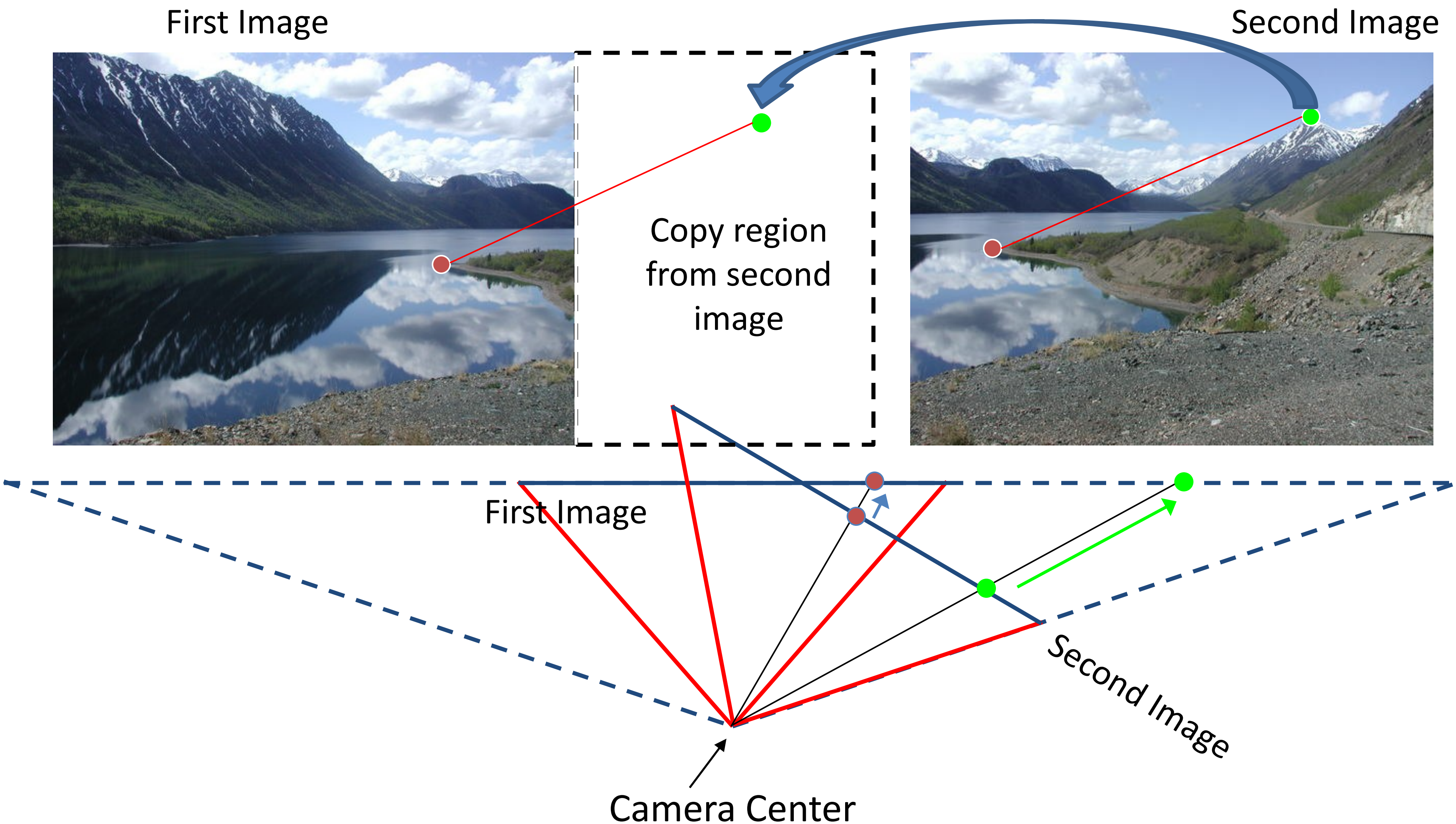
Homography Computation

Homography transformation

- An invertible transformation from one projective plane to the other that maps straight lines to straight lines.
- Any two images of the same planar surface in space are related by a homography transformation.

Views from rotating camera



Algorithm Overview

1. Detect keypoints: SIFT keypoint descriptor
2. Match keypoints: Matching is done through a Euclidean-distance based nearest neighbor approach. To increase robustness, matches are rejected for those keypoints for which the ratio of the nearest neighbor distance to the second nearest neighbor distance is greater than 0.8
3. Estimate homography with four matched keypoints (using RANSAC)
4. Project onto a surface

Computing homography

Assume we have four matched points: How do we compute homography **H**?

Direct Linear Transformation (DLT)

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \qquad \mathbf{x}' = \begin{bmatrix} w'u' \\ w'v' \\ w' \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

$$\begin{bmatrix} -u & -v & -1 & 0 & 0 & 0 & uu' & vu' & u' \\ 0 & 0 & 0 & -u & -v & -1 & uv' & vv' & v' \end{bmatrix} \mathbf{h} = \mathbf{0} \qquad \mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix}$$

Computing homography

Direct Linear Transform

$$\begin{bmatrix} -u_1 & -v_1 & -1 & 0 & 0 & 0 & u_1u_1' & v_1u_1' & u_1' \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & u_1v_1' & v_1v_1' & v_1' \\ & & & & \vdots & & & & \\ 0 & 0 & 0 & -u_n & -v_n & -1 & u_nv_n' & v_nv_n' & v_n' \end{bmatrix}\mathbf{h} = \mathbf{0} \Rightarrow \mathbf{A}\mathbf{h} = \mathbf{0}$$

- Apply SVD: $\boldsymbol{UDV}^T = \boldsymbol{A}$
- $\boldsymbol{h} = \boldsymbol{V}_{\text{smallest}}$ (column of $\boldsymbol{V}$ corr. to smallest singular value)

$$\mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix} \Rightarrow \mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

RANSAC: RANdom SAmple Consensus

Scenario: We've got way more matched points than needed to fit the parameters, but we're not sure which are correct

RANSAC Algorithm

- Repeat N times
  1. Randomly select a sample
     - Select just enough points to recover the parameters
  2. Fit the model with random sample
  3. See how many other points agree
- Best estimate is one with most agreement
  - can use agreeing points to refine estimate

Computing homography

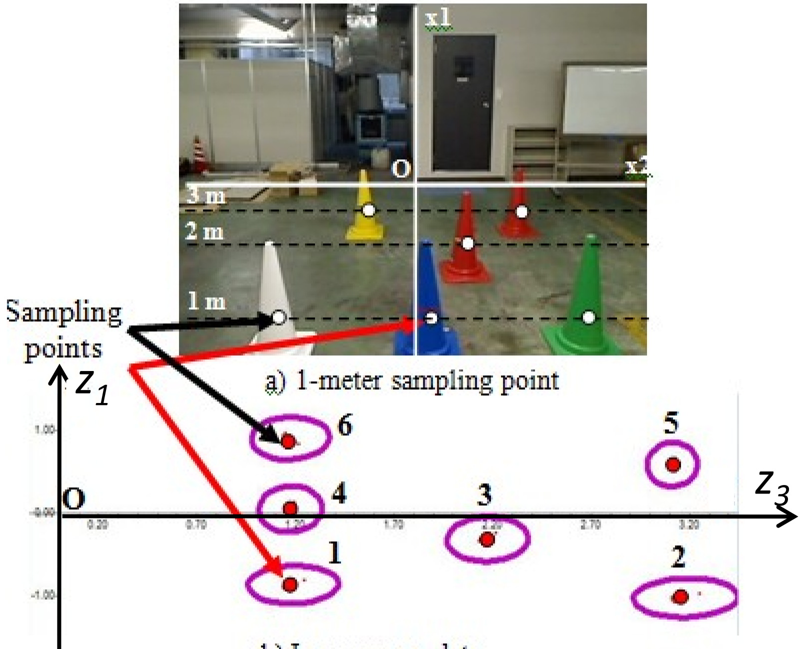
- Assume we have matched points with outliers: How do we compute homography **H**?

Automatic Homography Estimation with RANSAC

1. Choose number of samples *N*
2. Choose 4 random potential matches
3. Compute **H** using DLT
4. Project points from **x** to **x**’ for each potentially matching pair: $\mathbf{x}_i' = \mathbf{H}\mathbf{x}_i$
5. Count points with projected distance < t
   - E.g., t = 3 pixels
6. Repeat steps 2-5 *N* times
   - Choose **H** with most inliers

Examples

Examples



a) 1-meter sampling point

b) Laser-range data

Table 2: Coordinates of landmark points

Point order	Image (pixels)		Laser (mm)	
	x1	x2	z3	z1
1 (green)	-89	-114	1189	-855
2 (far red)	-18	-70	3181	-1016
3 (near red)	-41	-34	2157	-300
4 (blue)	-89	-8	1174	57
5 (yellow)	-18	33	3114	593
6 (white)	-89	94	1175	912

$$\begin{pmatrix} z_3 \\ z_1 \\ 1 \end{pmatrix} = H \begin{pmatrix} x1 \\ x2 \\ 1 \end{pmatrix}$$

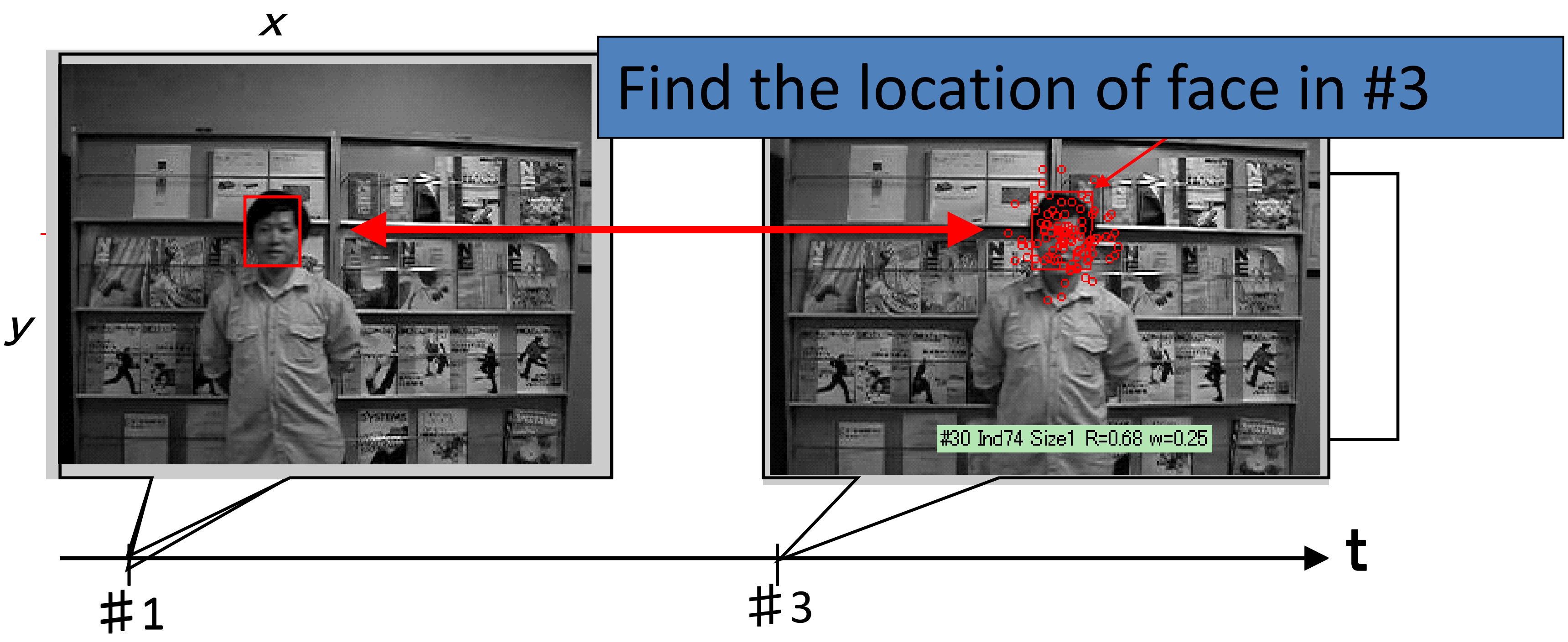
$$rad = \arctan(\frac{z_1}{z_3}) \text{ or } \arctan(\frac{-z_1}{z_3}) + \pi$$

z_2 : distance from the surface to laser device

Tracking Problem and Particle Filter

Tracking problem

Tracking facial object on the next frame in real time using **Particle Filter**



Particle Filters

- Represent belief by random samples
- Estimation of non-Gaussian, nonlinear processes

- Monte Carlo filter, Survival of the fittest, Condensation, Bootstrap filter, Particle filter

- Filtering: [Rubin, 88], [Gordon et al., 93], [Kitagawa 96]
- Computer vision: [Isard and Blake 96, 98]
- Dynamic Bayesian Networks: [Kanazawa et al., 95]d

Particle Filter Algorithm

1. Algorithm **particle_filter**(S_{t-1}, u_{t-1} z_t):
2. $S_t = \emptyset, \quad \eta = 0$
3. **For** $i = 1 \ldots n$
4. Resample index *j(i)* from the discrete distribution given by w_{t-1}
5. Sample x_t^i from $p(x_t \mid x_{t-1}, u_{t-1})$ using $x_{t-1}^{j(i)}$ and u_{t-1}
6. $w_t^i = p(z_t \mid x_t^i)$ ***Compute importance weight***
7. $\eta = \eta + w_t^i$ ***Update normalization factor***
8. $S_t = S_t \cup \{[x_t^i, w_t^i]\}$ ***Insert***
9. **For** $i = 1 \ldots n$
10. $w_t^i = w_t^i / \eta$ ***Normalize weights***

Initial step: Generate the Particles

Using N particles (N=500)

X1, Y1	X2, Y2	X3, Y3	X4, Y4	-----	X500, Y500

$$\begin{pmatrix} X_1 & Y_1 \\ \cdots & \cdots \\ X_{500} & Y_{500} \end{pmatrix} = U_{(500\times 2)} \begin{pmatrix} d & 0 \\ 0 & d \end{pmatrix} + \begin{pmatrix} x_1 \; y_1 \\ \cdots \; \cdots \\ x_1 \; y_1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Random Uniform in [-1, 1]

Weight (N=500) $Wi = \dfrac{1}{N}$

w1	w2	w3	w4	-----	w500

Particle Calculation

z_k: Euclidean distance of object intensity mean.

P(x(k)|x(k-1)): Propagation of position from t=k-1 to t=k.

Particle Calculation (Sequential Importance Sampling):

$$\{x_k^i, w_k^i\}_{i=1}^{N_s} = SIS(\{x_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, z_k)$$

For $i = 1 : N$

- Draw x_k^i from $\{x_{k-1}\}$ by distribution u_{t-1}

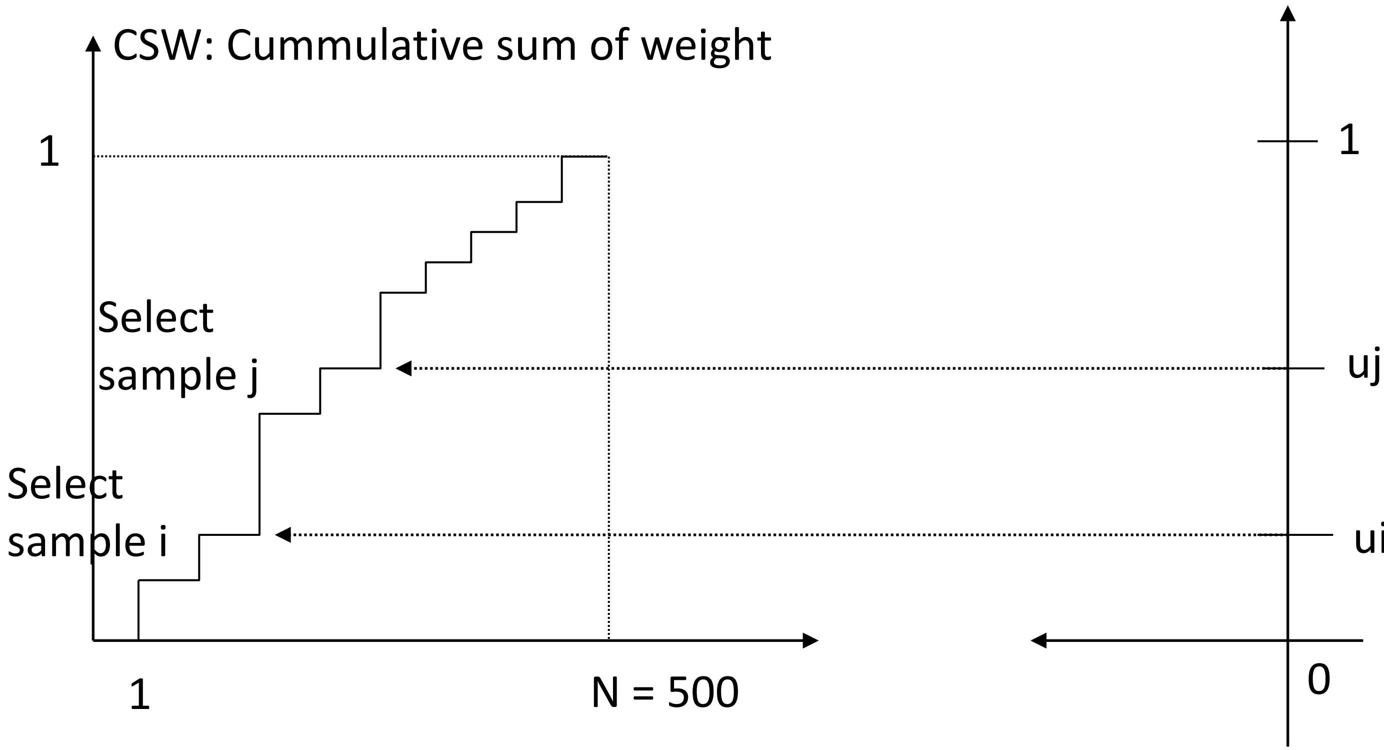
- Update weight

$$w_k^i \approx w_{k-1}^i p(z_k \mid x_k^i) \text{ or } w_k^i \approx p(z_k \mid x_k^i)$$

End For

Normalize weight $w_k^i = \dfrac{w_k^i}{\sum_i w_k^i}$

Resample calculation



$$\textit{Randomly}\ \text{select}\ \mathrm{u_i}, \mathrm{i}= 1,..,\mathrm{N}$$

$$NewSample_i = Sample_t \ \ with\ t = \arg_k(\min\{CSW(k)\}_{k=1,..,N} > u_i)$$

$$NewWeight_i = \frac{1}{N},$$

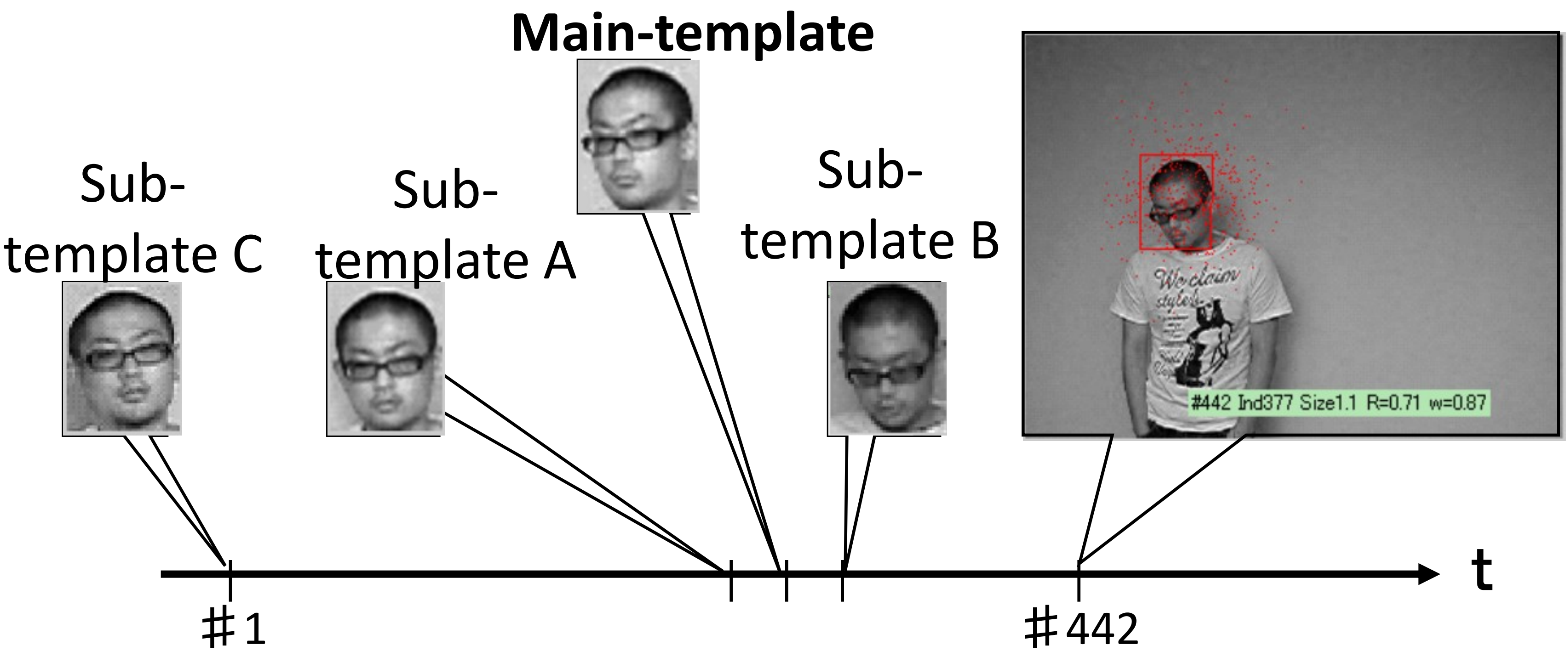
Tactics

- ISI (Important Sampling Sequence)
- **Background application**
  - Generate background
  - Observe background

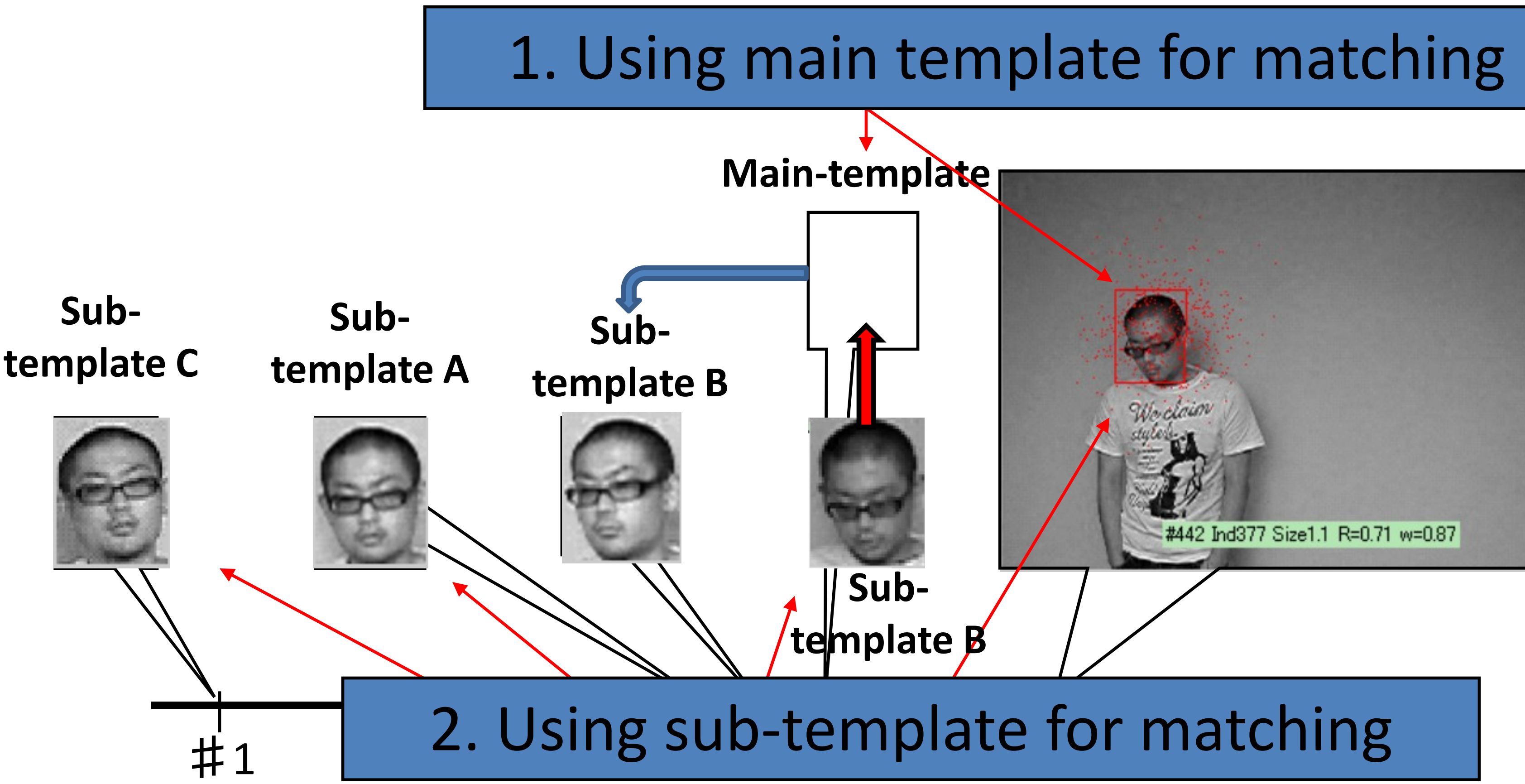
  Keep constraints for searching regions
- **Adaptive template**
  - Check sub-template
  - Register template and swap it to main template
- Resample

Iterative process

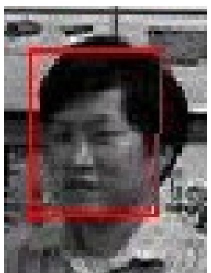
Adaptive Template



Adaptive Template



Using Adaptive Template



23 degree

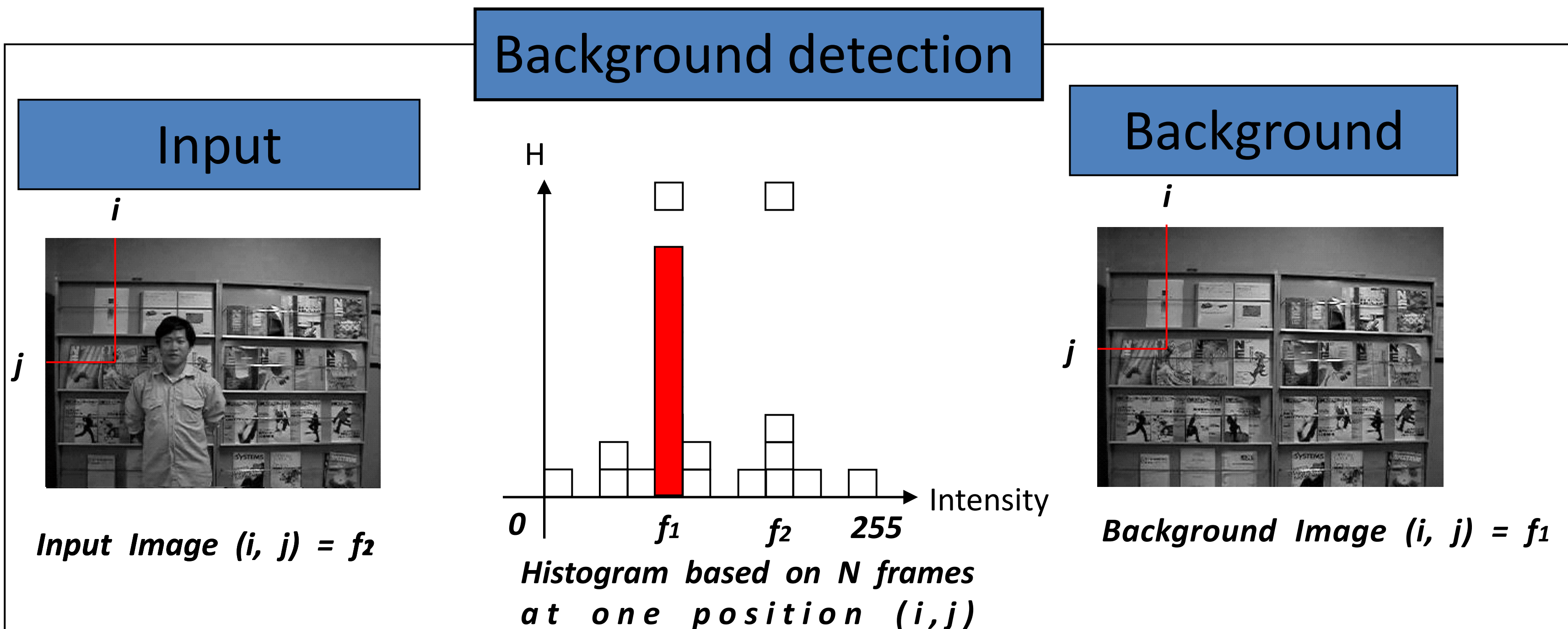
45 degree

Conventional

Adaptive template

Background Detection

Background detection is used when camera is always fixed at the same place and object is moving.



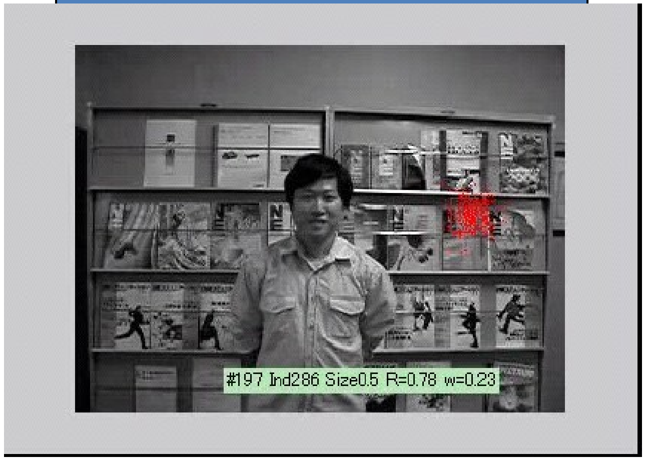
Background Detection

Texture in background

Matching → Tracking error

Conventional

Conventional error



BackgroundDetection

Texture in background

Matching → Tracking error

Background

Conventional

Background information helps to correct the error if it happens.

Discussions

- The structure of particle is designed such that the size of object can change adaptively.
- Adaptive template helps to track movements of one deformable object.
- Background detection helps to correct the error if it happens by tracking in a moving region.

Discrete Kalman Filter

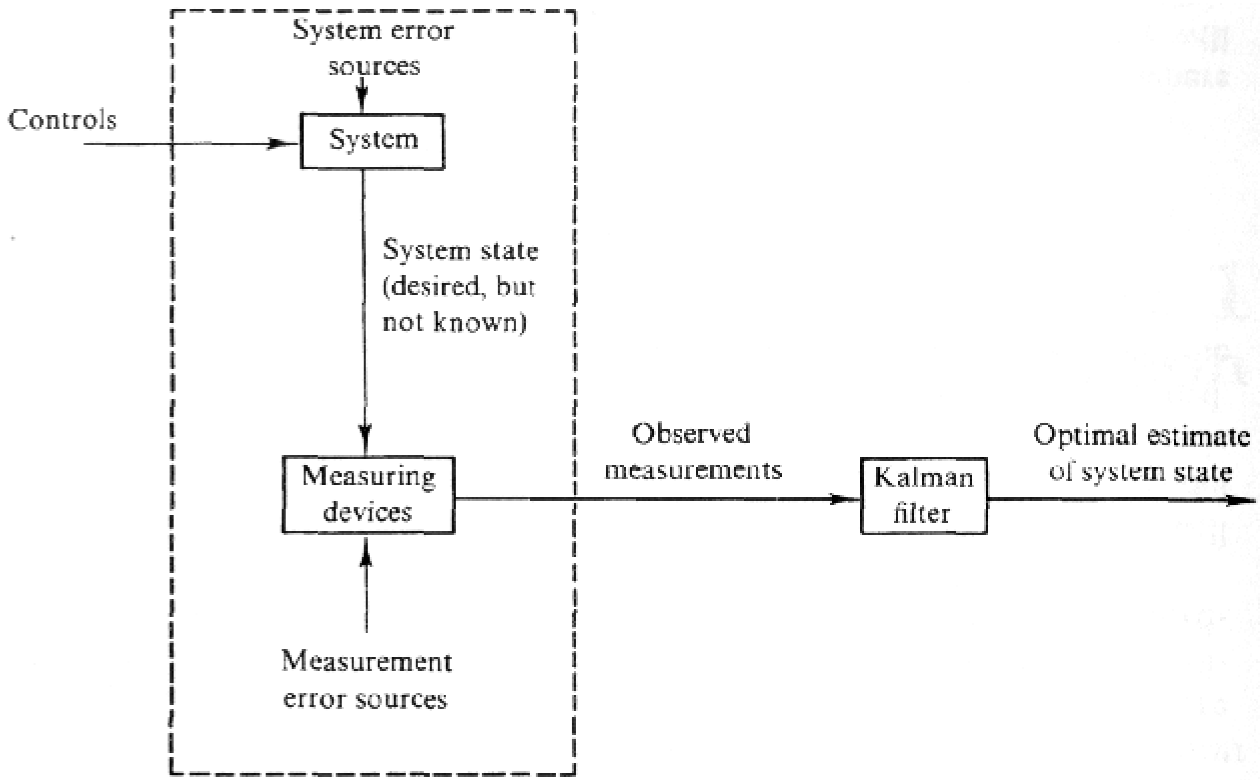
Kalman Filter Method

- How about noise in the system and measurements?
  - There will always be errors in measurements:
    - Imprecise instrumentation,
    - Ambiguity,
    - Finite resolution of measurements.
  - Model of the system *g()* might not be accurate:
    - Imprecise modeling of system.
    - Uncertainty of previously calculated state(s) due to imprecise measurements.
- Solution: tolerate the noise.

Kalman Filter Method

- Kalman filter is "optimal" because it incorporates:
  - knowledge of the system and measurement device dynamics,
  - the statistical description of the system noises, measurement errors, and uncertainty in the dynamics models,
  - any available information about initial conditions of the variables of interest.

Kalman Filter Method



Typical Kalman filter application

Kalman Filter Method

- Linear Kalman Filter Equations

  Assume we have the following plant and measurement equations:

$$a_i = Aa_{i-1} + Bu_{i-1} + w_{i-1}$$

$$x_i = Ha_i + v_i$$

Both w_{i-1} and v_i are error (or noise) and assumed to be normal-distribution random variables.

Kalman Filter Method

- Define R, Q:

R : measurement error covariance cov(v_i)
Q : state model error covariance cov(w_i)

Both R and Q can be calculated incrementally.

- Define P_i:

P_i : State estimation error covariance.

P_i can be calculated by $E[(a_i-\hat{a}_i)(a_i-\hat{a}_i)^T]$

Kalman Filter Method

Linear Kalman Equations are found by optimizing P_i:

Prediction Phase:

$$\bar{a}_i = A\hat{a}_{i-1} + C \quad \text{with } C \; = \; Bu_{i-1} = \text{const}$$

$$\bar{P}_i = AP_{i-1}A^T + Q$$

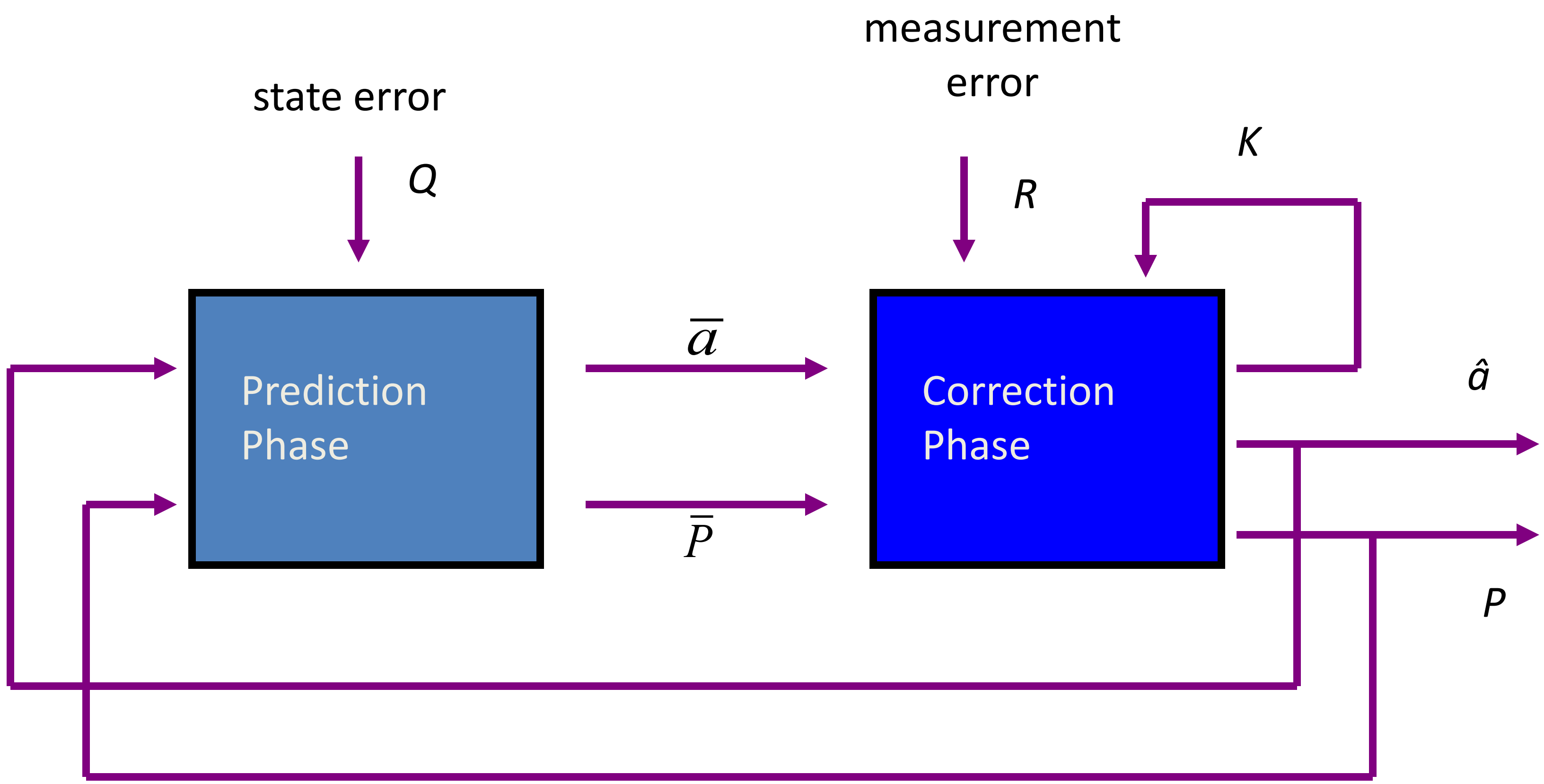
Correction Phase:

$$K_i = \bar{P}_i H^T (H\bar{P}_i H_i^{\ T} + R)^{-1}$$

$$\hat{a}_i = \bar{a}_i + K_i(x_i - H\bar{a}_i)$$

$$P_i = \bar{P}_i - K_i H\bar{P}_i$$

Kalman Filter Method



Kalman Filter Method

Startup of recursion requires:

1. Define state *A, H, C*
2. Initialize *P*, denoted as $P_{0.}$
3. Initialize $\overline{a}_0$
3. Define error covariances R and *Q*.

Kalman filter starts by calculating:

$$\overline{P}_1 = AP_0A^T + Q$$

Then, K_1,

2D Ball Tracking

- Input: A sequence of images with known temporal order.
- Goal: Given a ball in one image, track its movements on the sequence of images.
- Assumption:
  - Ball is existent in image.
  - It always moves in the horizontal line with one specific direction.

2D Ball Tracking

- Problem Abstraction:
  - Identify the “system states”
    - Find a representation of ball
    - The collection of ball position (center), radius, and maximum movement acceleration.
    - Error in the modeling is allowed.
  - Find the “measurement” equation
    - Found by matching with “predicted” ball location.

2D Ball Tracking

Represent a ball with [x-c*enter, radius, speed, acceleration*] where the "plant" equation can be written as followed:

$$a_i = Aa_{i-1} + C, \text{ where A} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$a_i = \begin{bmatrix} x \\ r \\ x^{/} \\ x^{//} \end{bmatrix}, C = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 6 \end{bmatrix}$$

2D Ball Tracking

Measurement function can be written as below:

$$x_i = Ha_i + v_i$$

where

To complete the modeling, we need covariance matrices Q and R.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, P_0 = 100 \times I_{4\times4}, Q = 0.01 \times I_{4\times4}$$

$$R = \begin{bmatrix} 0.28 & 0.0045 \\ 0.0045 & 0.0045 \end{bmatrix}$$

Mean-Shift Tracking

Mean-Shift Approach

Mean Shift [Che98, FH75, Sil86]

- An algorithm that iteratively shifts a data point to the average of data points in its neighborhood.
- Similar to clustering.
- Useful for clustering, mode seeking, probability density estimation, tracking, etc.

- Consider a set S of n data points $\mathbf{x}_i$ in d-D Euclidean space X.
- Let $K(\mathbf{x})$ denote a kernel function that indicates how much $\mathbf{x}$ contributes to the estimation of the mean.
- Then, the sample mean $\mathbf{m}$ at $\mathbf{x}$ with kernel K is given by

$$\mathbf{m}(\mathbf{x}) = \frac{\sum_{i=1}^{n} K(\mathbf{x} - \mathbf{x}_i)\,\mathbf{x}_i}{\sum_{i=1}^{n} K(\mathbf{x} - \mathbf{x}_i)}$$

- The difference $\mathbf{m}(\mathbf{x}) - \mathbf{x}$ is called mean shift.
- Mean shift algorithm: iteratively move date point to its mean.
- In each iteration, $\mathbf{x} \leftarrow \mathbf{m}(\mathbf{x})$.
- The algorithm stops when $\mathbf{m}(\mathbf{x}) = \mathbf{x}$.
- The sequence $\mathbf{x}, \mathbf{m}(\mathbf{x}), \mathbf{m}(\mathbf{m}(\mathbf{x})), \ldots$ is called the trajectory of $\mathbf{x}$.
- If sample means are computed at multiple points, then at each iteration, update is done simultaneously to all these points.

Mean-Shift Tracking

Basic Ideas [CRM00]:

- Model object using color probability density.
- Track target object in video by matching color density.
- Use mean shift to estimate color density and target location.

Object Modeling

- Let $\mathbf{x}_i$, $i = 1, \ldots, n$, denote pixel locations of model centered at $\mathbf{0}$.
- Represent color distribution by discrete m-bin color histogram.
- Let $b(\mathbf{x}_i)$ denote the color bin of the color at $\mathbf{x}_i$.
- Assume size of model is normalized; so, kernel radius $h = 1$.
- Then, the probability q of color u in the model is

$$q_u = C \sum_{i=1}^{n} k(\|\mathbf{x}_i\|^2)\, \delta(b(\mathbf{x}_i) - u)$$

- C is the normalization constant

$$C = \left[\sum_{i=1}^{n} k(\|\mathbf{x}_i\|^2) \right]^{-1}$$

- Kernel profile k weights contribution by distance to centroid.

- δ is the Kronecker delta function

$$\delta(a) = \begin{cases} 1 & \text{if } a = 0 \\ 0 & \text{otherwise} \end{cases}$$

That is, contribute $k(\|\mathbf{x}_i\|^2)$ to q_u if $b(\mathbf{x}_i) = u$.

Target Candidate

- Let $\mathbf{y}_i$, $i = 1, \ldots, n_h$, denote pixel locations of target centered at $\mathbf{y}$.
- Then, the probability p of color u in the target is

$$p_u(\mathbf{y}) = C_h \sum_{i=1}^{n_h} k\left(\left\|\frac{\mathbf{y}-\mathbf{y}_i}{h}\right\|^2\right)\delta(b(\mathbf{y}_i)-u)$$

- C_h is the normalization constant

$$C_h = \left[\sum_{i=1}^{n_h} k\left(\left\|\frac{\mathbf{y}-\mathbf{y}_i}{h}\right\|^2\right)\right]^{-1}$$

- Use Bhattacharyya coefficient ρ

$$\rho(p(\mathbf{y}), q) = \sum_{u=1}^{m} \sqrt{p_u(\mathbf{y})\, q_u}$$

- ρ is the cosine of vectors $(\sqrt{p_1}, \ldots, \sqrt{p_m})^\top$ and $(\sqrt{q_1}, \ldots, \sqrt{q_m})^\top$.
- Large ρ means good color match.
- For each image frame, find $\mathbf{y}$ that maximizes ρ.
- This $\mathbf{y}$ is the location of the target.

Tracking Algorithm

Given $\{q_u\}$ of model and location $\mathbf{y}$ of target in previous frame:

1. Initialize location of target in current frame as $\mathbf{y}$.
2. Compute $\{p_u(\mathbf{y})\}$ and $\rho(p(\mathbf{y}), q)$.
3. Apply mean shift: Compute new location $\mathbf{z}$ as

$$\mathbf{z} = \frac{\sum_{i=1}^{n_h} g\left(\left\|\frac{\mathbf{y}-\mathbf{y}_i}{h}\right\|^2\right)\mathbf{y}_i}{\sum_{i=1}^{n_h} g\left(\left\|\frac{\mathbf{y}-\mathbf{y}_i}{h}\right\|^2\right)}$$

4. Compute $\{p_u(\mathbf{z})\}$ and $\rho(p(\mathbf{z}), q)$.
5. While $\rho(p(\mathbf{z}), q) < \rho(p(\mathbf{y}), q)$, do $\mathbf{z} \leftarrow \frac{1}{2}(\mathbf{y}+\mathbf{z})$.
6. If $\|\mathbf{z}-\mathbf{y}\|$ is small enough, stop. Else, set $\mathbf{y} \leftarrow \mathbf{z}$ and goto (1).

- Step 3: In practice, a window of pixels $\mathbf{y}_i$ is considered. Size of window is related to h.
- Step 5 is used to validate the target's new location. Can stop Step 5 if $\mathbf{y}$ and $\mathbf{z}$ round off to the same pixel.
- Tests show that Step 5 is needed only 0.1% of the time.
- Step 6: can stop algorithm if $\mathbf{y}$ and $\mathbf{z}$ round off to the same pixel.
- To track object that changes size, varies radius h (see [CRM00] for details).

Kernel

- Typically, kernel K is a function of $\|\mathbf{x}\|^2$:

$$K(\mathbf{x}) = k(\|\mathbf{x}\|^2)$$

- k is called the profile of K.

Properties of Profile:

1. k is nonnegative.
2. k is nonincreasing: $k(x) \geq k(y)$ if $x < y$.
3. k is piecewise continuous and

$$\int_0^\infty k(x)dx < \infty$$

Kernel

- Flat kernel:

$$K(\mathbf{x}) = \begin{cases} 1 & \text{if } \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases}$$

- The mean square error is minimized by the Epanechnikov kernel:

$$K_E(\mathbf{x}) = \begin{cases} \frac{1}{2C_d}(d+2)(1 - \|\mathbf{x}\|^2) & \text{if } \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases}$$

where C_d is the volume of the unit d-D sphere, with profile

$$k_E(x) = \begin{cases} \frac{1}{2C_d}(d+2)(1 - x) & \text{if } 0 \le x \le 1 \\ 0 & \text{if } x > 1 \end{cases}$$

Kernel

- A more commonly used kernel is Gaussian

$$K(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right)$$

with profile

$$k(x) = \frac{1}{\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2}x\right)$$

- Define another kernel $G(\mathbf{x}) = g(\|\mathbf{x}\|^2)$ such that

$$g(x) = -\frac{dk(x)}{dx}$$

References

[1] Multiple View Geometry in Computer Vision, Richard Hartley, Andrew Zisserman, Cambridge Unv. Press, 2003.

[2] M.S. Arulampalam, et.al., A tutorial on particle filters for online nonlinear/non-GaussianBayesian tracking, Trans. IEEE Signal Processing, Vol. 50(2), 2002, pp. 174-188.

[3] Computer Vision: Algorithms and Applications, Richard Szeliski, Springer, 2011.

[4] L.W. Kheng, Course slide, CS4243, National University of Singapore.