Computer Vision
Motion

Professor Hager
http://www.cs.jhu.edu/~hager

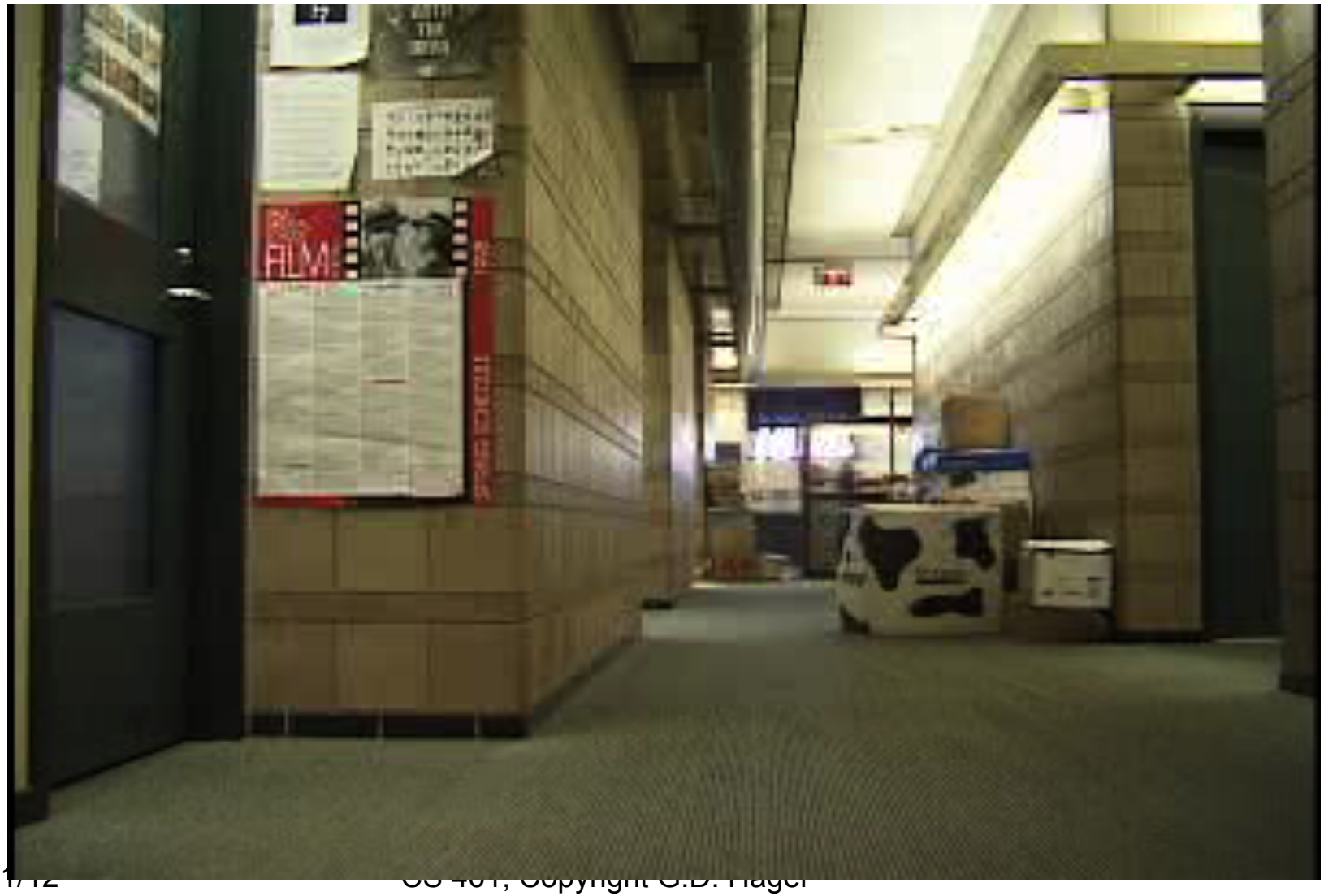
Outline

From Stereo to Motion

The motion field and optical flow (2D motion)

Factorization methods for structure and motion (3D motion)

DYNAMIC VISION: THE PROBLEM



SOME CORE PROBLEMS

- Image (2D) motion estimation
  - motion detection
  - The motion field
  - optical flow calculation
- 3D motion estimation
  - ego (self) motion
  - target motion
  - structure from motion differential
  - matching methods
- Structured motion recovery
  - visual tracking

For simplicity
we will assume a unit
focal length, nonmetric camera
(e.g. projective coordinates from
a calibrated camera)

What is Visual Motion?

1. A body moving in space experiences
   - translation (Tr)
   - rotation (R)

2. We are interested in time change
   - trans. velocity (T = dTr/dt)
   - rot. velocity (ω = "dR/dt")

3. Other rigid bodies appear to move in the opposite direction

$^{c}p = {}^{w}p - Tr$

What is Visual Motion?

1. A body moving in space experiences
   - translation (Tr)
   - rotation (R)

2. We are interested in time change
   - trans. velocity (T = dTr/dt)
   - rot. velocity (ω = "dR/dt")

3. Other rigid bodies appear to move in the opposite direction

4. Points on those bodies project into camera images

$$u = x/z$$
$$v = y/z$$

$$^{c}p = {}^{w}p - Tr$$

What is Visual Motion?

1. A body moving in space experiences
   - translation (Tr)
   - rotation (R)

2. We are interested in time change
   - trans. velocity (T = dTr/dt)
   - rot. velocity (ω = "dR/dt")

$^{c}x = {}^{w}x - Tr_x \rightarrow dx/dt = -T_x$
$^{c}y = {}^{w}y - Tr_y \rightarrow dy/dt = -T_y$
$^{c}z = {}^{w}z - Tr_z \rightarrow dz/dt = -T_z$

3. Other rigid bodies appear to move in the opposite direction

4. Points on those bodies project into camera images

$du/dt = (dx/dt\ {}^{c}z - dz/dt\ {}^{c}x)/{}^{c}z^2$

$u = x/z$
$v = y/z$

What is Visual Motion?

1. A body moving in space experiences
   - translation (Tr)
   - rotation (R)

2. We are interested in time change
   - trans. velocity (T = dTr/dt)
   - rot. velocity (ω = "dR/dt")

$^{c}x = {}^{w}x - Tr_x \rightarrow dx/dt = -T_x$
$^{c}y = {}^{w}y - Tr_y \rightarrow dy/dt = -T_y$
$^{c}z = {}^{w}z - Tr_z \rightarrow dz/dt = -T_z$

3. Other rigid bodies appear to move in the opposite direction

4. Points on those bodies project into camera images

$u = x/z$
$v = y/z$

$du/dt = (dx/dt - dz/dt\ {}^{c}x/{}^{c}z)/{}^{c}z^2$
$du/dt = -(T_x - u\ T_z)/z$

What is Visual Motion?

1. A body moving in space experiences
   - translation (Tr)
   - rotation (R)

2. We are interested in time change
   - trans. velocity (T = dTr/dt)
   - rot. velocity (ω = "dR/dt")

3. Other rigid bodies appear to move in the opposite direction

4. Points on those bodies project into camera images

$$u = x/z$$
$$v = y/z$$

$$^{c}x = {}^{w}x - Tr_x \rightarrow dx/dt = -T_x$$
$$^{c}y = {}^{w}y - Tr_y \rightarrow dy/dt = -T_y$$
$$^{c}z = {}^{w}z - Tr_z \rightarrow dz/dt = -T_z$$

$$du/dt = (dx/dt - dz/dt\ {}^{c}x/{}^{c}z)/{}^{c}z^2$$
$$du/dt = -(T_x - u\,T_z)/z$$
$$dv/dt = -(T_y - v\,T_z)/z$$

What is Visual Motion?

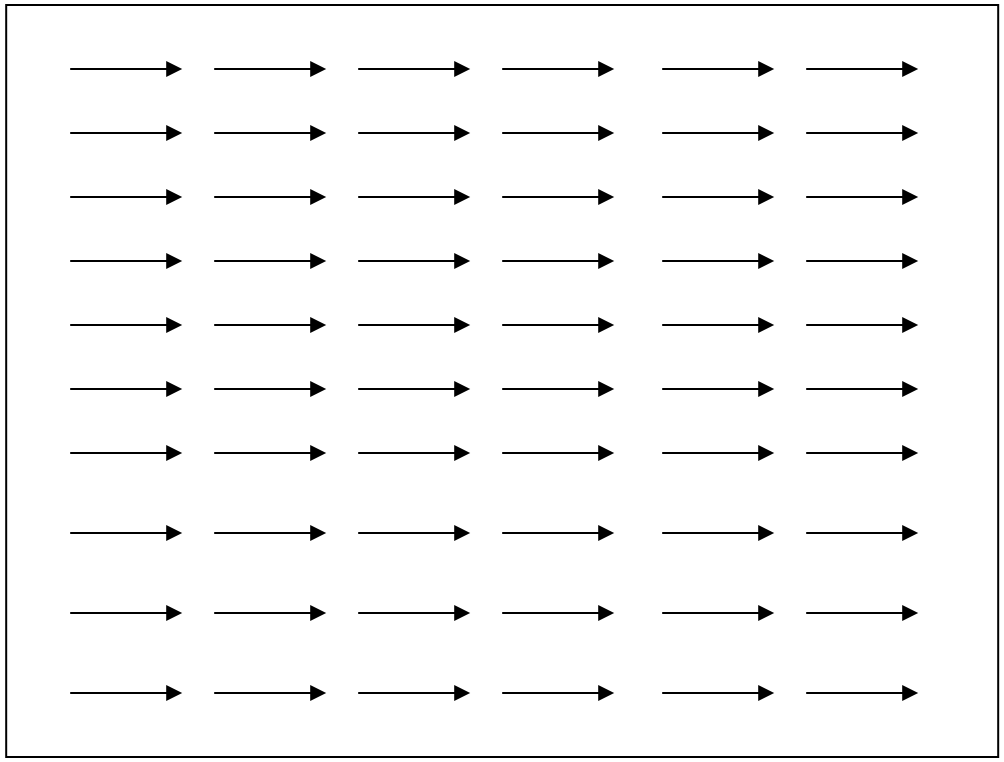
$$du/dt = -(T_x - u\,T_z)/z$$
$$dv/dt = -(T_y - v\,T_z)/z$$

This expression defines the motion field for the image under pure translation

Note that, given a velocity vector, we can determine precisely what the motion field is *up to a scale factor related to the depths of the points that are observed*

THE MOTION FIELD

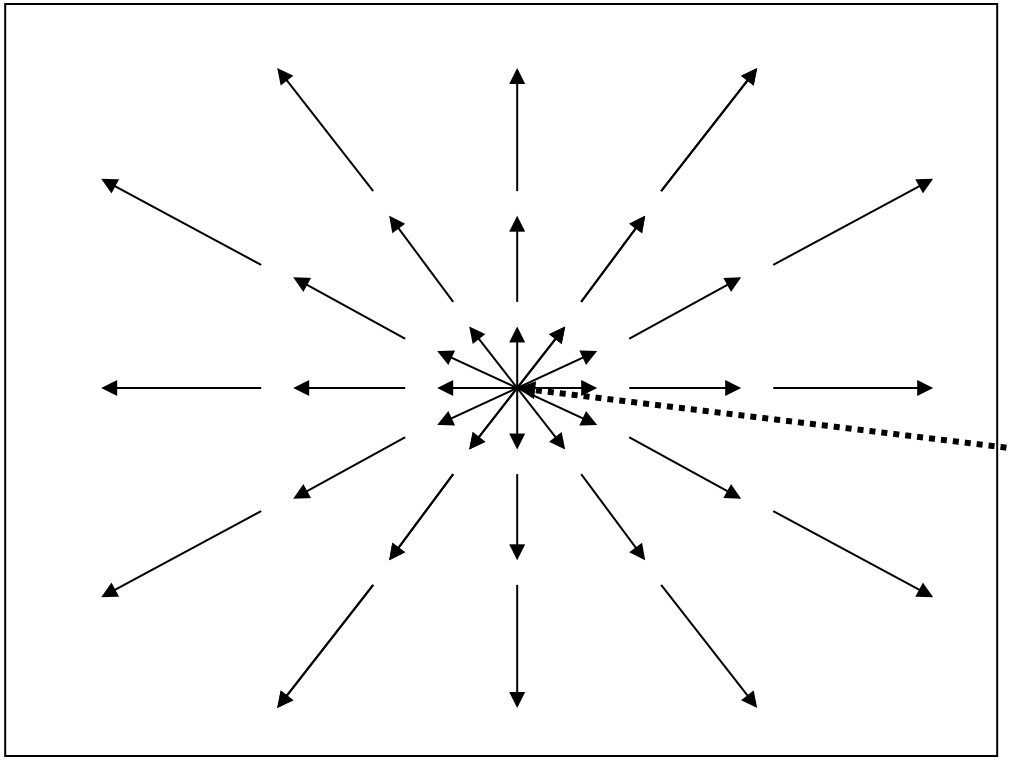
The "instantaneous" velocity of points in an image



TRANSLATION
(OR ROTATION)
TO THE LEFT

THE MOTION FIELD

The "instantaneous" velocity of points in an image



LOOMING

The *focus of expansion*

With just this information it is possible to calculate:

1. Direction of motion
2. Time to collision

Calculations

$du/dt = -(T_x - u\,T_z)/z$
$dv/dt = -(T_y - v\,T_z)/z$

- Solve for du/dt = 0 and dv/dt = 0 to find the focus of expansion

  Define $u_0 = T_x/T_z$ $(T_z := 0)$; $v_0 = T_y/T_z$
  - note $(u_0, v_0, 1)$ is direction of translation

- Pulling out a T_z yields
  - $u' = -(u - u_0)\,T_z/Z$; $v' = -(v - v_0)\,T_z/Z$ -> $p' = (p - p_0)\,T_z/Z$
  - therefore, the field is radial and rooted at $(T_x/T_z, T_y/T_z) = p_0$

- Z/T_z is the number of seconds until the point crosses the image plane ("time to collision") *for this point*
  - $Z/T_z = (u-u_0)/u'$; $Z/T_z = (v - v_0)/v'$

General Visual Motion

Let us assume there is one rigid object and the camera moves with velocities T and ω

For a given point P on the object, we have

$$p = P/z$$

From before, if P moves with velocity q we have

$$dp/dt = (z\ q - q_z\ P)/z^2 = 1/z\ (q - q_z\ p)$$

But q is

$$q = -T - \omega \times P$$

The Result:

$$\dot{u} = \frac{T_z u - T_x}{z} - w_y + w_z v + w_x uv - w_y u^2$$

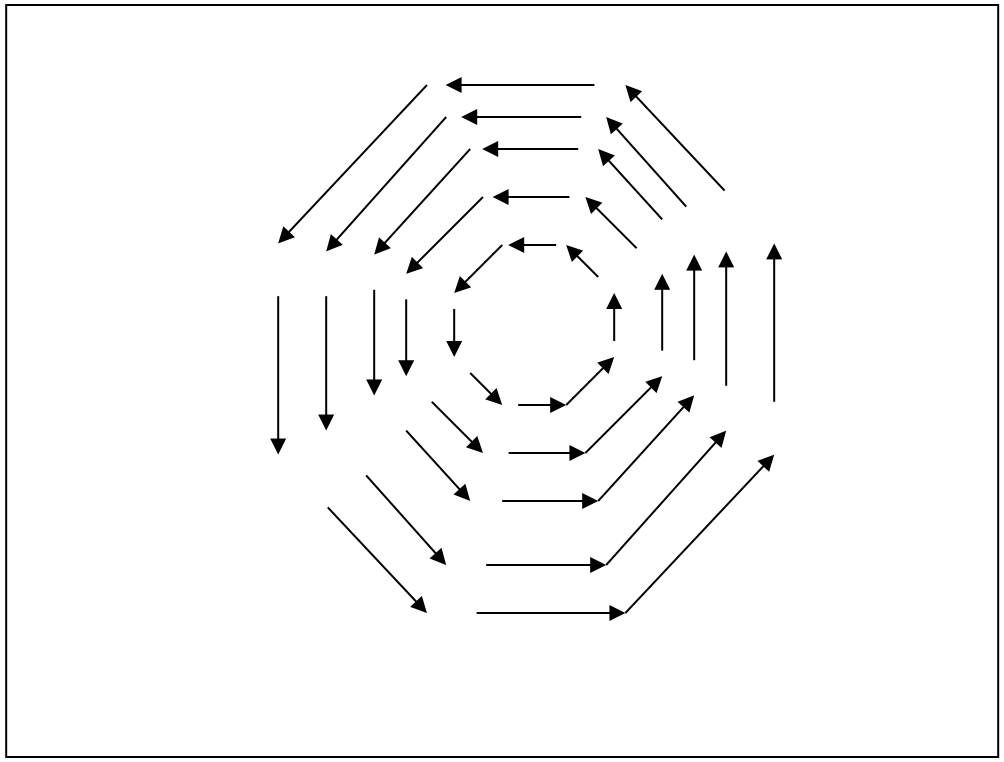
$$\dot{v} = \frac{T_z v - T_y}{z} + w_x - w_z u - w_y uv + w_x v^2$$

Motion due to translation: depends on depth

Motion due to rotation: independent of depth

THE MOTION FIELD

The "instantaneous" velocity of points in an image



PURE ROTATION

The Motion of a Planar Object

- $n^t P = d$ --- constraint on coordinates
  - $n_x X + n_y Y + n_z Z = d$
  - $n_x u + n_y v + n_z = d/Z$
  - $1/Z = (n_x u + n_y v + n_z)/d$

- Substitute back into optical flow equations

  $u' = a_1u^2 + a_2uv + a_3u + a_4v + a_5$
  $v' = a_1uv + a_2v^2 + a_7u + a_6v + a_8$

  where the a's depend on the motion and the plane parameters

Thus, planar motion can be computed *globally*

It is possible to show by a counting argument that the motion field of a plane is not unique --- there is more than one situation that gives rise to the same apparent motion

Motion Parallax: Perspective

- Consider two points on a rigid object that project to the same point instantaneously (think of a transparent plane upon which points are mounted)

- Note that in this case, the rotation fields are identical, hence taking a difference, we arrive at
  - $du' = (T_z u - T_x)(1/Z - 1/Z')$
  - $dv' = (T_z v - T_y)(1/Z - 1/Z')$

- These can be thought of as the relative motion field

- For example, we could think of motion as a background planar motion combined with a superimposed parallax ....

$v = v_p + dv$

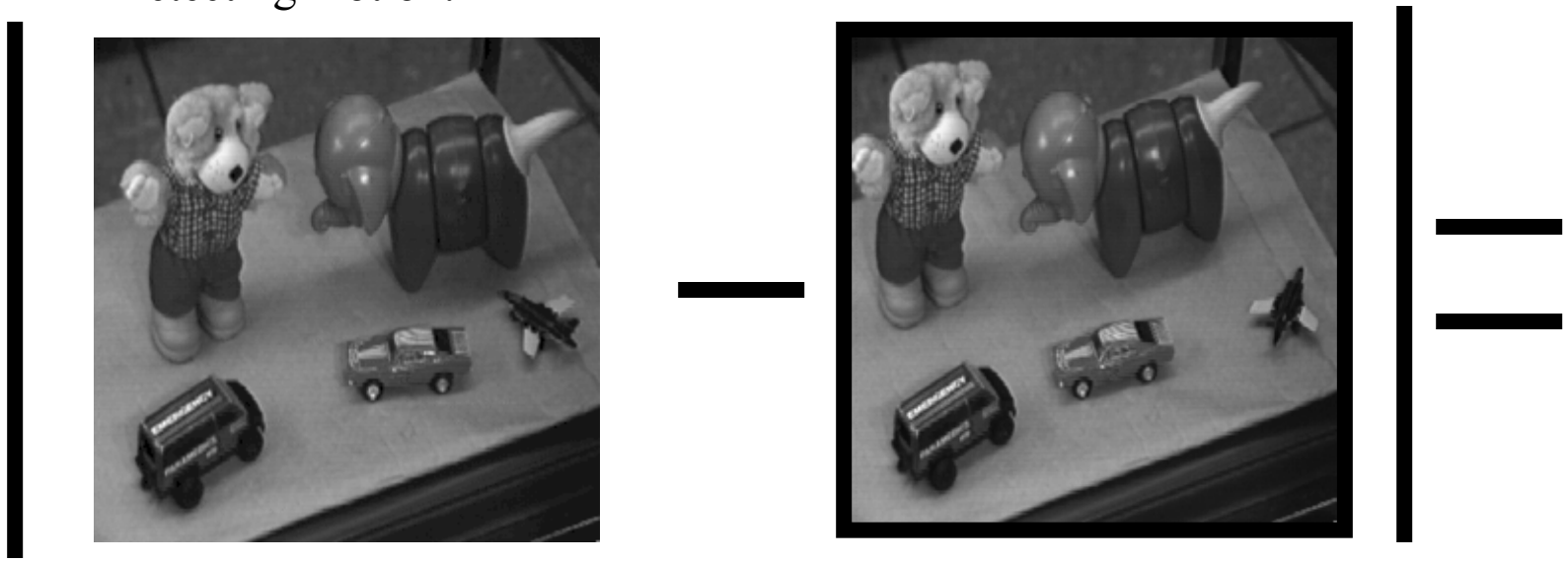
-- compute a planar motion

-- compute the effective planar motion field

-- compute dv and estimate depth from plane

MOTION IN REAL IMAGES

Detecting motion:



MOTION IN REAL IMAGES

Detecting motion:

$$\left| \text{  } - \text{  } \right| > 50$$

Candidate areas for motion

Computing Motion: Optical Flow

- Optical flow is the *apparent motion* in the image which, in some cases, corresponds to the motion field at that point.

- To compute it, think of image brightness as a function of time:
  - E(u(t),v(t),t)

- Let us assume that the image brightness of a point is constant. Then we have

  - $dE/dt = dE(x(t),y(t),t)/dt = E_x\, dx/dt + E_y\, dy/dt + E_t = 0$

  - equivalently, $rE\ \ v + E_t = 0$, where v is the velocity vector of an image pt.

- Note that locally, this implies that we can only compute motion perpendicular to the image gradient. This is commonly referred to as the aperture problem.
  - normal flow: $v_n = r\,E\,.v\ /\ ||\ r\,E|| = -\ E_t/||r\,E\ ||$

Computing Derivaties: An Aside

- The simplest way to compute the derivatives needed for optical flow is to think of a 2x2x2 cube (space x time) then we have

$$I_x(u,v,t) = [(I(u+1,v,t) - I(u,v,t)) + (I(u+1,v+1,t) - I(u,v+1,t)) + (I(u+1,v,t+1) - I(u,v,t+1)) + (I(u+1,v+1,t+1) - I(u,v+1,t+1))]/4$$

$$I_y(u,v,t) = [(I(u,v+1,t) - I(u,v,t)) + (I(u+1,v+1,t) - I(u+1,v,t)) + (I(u,v+1,t+1) - I(u,v,t+1)) + (I(u+1,v+1,t+1) - I(u+1,v,t+1))]/4$$

$$I_t(u,v,t) = [(I(u,v,t+1) - I(u,v,t)) + (I(u+1,v,t+1) - I(u+1,v,t)) + (I(u,v+1,t+1) - I(u,v+1,t)) + (I(u+1,v+1,t+1) - I(u+1,v+1,t))]/4$$

We can generalize to doing derivatives of Gaussians in space x time in the obvious way

Computing Optical Flow

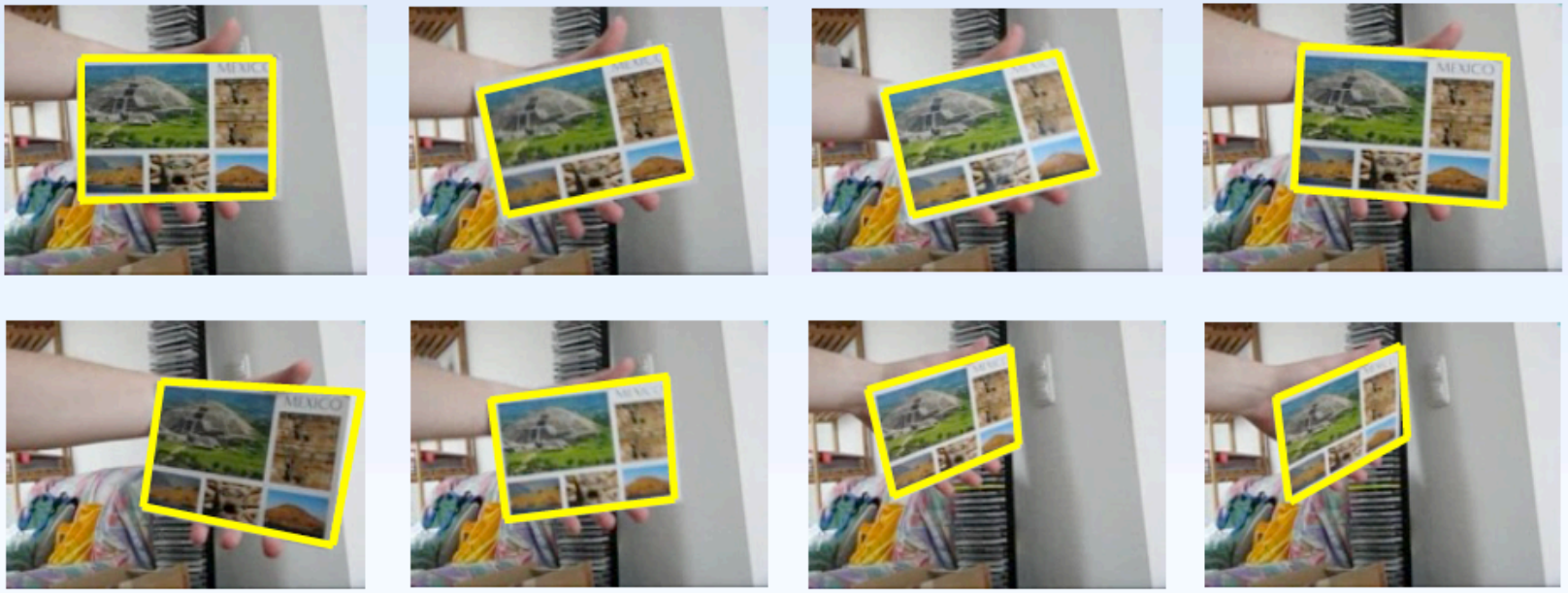
- Finite patch model:
  - $O(v(i)) = \Sigma_j (\nabla E(j) \cdot v(i) + E_t(j))^2$
  - $O(v(i)) = || A\, v(i) + b ||^2$

    where $A = [\nabla E(1); \nabla E(2); ... \nabla E(n)]$ $\quad b = [E_t(1); E_t(2); ... E_t(n)]$
    or $A = [vec(E_x), vec(E_y)]$ and $b = vec(E_t)$
  - $O'(v(i)) = A'(A\, v(i) + b) = 0 \rightarrow v(i) = -(A'A)^{-1} A'b$
- Algorithm:
  - Filter in u,v, and t using appropriate smoothing derivative
  - Solve previous equations for each point in image

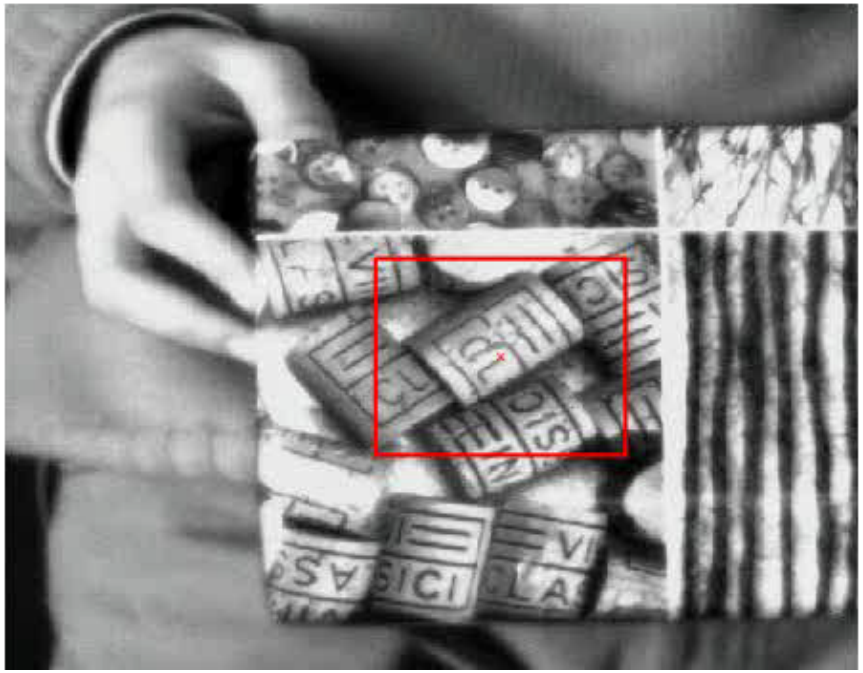
From Optical Flow to Tracking

template image

Follow a template image in an image sequence by estimating the warp.



Demo Videos



http://esm.gforge.inria.fr/

Template warping

Given the template T which we want to track:

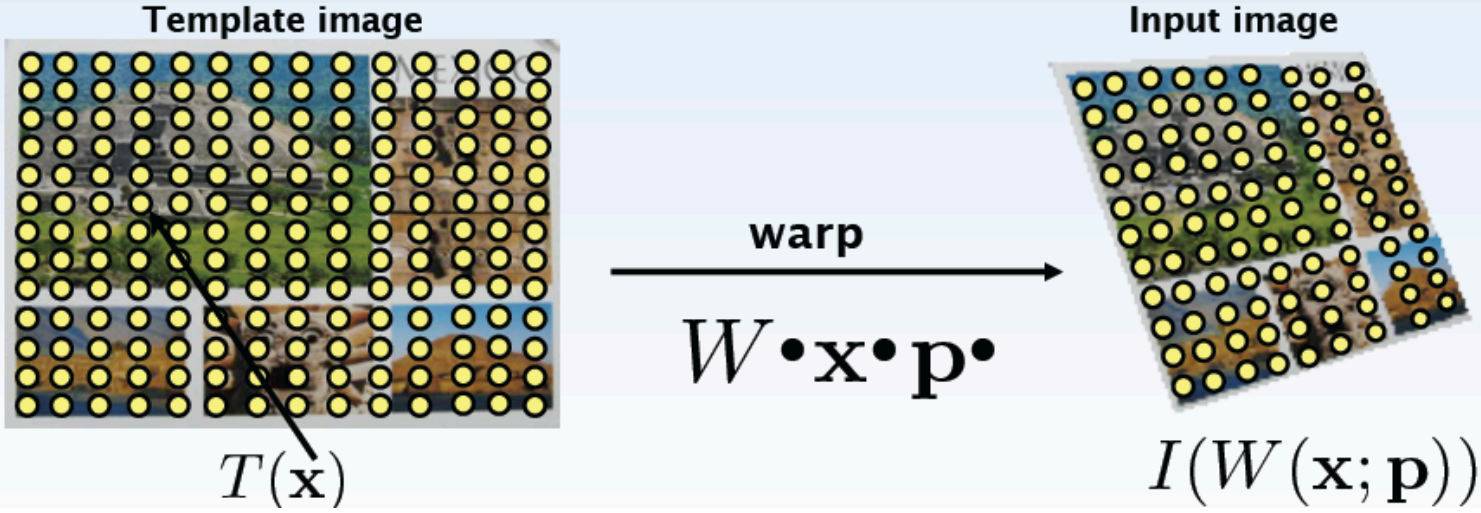
Template image

$T(\mathbf{x})$

Template warping

Given the template T which we want to track:

- Take all pixels $\mathbf{x}$ form the template and
- Warp them using the function $W(\mathbf{x};\mathbf{p})$ parameterized in terms of parameters $\mathbf{p}$ to the input image
- Assign the pixel value of the input image at the warped location to the template image



Tracking task

The **goal** of template-based tracking is to **find the parameters** $\mathbf{p}$ such that:

$$I(\mathbf{W}(\mathbf{x};\mathbf{p})) = T(\mathbf{x})$$

Using a prediction $\hat{\mathbf{p}}$ as an approximation of $\bar{\mathbf{p}}$ we can **reformulate the goal**:

- Find the **parameter increments** $\Delta\mathbf{p}$ such that:

$$\bar{\mathbf{p}} \leftarrow \hat{\mathbf{p}} + \Delta\mathbf{p}$$

- by **minimizing** the norm of:

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})) - T(\mathbf{x}) \,]^2 \qquad (1)$$

This is a **non-linear minimization problem**.

Assumptions

- **No errors in the template image boundaries**:
  only the appearance of the object to be tracked appears in the template image.

- **No occlusion**: the entire template is visible in the input image.

- **Brightness constancy assumption**: the intensity of the object appearance is always the same.

Lucas-Kanade approach

Uses the **Gauss-Newton method** for minimization:

- Applies a first-order approximation
- Minimizes iteratively

The warp has to be differentiable.

From Optical Flow to Tracking

- Now, let's introduce a warping operator
  - Simple translation: $R(t; d)(u,v) = I(u+d_u, v+d_v, t)$
  - Note that d is not guaranteed to be integral!
    - Typically sample using bilinear interpolation (look at the interp2 function in matlab).
    - Typically we end up warping **backwards** rather than forwards
    - Define a warped region by $R(t; d) = I(u+d_x, v+d_y, t)$ u in [-w,w]; v in [-h,h]
  - Now, we can do a simple iteration to define $Oflow(d, \Delta d, t_1, t_2)$
    1. While ($|| vec(R(d+\Delta d, t_2)) - vec(R(d,t_1)) || > \varepsilon$)
       1. Compute E_x, E_y, and E_t from $R(d+\Delta d, t_2)$ and $R(d,t_1)$
       2. set $\Delta d = \Delta d - [vec(E_x), vec(E_y)] \setminus vec(E_t)$
    2. Return Δd

More Tracking

- Now, we can compute multi-frame flow (tracking) in two ways:

- Incremental integration
  - $d_{t+1} = d_t + \text{Oflow}(d_t, 0, t, t+1)$
    - this computes interframe motion for a given patch
    - note that any error in Oflow is propagated to the next iteration
    - we could also predict the offset to the next frame for better convergence

- Reference tracking
  - $d_{t+1} = d_0 + \text{Oflow}(d_0, d_t - d_0, t_0, t+1)$
    - this always references images back to the same reference image
    - not subject to slippage, but calls the image constancy assumption more into question

- Note that in both cases, it makes sense to first subtract the mean from both images and possibly divide by variance
  - interesting exercise: can you include these photometric components in the calculation?

Remarks

- When is A'A full rank?
  - when spatial gradient spans R^2
  - SVD of A gives a measure of confidence
  - this is a good heuristic for finding "good features to track"

- How fast can we track?
  - derivatives are good for 1/2 pixel
  - smoothing correlates pixels and increases attraction
  - subsampling decreases # of pixels and increases effective pixel size
    → usually Oflow is done in a heirarchical computation.

A more elaborate example

Tracking techniques

- Direct methods (e.g. template tracking)
  - Directly recover image motion at each pixel from spatio-temporal image brightness variations
  - Dense motion fields, but sensitive to appearance variations

- Feature-based methods
  - Extract visual features (corners, textured areas, color blobs) and track them over multiple frames
  - Allows for more “customized” choice of methods

- Kernel-based methods
  - Generalization of “voting” to continuous domain
  - Maximize

$$\sum K(x-c)F(x) = \sum k(\| x-c \|^2)F(x)$$

  - where K is a kernel (e.g. a zero mean Gaussian) and F is some feature measure (e.g. histogram similarity) at location x

Kalman filter

(Adapted from slides by Kristen Grauman)

Detection vs. tracking



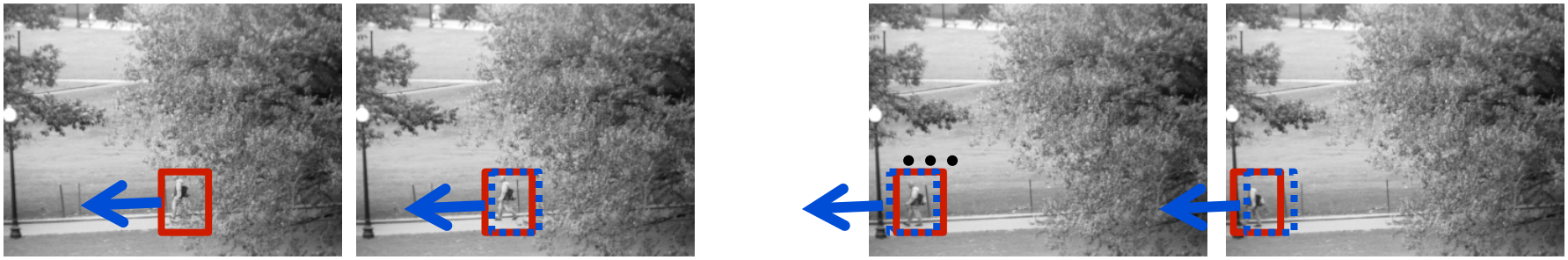

t=1 t=2 t=20 t=21

Detection vs. tracking

Detection: We detect the object independently in each frame and can record its position over time, e.g., based on blob's centroid or detection window coordinates

Kristen

Detection vs. tracking



Tracking with *dynamics*: We use image measurements to estimate position of object, but also incorporate position predicted by dynamics, i.e., our expectation of object's motion pattern.

Kristen

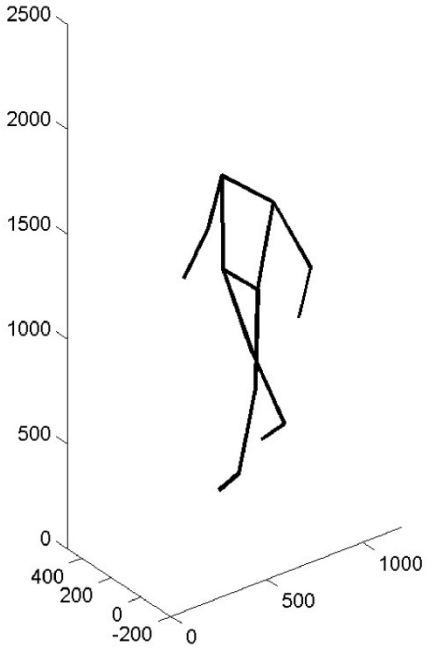
Tracking with dynamics

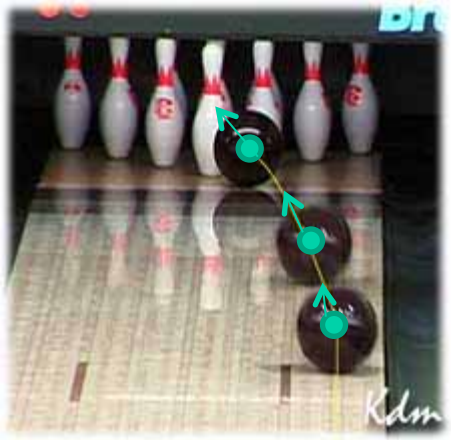
- Use model of expected motion to *predict* where objects will occur in next frame, even before seeing the image.
- **Intent**:
  - Do less work looking for the object, restrict the search.
  - Get improved estimates since measurement noise is tempered by smoothness, dynamics priors.
- **Assumption**: continuous motion patterns:
  - Camera is not moving instantly to new viewpoint
  - Objects do not disappear and reappear in different places in the scene
  - Gradual change in pose between camera and scene

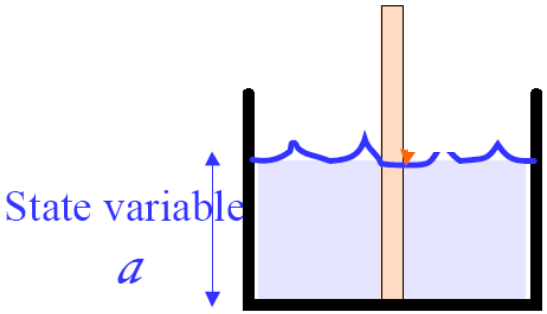
Tracking as inference

- The *hidden state* consists of the true parameters we care about, denoted *X*.
- The *measurement* is our noisy observation that results from the underlying state, denoted *Y*.
- At each time step, state changes (from X_{t-1} to X_t) and we get a new observation Y_t.

State vs. observation







- Hidden state : parameters of interest
- Measurement : what we get to directly observe

Tracking as inference

- The *hidden state* consists of the true parameters we care about, denoted *X*.
- The *measurement* is our noisy observation that results from the underlying state, denoted *Y*.
- At each time step, state changes (from X_{t-1} to X_t) and we get a new observation Y_t.
- Our goal: recover most likely state X_t given
  - All observations seen so far.
  - Knowledge about dynamics of state transitions.

Tracking as inference: intuition

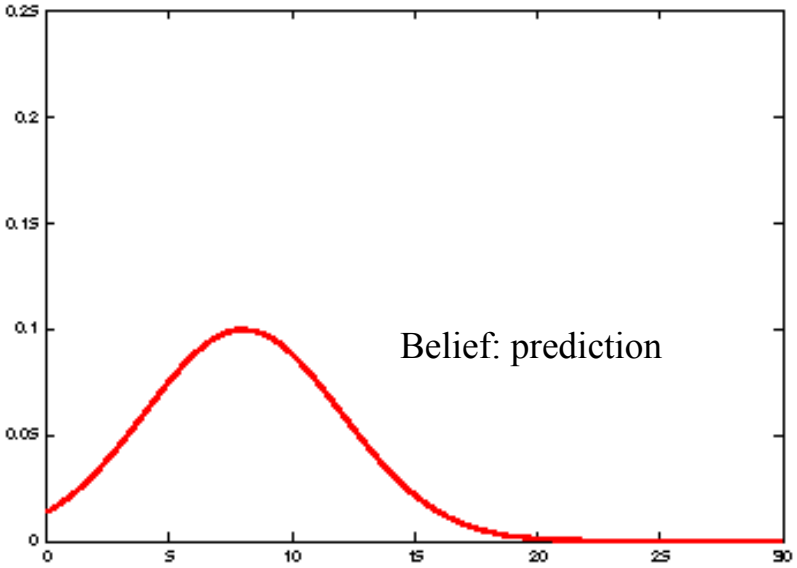
Time t Time t+1

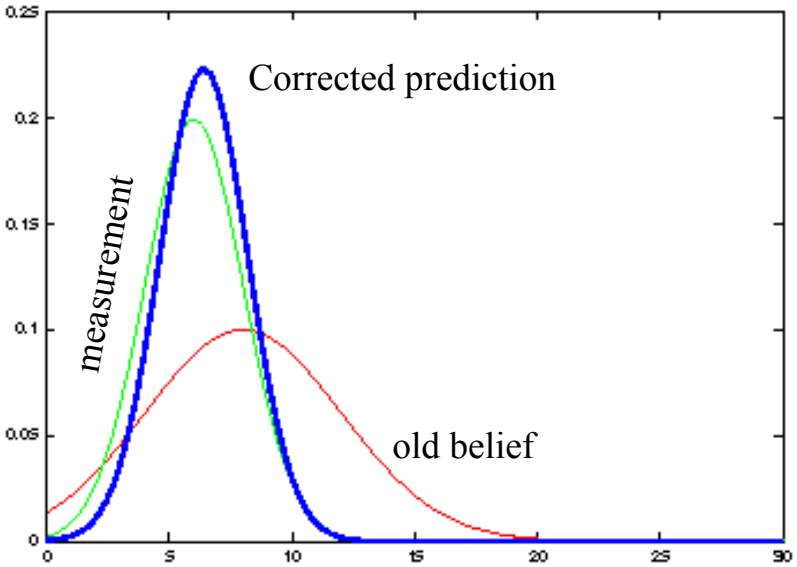
Belief

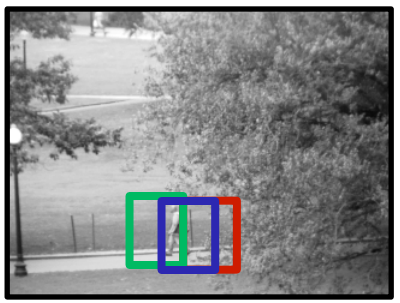
Measurement

Corrected prediction

Tracking as inference: intuition







Time t Time t+1

Kristen

Questions

- How to represent the known dynamics that govern the changes in the states?
- How to represent relationship between state and measurements, plus our uncertainty in the measurements?
- How to compute each cycle of updates?

Representation: We'll consider the class of *linear* dynamic models, with associated Gaussian pdfs.

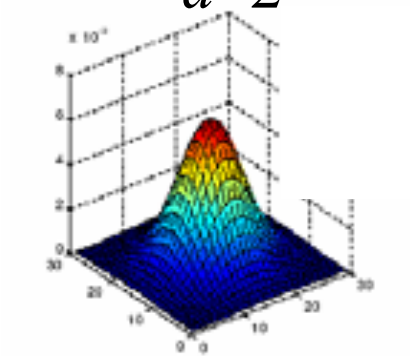
Updates: via the Kalman filter.

Notation reminder

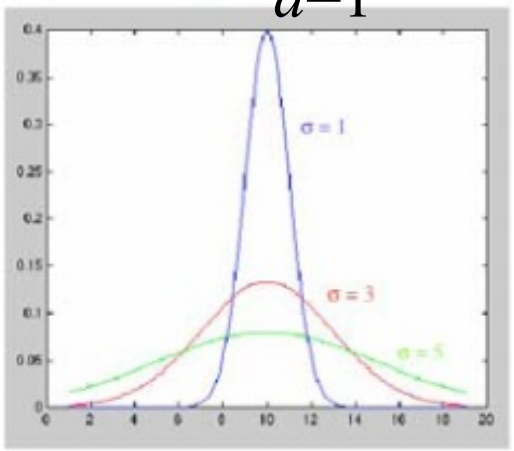
$$\mathbf{x} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

- Random variable with Gaussian probability distribution that has the mean vector **μ** and covariance matrix **Σ**.
- **x** and **μ** are *d*-dimensional, **Σ** is *d* x *d*.

d=2



d=1



If x is 1-d, we just have one **Σ** parameter - → the variance: σ^2

Linear dynamic model

- Describe the *a priori* knowledge about
  - System dynamics model: represents evolution of state over time.

$$\mathbf{x}_t \sim N(\mathbf{D}\mathbf{x}_{t-1}; \boldsymbol{\Sigma}_d)$$

n x 1 n x n n x 1

  - Measurement model: at every time step we get a noisy measurement of the state.

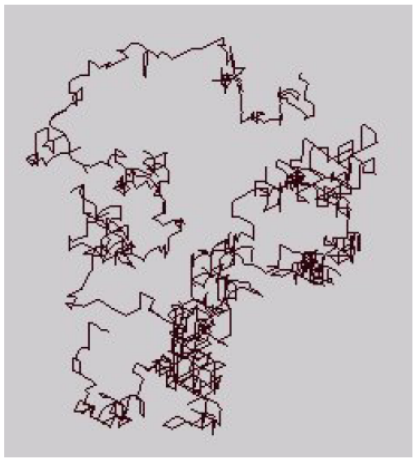
$$\mathbf{y}_t \sim N(\mathbf{M}\mathbf{x}_t; \boldsymbol{\Sigma}_m)$$

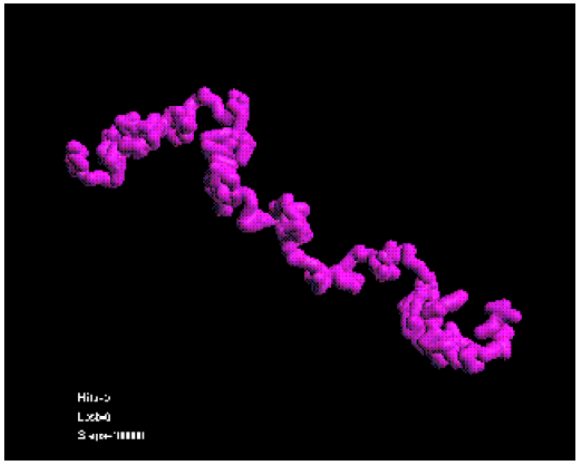
m x 1 m x n n x 1

Example: randomly drifting points

$$\mathbf{x}_t \sim N(\mathbf{D}\mathbf{x}_{t-1}; \boldsymbol{\Sigma}_d)$$

- Consider a stationary object, with state as position
- Position is constant, only motion due to random noise term.
- State evolution is described by identity matrix **D=I**

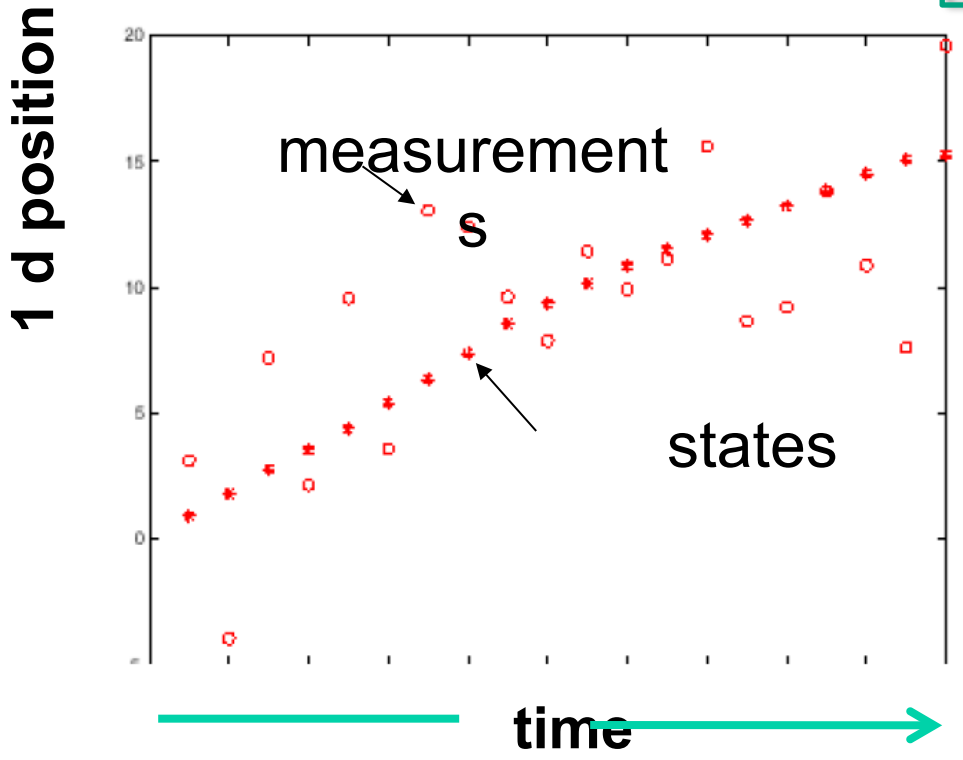




cic.nist.gov/lipman/sciviz/images/random3.gif
http://www.grunch.net/synergetics/images/rand
3.jpg

Example: Constant velocity (1D points)

1 d positior



Example: Constant velocity (1D points)

$$\mathbf{x}_t \sim N(\mathbf{D}\mathbf{x}_{t-1}; \boldsymbol{\Sigma}_d)$$
$$\mathbf{y}_t \sim N(\mathbf{M}\mathbf{x}_t; \boldsymbol{\Sigma}_m)$$

- State vector: position *p* and velocity *v*

$$x_t = \begin{bmatrix} p_t \\ v_t \end{bmatrix} \qquad \begin{aligned} p_t &= p_{t-1} + (\Delta t) v_{t-1} + \varepsilon \\ v_t &= v_{t-1} + \xi \end{aligned}$$

$$x_t = D_t x_{t-1} + noise = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{t-1} \\ v_{t-1} \end{bmatrix} + noise$$

- Measurement is position only

$$y_t = M x_t + noise = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} p_t \\ v_t \end{bmatrix} + noise$$

Questions

- How to represent the known dynamics that govern the changes in the states?
- How to represent relationship between state and measurements, plus our uncertainty in the measurements?
- How to compute each cycle of updates?

Representation: We'll consider the class of *linear* dynamic models, with associated Gaussian pdfs.

Updates: via the Kalman filter.

Kristen

The Kalman filter

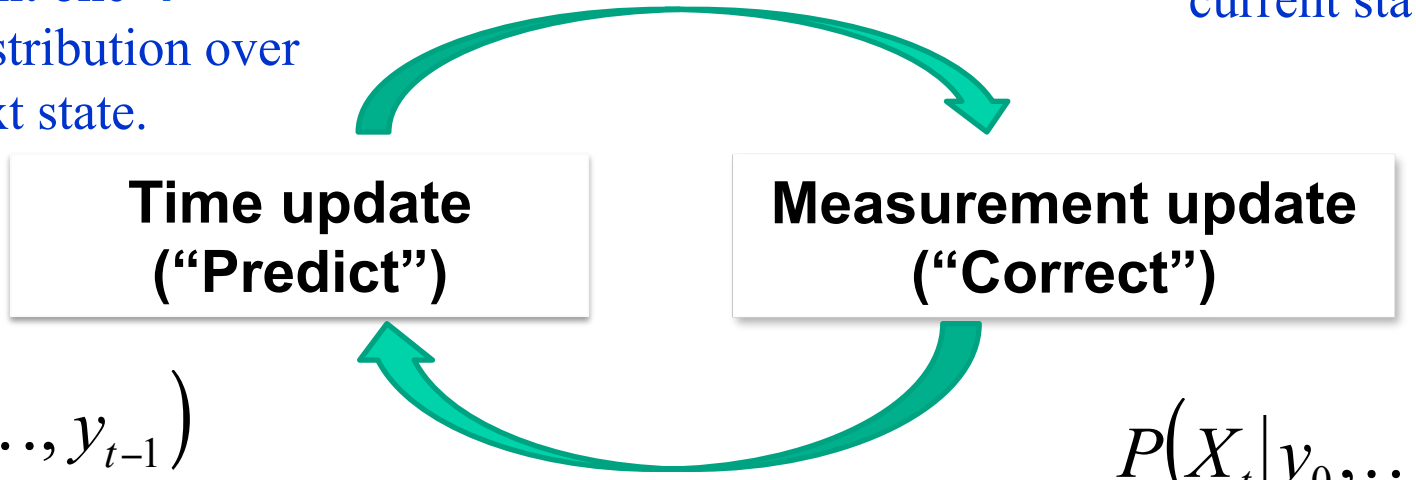
- Method for tracking linear dynamical models in Gaussian noise
- The predicted/corrected state distributions are Gaussian
  - Only need to maintain the mean and covariance
  - The calculations are easy

Kalman filter

Know corrected state from previous time step, and all measurements up to the current one →
Predict distribution over next state.

Receive measurement

Know prediction of state, and next measurement →
Update distribution over current state.



$P(X_t | y_0, \ldots, y_{t-1})$

Mean and std. dev. of predicted state:

Time advances: t++

$P(X_t | y_0, \ldots, y_t)$

Mean and std. dev. of corrected state:

μ_t^+, σ_t^+

Kristen

1D Kalman filter: **Prediction**

- Have linear dynamic model defining predicted state evolution, with noise
- $$X_t \sim N\left(dx_{t-1}, \sigma_d^2\right)$$
- Want to estimate predicted distribution for next state

$$P\left(X_t \middle| y_0, \ldots, y_{t-1}\right) = N\left(\mu_t^-, (\sigma_t^-)^2\right)$$

- Update the mean:

$$\mu_t^- = d\mu_{t-1}^+$$

- Update the variance:

$$(\sigma_t^-)^2 = \sigma_d^2 + (d\sigma_{t-1}^+)^2$$

1D Kalman filter: **Correction**

- Have linear model defining the mapping of state to measurements:

$$Y_t \sim N\left(mx_t, \sigma_m^2\right)$$

- Want to estimate corrected distribution given latest meas.:

$$P\left(X_t \middle| y_0, \ldots, y_t\right) = N\left(\mu_t^+, (\sigma_t^+)^2\right)$$

- Update the mean:

$$\mu_t^+ = \frac{\mu_t^- \sigma_m^2 + m y_t (\sigma_t^-)^2}{\sigma_m^2 + m^2 (\sigma_t^-)^2}$$

- Update the variance:

$$(\sigma_t^+)^2 = \frac{\sigma_m^2 (\sigma_t^-)^2}{\sigma_m^2 + m^2 (\sigma_t^-)^2}$$

L

Prediction vs. correction

$$\mu_t^+ = \frac{\mu_t^- \sigma_m^2 + m y_t (\sigma_t^-)^2}{\sigma_m^2 + m^2 (\sigma_t^-)^2} \qquad (\sigma_t^+)^2 = \frac{\sigma_m^2 (\sigma_t^-)^2}{\sigma_m^2 + m^2 (\sigma_t^-)^2}$$

- What if there is no prediction uncertainty $(\sigma_t^- = 0)$?
- $$\mu_t^+ = \mu_t^- \qquad (\sigma_t^+)^2 = 0$$

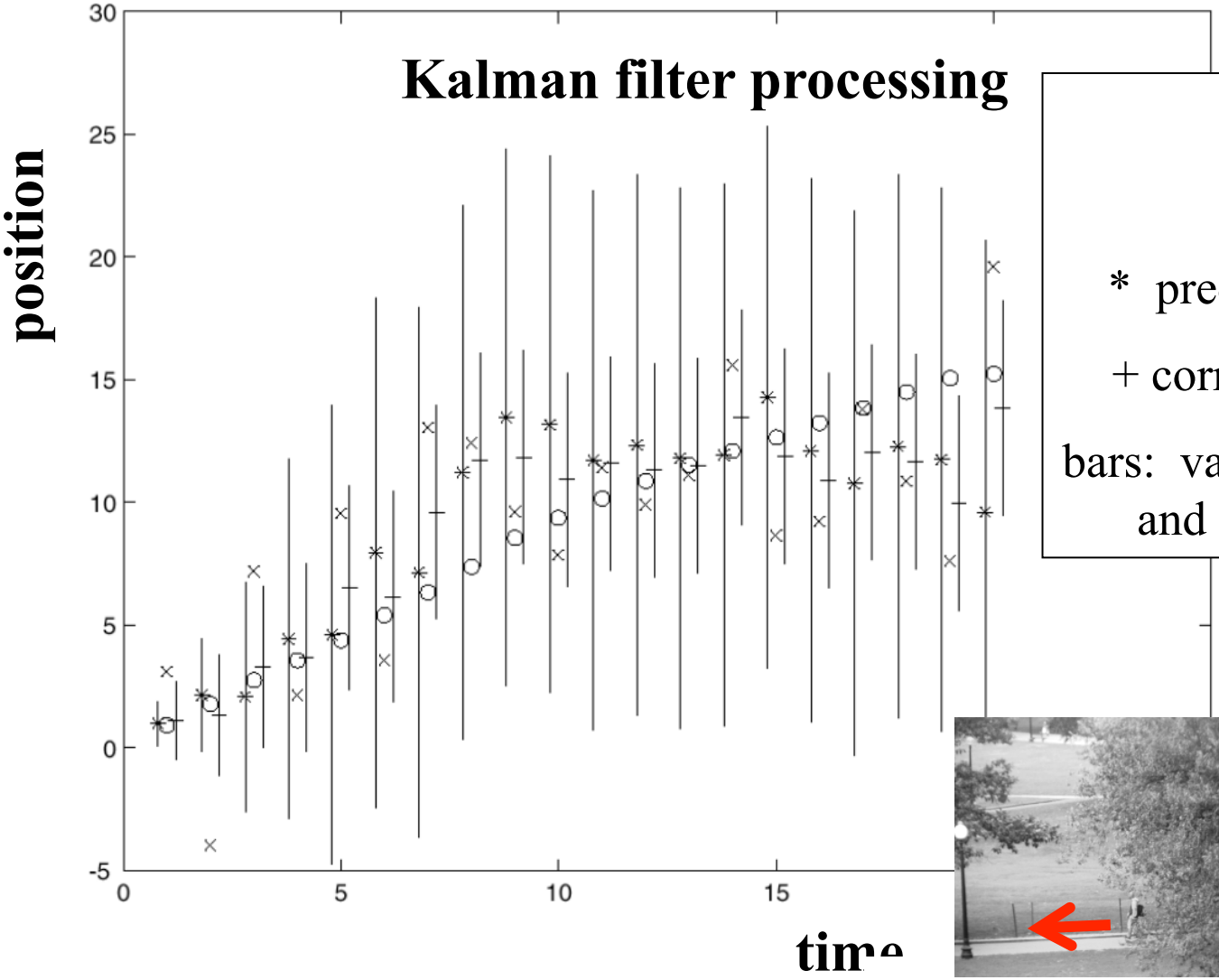
The measurement is ignored!

- What if there is no measurement uncertainty $(\sigma_m = 0)$?

$$\mu_t^+ = \frac{y_t}{m} \qquad (\sigma_t^+)^2 = 0$$

The prediction is ignored!

Constant velocity model



o state

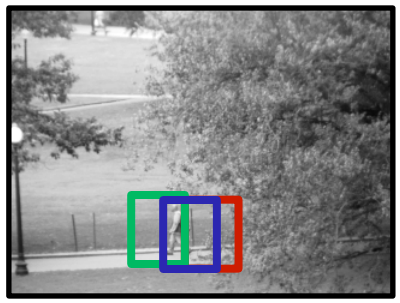
x measurement

* predicted mean estimate

\+ corrected mean estimate

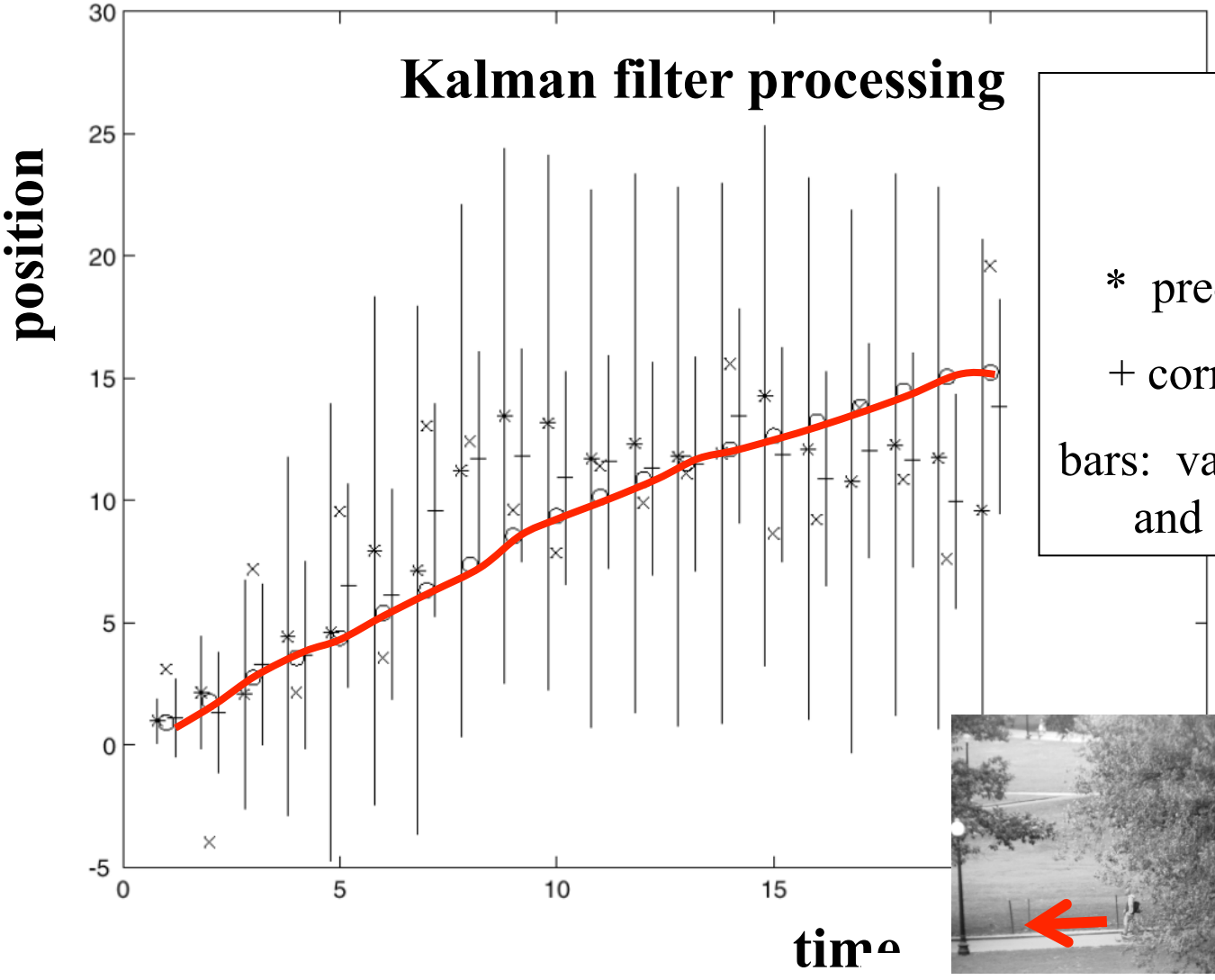
bars: variance estimates before and after measurements

Time t



Time t+1

Constant velocity model



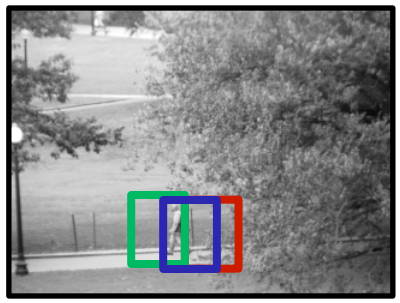
o state

x measurement

* predicted mean estimate

\+ corrected mean estimate

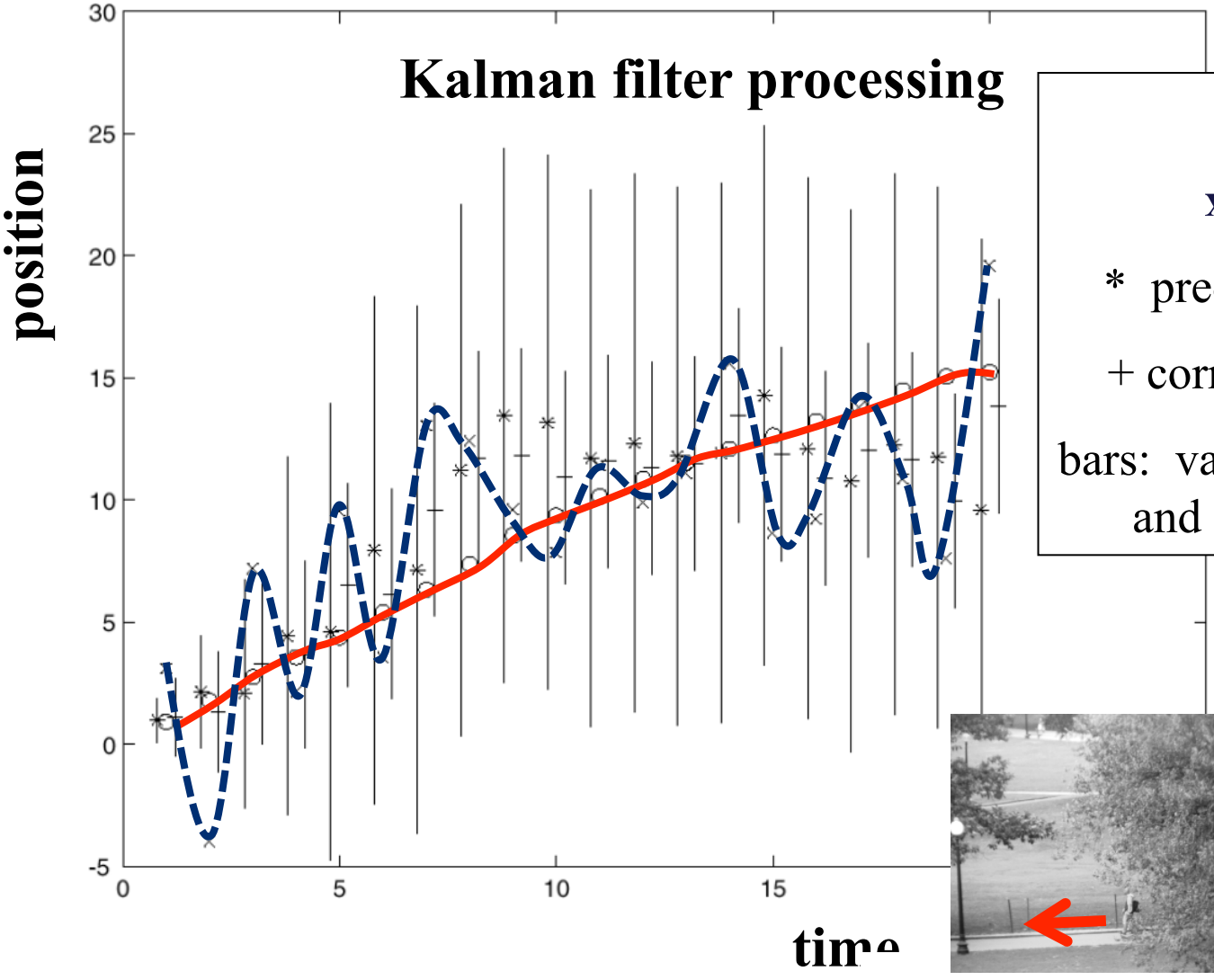
bars: variance estimates before and after measurements



Time t

Time t+1

Constant velocity model



o state

x measurement

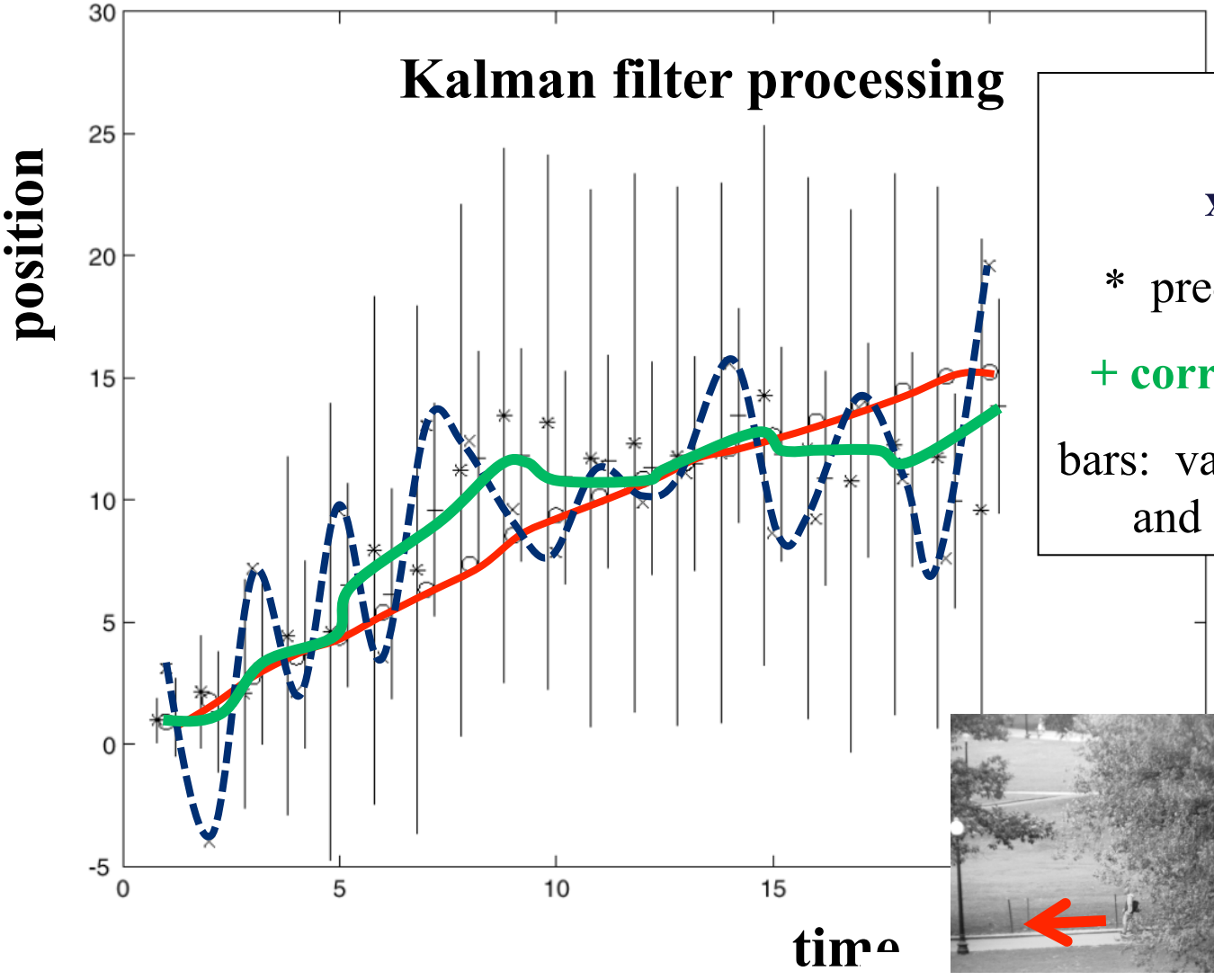
* predicted mean estimate

\+ corrected mean estimate

bars: variance estimates before and after measurements

Time t Time t+1

Constant velocity model



o state

x measurement

* predicted mean estimate

+ corrected mean estimate

bars: variance estimates before and after measurements

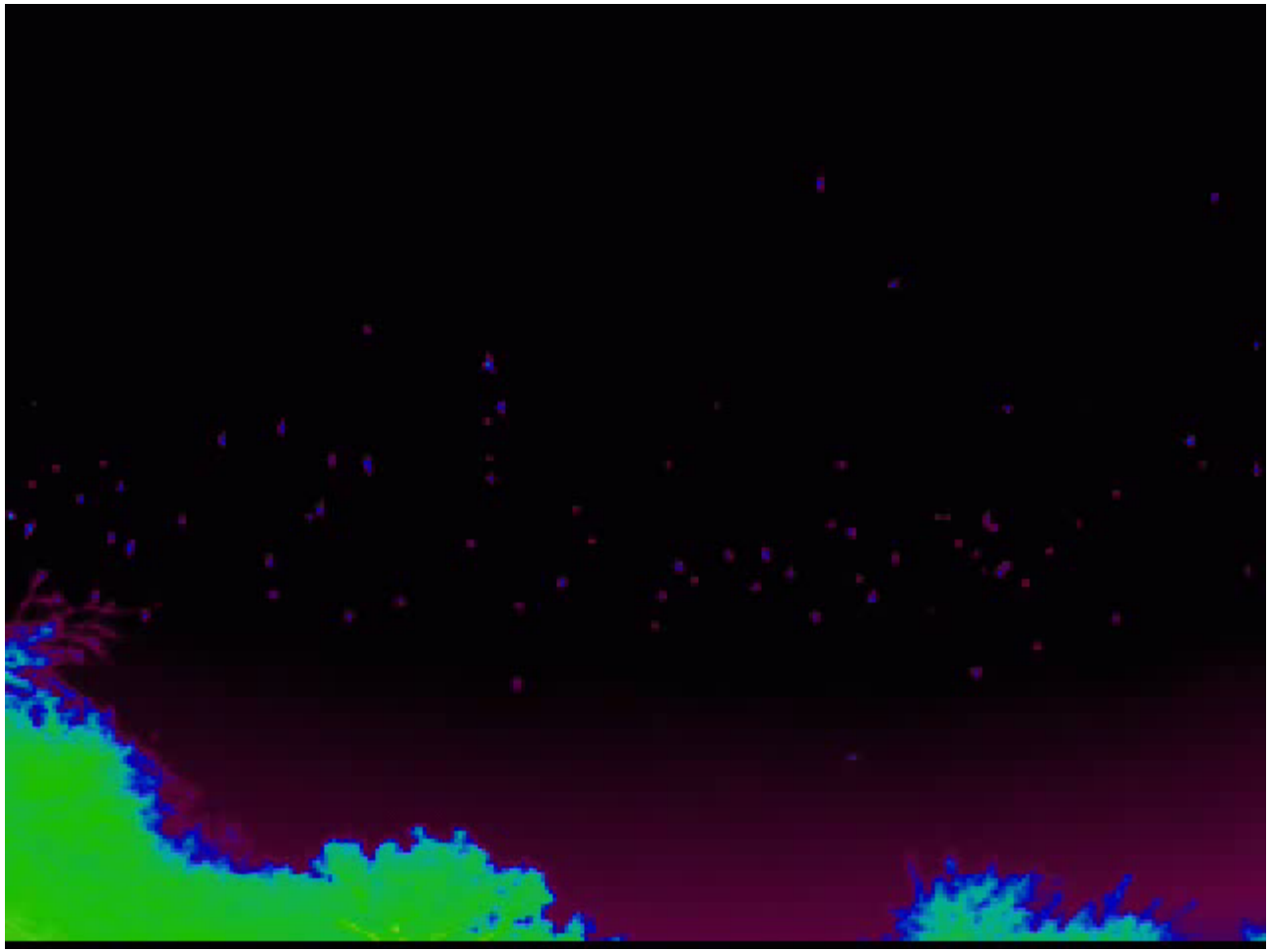
Time t Time t+1

Summary

- Tracking as inference
  - Goal: estimate posterior of object position given measurement
- Linear models of dynamics
  - Represent state evolution and measurement models
- Kalman filters
  - Recursive prediction/correction updates to refine measurement

Tracking: applications, challenges

A bat census



http://www.cs.bu.edu/~betke/research/bats/

Video synopsis

- http://www.vision.huji.ac.il/video-synopsis/

Tracking: issues

- **Initialization**
  - Often done manually
  - Background subtraction, detection can also be used
- **Data association**, multiple tracked objects
  - Occlusions, clutter

Tracking: issues

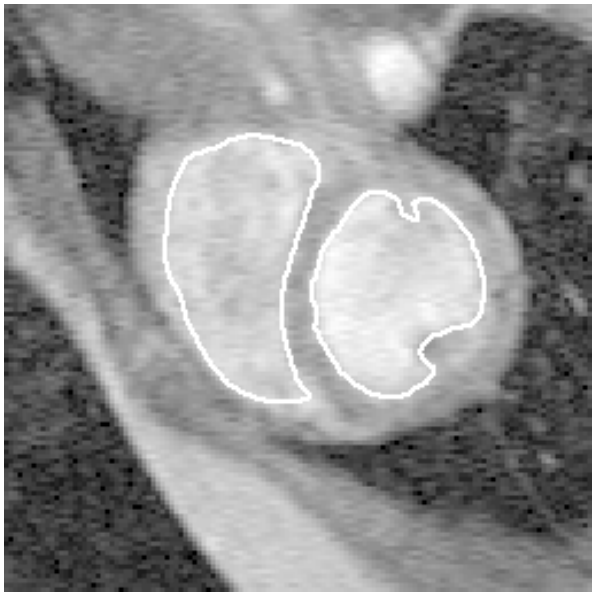
- **Initialization**
  - Often done manually
  - Background subtraction, detection can also be used
- **Data association**, multiple tracked objects
  - Occlusions, clutter
  - Which measurements go with which tracks?

Tracking: issues

- **Initialization**
  - Often done manually
  - Background subtraction, detection can also be used
- **Data association**, multiple tracked objects
  - Occlusions, clutter
- **Deformable** and articulated objects

Tracking via deformable contours

1. Use contour/model extracted at frame t as an initial solution for frame $t+1$
2. Evolve initial contour to fit exact object boundary at frame $t+1$
3. Repeat, initializing with most recent frame.

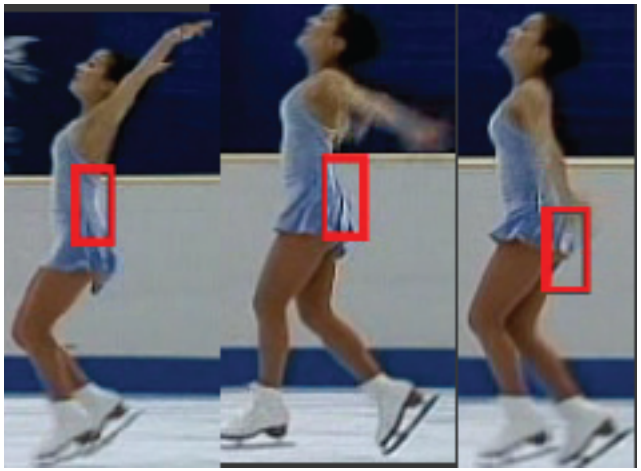


Visual Dynamics Group, Dept. Engineering Science, University of Oxford.

Tracking: issues

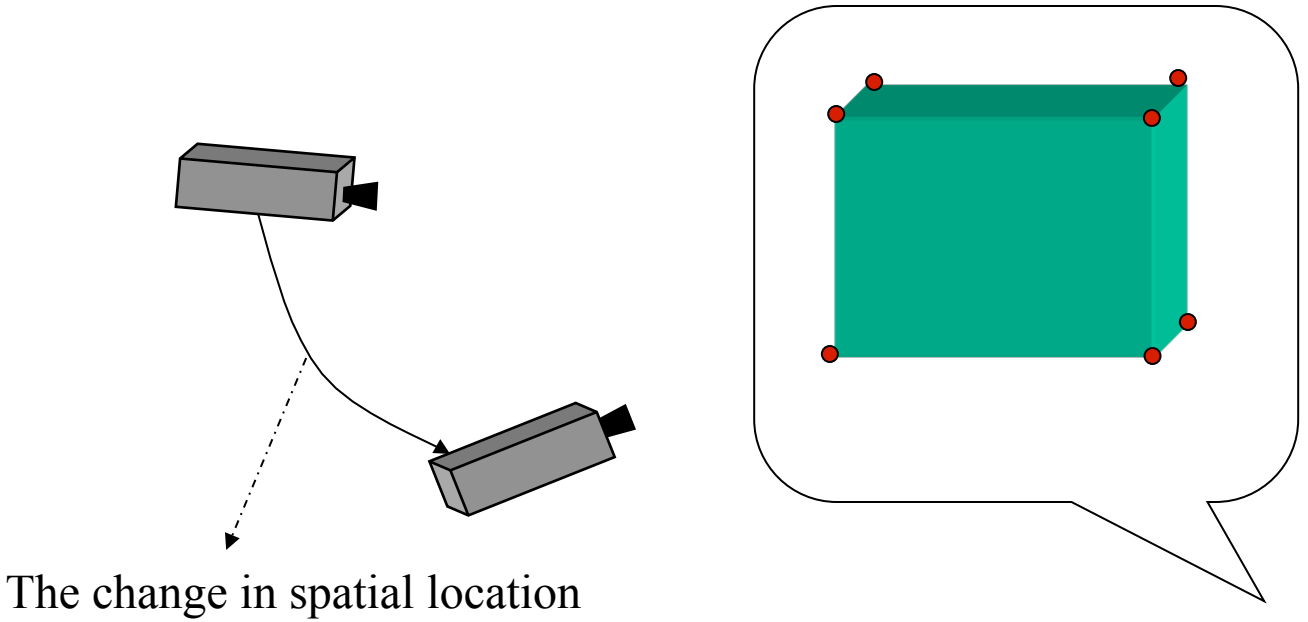
- **Initialization**
  - Often done manually
  - Background subtraction, detection can also be used
- **Data association**, multiple tracked objects
  - Occlusions, clutter
- **Deformable** and articulated objects
- **Constructing accurate models** of dynamics
  - E.g., Fitting parameters for a linear dynamics model
- **Drift**
  - Accumulation of errors over time

Drift



D. Ramanan, D. Forsyth, and A. Zisserman.
Tracking People by Learning their Appearance. PAMI 2007.

MOVING CAMERAS ARE LIKE STEREO



The change in spatial location
between the two cameras (the "motion")

Locations of
points on the object
(the "structure")

Now we consider continuous change over time

On to the Factorization Method ...

- Assumptions:
  - orthographic camera
  - n non-coplanar points tracked in N >= 3 frames

- Form the registered measurement matrix W* = [X*;Y*]
  - where $x^*_{i,j} = x_{i,j} - mx_i$
  - where $y^*_{i,j} = y_{i,j} - my_i$
  - mx and my are the means of the points in frame i
  - j ranges over points in a frame

- The rank theorem: The registered measurement matrix has rank at most 3

- By showing the rank theorem, we will also develop an algorithm for computing structure and motion.

The Rank Theorem

- First, formulate orthographic projection for points
  - $p_{ij} = O\ R_i\ (P_j - T_i)$ [we'll omit scaling which doesn't matter]
  - Choose origin at centroid of points
  - Think of first two rows of R as x and y axis of camera frame

- Note by subtracting $pm_j = \sum_j p_{i,j}/n$ from $p_{i,j}$ we have
  - $p'_{i,j} = O\ R_i(P_j - Pm)$, but Pm is at the origin so we have
  - $x'_{i,j} = \mathbf{x}_i\ P_j$
  - $y'_{i,j} = \mathbf{y}_i\ P_j$

- In this case, we can now create a 2N x 3 rotation matrix R from the i's and j's and a 3 x n shape matrix S from the points, and
  - W = R S.
  - It follows that W has rank 3 provide N >= 3 and the N points are not coplanar

- SVD: $W = U\ D\ V^t$ --- drop all but top 3 singular values and factor this into A and P: $A = U\ D^{1/2}$; $S = D^{1/2}\ V^t$

Orthogonalization

- A is of course not orthogonal;however for each position we can pull two rows (the i and j). Let $E_i = [i_i;j_i]$; and let A_i be the two corresponding rows of A; then
  - $E_i = A_i Q$
- This implies that
  - $E_i E_i' = A_i Q Q' A_i' = A_i M A_i' = I_{2x2}$
- Note that M appears linearly, so we can also write
  - $F(A_i)\ vec(M) = vec(I)$
  - We can stack this into a linear system of the form F vec(M) = b and solve for M (note M must be symmetric so we really only have 6 free values)
  - By eigenvalue decomposition $M = R D R' \dashrightarrow Q = R D^{1/2}$

The Algorithm

0. Compute the registered measurement matrix W* for N frames and n points
1. Compute the SVD of W*; $W^* = U D V^t$
   - U is 2N x 2N
   - V is n x n
   - D is 2N x n
2. Zero all but the first 3 singular values in D
   - D' is upper 3x3 of D
   - U' is corresponding 2Nx3 matrix from U
   - V' is corresponding nx3 from V
3. Define
   - $A = U' D'^{1/2}$; $S = D'^{1/2} V'^t$
4. Solve for Q s.t. Q orthogonalizes A
5. $R^* = A Q$ and $S^* = Q^{-1} S$

The Results

General Perspective and Motion

- There are iterative methods for differential motion (see book); we will not cover these.
  - In general, any motion and structure method is extremely sensitive for small motion (i.e. in the optical flow case).

- There are extensions of factorization to the perspective case; the method (see Ponce and Forsyth)

- For large motions, E-matrix computation and stereo-like methods are reasonable solutions to get dense estimates of depth

- Motion segmentation (multiple motions) is an important problem. GPCA-like methods have recently been developed (Vidal, Ma) as a way of describing the generalized epipolar constraints that arise in this case.

Perspective Motion Factorization

(Courtesy Marc Pollefeys)

Summary

- Motion field and its structure
  - translational motion (depends on depth)
  - rotational motion (quadratic on u/v)
- Optical flow vs. motion field
- Recovering optical flow and tracking
- Structure and motion from multiple views
- Using changes in illumination