Rolling Shutter and Radial Distortion are Features for High Frame Rate Multi-camera Tracking: Supplementary Material

Akash Bapat, True Price, Jan-Michael Frahm
Department of Computer Science, The University of North Carolina at Chapel Hill
{akash, jtprice, jmf}@cs.unc.edu

1. Derivation for additional constraints

The paper has a brief derivation for the linear constraints in Section 3.1.2. Here, we derive the additional constraints that are obtained from the cameras in the cluster with non-identity pose ${}^{n}\mathbf{T}_{cl}$ (Section 3.1.2.). Eqn.(4) from the paper is

$$X(n,t_2) = {}^{n}\mathbf{T}_{cl}\;\delta\,\mathbf{M}_{cl}\;{}^{cl}\mathbf{T}_{n}\;X(n,t_1). \tag{1}$$

In the paper, we show how with ${}^{n}\mathbf{T}_{cl}$ being the Identity pose, we obtain multiple linear constraints from a single row of a rolling shutter camera (Eqn.(7)). For non-identity pose ${}^{n}\mathbf{T}_{cl}$, 3D point $X(n,t_1)$ can be expressed in cluster space as:

$$X_{cl}(n,t_1) = {}^{cl}\mathbf{T}_{n}\;X(n,t_1) = [x\;y\;z\;1]^T \tag{2}$$

If we expand the matrix multiplication of $\delta\,\mathbf{M}_{cl}X_{cl}$ in Eqn.(1) using Eqn.(2), such that the unknowns are in a column vector, we can rewrite Eqn.(1) as follows:

$$\begin{aligned}\delta\mathbf{M}_{cl}X_{cl}(n,t_1) &= \begin{bmatrix}-1 & 0 & 0 & 0 & -z & y\\ 0 & -1 & 0 & z & 0 & -x\\ 0 & 0 & -1 & -y & x & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}t_x\\ t_y\\ t_z\\ \theta_x\\ \theta_y\\ \theta_z\end{bmatrix} + X_{cl}(n,t_1)\\ &= \qquad\qquad \mathbf{Q}_n \qquad\qquad\quad Y \;+ X_{cl}(n,t_1).\end{aligned}$$

Considering just the first row ρ_1 of Eqn.(1), we can write a constraint for any camera n as follows:

$$X(n,t_2)(1) - {}^{n}\mathbf{T}_{cl}.\rho_1 X_{cl}(n,t_1) = {}^{n}\mathbf{T}_{cl}.\rho_1\mathbf{Q}_n Y \tag{3}$$

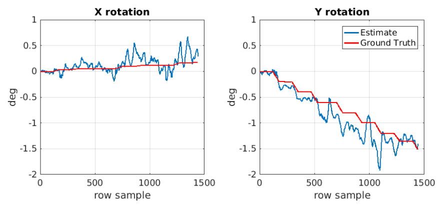
Note that $X(n,t_2)(1) - {}^{n}\mathbf{T}_{cl}.\rho_1 X_{cl}(n,t_1)$ is a single scalar and is an element of column vector B from the system of linear equations $\mathbf{C}\;\mathbf{A}\;Y = \mathbf{C}\;B$ described in Section 3.1. Similarly, ${}^{n}\mathbf{T}_{cl}.\rho_1\mathbf{Q}_n$ forms a row of matrix $\mathbf{A}$. Hence, multiple points observed in the same camera will share ${}^{n}\mathbf{T}_{cl}.\rho_1$ but will have different $\mathbf{Q}_n$, providing us with different constraints for each point.

2. Additional Results

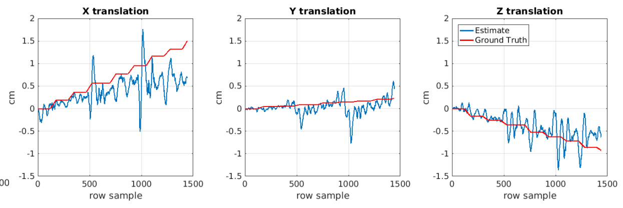
We provide the motion plots for the synthetic data experiments described in Section 4.1. Fig. 1 shows the motion plot for our 6-DoF pose estimates against Hi-Ball ground truth. As the 4-camera cluster has fewer of redundant constraints, it exhibits higher noise sensitivity as compared to the 6-camera case; see Fig. 2. The rendering pixel errors for these motion plots are in Fig.(4) of the paper.

3. Rendering pixel error visualization

Minimizing rendering pixel error is important in the augmented- and virtual-reality (AR/VR) applications. We provide a visualization of the rendering pixel error in the form of a video. In the associated video, we created a synthetic motion

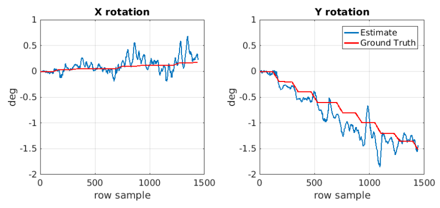


(a)

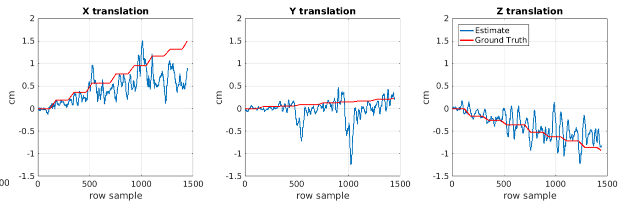


(b)

Figure 1: Tracking estimates of our 4-camera configuration using synthetic imagery and Hi-Ball tracking data for ground truth: (a) Rotation estimates in degrees and (b) translation estimates in cm.



(a)



(b)

Figure 2: Tracking estimates of our 6-camera configuration using synthetic imagery and Hi-Ball tracking data for ground truth: (a) Rotation estimates in degrees and (b) translation estimates in cm.

sequence with only rotation in Z direction. The red boxes in the video are the ground truth motion, the blue box has pose according to our tracking, and the green box has no motion. The rendering pixel error for the motion sequence in the video is 10.88px while tracking 17.34° of rotation in the Z direction. The video shows the visualization for rendering pixel error at 30x speed. We can see that our tracking estimates follow the Z rotation well but drifts and oscillates in the Y-translation.

4. Failure cases and degeneracies

Our approach depends upon densely matching the row-image pixels, and the system fails when there is a complete absence of texture or small texture gradient. In practice, however, this is remedied by using cameras looking in different directions. Degeneracies arise if the cluster consists of all cameras looking in the same direction with the same orientation, which makes the constraints linearly dependent. Our cluster design ensures that at least one row can sense motion in each direction. It uses pairs of cameras looking in orthogonal directions to give constraints in X, Y, and Z (see Sec. 3.3). The cameras in the pairs themselves are at 90° rotations, giving constraints in the local X and Y directions.

5. Implementation and pseudocode

We provide pseudo-code for robust filtering of Sec. 3.2.3 (Alg. 1) here for easier understanding of our robust shift filtering approach. In the interest of reproducibility, we will also make our source code available.

Algorithm 1 Robust shift smoothing

```
1: function RobustFilter(Input : raw value $m_t$)
2:     Compute one-step forecast $m_{t|t-1} = \tilde{m}_{t-1} + B_t$
3:     Estimate $\tau^2$-scale estimate $\hat{\sigma}_t$
4:     Refine $m_{t|t-1}$ to obtain $m_t^*$                        ▷ Eq. (9)
5:     Apply Holt-Winters smoothing on $m_t^*$ to obtain $\tilde{m}_t$   ▷ Eq. (8)
6:     return Filtered output $\tilde{m}_t$
7: end function
```