# Segmentation

> Often want to take a set of items (pixels, voxels, 3D points) and group them together.

"Blob Finding"

> How do we find clusters of related points?

idea: use pairwise similarity test

Consider a graph of "items". An edge between them if they should be part of the same object.

Ex: connect two voxels if dist < T.

Ex: connect two pixels if |color1 − color2| < T.

## Leads to "union find"

```
int  get_representative (int nodenum)
void connect(int nodea, int nodeb)
```

← e.g. minimum node id reachable from nodenum.

Two passes typical:

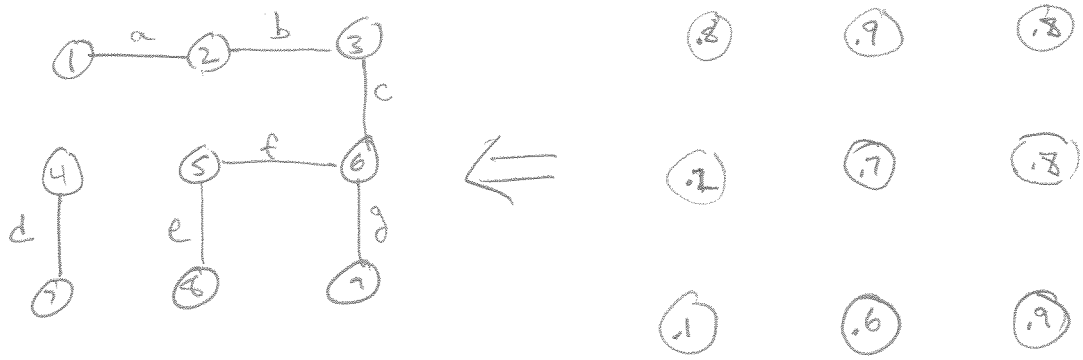
First pass: connect all nodes w/ edges.

Second pass: collect statistics for groups of items w/ same representative.

# Union Find

Create a "pointer" table:

every node points to a better parent node (or itself).




consider these pixel values with T=0.1

| node id | step (init) 0 | (1,2) a | (2,3) b | (3,6) c | (4,7) d | (5,8) e | (5,6) f | (6,9) g |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | | | | | | | |
| 2 | 2 | 1 | | | | | | |
| 3 | 3 | | 1 | | | | | |
| 4 | 4 | | | | | | | |
| 5 | 5 | | | | | | 1 | |
| 6 | 6 | | | 1 | | | | |
| 7 | 7 | | | | 4 | | | |
| 8 | 8 | | | | | 5 | | |
| 9 | 9 | | | | | | | 1 |

```
connect(a, b)
    ra = get_rep(a)
    rb = get_rep(b)
```

both root nodes

set the parent of the bigger one to the smaller one,
(or use a size heuristic)

```
get_rep(a)
    if a.parent == a
        return a
    a.parent = get_rep(a.parent)
    return a.parent
```

Flattens tree! ⟶

Runtime complexity (with flattening & a good merging policy)

$O(\alpha(n))$

↑ inverse ackerman

$$\text{ackerman}(4) \approx \underbrace{2^{2^{10^{19729}}}}_{\text{big! so } \alpha(n) < 5 \text{ for "any" } n.}$$

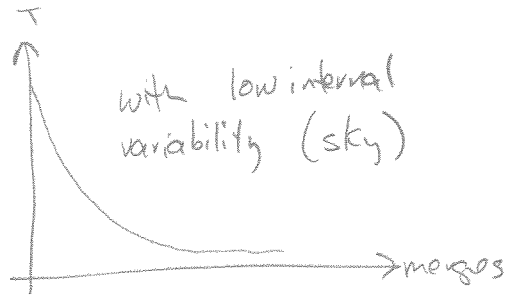
# Segmentation: Finding Threshold?

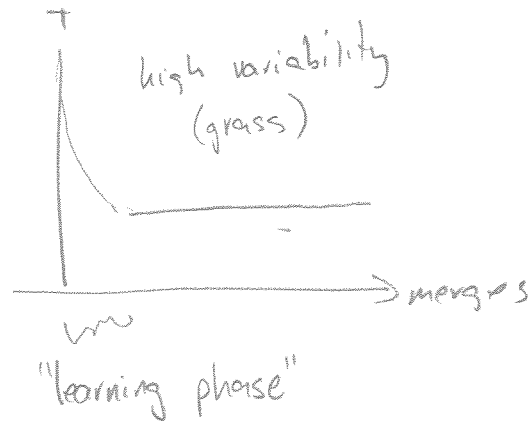
idea: set threshold dynamically based on properties of a cluster.

- begin with "large" threshold for each segment
- threshold for segment

$$\approx [\text{max edge cost already accepted}] + \frac{K}{\text{segment size}}.$$

Threshold for a segment.




- process all possible edges in best-to-worst order.
- reject edges with cost > per-segment cost.

Felzenszwalb-Huttenlocher (IJCV 2004)

# Computing Homographies

**A what?** "A picture (projective view) of a plane".

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & & c_x & 0 \\ & f_y & c_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Image point 3x1 homogeneous — Camera Matrix 3x4 $K$ — Extrinsics 4x4 $E$ — Point in World (4x1) homogeneous

Lets construct the problem so that the world plane is at $z=0$. We can drop z-coordinates and the corresponding column of $E$:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \underbrace{\underset{3\times 4}{K} \; \underset{4\times 3}{\begin{bmatrix} R_{3\times 2} & T \\ 0 & 1 \end{bmatrix}}}_{H\text{, which is } 3\times 3!} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Note: H's scale is irrelevant since it affects not just u, v, but also w.

Lets assume we have point correspondences: $y_i \overset{\text{correspond}}{\Longleftrightarrow} x_i$

$y_i$: point in image; $x_i$: point on plane in world

We want to compute H, but what error function to use?

(A) How about $y_i - Hx_i$?
Won't work well — scale of H would matter.

(B) Lets consider each point $y_i$ (or $Hx_i$) as a vector.

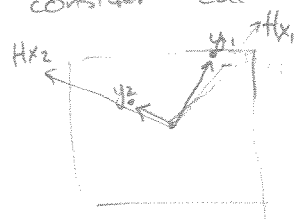



Image plane

We want $y_i$ and $Hx_i$ to have the same direction...
so:

$$e_i = y_i \underbrace{\times}_{\text{cross product}} Hx_i \qquad (\alpha \sin\theta,\ \theta \text{ angle between vectors})$$

Now, with $e_i = y_i \times Hx_i$, we can rewrite as

$$\underset{(3\times 1)}{e_i} = f_i(\underbrace{y_i, x_i}_{\text{These are known}}, \underbrace{h}_{9\times 1 \text{ unknown vector. Elements of } H})$$

$f$ is $3 \times 1$

- We build a least squares problem:

$$\begin{array}{c} \frac{\partial f_1}{\partial h} \rightarrow \\ \frac{\partial f_2}{\partial h} \rightarrow \\ \\ \end{array} \begin{bmatrix} \cdots \\ \cdots \\ \cdots \end{bmatrix} \begin{bmatrix} h \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdots \end{bmatrix}$$

$$J \quad h = 0$$

or: $J^T J h = J^T 0 = 0.$ ($J^T J$ matrix is $9\times 9$, no matter how many correspondences.)

- How many point correspondences do we need?

  $H$ has 8 DOF (scale ambiguity removes one)

  Each correspondence provides 3 eqns...

  So 3 correspondences? NO.

  Each correspondence yields only 2 independent eqns.

  Why? $Hx_i$ and $y_i$ are both in the plane — the direction of the cross product is constraine

- How do we solve?

  $Ah = 0$ trivial solution $h = 0$,

Our solution is $h$ = minimum eigen vector of $A$.

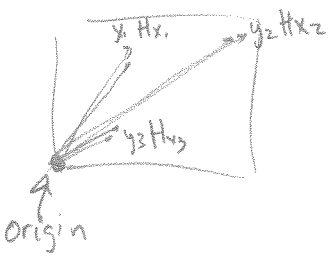
With 4 correspondences, $A$ has rank 8, but is $9\times 9$. So one eigenvector has $\lambda = 0$. With $>4$ correspondences, we pick the best $h$, but $Ah \approx 0$ only.

# Direct Linear Transform for Homographies

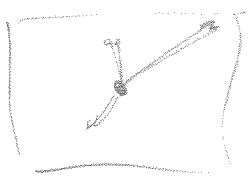
This method called <u>DLT</u>.

<u>Advantages</u>: Easy, Fast.
Can be used to bootstrap fancier methods.

<u>Disadvantages</u>:
- Wrong error function! Our error should be computed using reprojection error. (Assumption: our <u>noise</u> has to do with the accuracy with which we can measure $y_i$. Think blurry/noisy photo: feature correspondences will be worse). This is more like $|y_i - Hx_i|$ not $y_i \times Hx_i$
- Unenforced constraints, $H$ encodes a rotation, but we solved for 9 free parameters. May not correspond to a rigid rotation.
- Numerical precision and poor conditioning. Where is our origin when we treat $y_i$, $Hx_i$ as vectors?




or



← this is better; vectors $y_i$ and $Hx_i$ have bigger angles between them (shorter "lever" arm.).

Better yet, use centroid of $y$'s as origin.

$\Rightarrow$ With care, problem of conditioning is manageable. But this problem is an artifact of using the wrong cost function.

# Working backwards: Homography to Extrinsics.

- Suppose we know H. Can we recover E?

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \underbrace{\underbrace{\begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\substack{\text{Camera Matrix } 3\times 4 \\ K}} \underbrace{\begin{bmatrix} R_{3\times 2} & T \\ 0 & 1 \end{bmatrix}}_{\substack{\text{Extrinsics minus the } z \text{ column, } 4\times 3}}}_{\text{make equal to } H!} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Assume known K, this is straight-forward... mostly.

For example: $H_{11} = f_x R_{11} + c_x R_{31}$

$H_{31} = R_{31}$

Similarly, recover $R_{i,j}$ $T_x, T_y, T_z$.

- A couple wrinkles. H was only known to scale... but we need $|R| = 1$. So find/apply scale factor that fixes it!

- What about the 3rd column of R? (That we removed earlier since $z=0$)
  - R is orthonormal, so each column is perpendicular to the other 2.
  - Solve for $R(:,3) = R(:,1) \underbrace{\times}_{\text{cross product}} R(:,2)$ } make sure you have $|R|=1$, not $|R|=-1$.

- Are your Rs really perpendicular? Almost certainly not with DLT!

  Use Polar decomposition. Begin with SVD: $R = USV^T$

  U & V are proper rotation matrices. Only S is a problem!

  $\Rightarrow$ Improved $R = UV^T$.