Estimating Physical Camera Parameters based on Multi-Sprite Motion Estimation

Dirk Farin

Peter H. N. de With

Contact address:

Dirk Farin

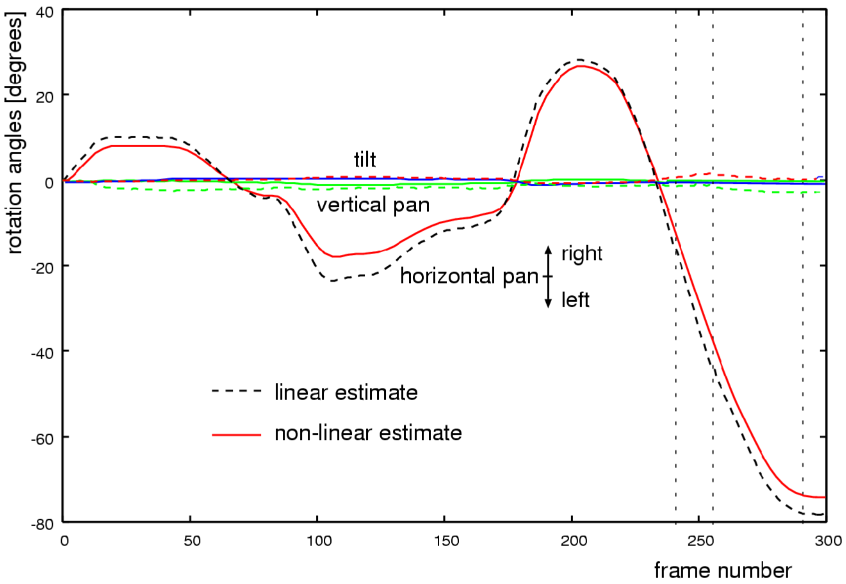
Eindhoven University of Technology (TU/e)

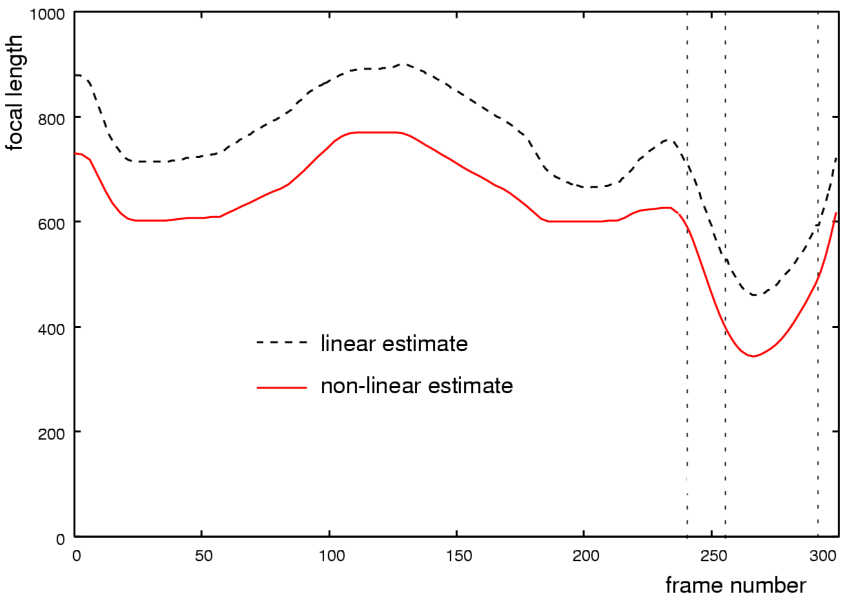
Design Technology Institute

5600 MB, Eindhoven, The Netherlands

d.s.farin@tue.nl

What are we going to do?





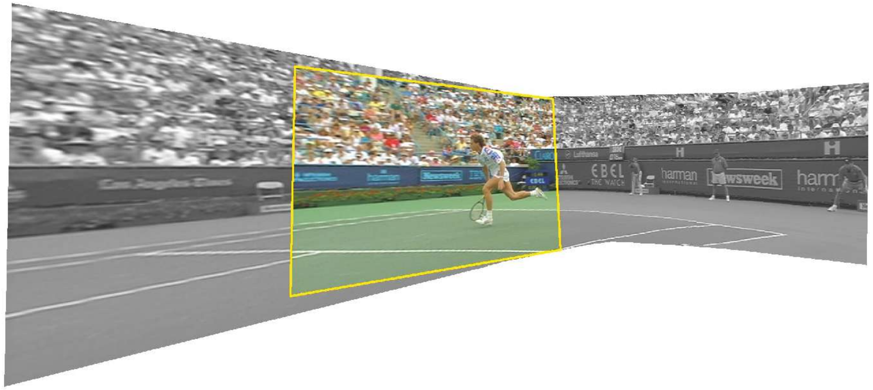
Introduction 1/2

- We consider the important special case of rotational camera motion.

- Model of rotational camera motion is employed in
  - video coding standards like MPEG-4 (GMC, background sprite coding),
  - video content analysis (MPEG-7 descriptor),

- Rotational camera motion is usually described with an 8-parameter model

$$x = \frac{h_{00}\hat{x} + h_{01}\hat{y} + h_{02}}{h_{20}\hat{x} + h_{21}\hat{y} + 1}, \quad y = \frac{h_{10}\hat{x} + h_{11}\hat{y} + h_{12}}{h_{20}\hat{x} + h_{21}\hat{y} + 1}$$



- However: no physical meaning can be assigned to these parameters.

Introduction 2/2

- Often, camera motion is required in physically meaningful parameters:
  - rotation angles,
  - focal-length (camera zoom).

- Applications
  - augmented reality (mixing natural video with synthetic 3-D objects),
  - video content analysis,
  - generation of video panoramas (requires focal-length).

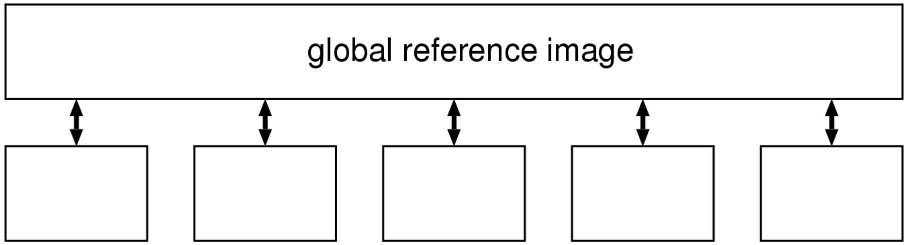
- Our goal: factorize perspective motion parameters into physical parameters
  - three rotation angles
  - focal length

$$\boxed{x = \frac{h_{00}\hat{x} + h_{01}\hat{y} + h_{02}}{h_{20}\hat{x} + h_{21}\hat{y} + 1}, \quad y = \frac{h_{10}\hat{x} + h_{11}\hat{y} + h_{12}}{h_{20}\hat{x} + h_{21}\hat{y} + 1}}$$

Global Motion Estimation

- Camera motion can be computed either
  - between successive frames (short-term prediction), or

  - relative to a global reference (long-term prediction).



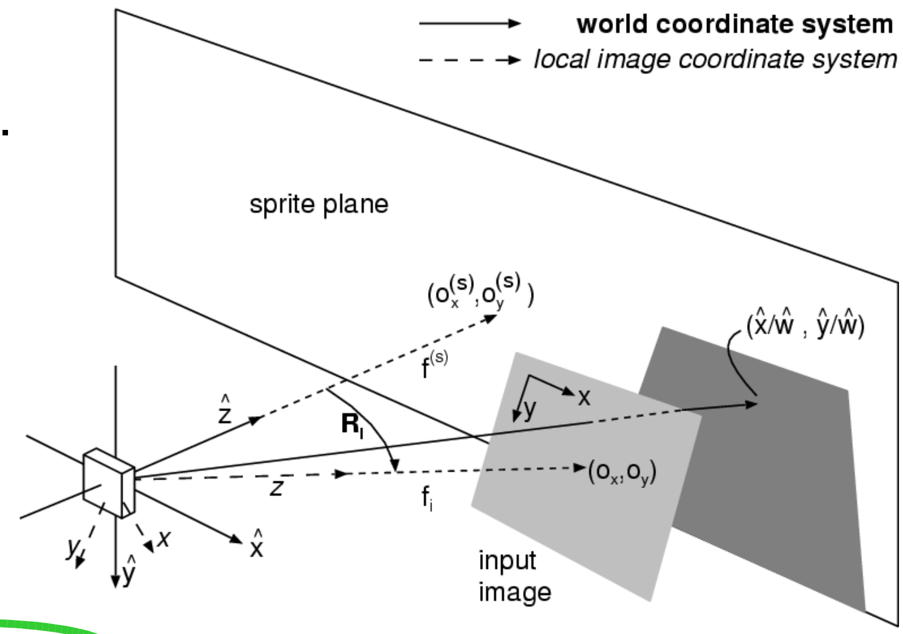
- For camera calibration, we need motion parameters for any pair of views.
  - Chaining of transforms between successive frames leads to error accumulation.
  - To prevent error accumulation, register frames to a common reference frame.

Geometry of Image Acquisition

- Internal camera parameter matrix **K** projects 3-D points onto image plane.

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \underbrace{\begin{bmatrix} f_i & 0 & o_x \\ 0 & f_i & o_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}_i} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

  - Focal length f_i
  - Principal point (o_x, o_y)



- Transformation from sprite to input image.

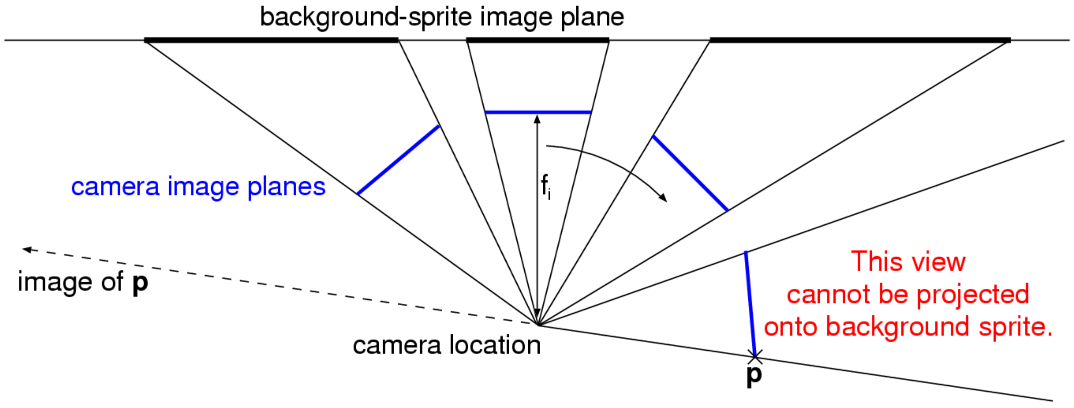
$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \mathbf{K}_i \mathbf{R}_i \mathbf{K}_{(s)}^{-1} \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{w} \end{pmatrix} = \mathbf{H}_i \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{w} \end{pmatrix}$$

- Multiplying matrices and converting to inhomogeneous formulation gives

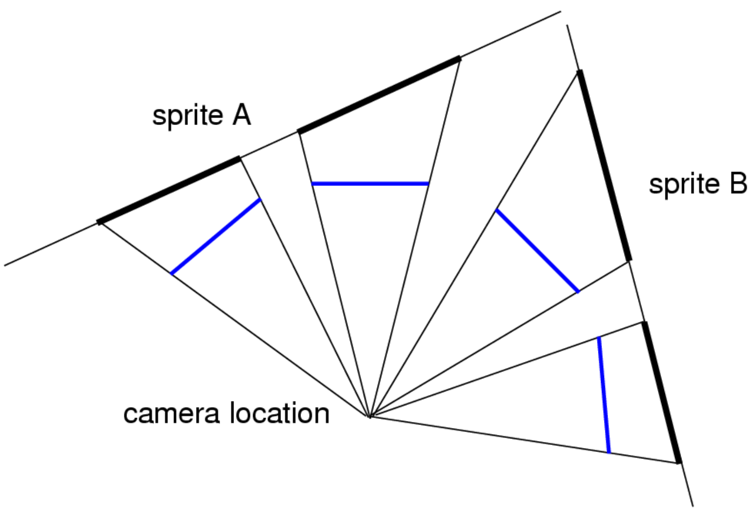
$$x = \frac{h_{00}\hat{x} + h_{01}\hat{y} + h_{02}}{h_{20}\hat{x} + h_{21}\hat{y} + 1}, \quad y = \frac{h_{10}\hat{x} + h_{11}\hat{y} + h_{12}}{h_{20}\hat{x} + h_{21}\hat{y} + 1}$$

Multi-Sprite Motion Estimation

- Perspective motion model does not work for large rotation angles.
  [Farin, VCIP 2004]



- Solution is to partition sequence into several sprites and perform global motion-estimation separately.
  (Multi-Sprite motion estimation)



Overview of our Camera Calibration Algorithm

- Input:
  - Perspective motion parameters (as obtained, e.g., by MPEG-4 sprite encoder)
- Output:
  - The equivalent motion, parameterized in physical parameters
    (rotation around elementary axes, and camera focal-length).
- Calibration is carried out in two steps
  - Step 1: fast camera calibration with a linear algorithm (Hartley 1997)
  - Step 2: refinement of camera parameters with a non-linear optimization.
- Both algorithms have been extended to multi-sprite motion estimation.
  - Main advantage: unlimited camera motion.

Linear Camera Calibration 1/3 [Hartley, 97]

- The algorithm examines images of the **absolute conic**, which is defined as

$$(x, y, z)\mathbf{I}(x, y, z)^\top = 0 \quad \text{and} \quad w = 0$$

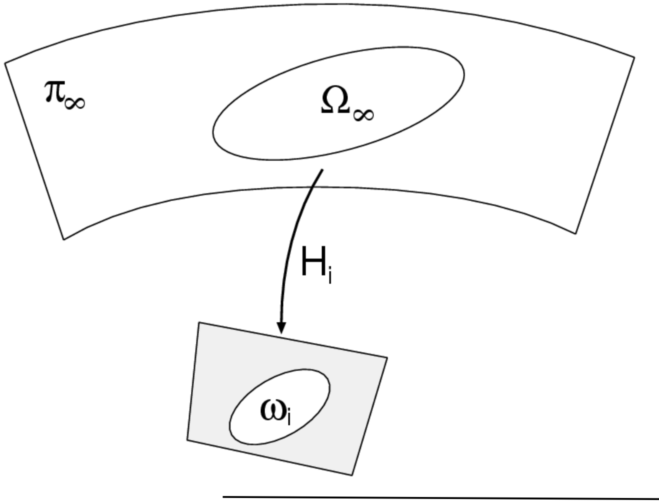
identity matrix

- With a camera transformation $\mathbf{H_i}$=$\mathbf{K_i R_i}$ for a view *i*,

  the *Image of the Absolute Conic (IAC)* $\omega^{(i)}$ is obtained as

$$\omega^{(i)} = \mathbf{H_i}^{-\top}\mathbf{I}\mathbf{H_i}^{-1} = \mathbf{K_i}^{-\top}\mathbf{R_i}^{-\top}\mathbf{R_i}^{-1}\mathbf{K_i}^{-1} = \mathbf{K_i}^{-\top}\mathbf{K_i}^{-1}$$

- Notice that the IAC
  - depends only on internal camera parameters,
  - is invariant to the camera rotations.

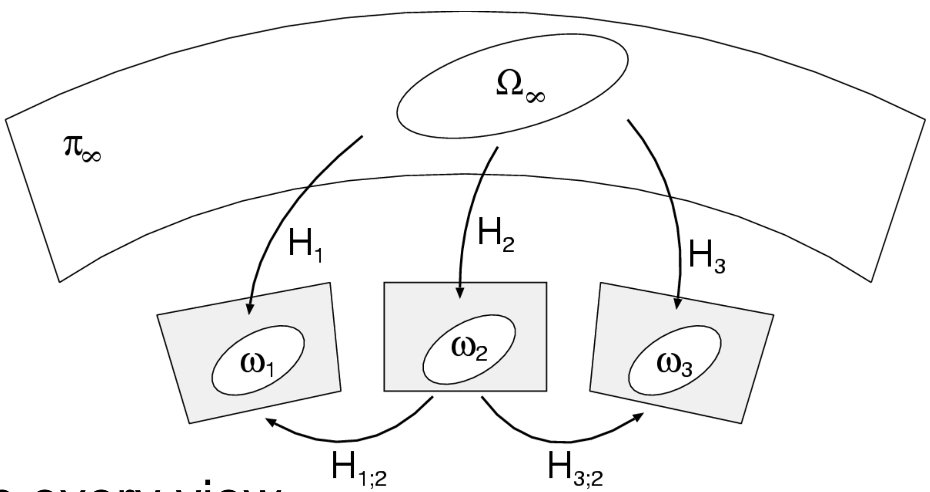


Linear Camera Calibration 2/3

- From $\omega^{(i)}=\mathbf{K}_i^{-T}\mathbf{K}_i^{-1}$ for a view, we obtain the IAC of view *i* as

$$\omega^{(i)} = \begin{bmatrix} 1/f_i^2 & 0 & -o_x/f_i^2 \\ 0 & 1/f_i^2 & -o_y/f_i^2 \\ -o_x/f_i^2 & -o_y/f_i^2 & o_x^2/f_i^2 + o_y^2/f_i^2 + 1 \end{bmatrix}$$

- We can set two constraints:
  - $\omega_{00}=\omega_{11}$ (square pixels)
  - $\omega_{10}=\omega_{01}=0$ (no image skew)



- These constraints can be imposed in every view.
- Using the transformation between views $\mathbf{H}_{i;r}$, constraints from one view can be mapped onto another:

$$\omega^{(i)} = \mathbf{H}_{i;r}^{-\top}\omega^{(r)}\mathbf{H}_{i;r}^{-1}$$

Linear Camera Calibration 3/3

- Transform the constraints from all views into a common view *i*.
- Stack all constraints into an equation system.

$$\begin{pmatrix} \text{constraints from view 1} \\ \text{constraints from view 2} \\ \vdots \\ \text{constraints from view N} \end{pmatrix} \begin{pmatrix} \omega_{00} \\ \omega_{01} \\ \omega_{02} \\ \omega_{11} \\ \omega_{12} \\ \omega_{22} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

- Solve with least-squares.
- Since $\omega^{(i)} = \mathbf{K}_i^{-T} \mathbf{K}_i^{-1}$, we get **K** using a Cholesky decomposition (factorization into triangular matrices).

- Once the internal parameters $\mathbf{K}_i$ are known, we obtain the rotation between views from

$$\mathbf{R} = \mathbf{K}_k^{-1} \mathbf{H}_k \mathbf{H}_i^{-1} \mathbf{K}_i$$

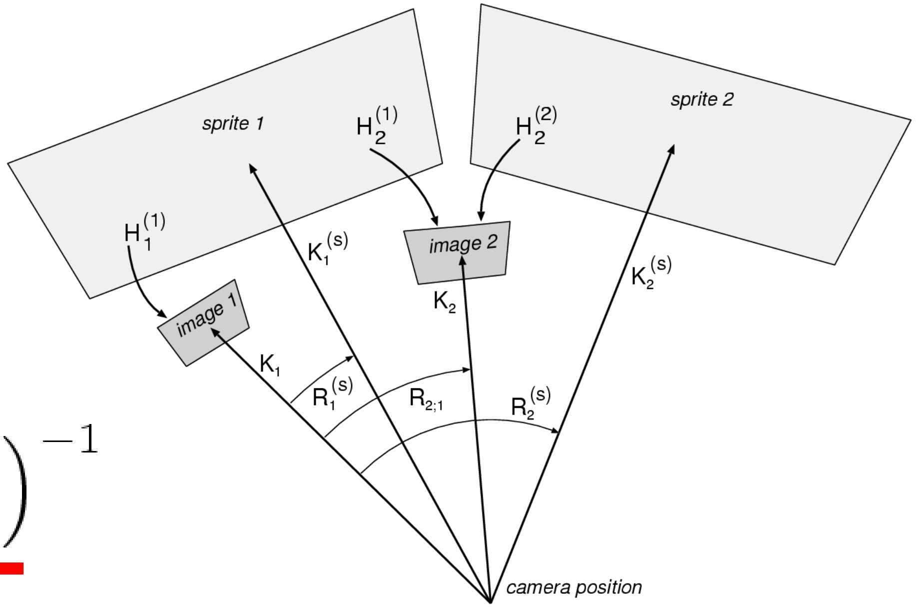
Extension to Multi-Sprite Motion Estimation

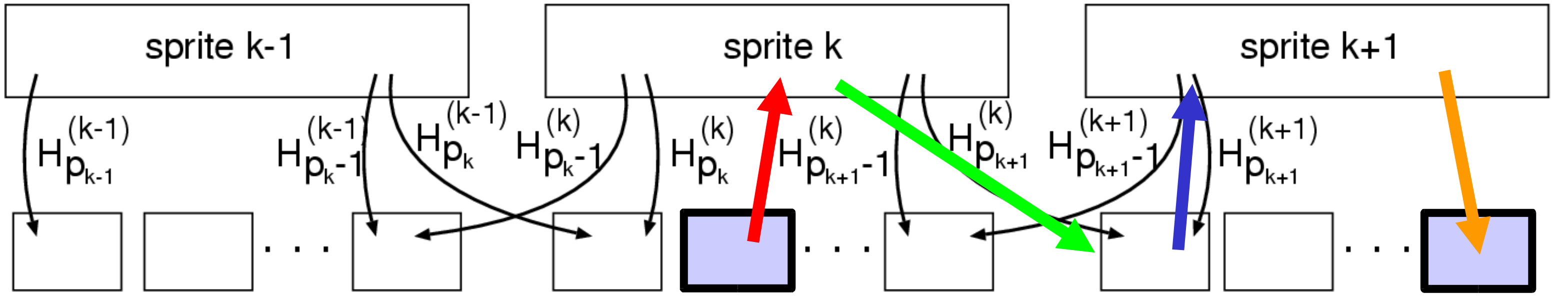
- Problem: we cannot transform constraints if they depend on different reference frames.
- Solution: Use linking transform to concatenate sprites. Transforms between frames are obtained by chaining.

- To find transformation
  - from view i_1 in sprite k
  - to view i_2 in sprite k+1,

  we compute:

$$\underline{\mathbf{H}_{i_2}^{(k+1)}} \underline{\left(\mathbf{H}_{p_{k+1}}^{(k+1)}\right)^{-1}} \underline{\mathbf{H}_{p_{k+1}}^{(k)}} \underline{\left(\mathbf{H}_{i_1}^{(k)}\right)^{-1}}$$





Non-linear Camera Calibration 1/3

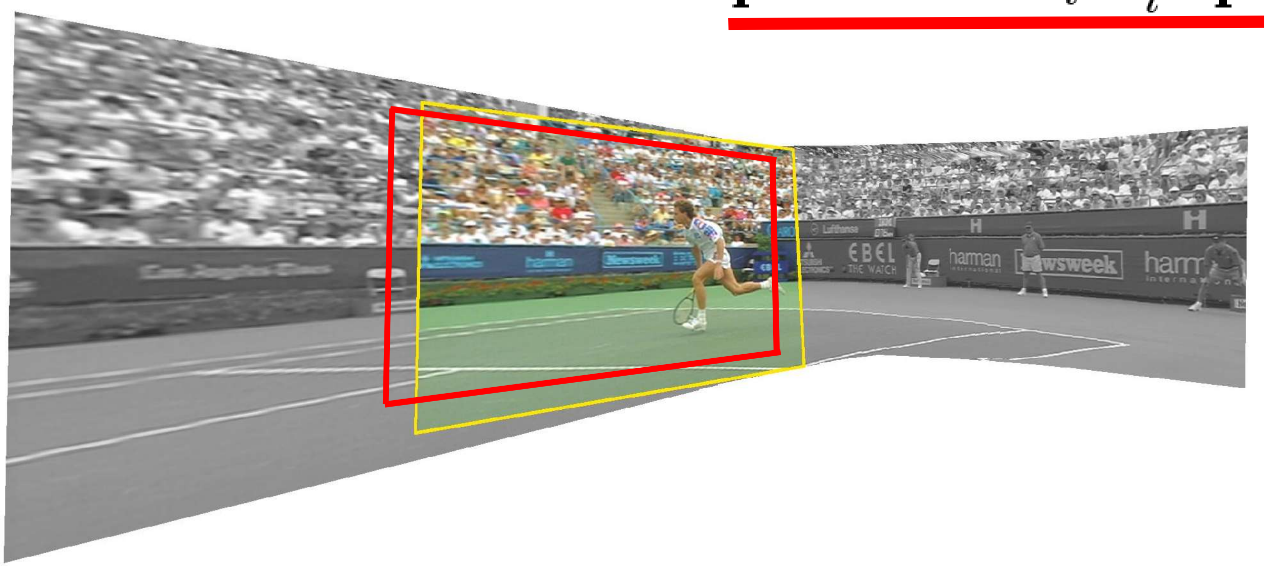
- Previous algorithm: minimize algebraic error
- Better approach: minimize reprojection error

- Define reprojection error as distance of image corners **p**, between
  - their position on the sprite, as obtained with the perspective motion model, and

$$\hat{\mathbf{p}} = \mathbf{H}_i^{-1}\mathbf{p}$$

  - their position as obtained with the physical camera parameters model.

$$\mathbf{p}' = \mathbf{K}^{(s)}\mathbf{R}_i\mathbf{K}_i^{-1}\mathbf{p}$$



Non-linear Camera Calibration 2/3

- Apply an iterative optimization, incorporating all frames at once.

$$\min_{\mathbf{K}_i,\mathbf{R}_i} \sum_k d(\hat{\mathbf{p}}_k, \mathbf{p}'_k) = \sum_k d(\underline{\mathbf{H}_i^{-1}\mathbf{p}_k} \; , \; \underline{\mathbf{K}^{(s)}\mathbf{R}_i\mathbf{K}_i^{-1}\,\mathbf{p}_k})$$

- Non-linear optimization enables to incorporate all known constraints
  - **R is a rotation matrix**,
  - **Principal point is fixed**, no skew, square pixels

- How to parameterize rotation (we have three rotation axes)
  - Rotation matrix (9p) – hard to enforce orthonormality constraint
  - Euler angles (3p) – numerical instabilities near poles
  - Quaternions (4p) – num. stable and easy to enforce constraint (unit norm)

$$\mathbf{R} = \begin{bmatrix} 1-2q_y^2-2q_z^2 & 2q_xq_y-2q_wq_z & 2q_xq_z+2q_wq_y \\ 2q_xq_y+2q_wq_z & 1-2q_x^2-2q_z^2 & 2q_yq_z-2q_wq_x \\ 2q_xq_z-2q_wq_y & 2q_yq_z+2q_wq_x & 1-2q_x^2-2q_y^2 \end{bmatrix}, \quad \text{where} \quad ||\mathbf{q}|| = 1$$

Non-linear Camera Calibration 3/3

- Optimize over camera parameters

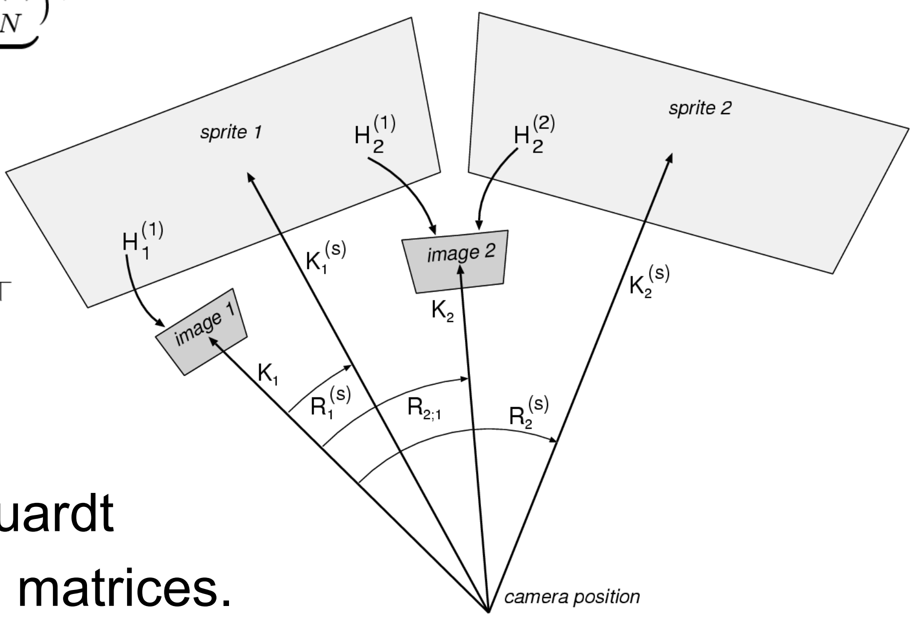
$$\mathbf{x} = (\underbrace{f_1,\ \mathbf{q}_2, f_2,\ \ldots,\ \mathbf{q}_N, f_N,\ o_x, o_y}_{\text{image view parameters}}\ ,\ \underbrace{\mathbf{q}_1^{(s)}, f_1^{(s)},\ o_{x;1}^{(s)}, o_{y;1}^{(s)},\ \ldots,\ \mathbf{q}_M^{(s)}, f_M^{(s)},\ o_{x;M}^{(s)}, o_{y;M}^{(s)}}_{\text{sprite view parameters}})^\top$$

- such that distance between the image corner positions ||y-y|| is minimal
  - corner positions using physical parameters

$$\mathbf{y} = (\underbrace{\underline{\mathbf{p}}'^{(1)}_1, \underline{\mathbf{p}}'^{(2)}_1, \underline{\mathbf{p}}'^{(3)}_1, \underline{\mathbf{p}}'^{(4)}_1}_{\text{first view}}, \ldots, \underbrace{\underline{\mathbf{p}}'^{(1)}_N, \underline{\mathbf{p}}'^{(2)}_N, \underline{\mathbf{p}}'^{(3)}_N, \underline{\mathbf{p}}'^{(4)}_N}_{\text{view } N})^\top$$

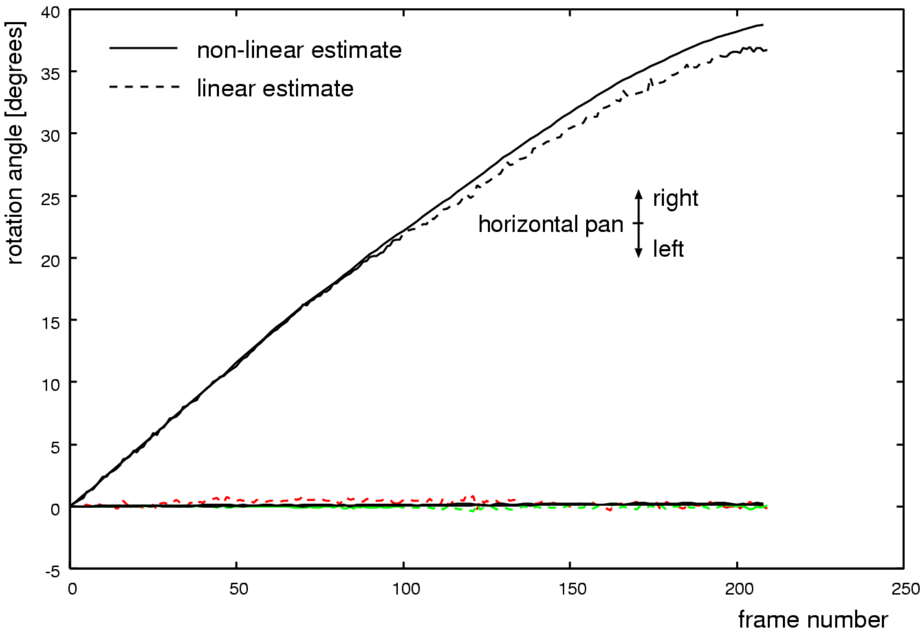
  - corner positions using homography **H**$_{\mathbf{i}}$

$$\hat{\mathbf{y}} = (\underbrace{\underline{\hat{\mathbf{p}}}^{(1)}_1, \underline{\hat{\mathbf{p}}}^{(2)}_1, \underline{\hat{\mathbf{p}}}^{(3)}_1, \underline{\hat{\mathbf{p}}}^{(4)}_1}_{\text{first view}}, \ldots, \underbrace{\underline{\hat{\mathbf{p}}}^{(1)}_N, \underline{\hat{\mathbf{p}}}^{(2)}_N, \underline{\hat{\mathbf{p}}}^{(3)}_N, \underline{\hat{\mathbf{p}}}^{(4)}_N}_{\text{view } N})^\top$$

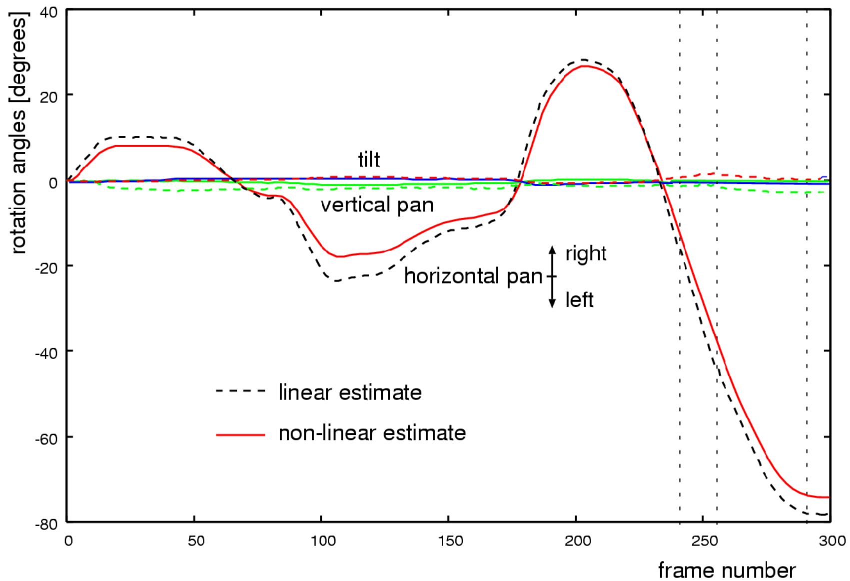


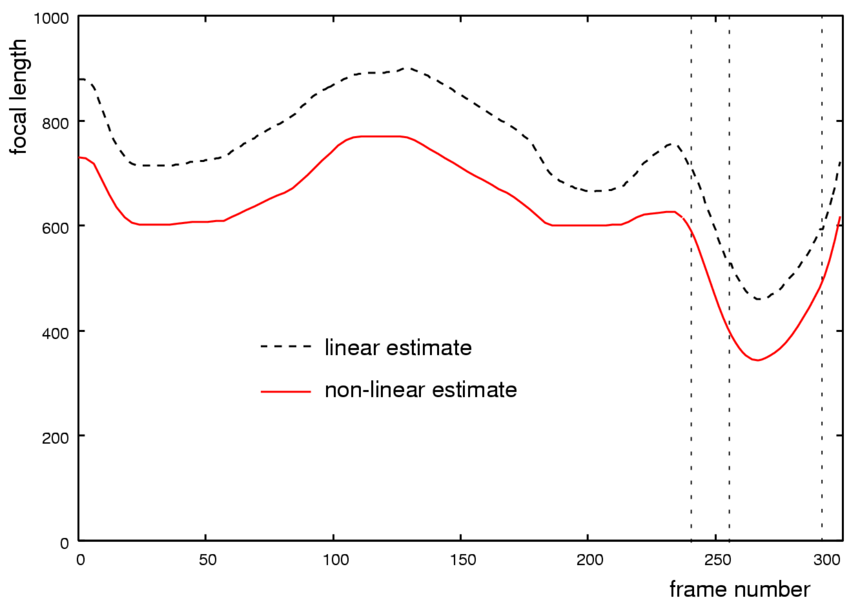
- Optimization using a Levenberg-Marquardt variant, optimized for sparse Jacobian matrices.

Results: horizontal pan



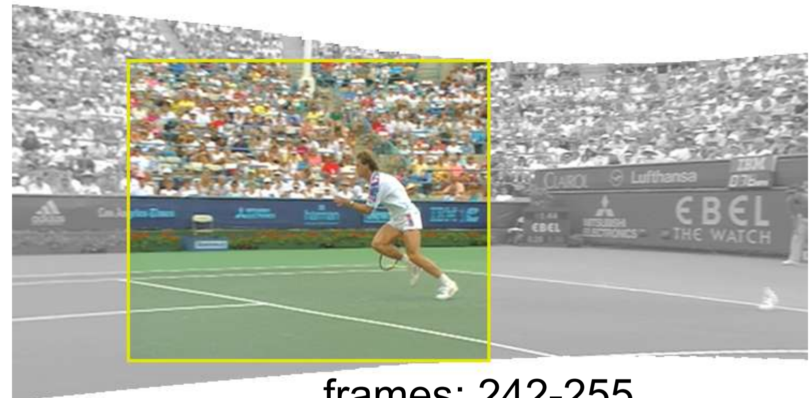
Results: *stephan* sequence



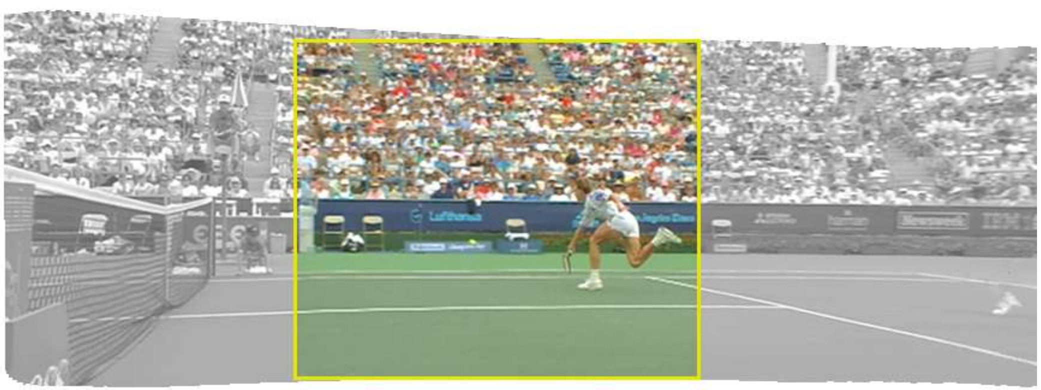


Results: *stefan* sequence   (sprites)

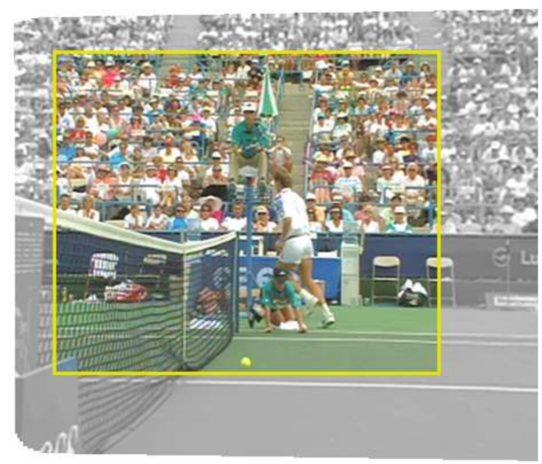
frames: 1-241



frames: 242-255



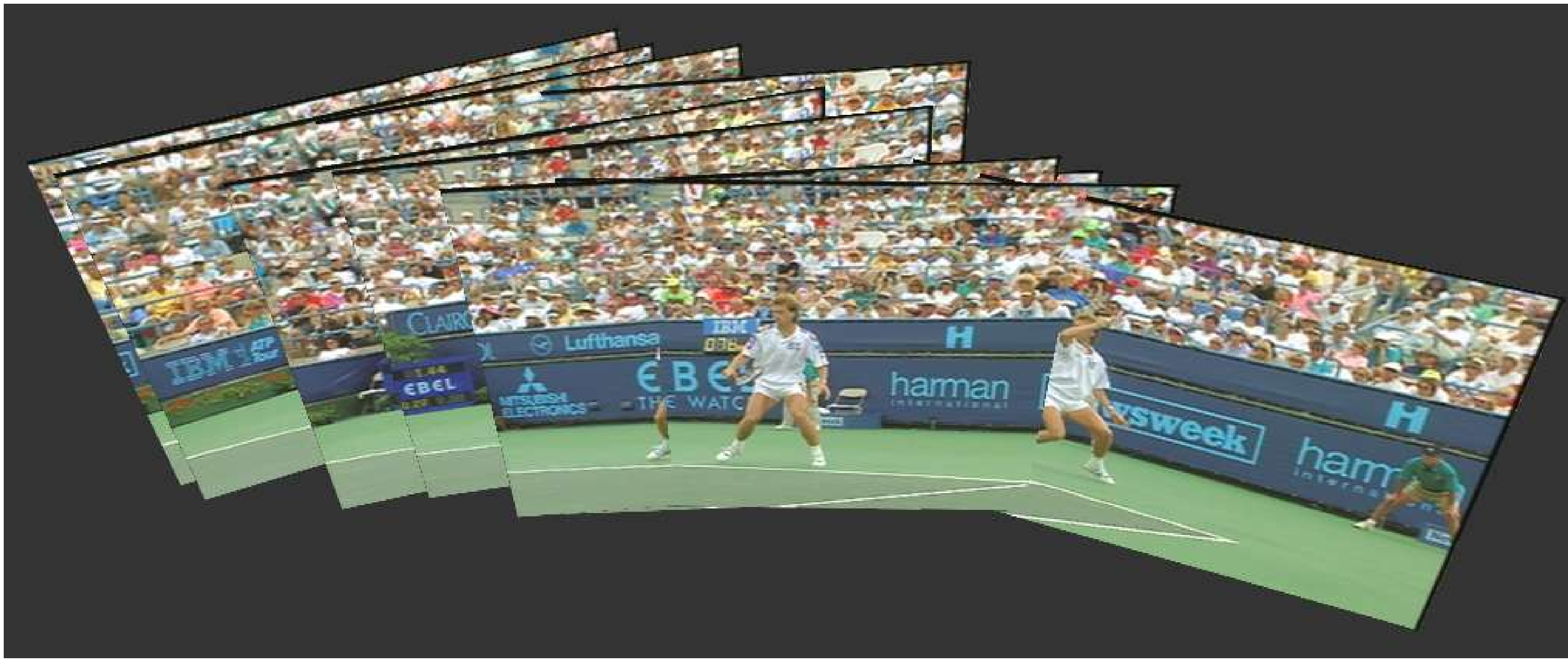
frames: 256-292



frames: 293-300

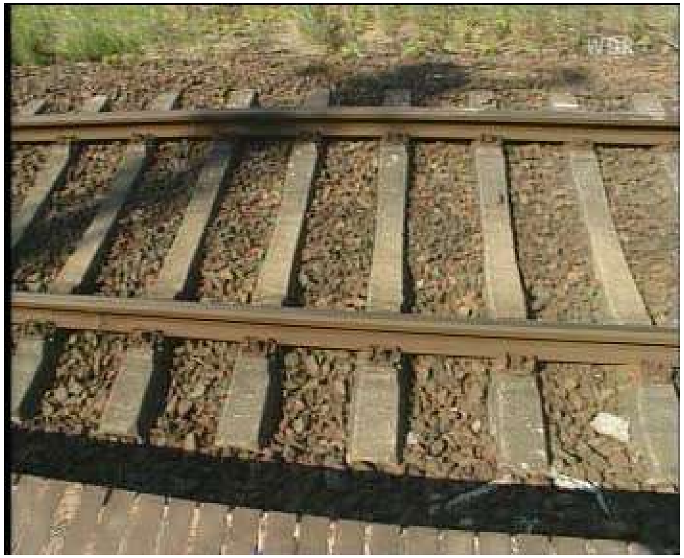
Results: *stephan* sequence

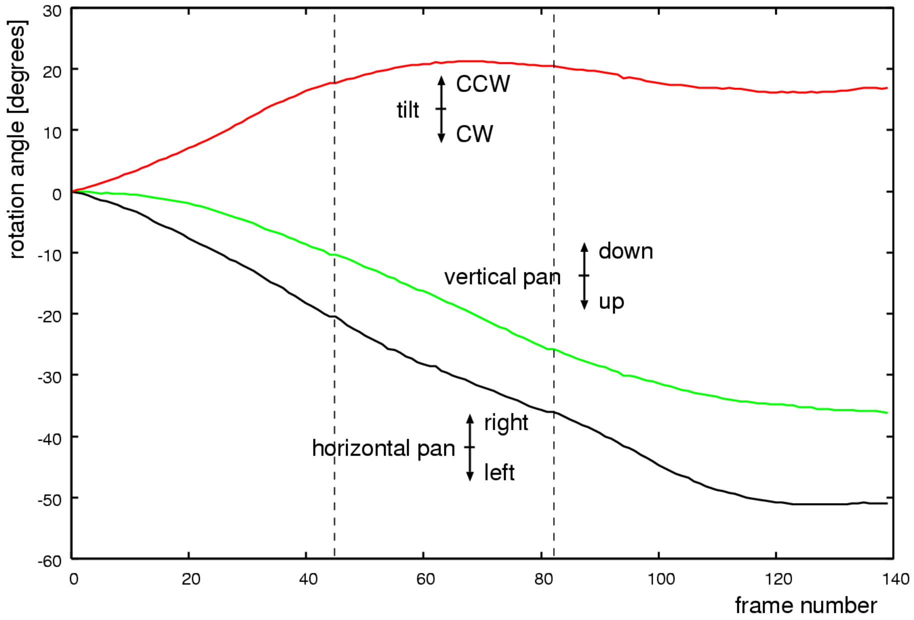
- Captured images are placed at their virtual image plane in 3D-space.

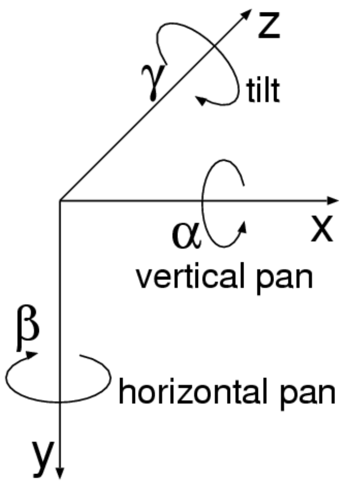


- Interactive OpenGL visualization.
- Note: no distortions observable along intersection lines.
- If viewed from camera position, images are aligned to panoramic view.

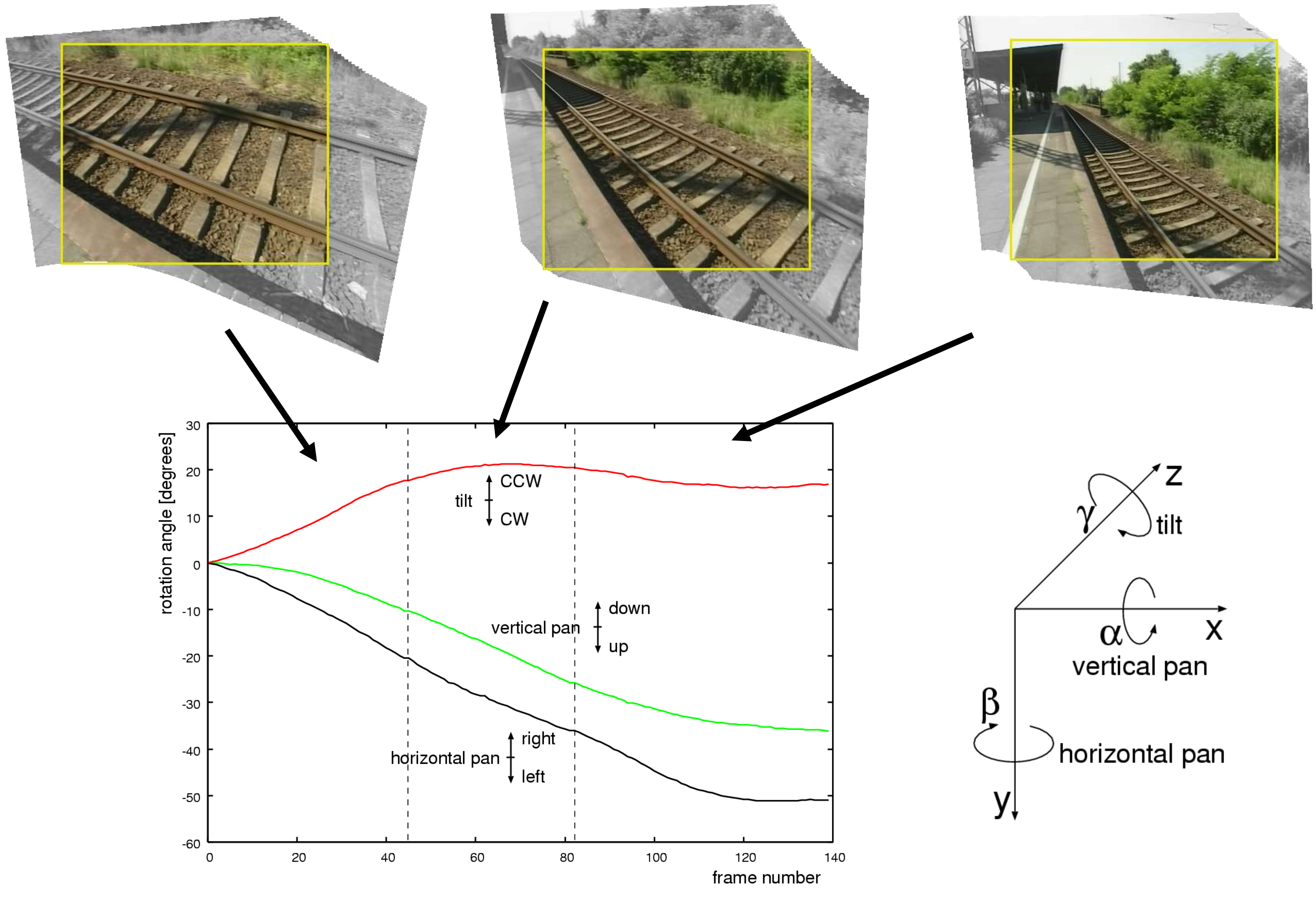
Results: complicated camera motion







Results: complicated camera motion



Conclusions

- We presented a new algorithm to factorize
  global motion parameters to physically meaningful parameters.
- Arbitrary camera motion is supported (due to multi-sprite motion. est.)
- Observations:
  - Linear algorithm (step 1 only) sufficient if qualitative analysis is sufficient.
  - Non-linear refinement is required if exact focal length is important.
- Future Work
  - Evaluation of absolute parameter accuracy.
  - Obtain physical camera parameters in an online process.