Image Registration
Lecture 14:
Distances and Least Squares

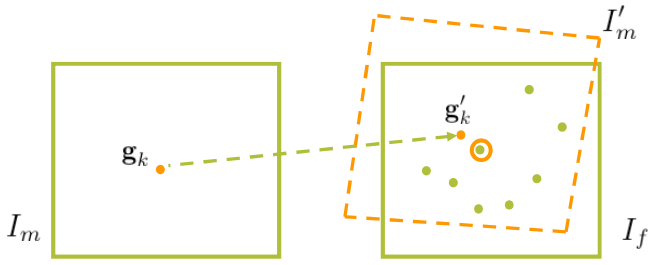
Prof. Charlene Tsai

Overview

- Distance measures
- Error projectors
- Implementation in toolkit
- Normalizing least-squares matrices
- Review

Reminder: Point-to-Point Euclidean Distance

- We started feature-based registration by thinking of our features as point locations.



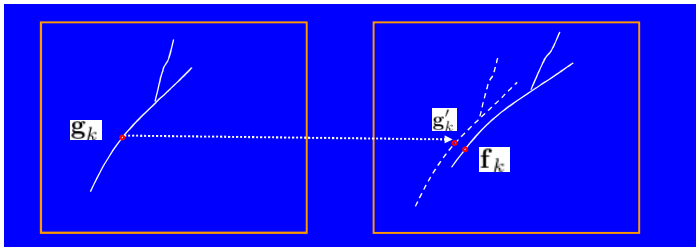
- This led to an alignment error measure based on Euclidean distances:

$$D(\mathbf{T}(\mathbf{g}_k;\boldsymbol{\Theta}),\mathbf{f}_k) = \|\mathbf{T}(\mathbf{g}_k;\boldsymbol{\Theta}) - \mathbf{f}_k\|$$

3

What About Linear Features?

- Suppose our features are center points along a blood vessel, and consider the following picture

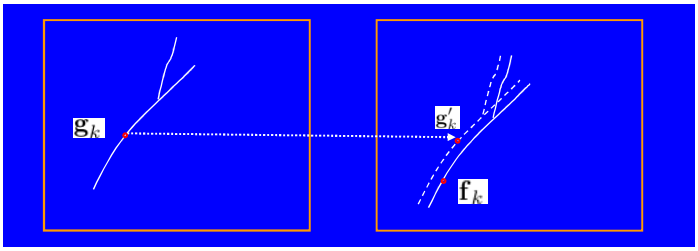


- Because of the misalignment on the right, the distance between the matching features is small

4

What About Linear Features?

- If the transformation moves the entire vessel toward the correction alignment, the distance **increases** for this correspondence!



- This prevents or at least slows correct alignment!

5

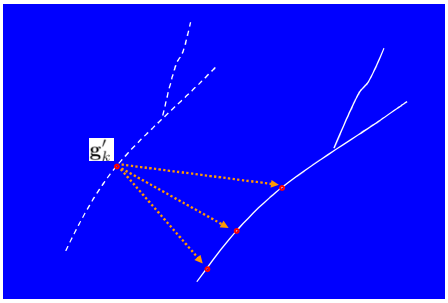
Potential Solutions

- Use only distinctive points:
  - Major disadvantage: could be very few in the image
  - Leads to less accurate or even failed alignment
- Augment features to make matching more distinctive:
  - Unfortunately, points along a contour tend to have similar appearance to each other, and
  - This is more effective for distinguishing between different contours
- A different error measure:
  - Focus of our discussion

6

Point-to-Line Distance

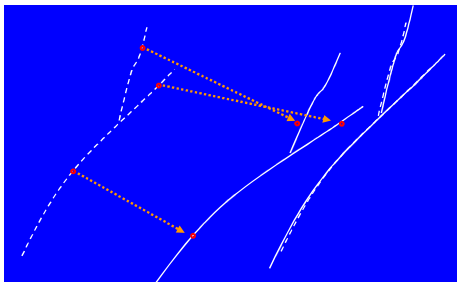
- Intuition:
  - Can't tell the difference between points along a contour
  - Therefore, want an error measure that is (close to) 0 when the alignment places the moving image feature along the contour.



7

Point-to-Line Distance

- Rely on correspondences from other correspondences to correct for errors along the contour:



- Only the (approximately) correct alignment satisfies all constraints simultaneously.
  - Of course this assumes points are matched to the right contour, which is why iterative rematching is needed.

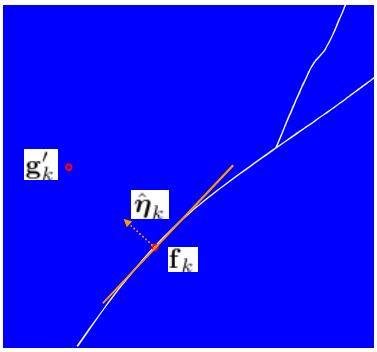
8

Measuring Point-To-Line Distance

- Linear approximation to contour through closest point
- Yields what is called the "normal distance"

$$(\mathbf{g}'_k - \mathbf{f}_k)^T \hat{\boldsymbol{\eta}}_k$$

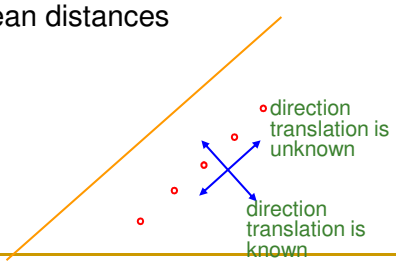
- This is the distance from the transformed point, $\mathbf{g}'_k$, to the line through $\mathbf{f}_k$ in direction $\hat{\boldsymbol{\eta}}_k$
- If the transformed point falls anywhere along this line, the distance is 0
- The transformation error distance is now $D(\mathbf{T}(\mathbf{g}_k;\boldsymbol{\Theta}),\mathbf{f}_k) = (\mathbf{T}(\mathbf{g}_k;\boldsymbol{\Theta}) - \mathbf{f}_k)^T \hat{\boldsymbol{\eta}}_k$



9

Advantages of Point-to-Line Distances

- Faster convergence
- Fewer problems with local minima
- No false sense of stability in the estimate
  - For example, estimating the translation will be (properly) ill-conditioned for points from a line using normal distances, but not Euclidean distances



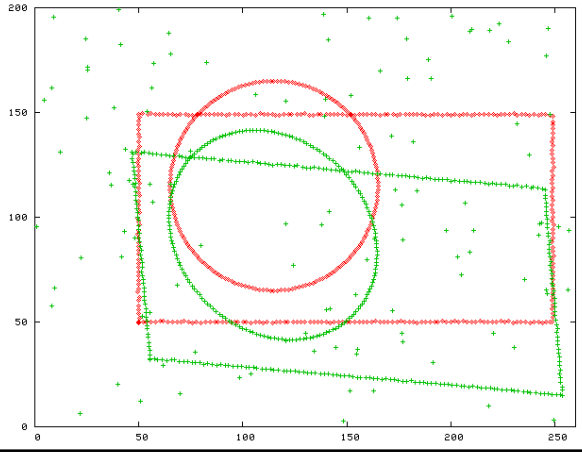
10

Disadvantages of Point-to-Line Distances

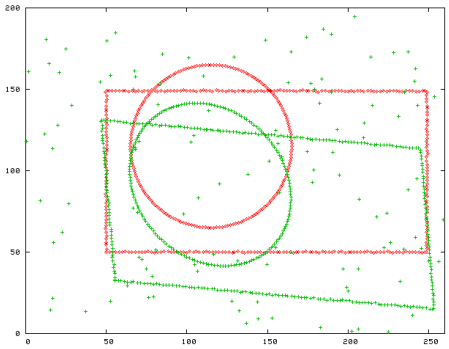
- Harder to initialize using just feature-points along contours
- Requires computation of tangent or normal direction

Example

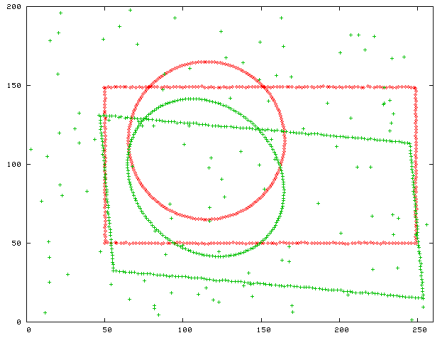
- Here is a case where using Euclidean distances fails to converge:



Example



Using normal distances (rgrl_feature_trace_pt), registration requires 8 rematching iterations to converge.



Using Euclidean distances (rgrl_feature_point), registration requires 22 rematching iterations to converge and the result is less accurate.

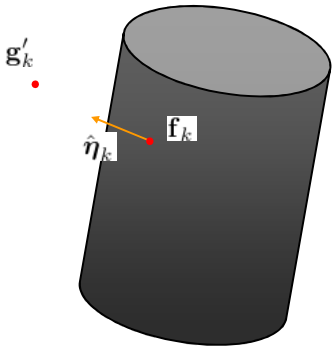
13

3D: Features from Surfaces

- Tangent-plane approximation to surface around fixed-image point
- Results in "normal-distance", just like point-to-line in 2d

$$D(\mathbf{T}(\mathbf{g}_k;\boldsymbol{\Theta}),\mathbf{f}_k) = (\mathbf{T}(\mathbf{g}_k;\boldsymbol{\Theta}) - \mathbf{f}_k)^T \hat{\boldsymbol{\eta}}_k$$

- Implemented as an `rgrl_feature_face_pt`



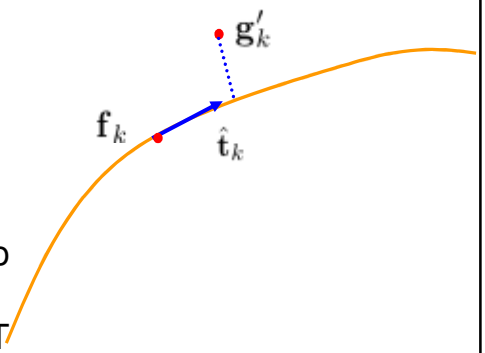
14

3D: Features from Contours

- Consider a contour in 3d and its tangent direction
- The distance of a point to this contour is

$$\|(\mathbf{g}'_k - \mathbf{f}_k) - [(\mathbf{g}'_k - \mathbf{f}_k)^T \hat{\mathbf{t}}]\hat{\mathbf{t}}\|$$

- The second term is the projection of the error onto the tangent direction
- This is the term we DON'T want, so we subtract from from the (first) error vector
  - The dotted line shows the error vector we want.

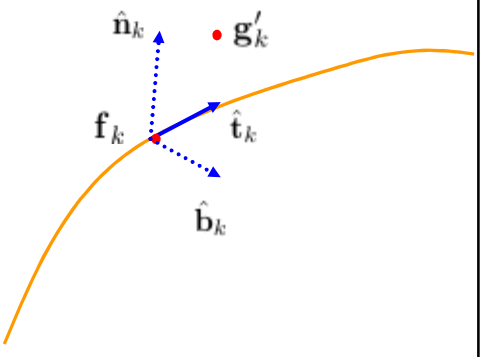


15

3D: Features from Contours (2)

- At a point on a smooth contour in 3d there are two normal directions.
  - Orthogonal basis vectors for these directions can be found from the null-space of the 1x3 matrix formed by the tangent vector.
- The distance we are interested in is the distance in this plane
- The distance is formed as the projection of the error vector onto the plane basis vectors:

$$[((\mathbf{g}'_k - \mathbf{f}_k)^T \hat{\mathbf{n}})^2 + ((\mathbf{g}'_k - \mathbf{f}_k)^T \hat{\mathbf{b}})^2]^{1/2}$$



16

Implementation

- rgrl_feature_trace_pt
  - This code is the same for a point in 2d and a point in 3d.
- Problem:
  - As written on the previous pages, there is a two-component error term in 3d and a one-component error term in 2d. How does this work?
- Answer:
  - An “error projector”
- As we will see, this will also help avoid having to write special cases for each feature type

17

Error Projector

- For data in *d* dimensions, the error projector is a *d*x*d* matrix **P** such that

$$(\mathbf{g}_k - \mathbf{f}_k)^T \mathbf{P} (\mathbf{g}_k - \mathbf{f}_k)$$

  is the square distance according to the feature type. (At this point, we are dropping the ‘ on the moving feature, just to avoid over-complexity in the notation.)
- For point features, where the Euclidean distance is used, the error projector is just the identity matrix:

$$\begin{aligned}(\mathbf{g}_k - \mathbf{f}_k)^T (\mathbf{g}_k - \mathbf{f}_k) &= (\mathbf{g}_k - \mathbf{f}_k)^T \mathbf{I} (\mathbf{g}_k - \mathbf{f}_k) \\ &= (\mathbf{g}_k - \mathbf{f}_k)^T \mathbf{P} (\mathbf{g}_k - \mathbf{f}_k)\end{aligned}$$

18

Error Projector for Normal Distances

- Start with the square of the normal distance:

$$[(\mathbf{g}_k - \mathbf{f}_k)^T \hat{\boldsymbol{\eta}}_k]^2$$

- Using properties of the dot product:

$$\begin{aligned}[(\mathbf{g}_k - \mathbf{f}_k)^T \hat{\boldsymbol{\eta}}_k]^2 &= [(\mathbf{g}_k - \mathbf{f}_k)^T \hat{\boldsymbol{\eta}}_k][\hat{\boldsymbol{\eta}}_k^T (\mathbf{g}_k - \mathbf{f}_k)] \\ &= (\mathbf{g}_k - \mathbf{f}_k)^T (\hat{\boldsymbol{\eta}}_k \hat{\boldsymbol{\eta}}_k^T)(\mathbf{g}_k - \mathbf{f}_k) \\ &= (\mathbf{g}_k - \mathbf{f}_k)^T \mathbf{P} (\mathbf{g}_k - \mathbf{f}_k)\end{aligned}$$

- So, the error projector is the outer product of the normal vector: $\mathbf{P} = \hat{\boldsymbol{\eta}}_k \hat{\boldsymbol{\eta}}_k^T$

19

Error Projector: Point-to-Contour

- We use the original form of the square point-to-line distance:

$$\|(\mathbf{g}'_k - \mathbf{f}_k) - [(\mathbf{g}'_k - \mathbf{f}_k)^T \hat{\mathbf{t}}]\hat{\mathbf{t}}\|^2$$

- Notes:
  - This is correct in any dimension, from 2 on up.
  - This uses the contour **tangent**, whereas the normal distance (face) uses the **normal**
- Now, look at just the projection of the error vector:

$$\begin{aligned}(\mathbf{g}_k - \mathbf{f}_k) - [(\mathbf{g}_k - \mathbf{f}_k)^T \hat{\mathbf{t}}]\hat{\mathbf{t}} &= (\mathbf{g}_k - \mathbf{f}_k) - \hat{\mathbf{t}}[\hat{\mathbf{t}}^T (\mathbf{g}_k - \mathbf{f}_k)] \\ &= \mathbf{I}(\mathbf{g}_k - \mathbf{f}_k) - [\hat{\mathbf{t}}\hat{\mathbf{t}}^T](\mathbf{g}_k - \mathbf{f}_k) \\ &= (\mathbf{I} - \hat{\mathbf{t}}\hat{\mathbf{t}}^T)(\mathbf{g}_k - \mathbf{f}_k)\end{aligned}$$

20

Error Projector: Point-to-Line

- Now look at the square magnitude of this vector:

$$
\begin{aligned}
&(\mathbf{g}_k - \mathbf{f}_k)^T (\mathbf{I} - \hat{\mathbf{t}}\hat{\mathbf{t}}^T)^T (\mathbf{I} - \hat{\mathbf{t}}\hat{\mathbf{t}}^T)(\mathbf{g}_k - \mathbf{f}_k) \\
&\quad = (\mathbf{g}_k - \mathbf{f}_k)^T (\mathbf{I} - \hat{\mathbf{t}}\hat{\mathbf{t}}^T)(\mathbf{I} - \hat{\mathbf{t}}\hat{\mathbf{t}}^T)(\mathbf{g}_k - \mathbf{f}_k) \\
&\quad = (\mathbf{g}_k - \mathbf{f}_k)^T (\mathbf{I} - \hat{\mathbf{t}}\hat{\mathbf{t}}^T - \hat{\mathbf{t}}\hat{\mathbf{t}}^T + \hat{\mathbf{t}}\hat{\mathbf{t}}^T\hat{\mathbf{t}}\hat{\mathbf{t}}^T)(\mathbf{g}_k - \mathbf{f}_k) \\
&\quad = (\mathbf{g}_k - \mathbf{f}_k)^T (\mathbf{I} - \hat{\mathbf{t}}\hat{\mathbf{t}}^T - \hat{\mathbf{t}}\hat{\mathbf{t}}^T + \hat{\mathbf{t}}\hat{\mathbf{t}}^T)(\mathbf{g}_k - \mathbf{f}_k) \\
&\quad = (\mathbf{g}_k - \mathbf{f}_k)^T (\mathbf{I} - \hat{\mathbf{t}}\hat{\mathbf{t}}^T)(\mathbf{g}_k - \mathbf{f}_k)
\end{aligned}
$$

- Which gives the error projector $\mathbf{P} = \mathbf{I} - \hat{\mathbf{t}}\hat{\mathbf{t}}^T$

21

Error Projector - Summary

Feature Type	Rgrl class	Projector	DoF to Constraint
Point	rgrl_feature_point	$\mathbf{P} = \mathbf{I}$	*m*
Point on contour	rgrl_feature_trace_pt	$\mathbf{P} = \mathbf{I} - \hat{\mathbf{t}}\hat{\mathbf{t}}^T$	*m-1*
Point on surface	kgrl_feature_face_pt	$\mathbf{P} = \hat{\boldsymbol{\eta}}_k \hat{\boldsymbol{\eta}}_k^T$	*1*

m is the dimension of the space containing the points - 2 or 3 so far

22

Error Projector and Least-Squares

- The error projector fits naturally with least-squares estimation. We'll consider the case of 2d affine estimation.
- Recall that the alignment error is

$$\mathbf{A}\mathbf{g}_k + \mathbf{t} - \mathbf{f}_k = \begin{pmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \end{pmatrix} \mathbf{g}_k + \mathbf{t} - \mathbf{f}_k$$

- We re-arrange this so that the parameters of the transformation are gathered in a vector:

$$\mathbf{A}\mathbf{g}_k + \mathbf{t} - \mathbf{f}_k = \begin{pmatrix} \mathbf{g}_k^T & \mathbf{0}^T & 1 & 0 \\ \mathbf{0}^T & \mathbf{g}_k^T & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{t} \end{pmatrix} - \mathbf{f}_k$$

$$= \mathbf{X}_k\mathbf{a} - \mathbf{f}_k$$

- Be sure that you understand this. Think about which terms here are known and which are unknown.

23

Error Projector and Least-Squares: 2d Affine

- The squared least-squares alignment error for correspondence *k* is now

$$(\mathbf{X}_k\mathbf{a} - \mathbf{f}_k)^T\mathbf{P}(\mathbf{X}_k\mathbf{a} - \mathbf{f}_k)$$

- And the weighted least-squares alignment error is

$$w_k(\mathbf{X}_k\mathbf{a} - \mathbf{f}_k)^T\mathbf{P}(\mathbf{X}_k\mathbf{a} - \mathbf{f}_k)$$

- The summed squared alignment error is

$$\sum_k w_k(\mathbf{X}_k\mathbf{a} - \mathbf{f}_k)^T\mathbf{P}(\mathbf{X}_k\mathbf{a} - \mathbf{f}_k)$$

- Finally, the weighted least-squares estimate is

$$\hat{\mathbf{a}} = \left(\sum_k w_k\mathbf{X}_k^T\mathbf{P}\mathbf{X}_k\right)^{-1}\left(\sum_k w_k\mathbf{X}_k^T\mathbf{P}\mathbf{f}_k\right)$$

24

Discussion

- From a coding perspective, the importance of this can not be over-estimated:
  - *We don't need special-purpose code in our estimators for each type of feature*
- Each instance of each feature object, when constructed, builds its own error projector
- This error projector matrix is provided to the estimator when building the two constraint matrices:

$$\left(\sum_k w_k \mathbf{X}_k^T \mathbf{P} \mathbf{X}_k\right) \quad \text{and} \quad \left(\sum_k w_k \mathbf{X}_k^T \mathbf{P} \mathbf{f}_k\right)$$

25

Conditioning: Centering and Normalization

- While we are on the issue of weighted least-squares, we'll consider two important implementation details:
  - Centering and normalization
- These are necessary to improve the conditioning of the estimates
- We'll look at the very simplest case of 2d affine with projection matrix **P=I**

26

Consider the Weighted "Scatter Matrix"

$$\sum w_k \mathbf{X}_k^T \mathbf{P} \mathbf{X}_k =$$

$$\begin{pmatrix} \sum w_k x_k^2 & \sum w_k x_k y_k & 0 & 0 & \sum w_k x_k & 0 \\ \sum w_k x_k y_k & \sum w_k y_k^2 & 0 & 0 & \sum w_k y_k & 0 \\ 0 & 0 & \sum w_k x_k^2 & \sum w_k x_k y_k & 0 & \sum w_k x_k \\ 0 & 0 & \sum w_k x_k y_k & \sum w_k y_k^2 & 0 & \sum w_k y_k \\ \sum w_k x_k & \sum w_k y_k & 0 & 0 & \sum w_k & 0 \\ 0 & 0 & \sum w_k x_k & \sum w_k y_k & 0 & \sum w_k \end{pmatrix}$$

- Observations:
  - The upper left 2x2 and center 2x2 are quadratic in x and y
  - The upper right 4x2 are linear in x and y
  - The lower right 2x2 is independent of x and y
- When x is in units of (up to) 1,000 then the quadratic terms will be in units of (up to) 1,000,000.
- This can lead to numerical ill-conditioning and other problems
- For non-linear models, the problems get much worse!

27

Solution, Part 1: Center the Data

- With $\mathbf{g}_k = (x_k, y_k)^T$ and $\mathbf{f}_k = (u_k, v_k)^T$
- compute weighted centers:

$$\mathbf{g}_c = \left(\sum_k w_k \mathbf{g}_k\right) / \left(\sum_k w_k\right) \quad \text{and} \quad \mathbf{f}_c = \left(\sum_k w_k \mathbf{f}_k\right) / \left(\sum_k w_k\right)$$

- Center the points: $\mathbf{g}'_k = \mathbf{g}_k - \mathbf{g}_c$ and $\mathbf{f}'_k = \mathbf{f}_k - \mathbf{f}_c$
- Estimate the transformation, obtaining parameter vector $\hat{\mathbf{a}}_c$
- Take this vector apart to obtain

$$\mathbf{A}_c, \mathbf{t}_c$$

28

Solution, Part 1: Center the Data

- Eliminate the centering:

$$\begin{aligned}\mathbf{A}_c\mathbf{g}' + \mathbf{t}_c - \mathbf{f}' &= \mathbf{A}_c(\mathbf{g} - \mathbf{g}_c) + \mathbf{t}_c - (\mathbf{f} - \mathbf{f}_c) \\ &= \mathbf{A}_c\mathbf{g} + (\mathbf{t}_c - \mathbf{A}_c\mathbf{g}_c + \mathbf{f}_c) - \mathbf{f}\end{aligned}$$

- Hence, the uncentered estimates are

$$\mathbf{A} = \mathbf{A}_c \qquad \text{and} \qquad \mathbf{t} = \mathbf{t}_c - \mathbf{A}_c\mathbf{g}_c$$

29

Solution, Part 2: Normalize the Data

$$\sum w_k \mathbf{X}_k^T \mathbf{P} \mathbf{X}_k =$$

$$\begin{pmatrix} \sum w_k x_k^2 & \sum w_k x_k y_k & 0 & 0 & \sum w_k x_k & 0 \\ \sum w_k x_k y_k & \sum w_k y_k^2 & 0 & 0 & \sum w_k y_k & 0 \\ 0 & 0 & \sum w_k x_k^2 & \sum w_k x_k y_k & 0 & \sum w_k x_k \\ 0 & 0 & \sum w_k x_k y_k & \sum w_k y_k^2 & 0 & \sum w_k y_k \\ \sum w_k x_k & \sum w_k y_k & 0 & 0 & \sum w_k & 0 \\ 0 & 0 & \sum w_k x_k & \sum w_k y_k & 0 & \sum w_k \end{pmatrix}$$

- Compute a scale factor:

$$s = \left(\frac{\max(\sum w_k x_k^2, \sum w_k y_k^2)}{\sum w_k} \right)^{1/2}$$

- Form diagonal matrix: $\mathbf{S} = \text{diag}(1, 1, 1, 1, s, s)$

30

Solution, Part 2: Normalize the Data

- Scale our two matrices:

$$\left(\sum_k w_k \mathbf{X}_k^T \mathbf{P} \mathbf{X}_k\right) \quad \text{and} \quad \left(\sum_k w_k \mathbf{X}_k^T \mathbf{P} \mathbf{f}_k\right)$$

- Become

$$\mathbf{S}\left(\sum_k w_k \mathbf{X}_k^T \mathbf{P} \mathbf{X}_k\right)\mathbf{S} \quad \text{and} \quad \mathbf{S}\left(\sum_k w_k \mathbf{X}_k^T \mathbf{P} \mathbf{f}_k\right)$$

- In detail,

$$\mathbf{S}\sum w_k \mathbf{X}_k^T \mathbf{P} \mathbf{X}_k \mathbf{S} =$$

$$\begin{pmatrix} \sum w_k x_k^2 & \sum w_k x_k y_k & 0 & 0 & s\sum w_k x_k & 0 \\ \sum w_k x_k y_k & \sum w_k y_k^2 & 0 & 0 & s\sum w_k y_k & 0 \\ 0 & 0 & \sum w_k x_k^2 & \sum w_k x_k y_k & 0 & s\sum w_k x_k \\ 0 & 0 & \sum w_k x_k y_k & \sum w_k y_k^2 & 0 & s\sum w_k y_k \\ s\sum w_k x_k & s\sum w_k y_k & 0 & 0 & s^2\sum w_k & 0 \\ 0 & 0 & s\sum w_k x_k & s\sum w_k y_k & 0 & s^2\sum w_k \end{pmatrix}$$

31

Solution, Part 2: Normalize the Data

- Aside:
  - When we've centered the data the terms in the upper right 4x2 and lower left 2x4 submatrices become 0
  - But, this is only true when **P=I**
  - In general, when **P != I**, the matrix has no 0 entries
- Now back to our solution…

32

Solution, Part 2: Normalize the Data

- To solve, invert to obtain a normalized estimate:

$$\hat{a}_N = \left(\mathbf{S}\sum_k w_k \mathbf{X}_k^T \mathbf{P} \mathbf{X}_k \mathbf{S}\right)^{-1} \left(\mathbf{S}\sum_k w_k \mathbf{X}_k^T \mathbf{P} \mathbf{f}_k\right)$$

- To obtain the unnormalized estimate we simply compute:

$$\hat{\mathbf{a}} = \mathbf{S}\,\hat{\mathbf{a}}_N$$

33

Centering and Normalization, Summary

- Applying centering first
- Apply normalization
- Compute normalized, centered estimate
- Undo normalization to obtain unnormalized, centered estimate
- Undo centering to obtain the final, uncentered estimate
  - In practice, the internal representation of the transformation is centered.

34

Lecture 14: Summary

- Different feature types produce different error measures (distance constraints)
- These constraints may be described succinctly using an error projector matrix
- The error projector matrix simplifies the code in the estimators, removing any need to specialize based on feature types
- Effective estimation requires use of centering and normalization of the constraints

35