Algorithm 14.3: Inferring 3D world points (reconstruction)

Given J calibrated cameras in known positions (i.e. cameras with known $\boldsymbol{\Lambda}, \boldsymbol{\Omega}, \boldsymbol{\tau}$), viewing the same three-dimensional point $\mathbf{w}$ and knowing the corresponding projections in the images $\{\mathbf{x}_j\}_{j=1}^J$, establish the position of the point in the world.

As for the previous algorithms the final solution depends on a non-linear minimization of the reprojection error between $\mathbf{w}$ and the observed data $\mathbf{x}_j$,

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\text{argmin}} \left[\sum_{j=1}^{J} (\mathbf{x}_j - \mathbf{pinhole}[\mathbf{w}, \boldsymbol{\Lambda}_j, \boldsymbol{\Omega}_j, \boldsymbol{\tau}_j])^T (\mathbf{x}_j - \mathbf{pinhole}[\mathbf{w}, \boldsymbol{\Lambda}_j, \boldsymbol{\Omega}_j, \boldsymbol{\tau}_j]) \right]$$

The algorithm below finds a good approximate initial conditions for this minimization using a closed-form least-squares solution.

Algorithm 14.3: Inferring 3D world position

Input : Image points $\{\mathbf{x}_j\}_{j=1}^J$, camera parameters $\{\boldsymbol{\Lambda}_j, \boldsymbol{\Omega}_j, \boldsymbol{\tau}_j\}_{j=1}^J$
Output: 3D world point $\mathbf{w}$
begin
  for j=*1* **to** J **do**
    `// Convert to normalized camera coordinates`
    $\mathbf{x}'_j = \boldsymbol{\Lambda}_j^{-1}[x_j, y_j, 1]^T$
    `// Compute linear constraints`
    $a_{1j} = [\omega_{31j}x'_j - \boldsymbol{\omega}_{11j}, \omega_{32j}x'_j - \omega_{12j}, \omega_{33j}x'_j - \omega_{13j}]$
    $a_{2j} = [\omega_{31j}y'_j - \boldsymbol{\omega}_{11j}, \omega_{32j}y'_j - \omega_{12j}, \omega_{33j}y'_j - \omega_{13j}]$
    $b_j = [\tau_{xj} - \tau zjx'_j; \tau_{yj} - \tau_{zj}y'_j]$
  end
  `// Stack linear constraints`
  $\mathbf{A} = [a_{11}; a_{21}; a_{12}; a_{22}; \ldots a_{1J}; a_{2J}]$
  $\mathbf{b} = [b_1; b_2; \ldots b_J]$
  `// LS solution for parameters`
  $\mathbf{w} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$
  `// Refine parameters with non-linear optimization`
  $\hat{\mathbf{w}} = \text{argmin}_{\mathbf{w}} \left[\sum_{j=1}^{J} (\mathbf{x}_j - \mathbf{pinhole}[\mathbf{w}, \boldsymbol{\Lambda}_j, \boldsymbol{\Omega}_j, \boldsymbol{\tau}_j])^T (\mathbf{x}_j - \mathbf{pinhole}[\mathbf{w}, \boldsymbol{\Lambda}_j, \boldsymbol{\Omega}_j, \boldsymbol{\tau}_j]) \right]$
end

Algorithm 16.1: Camera geometry from point matches

This algorithm finds approximate estimates of the rotation and translation (up to scale) between two cameras given a set of I point matches $\{\mathbf{x}_{i1}, \mathbf{x}_{i2}\}_{i=1}^{I}$ between two images. More precisely, the algorithm assumes that the first camera is at the world origin and recovers the extrinsic parameters of the second camera.

There is a fourfold ambiguity in the possible solution due to the symmetry of the camera model - it allows for points that are behind the camera to be imaged, although this is clearly not possible in the real world. This algorithm distinguishes between these four solutions by reconstructing all of the points with each and choosing the solution where the largest number are in front of both cameras.

Algorithm 16.1: Extracting relative camera position from point matches

Input : Point pairs $\{\mathbf{x}_{i1}, \mathbf{x}_{i2}\}_{i=1}^{I}$, intrinsic matrices $\boldsymbol{\Lambda}_1, \boldsymbol{\Lambda}_2$
Output: Rotation $\boldsymbol{\Omega}$, translation $\boldsymbol{\tau}$ between cameras
begin
  // Compute fundamental matrix (algorithm 16.2)
  $\mathbf{F} = \text{ComputeFundamental}[\{\mathbf{x}_{1i}, \mathbf{x}_{2i}\}_{i=1}^{I}]$
  // Compute essential matrix
  $\mathbf{E} = \boldsymbol{\Lambda}_2^T \mathbf{F} \boldsymbol{\Lambda}_1$
  // Extract four possible rotation and translations from $\mathbf{E}$
  $\mathbf{W} = [0, -1, 0; 1, 0, 0; 0, 0, -1]$
  $[\mathbf{U}, \mathbf{L}, \mathbf{V}] = \mathbf{svd}[\mathbf{E}]$
  $\tau_1 = \mathbf{ULWU}^T; \boldsymbol{\Omega}_1 = \mathbf{UW}^{-1}\mathbf{V}^T$
  $\tau_2 = \mathbf{ULW}^{-1}\mathbf{U}^T; \boldsymbol{\Omega}_2 = \mathbf{UWV}^T$
  $\tau_3 = -\tau_1; \boldsymbol{\Omega}_3 = \boldsymbol{\Omega}_1$
  $\tau_4 = -\tau_2; \boldsymbol{\Omega}_4 = \boldsymbol{\Omega}_1$
  // For each possibility
  for k=*1* **to** K **do**
    $t_k = 0$ // number of points in front of camera for k^{th} soln
    // For each point
    for i=*1* **to** I **do**
      // Reconstruct point (algorithm 14.3)
      $\mathbf{w} = \text{Reconstruct}[\mathbf{x}_{i1}, \mathbf{x}_{i2}, \boldsymbol{\Lambda}_1, \boldsymbol{\Lambda}_2, \mathbf{0}, \mathbf{I}, \boldsymbol{\Omega}_k, \boldsymbol{\tau}_k]$
      // Compute point in frame of reference of second camera
      $\mathbf{w}' = \boldsymbol{\Omega}_k + \boldsymbol{\tau}_k$
      // Test if point reconstructed in front of both cameras
      if $\mathbf{w}_3 > 0$ *&* $\mathbf{w}'_3 > 0$ **then**
        $t_k = t_k + 1$
      end
    end
  end
  // Choose solution with most support
  $k = \text{argmax}_k[t_k]$
  $\boldsymbol{\Omega} = \boldsymbol{\Omega}_k$
  $\boldsymbol{\tau} = \boldsymbol{\tau}_k$
end