Lecture 5.
Dense Reconstruction and Tracking with Real-Time Applications

Part 1: Tracking

Dr Richard Newcombe and Dr Steven Lovegrove

Slide content developed from:

[Newcombe, “Dense Visual SLAM”, 2015] [Lovegrove, “Parametric Dense Parametric SLAM”]

and [Szeliski, Seitz, Zitnick UW CSE576 CV lectures]

Recap

Overview: Parts of the 3D Computer Vision Pipeline

Feature based methods for estimating camera motion and scene geometry

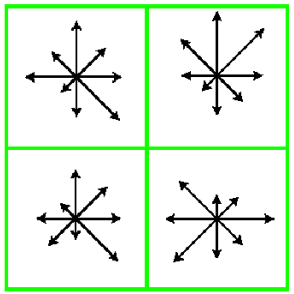
Two Frames: with Computer Vision what can we estimate?

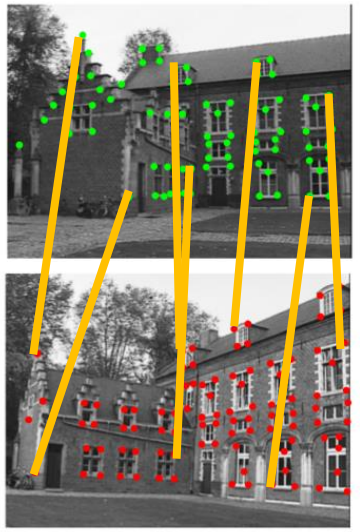
Frame 1:

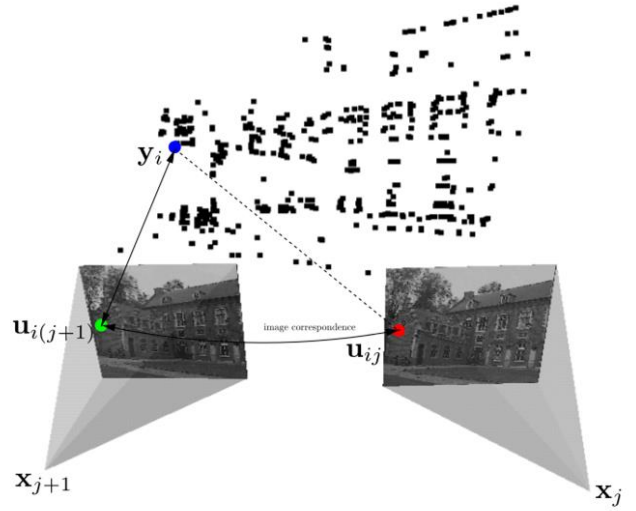
Two Frames: with Computer Vision what can we estimate?

Frame 2:

Recap

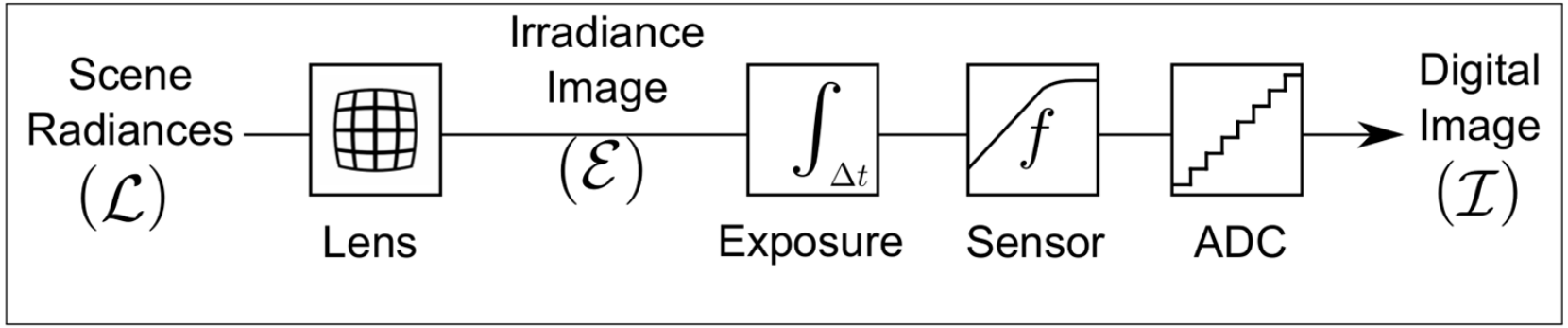






Detect Salient Feature Points	Extract Descriptors	Match Descriptors	Initial Parameter Estimates	Parameter Refinement
Detect Salient Features • Features should be stable under geometric distortion	Extract Descriptors • Descriptors should be invariant to geometric distortion	Match Descriptors • Similarity between descriptors • Rank Matches	Parameter Initialization • Scene Geometry • Camera Motion • …Scene Motion?	Refinement • MLE Optimization • Joint SFM estimation • Bundle Adjustment • SE(N) Representations
Lecture 2	Lecture 2	Lecture 2	Lecture 3	Lecture 4

Refresher on the Image Formation process



- Scene Radiance: Scene geometry, lighting structures and material reflectance
- Geometric and radiometric distortion due to the camera lens,
  - e.g. lens distortion, vignetting.
- Motion blur due to long exposures
- Image noise for short exposures and low lighting
- Non-Linear sensor response and unknown exposure settings, saturation

Example transformations: perils of feature matching

I.e. what if there is image degradation?

Reference Image

Geometric transformation and blur

Geometric, blur and noise

Geometric, motion blur

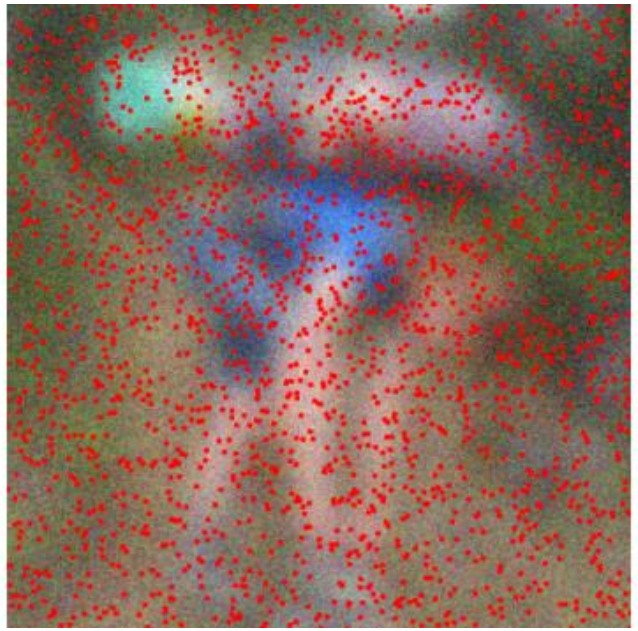
[Newcombe, 2013]

Sparse pipelines need image features

Example FAST detections (Rosten and Drummond, ECCV 2006)

Reference Image

Geometric transformation and blur



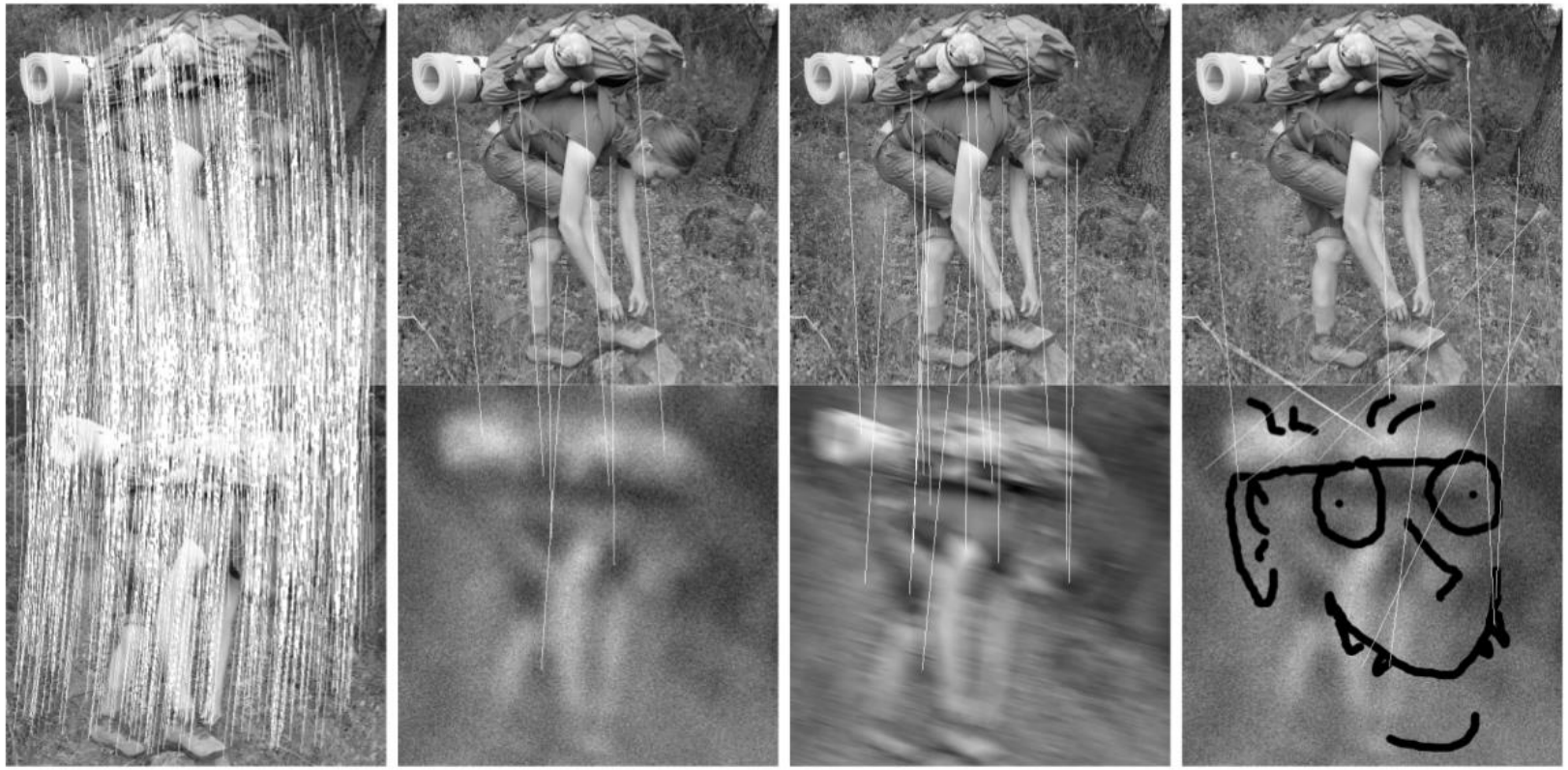
Geometric, blur and noise

Geometric, motion blur

Sparse (1) extraction and (2) matching

Descriptor extraction and matching using naively applied SIFT (Lowe, ICCV 2004)

How can we estimate camera pose or scene reconstruction if this process fails?



Geometric only

Geometric, blur and noise

Geometric, motion blur

Geometric, blur, noise, occlusion

Classes of Techniques

Feature-based methods

- Extract visual features (corners, textured areas) and then Estimate Paramters
- Suitable especially when image motion is large (10s of pixels)

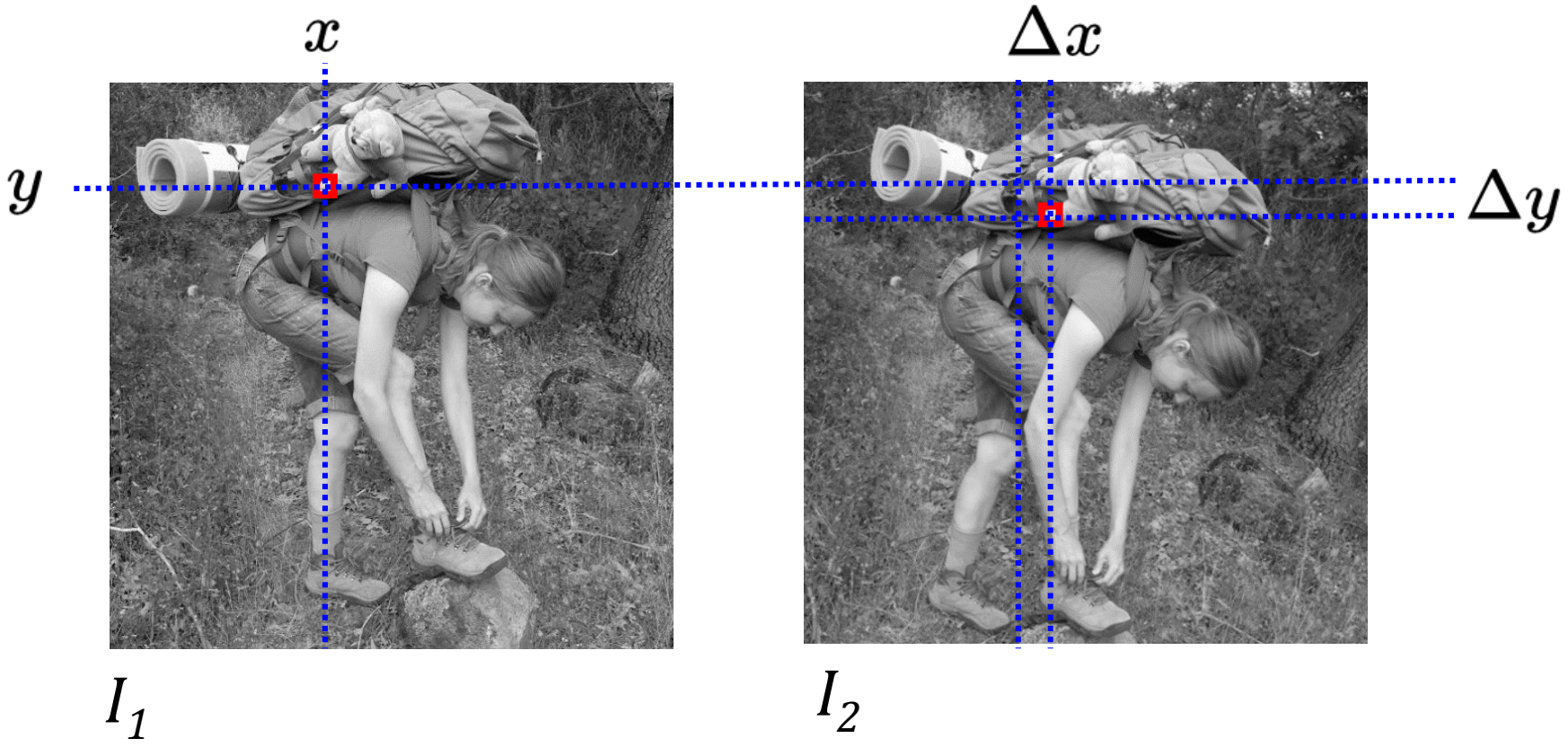
Direct-methods

- Directly recover image motion from spatio-temporal image brightness variations
- Global motion parameters directly recovered without an intermediate feature motion calculation
- Suitable for video and Real-Time applications

Correspondence from Tracking

Assumptions on image appearance and motion

Assumption I: Brightness constancy



$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

Assumption II: Small Motion in the image

Taylor Series expansion of $I(t+\Delta t)$:

$$I(x+\Delta x, y+\Delta y, t+\Delta t) = I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + \text{H.O.T}$$

Truncation of higher order terms:

$$I(x,y,t) \approx I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$$

Optical Flow Constraint

Relation to Image Pixel Velocity *V*:

$$I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t - I(x,y,t) = 0$$

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0$$

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0$$

O.F. Constraint

$$I_x V_x + I_y V_y = -I_t$$

$$\nabla I^T \cdot \vec{V} = -I_t$$

Optical Flow – Instantaneous Correspondence

2 unknowns per scalar image pixel:

$$\nabla I^T \cdot \vec{V} = -I_t$$

How to solve for V, either ?

- Assume local pixels in a patch have the same motion

Assumption III: Neighboring Pixels have the same motions

Assume a single velocity for all pixels within an image patch

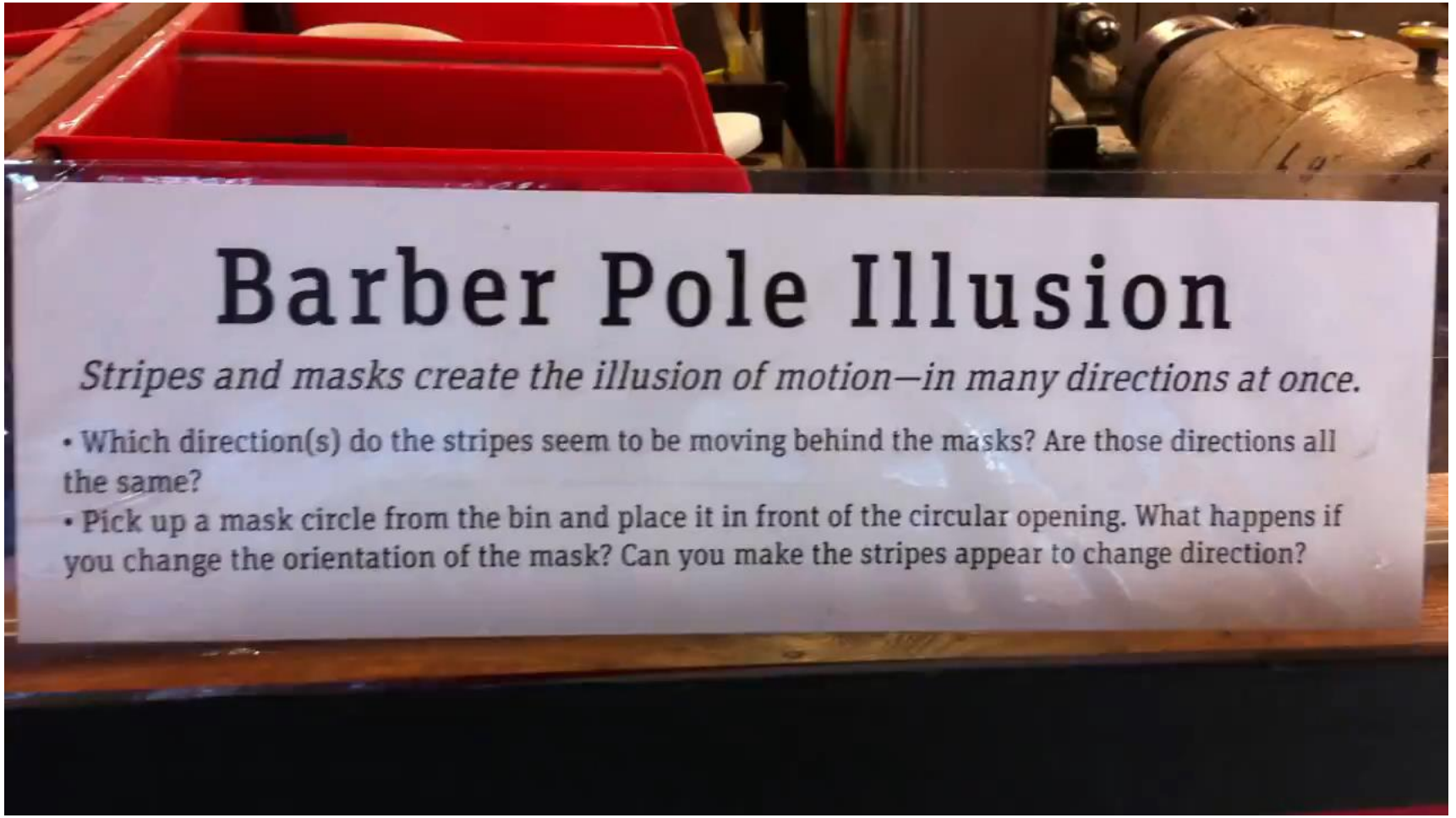
$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \qquad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \qquad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

Solve the *Normal Equations* $A^{\mathrm{T}}A\ v = A^{\mathrm{T}}b$:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i)I_y(q_i) \\ \sum_i I_y(q_i)I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i)I_t(q_i) \\ -\sum_i I_y(q_i)I_t(q_i) \end{bmatrix}$$

Lucas-Kanade Patch Tracking (1981) utilized these three assumptions.
https://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade_method

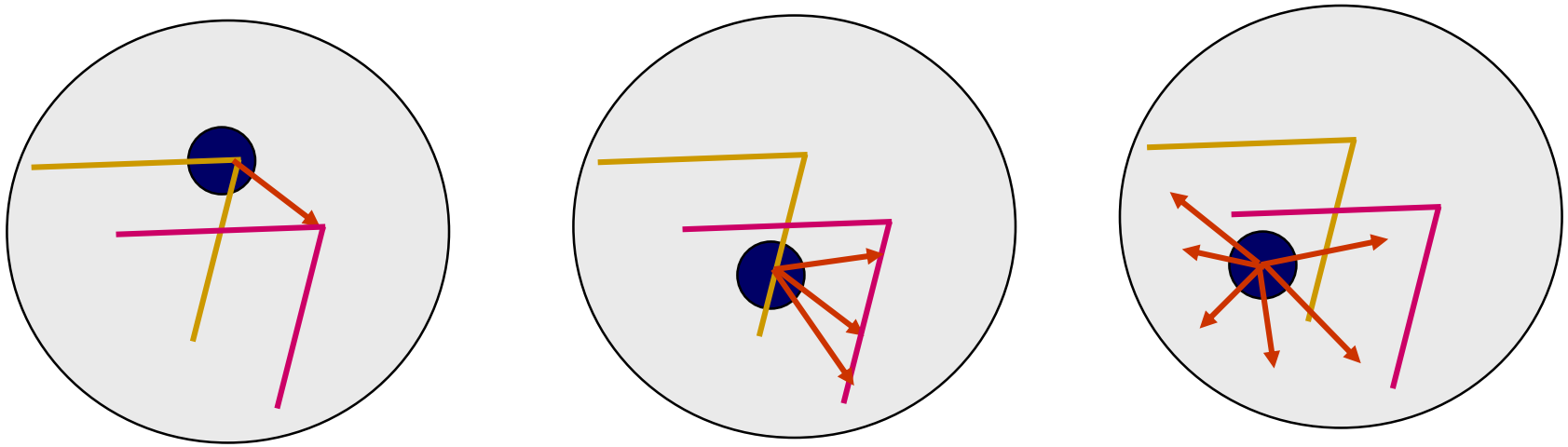
Is the solution always unique?



Nature Clips
Published on 9 Dec 2016 https://www.youtube.com/watch?v=QkmSFWQzFA8

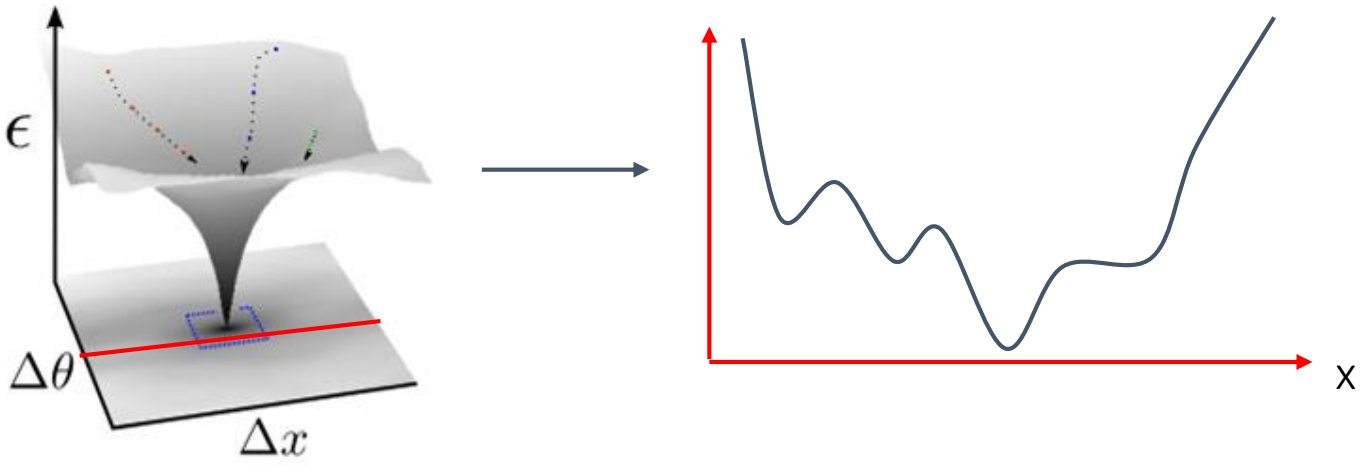
Local Patch Analysis

- How *certain* are the motion estimates?



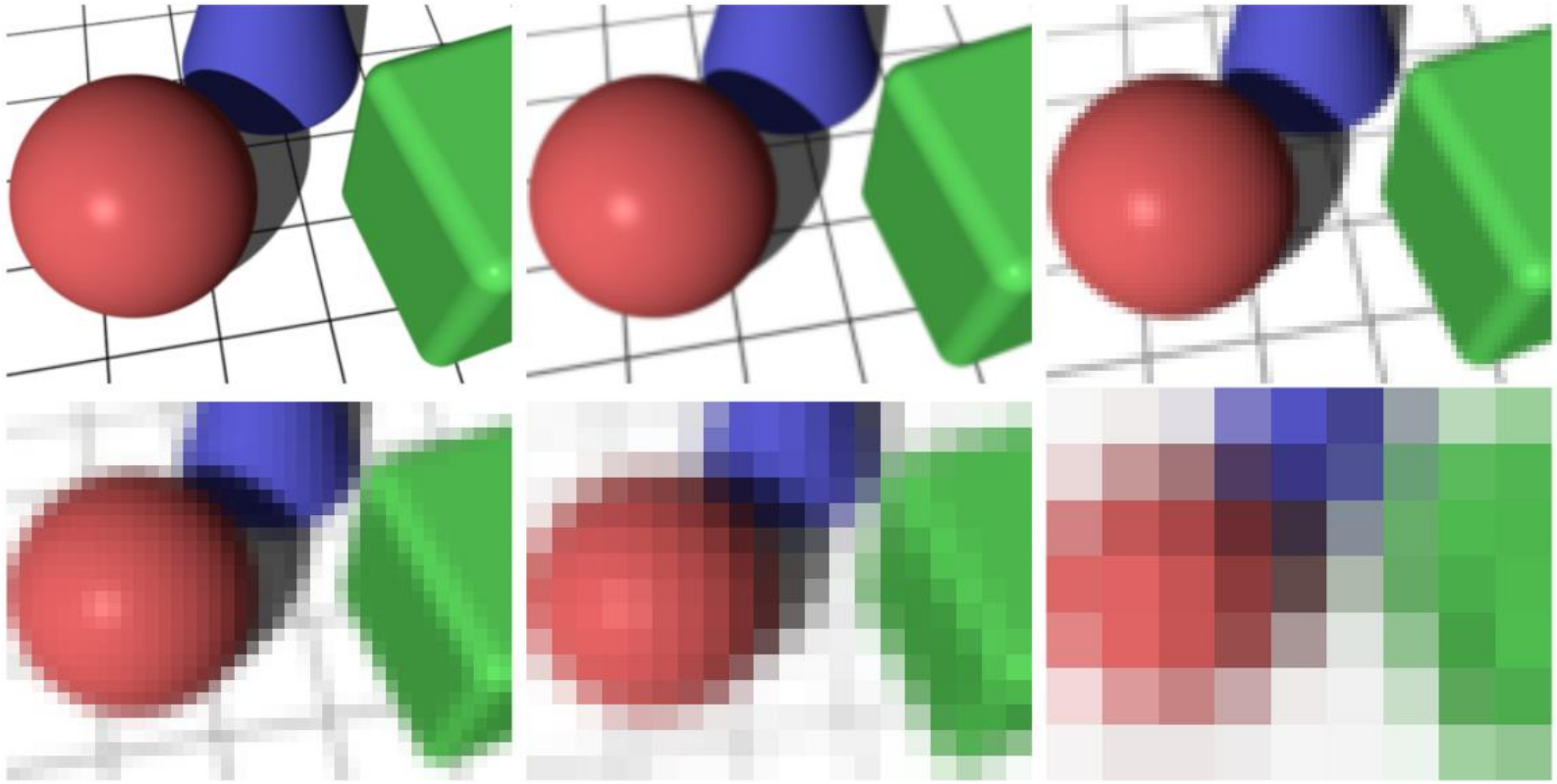
Linearization Assumption and Coarse-to-fine optimization

Assumption Easily broken in real images, as the transformation magnitude increases, the cost function becomes clearly non-convex, and the small-motion assumption is invalid.



Coarse-to-fine optimisation: downsampling

Removing higher frequency components in the images increases the parameter range for which the linearisation holds.



[Lovegrove, 2011]

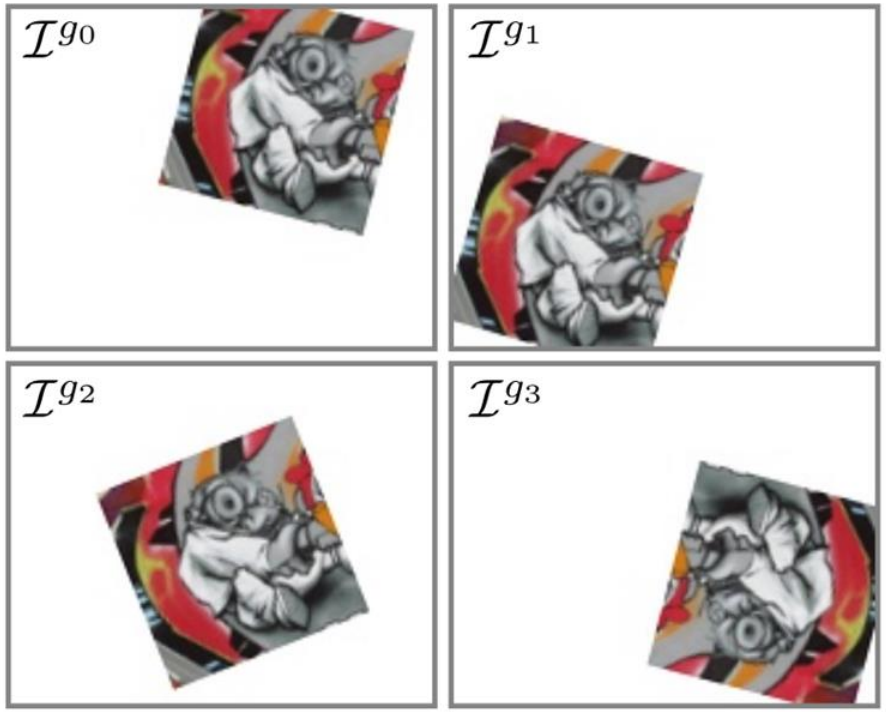
Switch to live coding demo of Lucas-Kanade

- And take a break!

The Generative Model & Parametric Tracking

Enabling use of more data for estimation of fewer parameters

Example Basic Generative Models



$$\mathtt{H}(\mathbf{x}) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathtt{I}^g \begin{pmatrix} u \\ v \end{pmatrix} = \mathtt{I}^* \left(\pi \left(\mathtt{H}(\mathbf{x}) \, (u, v, 1)^\top \right) \right).$$

Lucas-Kanade Patch Tracking Revisted (1981)

Direct alignment for 2D image patch translation with warp function ***w***(u) = u+t, and with a quadratic penalty function:

$$\underset{t \in \mathrm{R}^2}{\operatorname{argmin}} \left\{ E(t) = \sum_{u \in \Omega} (\mathcal{I}_l(u+t) - \mathcal{I}_r(u))^2 \right\} .$$

This was the underlying optimization problem for the Lucas-Kanade Patch tracking.

Dense whole image alignment technique: *warp*

1. Define the geometric model *W*, with parameters **x**, that transforms a pixel in one frame into another:

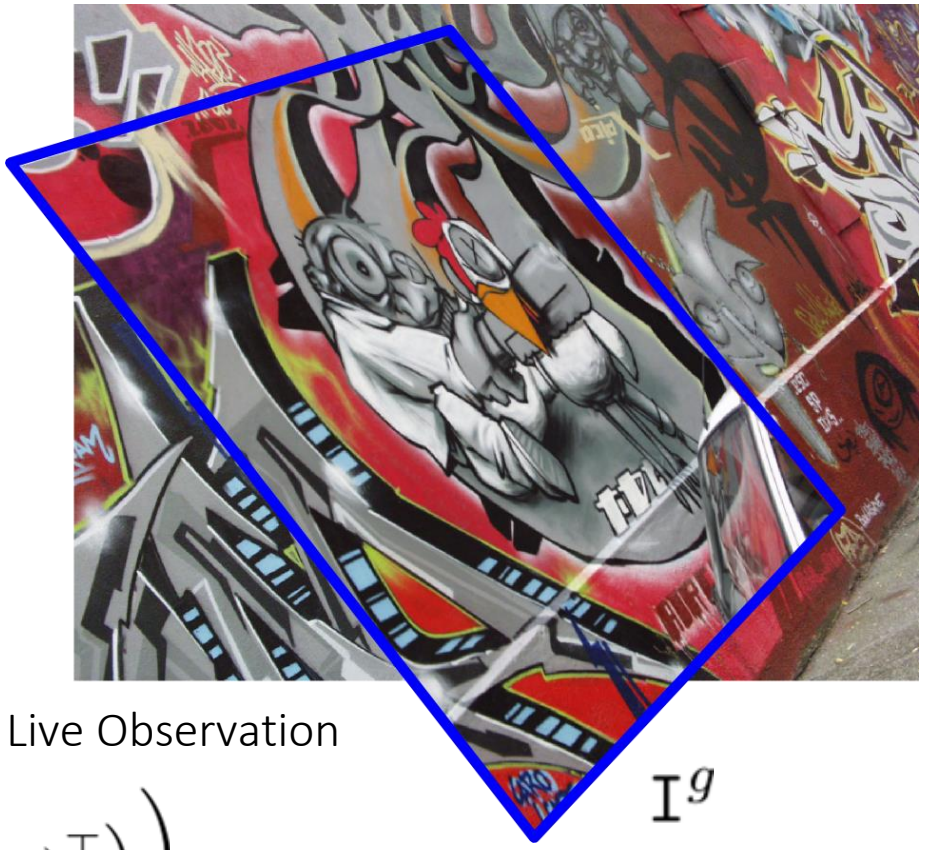
$$\mathtt{I}^g\begin{pmatrix} u \\ v \end{pmatrix} = \mathtt{I}^*\left(W\left(\mathbf{x}; (u,v)^\top\right)\right),$$

Example generative model

Reference Image

$\mathtt{I}^*$

$$\mathtt{I}^*\left(W\left(\mathbf{x};(u,v)^\top\right)\right)$$



Live Observation

$\mathtt{I}^g$

$$\mathtt{I}^g\begin{pmatrix}u\\v\end{pmatrix} = \mathtt{I}^*\left(W\left(\mathbf{x};(u,v)^\top\right)\right)$$

Dense whole image alignment technique: *error*

2. Define a Frame to Frame image alignment **error** and **cost function** that computes a similarity score between the image values $\mathbf{I}^{\mathrm{r}}(\mathbf{u})$ and $\mathbf{I}^{\mathrm{r}}(W(\mathbf{x},\mathbf{u}))$

$$e(u,\mathbf{x}) = \frac{1}{2}\sum_{\mathbf{u}_r \in \Omega_r}\left(\mathbf{I}^{l}\left(W\left(\mathbf{x};\mathbf{u}_r\right)\right) - \mathbf{I}^{r}\left(\mathbf{u}_r\right)\right)^2$$

Error function computed at each pixel

Reference Image

I^*

$e(u, \mathbf{x})$

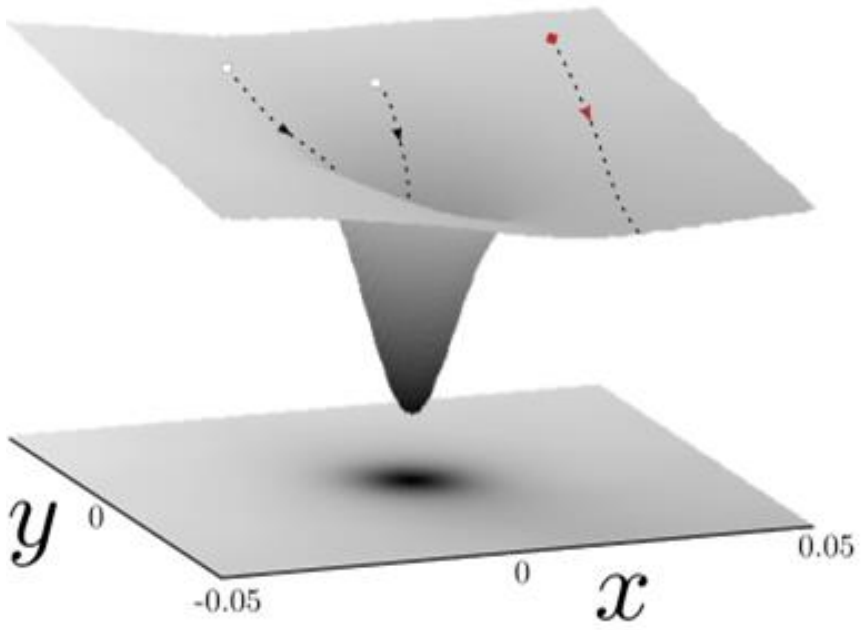
Live Observation

I^g

$$e(u, \mathbf{x}) = \mathrm{I}^l\Big(W\left(\mathbf{x}; \mathbf{u}_r\right)\Big) - \mathrm{I}^r\left(\mathbf{u}_r\right)$$

Dense whole image alignment (Direct Parametric Tracking)

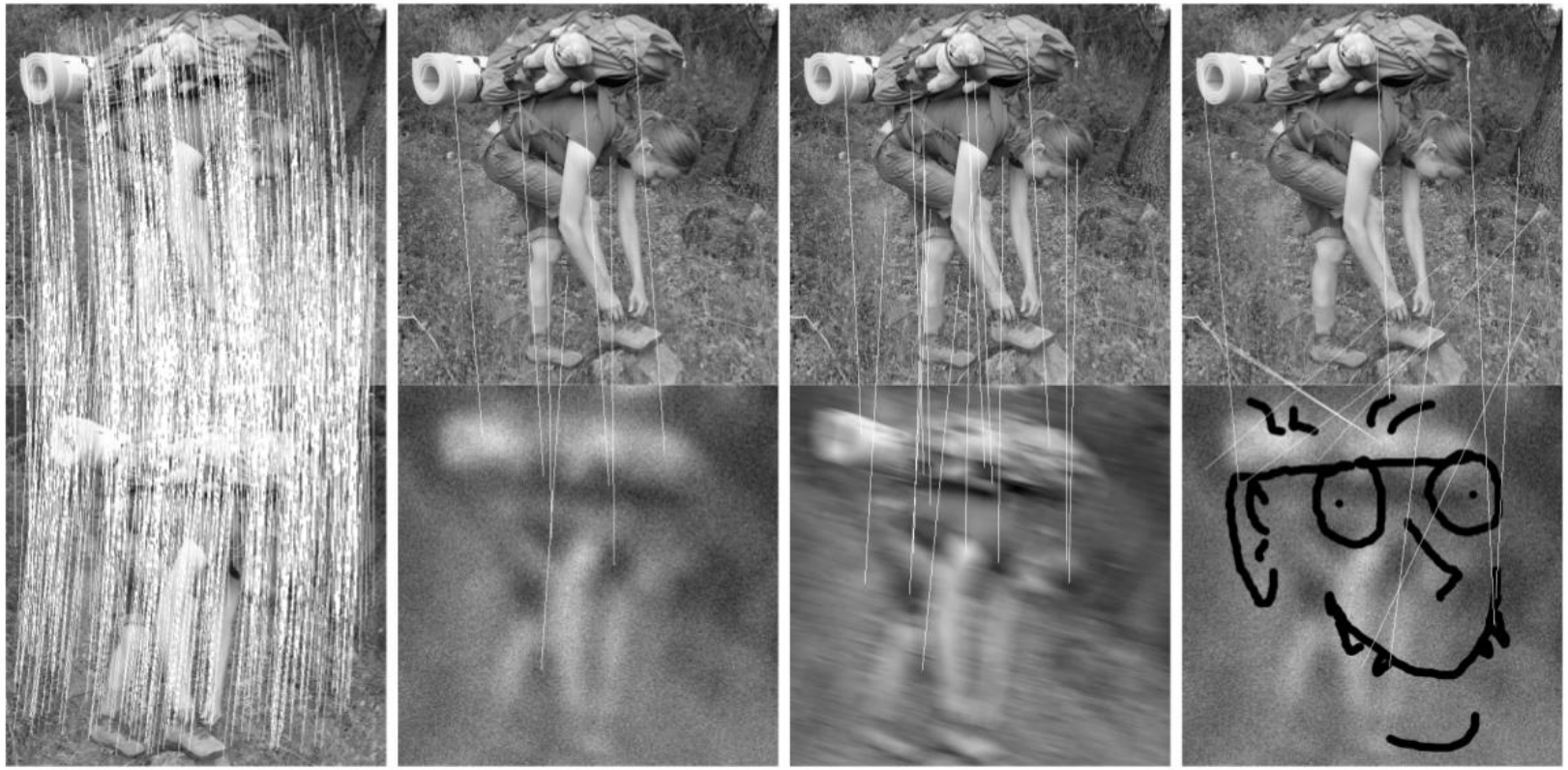
3. We obtain the estimated alignment parameters **x** at the *minimum of* the photometric cost function:



$$\mathbf{x}^{\circ} = \underset{\mathbf{x} \in \mathbb{R}^N}{\arg\min}\ F(\mathbf{x})$$

Comparison: Sparse (1) extraction and (2) matching

Descriptor extraction and matching using naively applied SIFT (Lowe, ICCV 2004)



Geometric only

Geometric, blur and noise

Geometric, motion blur

Geometric, blur, noise, occlusion

Comparison: Cost function using *dense* pixel errors

Source Frame

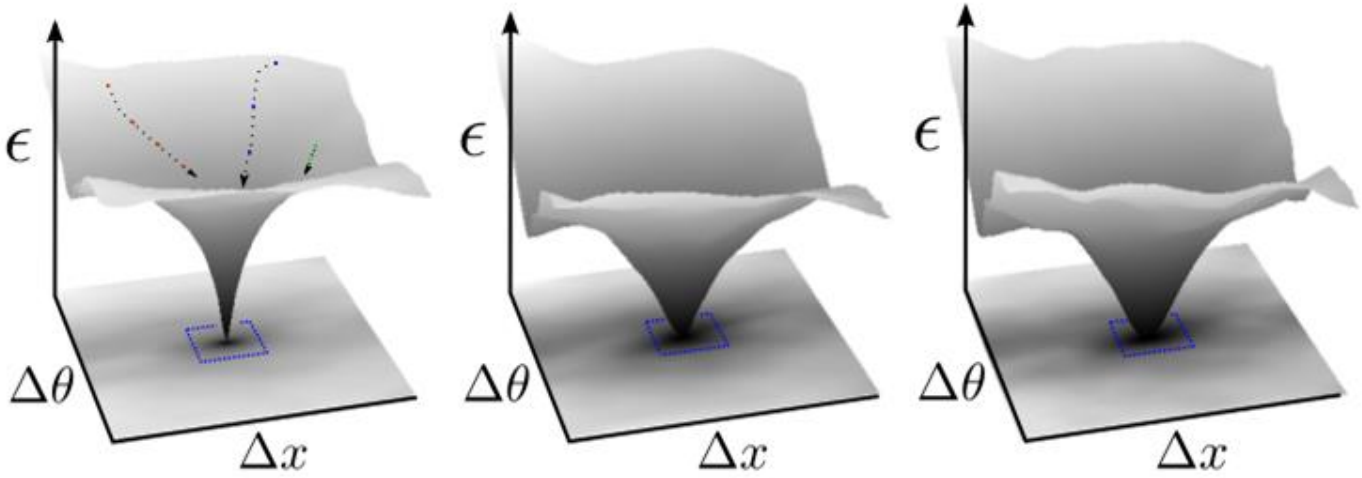
→

Geometric

Geometric, blur, noise

Geometric, blur, noise, occlusion

- ➔ Despite using a simple single pixel error term, there exists a clear global minimum
- ➔ However, there are local minima!



Parameter range: $\Delta\theta \pm \pi/2$, $\Delta x \pm 100$ pixels

Parametric Direct Optimization

Application of Quasi-Newton and Gauss-Newton Optimization

Direct Parametric Optimisation

We want to estimate the unknown transform parameters $\mathbf{x}$ between two image frames by minimising a whole image error:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \{E_w(\mathbf{x})\} \ ,$$

$$E_w(\mathbf{x}) = \sum_{u \in \Omega} \psi\left(e(u, \mathbf{x})\right) \ .$$

$$e(u, \mathbf{x}) = \mathcal{I}_l(\mathbf{w}(u, \mathbf{x})) - \mathcal{I}_r(u)$$

Where:

- $\mathbf{w}$ is the generative model warp function
- e is the whole image error induced by $\mathbf{w}$
- $\psi(e)$ is a penalty over the image error e $(\mathbf{e.g.}\ e^2)$, and Ω is the image domain.

Direct Parametric Optimization

Note the image error (e), can be as simple as the per pixel difference of the images **given the** generative model warp (w) with current parameters ***x***:

$$e(u, \mathbf{x}) = \mathcal{I}_l(\mathbf{w}(u, \mathbf{x})) - \mathcal{I}_r(u)$$

We will use an Iterative Gauss-Newton Gradient descent on E_w to estimate the parameters **x**.

Taylor series expansion of $E_w(\mathbf{x}_0 + \Delta x)$

$$\tilde{E}_w(\mathbf{x}_0 + \Delta x) \approx E_w(\mathbf{x}_0) + \nabla_{\mathbf{x}} E_w(\mathbf{x}_0)\Delta x + \frac{1}{2}\Delta x^\top \nabla_{\mathbf{x}}^2 E_w(\mathbf{x}_0)\Delta x \ ,$$

Solve convex form at Stationary Point:

$$\nabla_{\Delta x} \tilde{E}_w = 0.$$

Gauss-Newton Approximation

Approximate the Hessian by truncating to the first order components:

$$\tilde{E}_w(\mathbf{x}_0 + \Delta x) \approx E_w(\mathbf{x}_0) + \nabla_{\mathbf{x}} E_w(\mathbf{x}_0)\Delta x + \tfrac{1}{2}\Delta x^\top \nabla_{\mathbf{x}}^2 E_w(\mathbf{x}_0)\Delta x \,,$$

$$\tfrac{1}{2}\sum_{u\in\Omega} \Delta x^\top J(u,\mathbf{x}_0)^\top J(u,\mathbf{x}_0)\Delta x$$

The result is an approximated 2nd order linearisation:

$$\tilde{E}_w(\mathbf{x}_0 + \Delta x) = E_w(\mathbf{x}_0) + \sum_{u\in\Omega} \psi'(e(u,\mathbf{x}_0))J(u,\mathbf{x}_0)\Delta x + \tfrac{1}{2}\sum_{u\in\Omega} \Delta x^\top J(u,\mathbf{x}_0)^\top J(u,\mathbf{x}_0)\Delta x \,,$$

Gradient of the cost function

$$\tilde{E}_w(\mathbf{x}_0 + \Delta x) = E_w(\mathbf{x}_0) + \sum_{u \in \Omega} \psi'(e(u, \mathbf{x}_0)) J(u, \mathbf{x}_0) \Delta x + \frac{1}{2} \sum_{u \in \Omega} \Delta x^\top J(u, \mathbf{x}_0)^\top J(u, \mathbf{x}_0) \Delta x \ ,$$

Derivative of the penalty function:

$$\psi'(e(u, \mathbf{x}_0)) = \left. \frac{\partial \psi(e(u, \mathbf{x}))}{\partial e(u, \mathbf{x})} \right|_{\mathbf{x}_0} ,$$

i.e. $2e(u, x_0)$ for $\psi(e(u,x))=e(u,x)^2$

Derivative of the observation prediction function:

$$J(u, \mathbf{x}_0) = \left. \frac{\partial \mathcal{I}_l(\mathbf{w}(u, \mathbf{x}))}{\partial \mathbf{x}} \right|_{\mathbf{x}_0} .$$

Example Jacobians to follow for SO(3) Camera, SE(3) RGB, SE(3) RGB-D **Warp Functions**

Solve for the linearised Cost function

Remember, a minimising argument is achieved as a function extremum:

$$\nabla_{\Delta x} \tilde{E}_w = 0.$$

Taking the derivative of the linearised cost function:

$$\cancel{E_w(\mathbf{x}_0)} + \sum_{u \in \Omega} \psi'(e(u, \mathbf{x}_0)) J(u, \mathbf{x}_0) \cancel{\Delta x} + \cancel{\frac{1}{2}} \sum_{u \in \Omega} \cancel{\Delta x^\top} J(u, \mathbf{x}_0)^\top J(u, \mathbf{x}_0) \Delta x \ ,$$

$$\sum_{u \in \Omega} \psi'(e(u, \mathbf{x}_0)) J(u, \mathbf{x}_0) \quad + \quad \sum_{u \in \Omega} J(u, \mathbf{x}_0)^\top J(u, \mathbf{x}_0) \Delta x$$

Solving for the incremental update

Resulting in the *normal equations*, *we solve this linear system*:

$$\sum_{u\in\Omega} J(u,\mathbf{x}_0)^\top J(u,\mathbf{x}_0)\Delta x = -\sum_{u\in\Omega} \psi'(e(u,\mathbf{x}_0))J(u,\mathbf{x}_0)\ ,$$

$$\Rightarrow \Delta x = -(\sum_{u\in\Omega} J(u,\mathbf{x}_0)^\top J(u,\mathbf{x}_0))^{-1} \sum_{u\in\Omega} \psi'(e(u,\mathbf{x}_0))J(u,\mathbf{x}_0)\ .$$

The parameter vector is then updated:

$$\mathbf{x} \leftarrow \mathbf{x}_0 + \Delta x\ .$$

Luckas-Kanade tracking can be derived in this more general way for the simple 2D image translation only generative model.

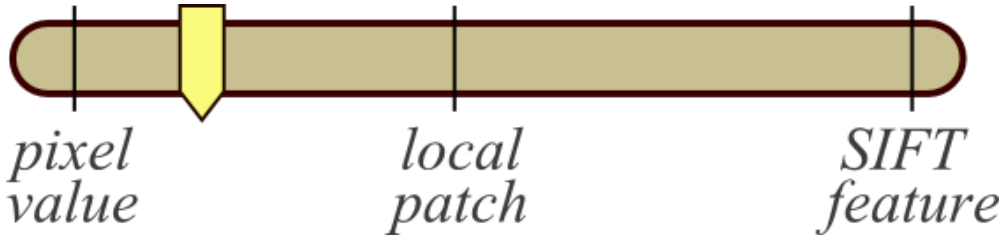
General Parametric Optimization Recap

- Given linearizable Warp, Image Error and Penalty Function

- We can use this Parametric optimization for many applications:
  - Image Correspondence
  - Camera Tracking
  - Model Tracking (incl. Rigid, Non-rigid and Articulated)
  - Geometric Model refinement
  - Estimation of other scene parameters like Lighting, Reflectance,…, etc

Direct Image Errors vs. Image Descriptors

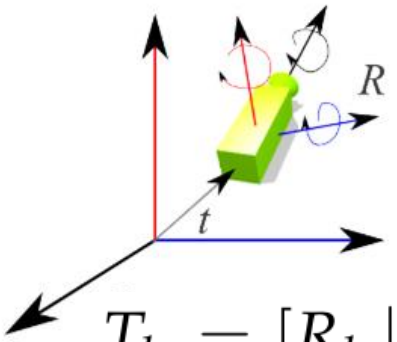
Generally we can trade off between complexity of the descriptor size and density of descriptor extraction to obtain a more robust error f:



- For whole image alignment, there is great redundancy for the few parameters being estimated, which can increase tracking robustness
- But gradient descent on the whole image cost function requires initialisation near to the global minimum (i.e. not for wide baseline)
- Many variations on how robustify against, or model photometric transformations

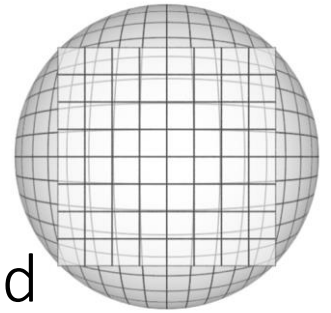
Application to Real-time Incremental Camera Tracking

For Passive and RGB-D Cameras



$$T_{lr} = [R_{lr}|t_{lr}] \in SE_3$$

Incremental Transformations Recap



We can parameterise the relative camera motion between reference and live frames:

$$T_{lr} = [R_{lr}|t_{lr}] \in SE_3$$

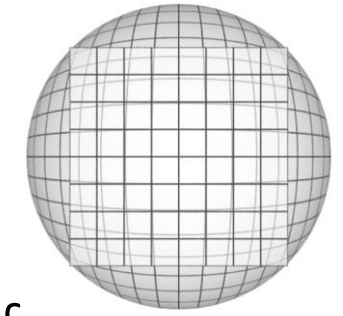
A minimal parameterisation of a rigid body transform is given by:

$$\mathbf{x} = \begin{pmatrix} \omega \in \mathbb{R}^3 \\ v \in \mathbb{R}^3 \end{pmatrix},$$

Where the parameters define an element of the Lie Algebra as :

$$\hat{\mathbf{x}} = \begin{pmatrix} [\omega]_\times & v \\ 0 & 0 \end{pmatrix} \in \mathfrak{se}_3 \quad \text{where} \quad [\omega]_\times = \begin{pmatrix} 0 & -\omega_2 & \omega_1 \\ \omega_2 & 0 & -\omega_0 \\ -\omega_1 & \omega_0 & 0\,. \end{pmatrix}$$

Incremental Transformations Recap



The derivative of the non-linear exponential map that takes $[\omega]_x$ to the SO3 rotation matrix can be obtained by truncating to the linear term of the matrix exponential:

$$\exp([\omega]_\times) \mapsto I + [\omega]_\times + \tfrac{1}{2}[\omega]_\times^2 + \cdots + \tfrac{1}{k!}[\omega]_\times^k + \cdots$$

The linearisation of the exponential map to first order for ω around 0 is useful in practice, i.e. cos(θ)~1 and sin(θ)~0.

We will compose resulting incremental small SO3 (or SE3) transformations together via the exponential map:

$$T_{lr} = \exp(\hat{\mathbf{x}}^n)\exp(\hat{\mathbf{x}}^{n-1})\ldots\exp(\hat{\mathbf{x}}^0)\tilde{T}_{lr}$$

Generative Model for a ***Rotating*** Camera

The transformation of a pixel from one frame into another is *independent* of the scene geometry if $t = (0\ 0\ 0)^{\mathsf{T}}$:

$$u_l = \pi\left(K R_{lr} K^{-1} \dot{u}_r\right) .$$

Here $K^{-1}u_r$ defines a ray through pixel u_r and the camera center that is rotated and projected into the live frame.

Given an incremental compositional update to the rotation between the reference and live frames, the **warp function** is therefore:

$$\mathbf{w}_{SO_3}(u_r, \omega) = \pi\left(K \exp([\omega]_\times) \tilde{R}_{lr} K^{-1} \dot{u}_r\right) .$$

Optimization for a *Rotating* Camera

$$E_w(\mathbf{x}) = \sum_{u \in \Omega} \psi\left(e(u, \mathbf{x})\right)$$

$$e(u, \mathbf{x}) = \mathcal{I}_l(\mathbf{w}(u, \mathbf{x})) - \mathcal{I}_r(u)$$

Whole Image Error: E

Inserting $\mathbf{w}_{SO3}$ into the whole image error we now perform the linearisation of $E_w(\mathbf{x}_0 + \Delta)$ with $\Delta = \omega$, hence we compute the per pixel image **error derivative** as:

$$J(u, \omega) = \frac{\partial I_l(\mathbf{w}_{SO3})}{\partial \mathbf{w}_{SO3}} \frac{\partial \mathbf{w}_{SO3}(u, \omega)}{\partial K \exp([\omega]_\times) \tilde{R}_{lr} K^{-1} \dot{u}_r} \frac{\partial \mathrm{K} \exp([\omega]_\times) \tilde{R}_{lr} K^{-1} \dot{u}_r}{\partial \omega}$$

Optimization for a *Rotating* Camera

Pre-computing the currently rotated ray

$$(x, y, z)^\top = \tilde{R}_{lr} K^{-1} u_r$$

The resulting **error gradient** vector for pixel *u* is:

$$J(u, \omega) = \begin{pmatrix} \nabla_x I_l \\ \nabla_y I_l \end{pmatrix}^\top \begin{pmatrix} \frac{f_x}{z} & 0 & -\frac{x f_x}{z^2} \\ 0 & \frac{f_y}{z} & -\frac{y f_y}{z^2} \end{pmatrix} \begin{pmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{pmatrix}.$$

Optimization for a *Rotating* Camera

Evaluating the total Jacobian together with the chosen penalty function, we solve the resulting *normal equations:*

$$\Delta x = -\left(\sum_{u\in\Omega} J(u,\mathbf{x}_0)^\top J(u,\mathbf{x}_0)\right)^{-1} \sum_{u\in\Omega} \psi'(e(u,\mathbf{x}_0))J(u,\mathbf{x}_0)$$

Finally, form the SO3 matrix by exponentiation, and compose onto the initial transform:

$$\tilde{R}_{lr} \leftarrow \exp([\omega]_\times)\tilde{R}_{lr} \ .$$

Application: Real-time Spherical Mosaicing using Whole Image Alignment

[Lovegrove & Davison, ECCV 2010]

Generative Model for 6DOF RGB-D Camera Tracking

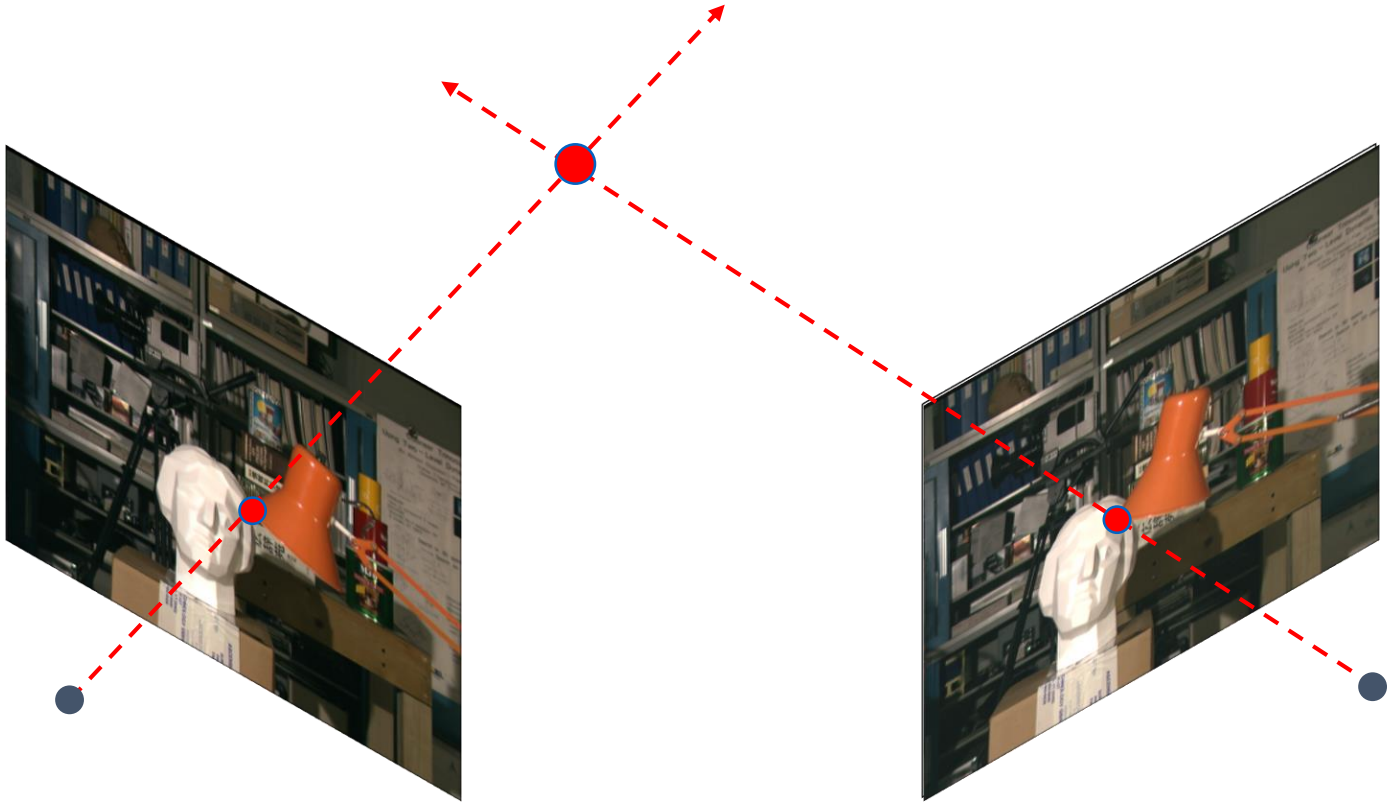
When a depth map is also available in one frame, we can compute pixel transfer of points in one frame given the relative **SE3** transform T_{lr}:

$$u_l = \pi \left(K T_{lr} K^{-1} \mathcal{D}_r(u_r) \dot{u}_r \right)$$

Given an incremental compositional update to the transformation between the reference and live frames, the **warp function** is therefore:

$$\mathbf{w}_{SE_3}(u, \mathbf{x}) = \pi \left(K \exp(\hat{\mathbf{x}}) \tilde{T}_{lr} K^{-1} \mathcal{D}_r(u_r) \dot{u}_r \right)$$

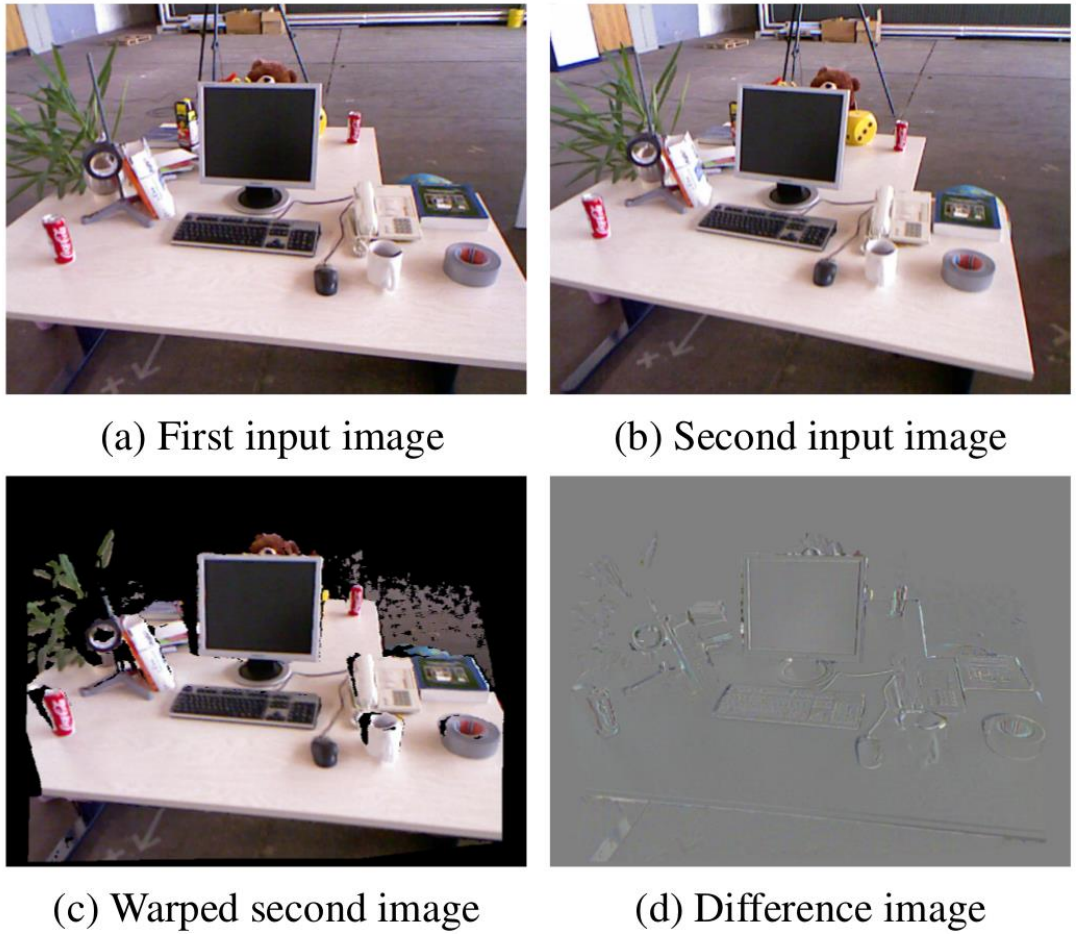
Dense Scene Geometry Generative Model



$$\mathbf{w}_{SE_3}(u, \mathbf{x}) = \pi \left(K \exp(\hat{\mathbf{x}}) \tilde{T}_{lr} K^{-1} \mathcal{D}_r(u_r) \dot{u}_r \right)$$

Dense Pixel Transfer through the Depth Image

- ➔ Note: we can use rendering engine (e.g. OpenGL) to achieve the observation prediction.
- ➔ Requires a triangle mesh representation of the depth map.
- ➔ Can correctly predict self occlusion since it is a surface.



(a) First input image (b) Second input image

(c) Warped second image (d) Difference image

[image from: Real-Time Visual Odometry from Dense RGB-D Images. Steinbrücker, Sturm, Cremers, 2011]

Optimization for *6DOF* RGB-D Camera Tracking

Inserting $\mathbf{w}_{SE3}$ into the whole image error we now perform the **linearisation** of $E_w(\mathbf{x}_0 + \Delta)$ with rigid body parameters $\Delta = \mathbf{x}$:

$$J(u, \mathbf{x}) = \frac{\partial I_l(\mathbf{w}_{SE_3})}{\partial \mathbf{w}_{SE_3}} \frac{\partial \mathbf{w}_{SE_3}(u, \mathbf{x})}{\partial K \exp(\hat{\mathbf{x}}) \tilde{T}_{lr} K^{-1} \mathcal{D}_r(u) \dot{u}_r} \frac{\partial K \exp(\hat{\mathbf{x}}) \tilde{T}_{lr} K^{-1} \mathcal{D}_r(u) \dot{u}_r}{\partial \mathbf{x}}$$

Pre-computing the currently transformed per pixel vertex:

$$(x, y, z)^\top = \tilde{T}_{lr} K^{-1} \mathcal{D}_r(u) \dot{u}_r$$

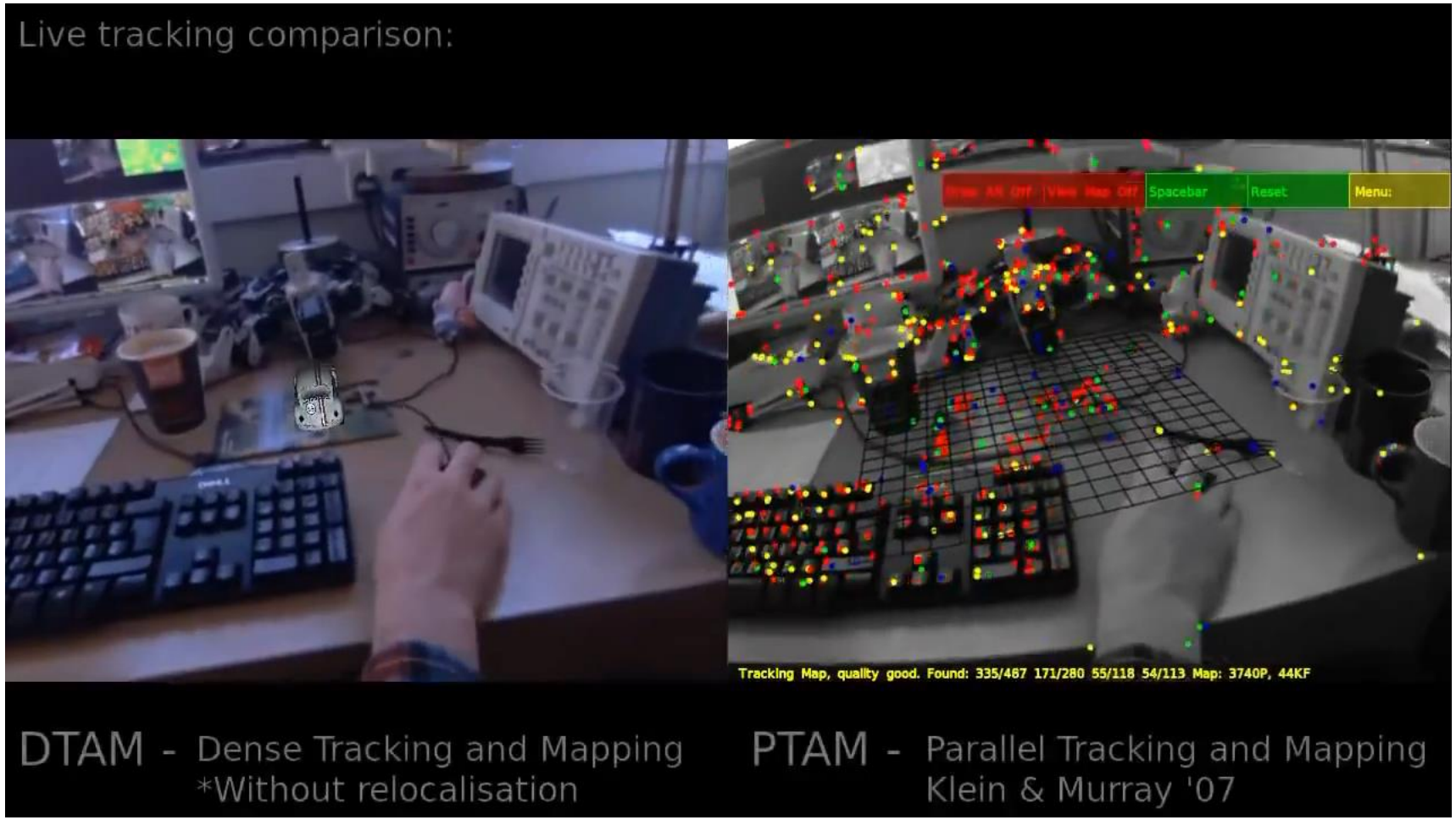
The resulting image **error gradient** vector for pixel *u* is:

$$J(u, \mathbf{x}) = \begin{pmatrix} \nabla_x I_l \\ \nabla_y I_l \end{pmatrix}^\top \begin{pmatrix} \frac{f_x}{z} & 0 & -\frac{x f_x}{z^2} \\ 0 & \frac{f_y}{z} & -\frac{y f_y}{z^2} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{pmatrix}$$

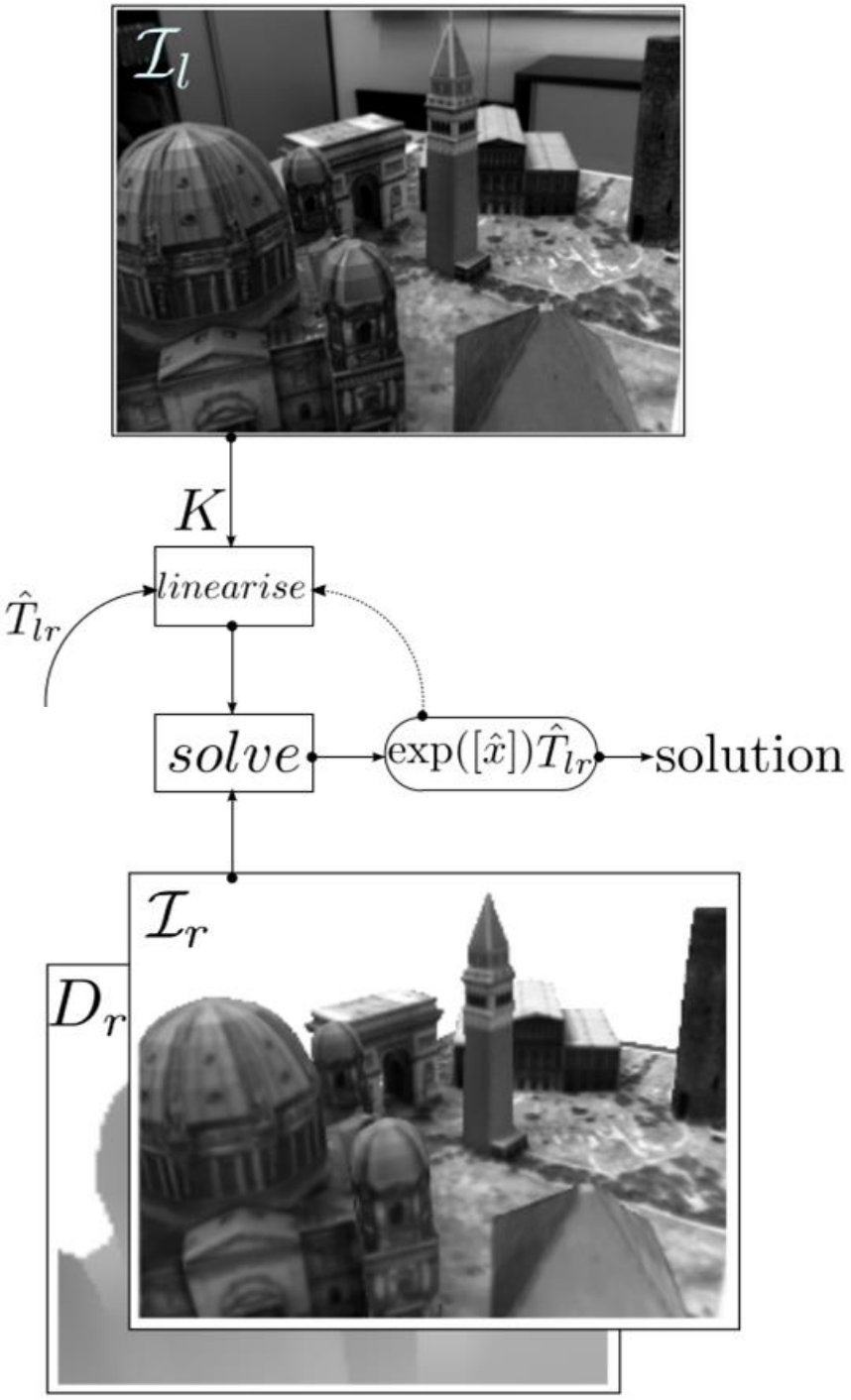
Solve normal equations and compose:

$$\tilde{T}_{lr} \leftarrow \exp(\hat{\mathbf{x}}) \tilde{T}_{lr}$$

Application: Dense Tracking and Mapping in Real-Time



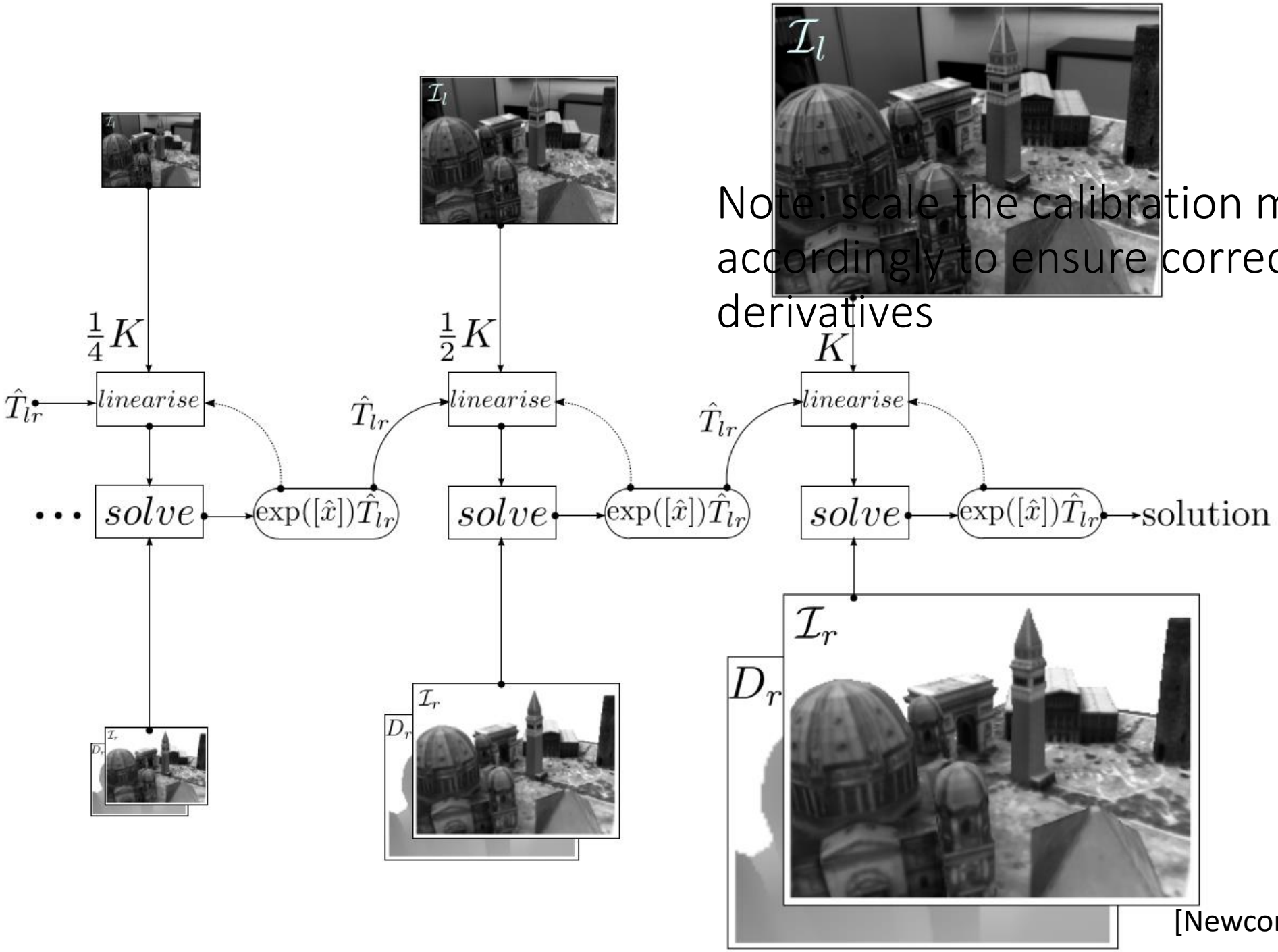
[Newcombe, Lovegrove, Davison, ICCV 2011]



The linearisation assumption:

$$\mathcal{I}_l(\mathbf{w}(u, \Delta\mathbf{x})) \approx \mathcal{I}_l(\mathbf{w}(u, \mathbf{0})) + J(\mathbf{0})\Delta\mathbf{x} \,,$$

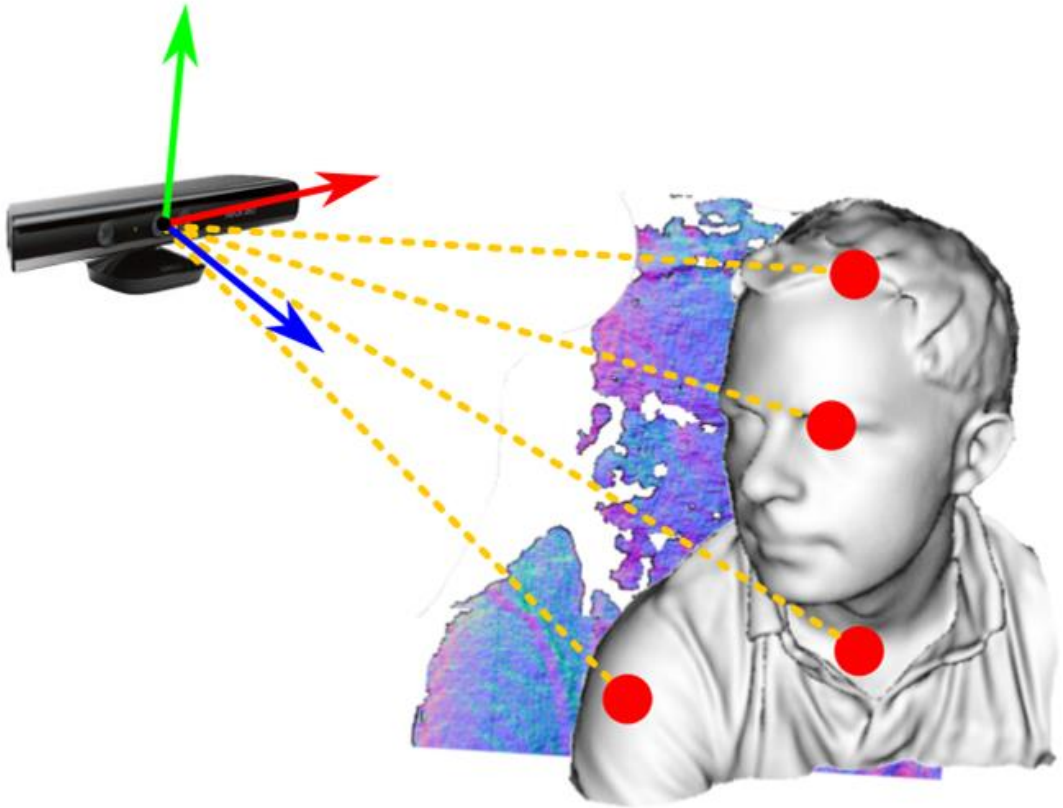
$$\Rightarrow \mathcal{I}_r(u) - \mathcal{I}_l(\mathbf{w}(u, \mathbf{0})) \approx J(\mathbf{0})\Delta\mathbf{x} \,.$$



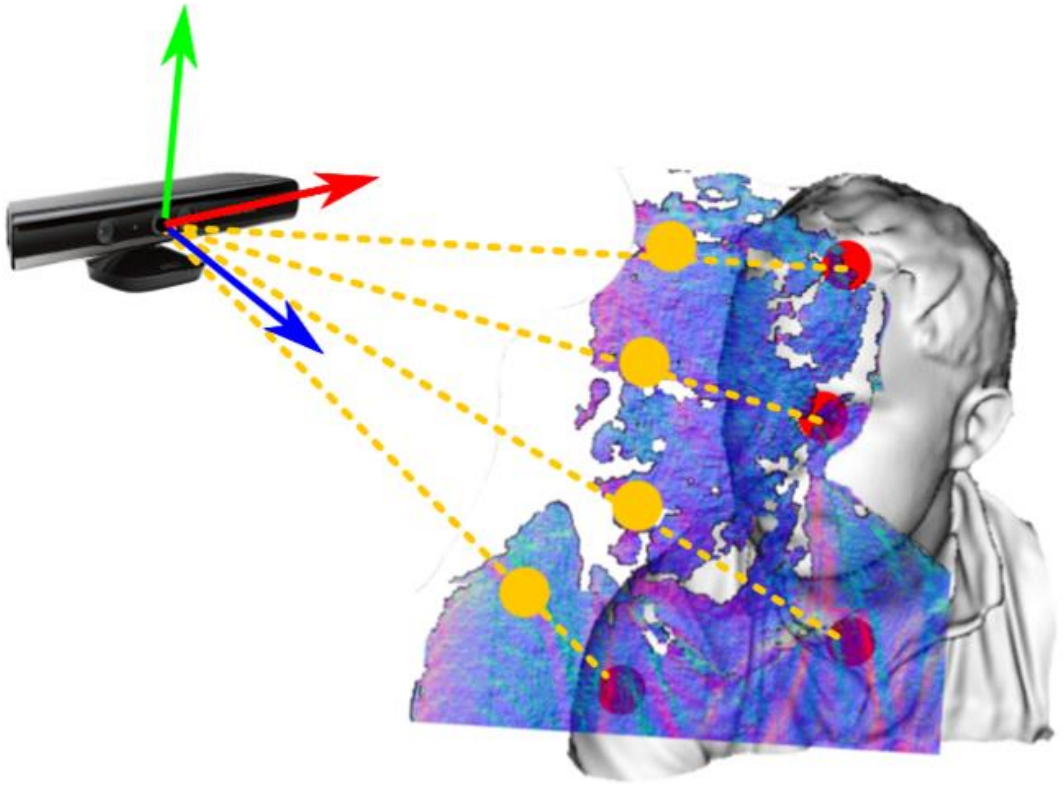
Note: scale the calibration matrix accordingly to ensure correct derivatives

[Newcombe, 2013]

General *6DOF* depth tracking (ICP)



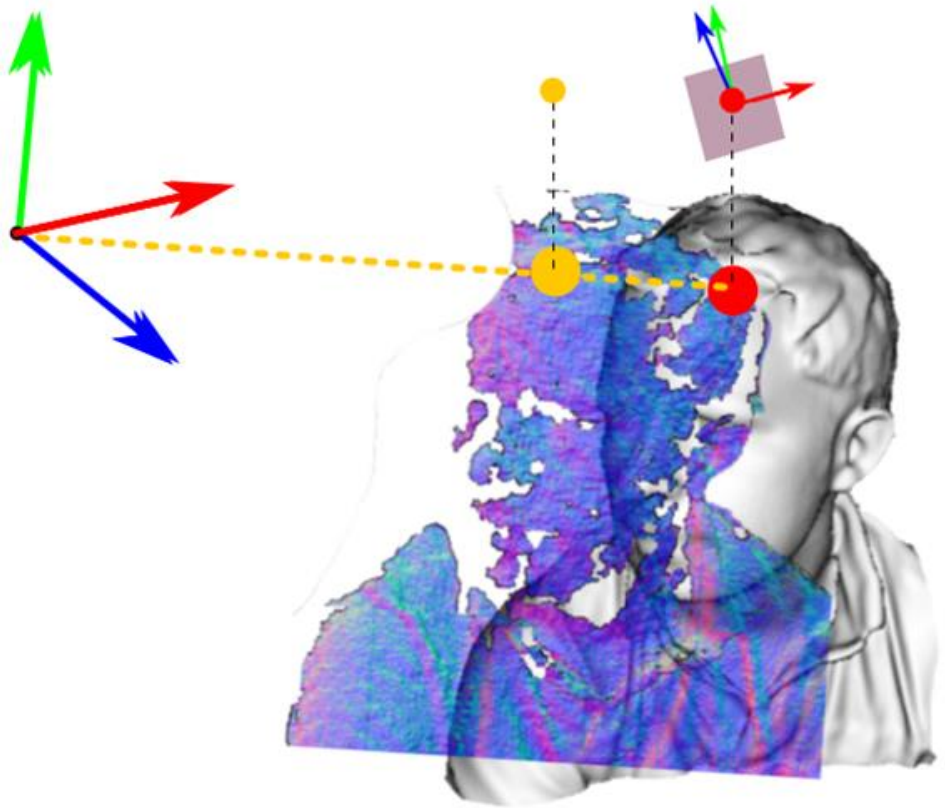
(a) The model (grey) is rendered into the estimated frame. We can sample points from this model in image space (red dots).



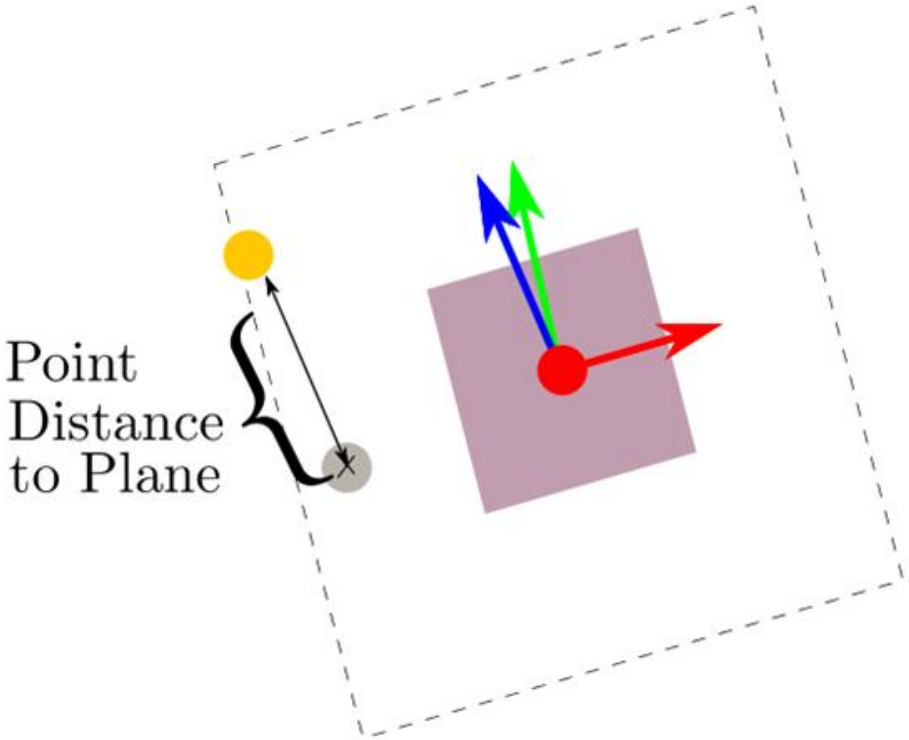
(b) Projective data-association with the live frame: Corresponding are selected by pairing points which lie on the same ray (red-yellow dots).

[From KinectFusion: Newcombe et al , 2011]

General *6DOF* depth tracking (ICP)

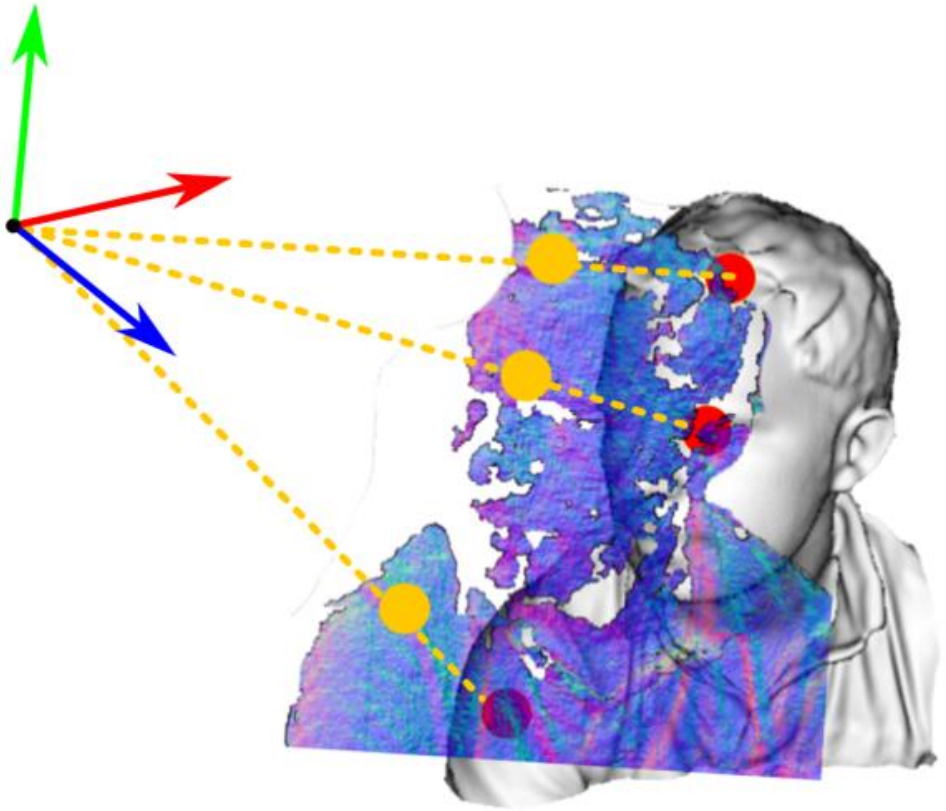


(c) Each pair, has a point-plane constraint: the surface normal estimated from the model provides the normal since it is higher quality.

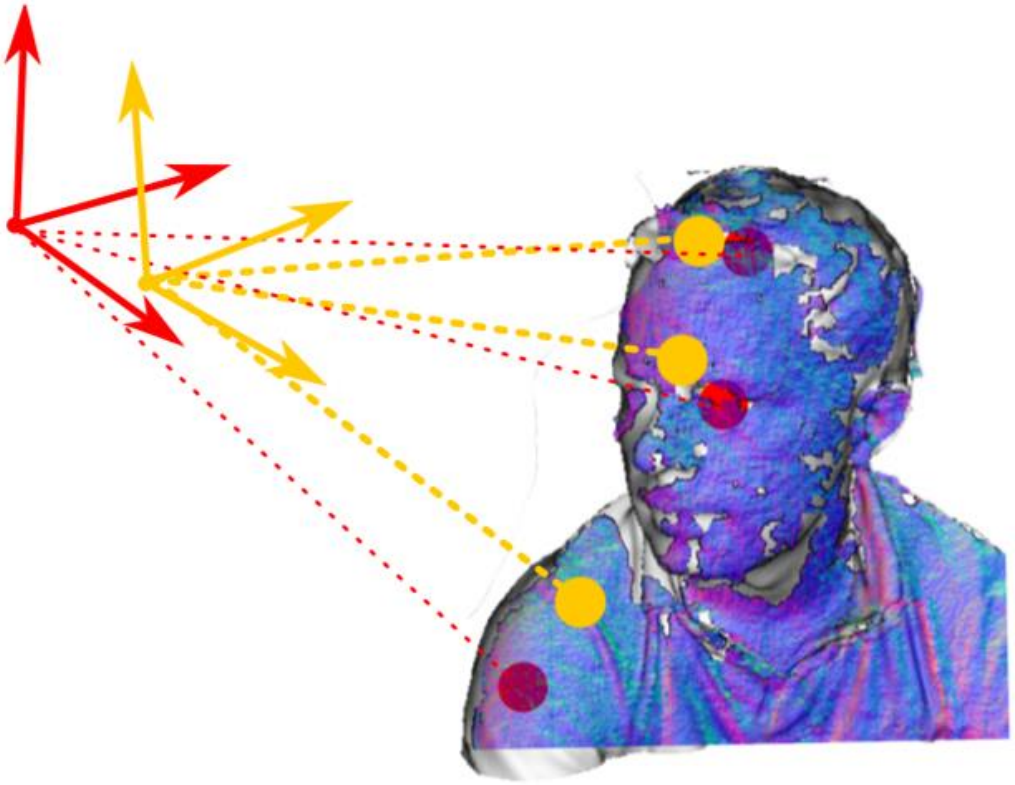


(d) Each point-plane constraint provides an error measure as the shortest distance of the live image point to the tangent plane of the corresponding model point.

General *6DOF* depth tracking (ICP)



(e) Pairs fail a point-plane compatibility if the point-point Euclidean distance or normal-normal angle exceed thresholds.



(f) A Gauss-Newton based iterative gradient descent minimisation of the sum of point-plane error induced by the remaining pairs results in the new pose estimate.

Generative Model for depth Camera tracking (dense ICP)

Given 2 depth images, we define a generative model over the vertex maps:

$$v_r(u) = K^{-1}\dot{u}\mathcal{D}_r(u) \ ,$$

Warp the surface in the reference image into the live image given the relative SE3 transform:

$$v_l(u') = \tilde{T}_{rl}K^{-1}\dot{u}'\mathcal{D}_l(u') \ ,$$

$$u' = \mathbf{w}_{se3}(u, \mathbf{x}_0) = \pi(K\tilde{T}_{lr}v_r) \ .$$

We can use the per depth pixel point-plane error, instead of a euclidean distance of the vertices:

$$e(u, \mathbf{x}) = N_r^\top(u)(\exp(\hat{\mathbf{x}})v_l(u') - v_r(u)) \ ,$$

[Newcombe, 2013]

Whole image depth image tracking (dense ICP)

Plugging the point-plane error into the whole image cost function, we again perform **linearisation** of $E_w(\mathbf{x}_0 + \Delta)$ with rigid body parameters

$\Delta = \mathbf{x}$. Pre-computing the currently transformed per pixel vertex:

$$(x, y, z)^\top = \tilde{T}_{rl} v_l(\mathbf{w}(u, \mathbf{x}))$$

The resulting image **error gradient** vector for pixel *u* is:

$$J(u, \mathbf{x}) = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}^\top \begin{pmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{pmatrix},$$

Solve normal equations and compose: $\tilde{T}_{lr} \leftarrow \exp(\hat{\mathbf{x}}) \tilde{T}_{lr}$.