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 node_num)
  void connect(int node_a, int node_b)
  ```

  E.g. minimum node id reachable from node_num.

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

<table>
<thead>
<tr>
<th>node id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>step</td>
<td>(0,2)</td>
<td>(2,3)</td>
<td>(3,6)</td>
<td>(4,7)</td>
<td>(5,8)</td>
<td>(5,6)</td>
<td>(6,9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

connect (a, b)

ra = get_rep(a) \leftarrow \text{both root nodes}
rb = get_rep(b)

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
Flattens tree!

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

Runtime complexity (with flattening & a good merging policy)

$O(\alpha(n))$

\text{Inverse Ackermann}

\text{Ackerman}(4) \approx 2^{6.0}

\text{Big, so } \alpha(n) < 5

for any \text{n}. 
Segmentation: Finding Threshold?

Idea: set threshold dynamically based on properties of a cluster,

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

\[ \approx \begin{cases} \text{max edge cost already accepted} \end{cases} + \frac{K}{\text{segment size}} \]

Threshold for a segment:

\[ T \]

- with low internal variability (sky)
- high variability (grass)

- process all possible
- reject edges with cost > per-segment cost.

Felzenszwalb, Huttenlocher (IJCV 2004)
Computing Homographies

A picture (projective view) of a plane.

\[
\begin{pmatrix}
U \\
V \\
W
\end{pmatrix}
= 
\begin{pmatrix}
K & R & T \\
0 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix}
\]

\(K\) is a 3x3 matrix, \(R\) is a 3x3 rotation matrix, \(T\) is a 3x3 translation matrix, and \(X, Y, Z\) are the coordinates in the world.

Let's construct the problem so that the world plane is at \(Z = 0\). We can drop \(Z\)-coordinates and the corresponding column of \(E\).

\[
\begin{pmatrix}
U \\
V \\
W
\end{pmatrix}
= 
\begin{pmatrix}
R & T \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
X \\
Y \\
1
\end{pmatrix}
\]

Note: \(H\)’s scale is irrelevant since it affects not just \(u, v\), but also \(w\).

Let's assume we have point correspondences: \(Y_i \leftrightarrow X_i\)

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. Let's consider each point \(Y_i\) (or \(HX_i\)) as a vector,

We want \(Y_i\) and \(HX_i\) to have the same direction...

So:

\[e_i = Y_i \times HX_i\] (cross product of vectors)
Now, with \( e_i = y_i \times Hx_i \), we can rewrite as
\[
e_i = f_i(y_i, x_i, h)
\]
\[(3x1)\] These are \( 9x1 \) unknown vector. Elements of \( H \)
\( f \) is \( 3 \times 1 \)

We build a least squares problem:

\[
\begin{align*}
\frac{2h_1}{dh} & \rightarrow \frac{2h_2}{dh} \\
& \vdots \\
Jh &= 0 \\
\text{or: } J^TJh &= J^TO = 0. \quad \left( J^TJ \text{ matrix is } 9x9, \right) \\
& \quad \text{no matter how many} \\
& \quad \text{correspondences.}
\end{align*}
\]

How many point correspondences do we need?

\( H \) has 8 DOF (Scale ambiguity removes one)

Each correspondence provides 3 eqns...

So 3 correspondences? \( \text{No.} \)

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 constrained

How do we solve?

\[
Ah = 0 \quad \text{trivial solution } h = 0,
\]

Our solution is \( h = \text{minimum eigenvector of } A \).

With 4 correspondences, \( A \) has rank 8, but is 9x1. 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 DLT.

Advantages: Easy, Fast
Can be used to bootstrap fancier methods.

Disadvantages:
- Wrong error function! Our error should be computed using reprojection error. (Assumption: our noise has to do with the accuracy with which we can measure yi. Think blurry/noisy photo: feature correspondences will be worse). This is more like \(|yi - Hxi|\), not \(yi \times Hxi\).

- 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 \(yi, Hxi\) as vectors?

Better yet, use centroid of \(yi\)'s as origin.

⇒ With care, problem of conditioning is manageable. But this problem is an artifact of using the wrong cost function.
Suppose we know $H$. Can we recover $E$?

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & c_x & 0 \\ f_y & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{3x3} & T_z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Camera Matrix $3x4$

Extrinsics minus the $z$ column, $4x3$

Make equal to $H$!

Assume known $K$, this is straightforward... mostly.

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

$H_{31} = R_{31}$

A couple wrinkles. $H$ was only known to scale... but we need $|R|=1$. So find 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 $R(:,3) = R(:,1) \times R(:,2)$

> Make sure you have $|R|=1$, not $|R|=-1$.

Are your $R$s really perpendicular? Almost certainly not with DLT!

Use Polar decomposition. Begin with SVD: $R = U S V^T$

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

Improve $R = U V^T$. 