AprilTag visual error measurement

Fixed inputs:

1. s: known AprilTag size in meters
2. k known AprilTag corner coordinates in a reference frame. There may be more feature point in the tag, but at least the 4 corners ($k = 4$): $z_1 = (0, 0), z_2 = (0, s), z_3 = (s, 0), z_4 = (s, s)$. Mark the homogeneous 4D-coorinates as, e.g, $\hat{z}_2 = [0, s, 0, 1]^T$.

Variable inputs (per session): For each frame $j = 1, 2, \ldots, n$:

- World-to-camera matrix from a VIO method (e.g., ARCore) $V_j \in \mathbb{R}^{4\times4}$
- Projection matrix from the VIO method (may be fixed) $P_j \in \mathbb{R}^{3\times4}$. NOTE: remove the row that computes the depth coordinate from the 4x4 matrix. Also make sure it is scaled so that the output, after "dividing by w" is in pixel coordinates and not some normalized screen coordinates.
- List of AprilTag pixel coordinates for a single tag $y_1^j, \ldots, y_k^j$.

Now if we assume we know the pose $M \in \mathbb{R}^{4\times4}$ (model-to-world matrix) of the AprilTag in the AR/VIO coordinates, we can compute the projection error of the pixel coordinates as

$$E(M) = \sum_j \sum_k \left\| g(P_j V_j M \hat{z}_k) - y_k^j \right\|^2 \tag{1}$$

where

$$g([x, y, w]^T) = \left[\frac{x}{w}, \frac{y}{w}\right]^T. \tag{2}$$

The error we are interested in is computed by finding the pose M that minimizes it:

$$E_* = \min_{M \in SE(3)} E(M) \tag{3}$$

It can be computed by first approximating the model-to-camera matrix, C_j for some frame j based on the AprilTags (with the known projection matrix P_j), then computing

$$M_0 = V_j^{-1} C_j \tag{4}$$

and finally computing the optimal value M_* by iteratively minimizing the the error in equation 1 using the Gauss–Newton method, starting from the initial guess M_0. It probably makes sense to parametrize the SE(3)-matrices (rotation + translation) using quaternion rerpresentation while doing the minimization.

Minimizing the error

State representation

To minimize the error, the matrix M should be parametrized as a quaternion $q \in \mathbb{R}^4$ and a translation vector $t \in \mathbb{R}^3$. To make this fit into a standard optimization framework, these are concatenated into a 7-dimensional vector

$$\mathbf{x} = \begin{bmatrix} t \\ q \end{bmatrix} = [t_x, t_y, t_z, q_w, q_x, q_y, q_z]^T. \tag{5}$$

Then

$$M(\mathbf{x}) = M(t, q) = \begin{bmatrix} R(q) & t \\ & 1 \end{bmatrix}, \tag{6}$$

where $R(q)$ is a quaternion-to-matrix operation, see, *Simo Särkkä's quaternion notes (link)*, equation 27.

Differentiation

To be able to use the Gauss–Newton algorithm, one must be able to compute the Jacobian matrices of things inside E with respect to the state $\mathbf{x}$. This is best done by repeatedly utilizing the chain rule of differentiation:

$$J_{g(f(\mathbf{x}))}(x) = J_g(f(\mathbf{x}))J_f(x) \tag{7}$$

Here J denotes the Jacobian matrix of a function and the products above are matrix-matrix multiplications.

For equation 1, the relevant Jacobians need to be formed row-by-row as

$$J_{j,k}(\mathbf{x})(i,:) = J_g(P_j V_j M(\mathbf{x})\hat{z}_k) P_j V_j \frac{\partial M}{\partial \mathbf{x}_i}(\mathbf{x})\hat{z}_k \tag{8}$$

where

$$J_g([x, y, w]^T) = \begin{bmatrix} \frac{1}{w} & & -\frac{x}{w^2} \\ & \frac{1}{w} & -\frac{y}{w^2} \end{bmatrix} \tag{9}$$

and

$$\frac{\partial M}{\partial \mathbf{x}_i}(\mathbf{x}) = \begin{cases} \begin{bmatrix} 0 & 0 & 0 & \delta_{i,1} \\ 0 & 0 & 0 & \delta_{i,2} \\ 0 & 0 & 0 & \delta_{i,3} \\ 0 & 0 & 0 & \delta_{i,4} \end{bmatrix} & \text{if } i \leq 3 \\ \frac{\partial R(q)}{\partial q_{i-3}} & \text{if } i > 3 \end{cases} \tag{10}$$

The partial derivatives of the quaternion-to-matrix operation $\frac{\partial R(q)}{\partial q_l} \in \mathbb{R}^{3\times 3}$ are matrices that can be computed by differentiating the matrix in equation 27 in the quaternion notes with respect to the relevant component.

Gauss–Newton step

Nominally, the Gauss–Newton step is computed by solving

$$J^T J \Delta \mathbf{x} = -J^T \Delta \mathbf{y} \tag{11}$$

where,

$$\Delta \mathbf{y} = \begin{bmatrix} g(P_1 V_1 M \hat{z}_1) - y_1^1 \\ \vdots \\ g(P_n V_n M \hat{z}_4) - y_n^4 \end{bmatrix} \tag{12}$$

is a long vector. However, instead of actually forming the long vectors and matrices, it is more efficient and convenient to just compute the left and right hand side of the equation in a summation loop as

$$A(\mathbf{x}) = J^T J = \sum_j \sum_k J_{j,k}^T(\mathbf{x}) J_{j,k}(\mathbf{x}) \tag{13}$$

and

$$b(\mathbf{x}) = -J^T \Delta \mathbf{y} = -\sum_j \sum_k J_{j,k}^T(\mathbf{x})(g(P_j V_j M(\mathbf{x}) \hat{z}_k) - y_k^j). \tag{14}$$

Then the increment $\Delta \mathbf{x} \in \mathbb{R}^7$ is computed by solving the linear system

$$A \Delta \mathbf{x} = b, \tag{15}$$

that is (formally)

$$\Delta \mathbf{x} = A^{-1} b. \tag{16}$$

The linear system is so small (7x7) and presumably stable that it may not matter much if the matrix A is actually inverted or if the system is solved for just one vector, which is usually a better idea from the computational point of view.

If there are any problems with stability, i.e., if A is singular or ill-conditioned in some cases, try the Levenberg–Marquardt algorithm: replace A with $\tilde{A} = A + \lambda I$, where I is the 7x7 identity matrix and λ is a small positive number of your choice (try, e.g., $\lambda = \frac{1}{1000}$).

Normalization

After each Gauss–Newton step, the quaternion q must be *normalized* $q \mapsto q/|q|$ to ensure the matrix $M(t, q)$ still represents a rigid transformation SE(3). This is needed since this property is not preserved by the Gauss–Newton steps. Mathematically, the normalization is a bit of a hack, but should work fine in practice.

Stopping criterion

Stop when $\|\Delta \mathbf{x}\| < \epsilon$ or alternatively when $\|\Delta \mathbf{y}\| < \epsilon'$. Try different values for the threshold. The units of $\Delta \mathbf{y}$ are pixels so it may be easier to interpret: further tweaking clearly does not matter anymore when, e.g., $\|\Delta \mathbf{y}\| < 0.01$ pixels.