Delta\mathbf{R}_y$ in equation (2.8) is projected to $SO(3)$ by using the Rodrigues' formula.

2.3.2 Absolute camera pose

Given 2D–3D matches between 2D feature points extracted from a query image and 3D points from the database map, we are ready to estimate the absolute pose of a camera. If the intrinsic matrix $\mathbf{K}$ is unknown, a Perspective-6-Point method (Hartley and Zisserman [2003]) is well-adopted. The merit of this P6P method is that it can be easily solved using the *Direct Linear Transform*, and used for end-to-end training a deep network. Our implementation in Tensorflow is available at https://github.com/Liumouliu/Deep_blind_PnP/blob/master/utils/tf_pose_loss.py.

When assuming the 2D pixel locations of the principle point of a camera lying on the center of an image, a P4P method (Bujnak et al. [2008]) which simultaneously solves the camera pose and focal length can be adopted. If a camera is fully calibrated with known $\mathbf{K}$, the most common one would be the P3P method (Kneip et al. [2011]). Furthermore, if the vertical direction can be obtained, the most accurate and efficient one is the P2P method (Sweeney et al. [2015]).

In the following, we present a P2P method which is our original contribution. The proposed method achieves exactly the same precision as the method (Sweeney et al. [2015]), with a 20% speed-up. Our code is available at https://github.com/Liumouliu/p2p_vertical_direction.

We again utilize Plücker line to represent each ray since it includes both the camera center position and ray direction. We denote a 6-vector Plücker line as $\boldsymbol{\ell}_j = \left[\mathbf{u}_j^{\top}, \mathbf{q}_j^{\top}\right]^{\top}$, where j is the line index, $\mathbf{u}_j$ is the ray direction and $\mathbf{q}_j$ is the ray starting point.

A 3D Point $\mathbf{X}_j^G$ lying on the line is given by (Lee et al. [2015])

$$\mathbf{X}_j^G = \mathbf{q}_j + \lambda_j \mathbf{u}_j, \tag{2.14}$$

where λ_j is the depth of the point and G stand for the local camera coordinate system.

Our goal is to solve the unknown rotation $\mathbf{R}$ and translation $\mathbf{t}$ such that they transform a 3D point $\mathbf{X}_j^W$ in the world frame W to the local frame G

$$\mathbf{X}_j^G = \mathbf{R}\mathbf{X}_j^W + \mathbf{t}. \tag{2.15}$$

With vertical direction, Plücker lines can be aligned to the world coordinate system, and the problem is transformed to solve the unknown rotation angle s about the vertical direction $\mathbf{g}$ and translation $\mathbf{t}$, such that

$$\mathbf{q}_j + \lambda_j \mathbf{u}_j = \mathbf{R}_y\left(\mathbf{g}, s\right)\mathbf{X}_j^W + \mathbf{t}. \tag{2.16}$$

Sweeney et al. [2015] utilize the distance preserving nature of $SE(3)$ transformation. They first solve depths of 3D points and then align points in the coordinate system G to W to solve $\mathbf{R}$ and $\mathbf{t}$.

In different from the method (Sweeney et al. [2015]), we directly solve $\mathbf{R}$ and $\mathbf{t}$ and do not estimate the unknown depths of 3D points. Equation (2.16) can be rearranged to eliminate the unknown depth λ_j,

$$\mathbf{u}_j \times \left[\mathbf{R}_y\left(\mathbf{g}, s\right)\mathbf{X}_j^W + \mathbf{t} - \mathbf{q}_j\right] = \mathbf{0}. \tag{2.17}$$

This gives three equations, only two of which are linearly independent since $\operatorname{rank}\left(\left[\mathbf{u}_j\right]_\times\right) = 2$. In total, we have four unknowns (one for the angle s and three for the translation $\mathbf{t}$), thus two 2D–3D correspondences are needed to solve the absolute pose.

The normalized rotation matrix $\mathbf{R}_y\left(\mathbf{g}, s\right)$ is given by

$$\mathbf{R}_y\left(\mathbf{g}, s\right) = \frac{1}{s^2+1}\begin{bmatrix} s^2-1 & -2s & 0 \\ 2s & s^2-1 & 0 \\ 0 & 0 & s^2+1 \end{bmatrix}. \tag{2.18}$$

Substituting equation (2.18) into equation (2.17), the first two linearly independent rows are

$$\begin{cases} a_1 t_2 + a_2 t_3 + \frac{a_3}{s^2+1} + \frac{a_4 s}{s^2+1} + \frac{a_5 s^2}{s^2+1} + a_6 = 0 \\ a_7 t_1 + a_8 t_3 + \frac{a_9}{s^2+1} + \frac{a_{10} s}{s^2+1} + \frac{a_{11} s^2}{s^2+1} + a_{12} = 0 \end{cases}, \tag{2.19}$$

where coefficients $a_1 \sim a_{12}$ are given by

$$\begin{cases} a_1 = -\mathbf{u}_{j,3} \\ a_2 = \mathbf{u}_{j,2} \\ a_3 = \mathbf{X}^W_{j,2}\mathbf{u}_{j,3} \\ a_4 = -2\mathbf{X}^W_{j,1}\mathbf{u}_{j,3} \\ a_5 = -\mathbf{X}^W_{j,2}\mathbf{u}_{j,3} \\ a_6 = \mathbf{X}^W_{j,3}\mathbf{u}_{j,2} + \mathbf{q}_{j,2}\mathbf{u}_{j,3} - \mathbf{q}_{j,3}\mathbf{u}_{j,2} \\ a_7 = \mathbf{u}_{j,3} \\ a_8 = -\mathbf{u}_{j,1} \\ a_9 = -\mathbf{X}^W_{j,1}\mathbf{u}_{j,3} \\ a_{10} = -2\mathbf{X}^W_{j,2}\mathbf{u}_{j,3} \\ a_{11} = \mathbf{X}^W_{j,1}\mathbf{u}_{j,3} \\ a_{12} = \mathbf{q}_{j,3}\mathbf{u}_{j,1} - \mathbf{q}_{j,1}\mathbf{u}_{j,3} - \mathbf{X}^W_{j,3}\mathbf{u}_{j,1}. \end{cases} \tag{2.20}$$

One more 2D–3D correspondence gives coefficients $b_1 \sim b_{12}$, which are in the same form as equation (2.20). Stacking the two gives

$$\begin{cases} a_1 t_2 + a_2 t_3 + \dfrac{a_3}{s^2+1} + \dfrac{a_4 s}{s^2+1} + \dfrac{a_5 s^2}{s^2+1} + a_6 = 0 & \text{(2.21a)} \\ a_7 t_1 + a_8 t_3 + \dfrac{a_9}{s^2+1} + \dfrac{a_{10} s}{s^2+1} + \dfrac{a_{11} s^2}{s^2+1} + a_{12} = 0 & \text{(2.21b)} \\ b_1 t_2 + b_2 t_3 + \dfrac{b_3}{s^2+1} + \dfrac{b_4 s}{s^2+1} + \dfrac{b_5 s^2}{s^2+1} + b_6 = 0 & \text{(2.21c)} \\ b_7 t_1 + b_8 t_3 + \dfrac{b_9}{s^2+1} + \dfrac{b_{10} s}{s^2+1} + \dfrac{b_{11} s^2}{s^2+1} + b_{12} = 0, & \text{(2.21d)} \end{cases}$$

which can be simplified following the steps:

1. Combing (2.21a) and (2.21c) to eliminate t_2;
2. Combing (2.21b) and (2.21d) to eliminate t_1;
3. Combing equations from step 1 and 2 to eliminate t_3.

This gives a quadratic polynomial with respect to s

$$As^2 + Bs + C = 0, \tag{2.22}$$

where coefficients A, B and C are given by

$$\begin{cases} A = \frac{a_5 b_1 - a_1 b_5 + a_6 b_1 - a_1 b_6}{a_2 b_1 - a_1 b_2} - \frac{a_{11} b_7 - a_7 b_{11} + a_{12} b_7 - a_7 b_{12}}{a_8 b_7 - a_7 b_8} \\ B = \frac{a_4 b_1 - a_1 b_4}{a_2 b_1 - a_1 b_2} - \frac{a_{10} b_7 - a_7 b_{10}}{a_8 b_7 - a_7 b_8} \\ C = \frac{a_3 b_1 - a_1 b_3 + a_6 b_1 - a_1 b_6}{a_2 b_1 - a_1 b_2} - \frac{a_9 b_7 - a_7 b_9 + a_{12} b_7 - a_7 b_{12}}{a_8 b_7 - a_7 b_8} \end{cases}. \tag{2.23}$$

Solving equation (2.22) gives at most 2 real solutions. After obtaining s, the translation $\mathbf{t}$ can be solved by the least-square fitting of the equation

$$\begin{pmatrix} 0 & a_1 & a_2 \\ a_7 & 0 & a_8 \\ 0 & b_1 & b_2 \\ b_7 & 0 & b_8 \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = -\frac{1}{s^2+1}\mathbf{D}(s), \tag{2.24}$$

where $\mathbf{D}(s)$ is given by

$$\mathbf{D}(s) = \begin{pmatrix} (a_5+a_6)s^2 + a_4 s + a_3 + a_6 \\ (a_{11}+a_{12})s^2 + a_{10}s + a_9 + a_{12} \\ (b_5+b_6)s^2 + b_4 s + b_3 + b_6 \\ (b_{11}+b_{12})s^2 + b_{10}s + b_9 + b_{12} \end{pmatrix}. \tag{2.25}$$

Using pseudo-inverse of the left 4×3 matrix of equation (2.24) to obtain the least-square solution of $(t_1, t_2, t_3)^{\mathsf{T}}$ is time-consuming. We can do much better by exploiting the structure of the left 4×3 matrix.

t_3 can be solved directly by combing the 1^{st} and 3^{rd} row. Similarly, t_3 can be also solved by combing the 2^{nd} and 4^{th} row. Finally, t_3 is averaged using the above two solutions. Once t_3 is obtained, it is substituted into the 1^{st} and 3^{rd} row to solve t_2 and the 2^{nd} and 4^{th} row to solve t_1.

Root polishing. In general, our method gives at most two solutions. Utilizing the positive depth constraint ($\lambda_j \geq 0$), we can quickly prune a false solution by checking

$$\mathbf{u}_j^T \cdot \left[\mathbf{R}_y(\mathbf{g}, s)\mathbf{X}_j^W + \mathbf{t} - \mathbf{q}_j\right] \geq \mathbf{0}. \tag{2.26}$$

2.4 Building a Large-Scale Map

A pre-stored map is a prerequisite for camera-based localization. There is no universal agreement on what should be included in the map. In this thesis, we exploit three easily available sources: (i) ground-view images; (ii) satellite images; (iii) 3D points cloud. The three sources can be pre-registered easily by using geographical coordinates. An example of the ANU campus map is given in Figure 1.5. In the following, we separately introduce engineering practices to build a map from the three sources.

2.4.1 Building a ground-view image database

Ground-view image database can be obtained from the Google street view, Flicker, or a handcrafted mapping system. In the following, we separately introduce two methods to obtain a ground-view image database: (i) the Google street view; (ii) a handcrafted GPS/IMU/stereo mapping system.