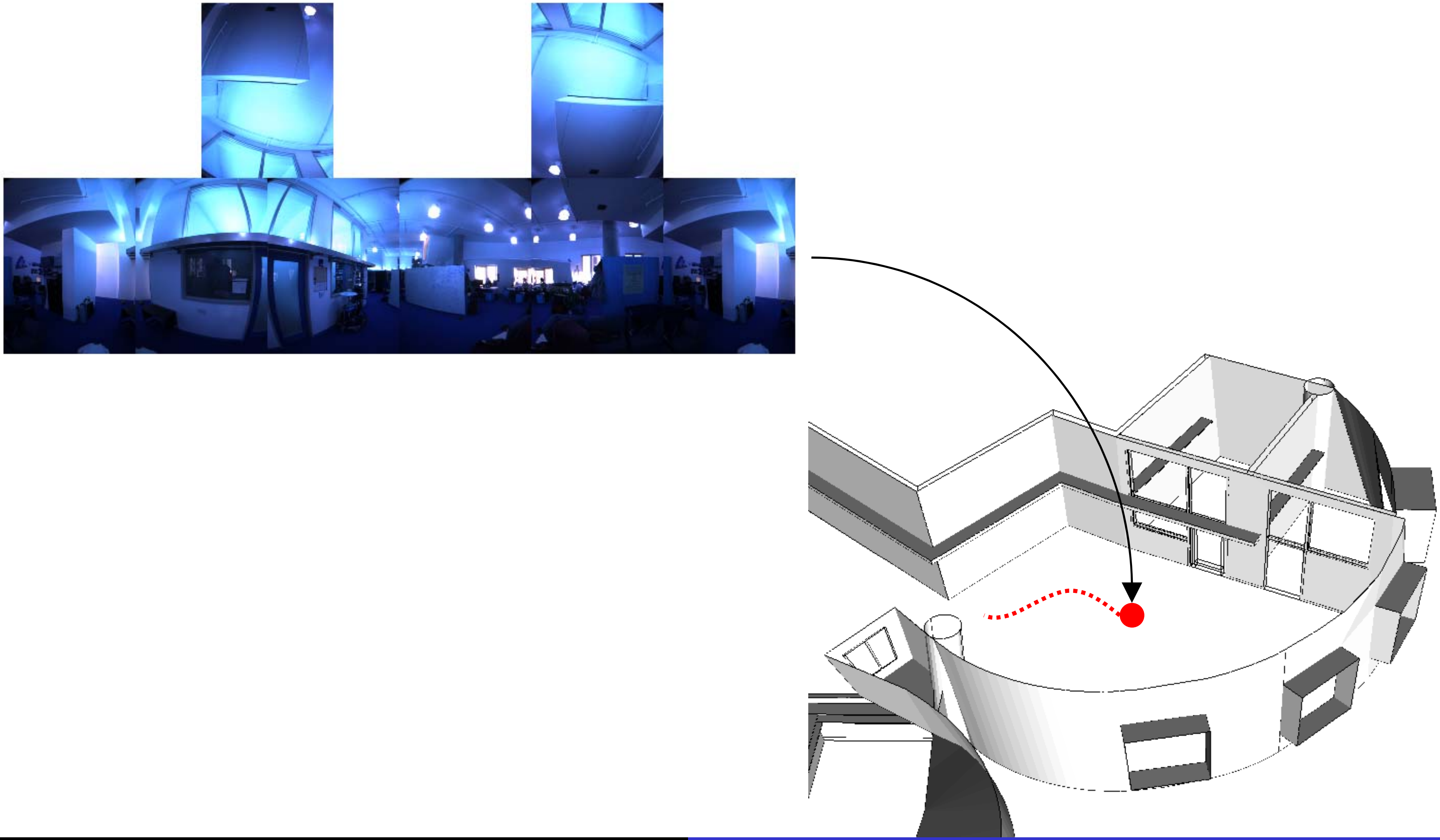
Wide-Area 3D Tracking From Omnivision and 3D Structure

Olivier Koch, Seth Teller

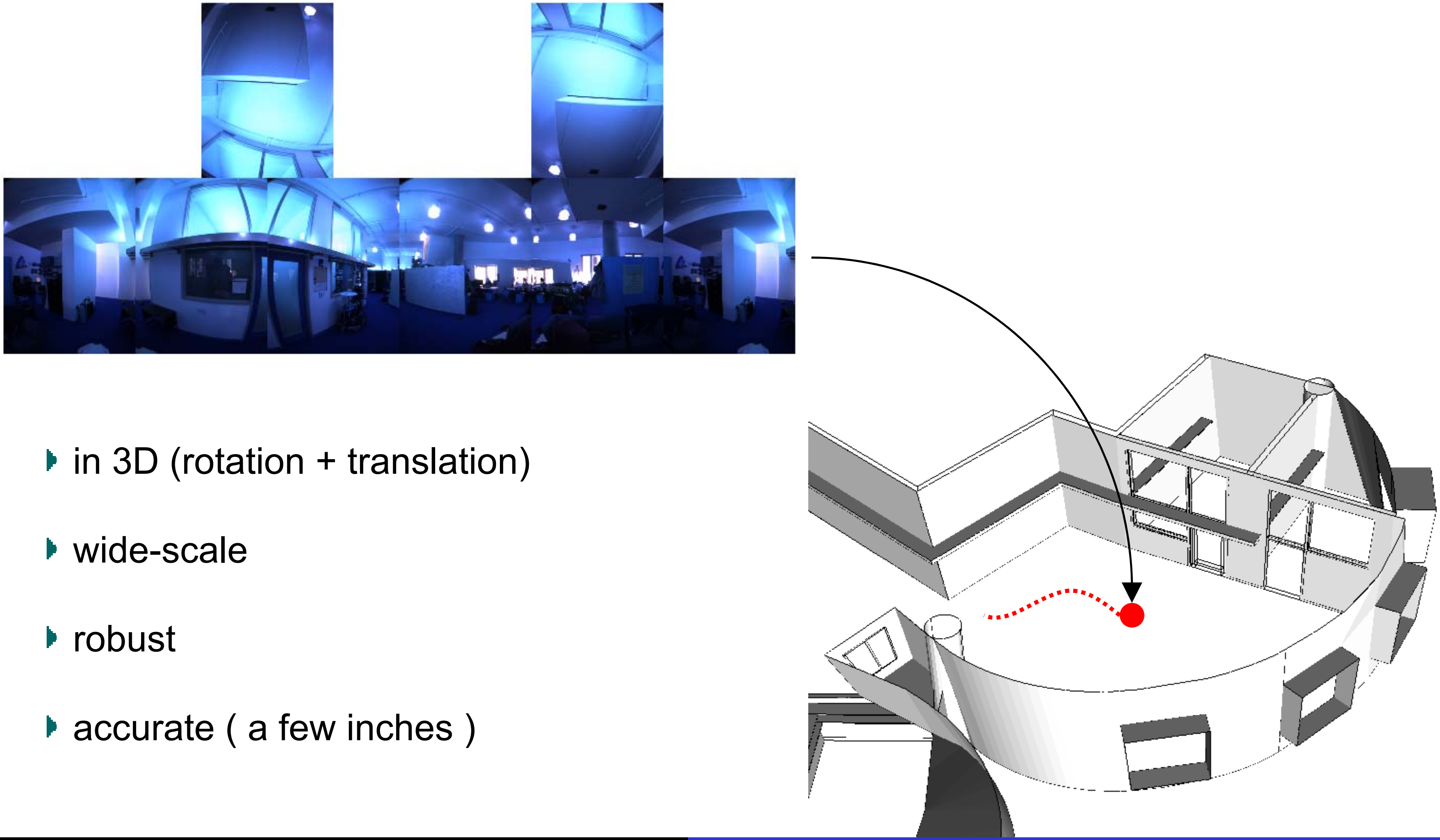
Problem Statement:

Track an omnivision camera given a coarse 3D structure of the environment.



Problem Statement:

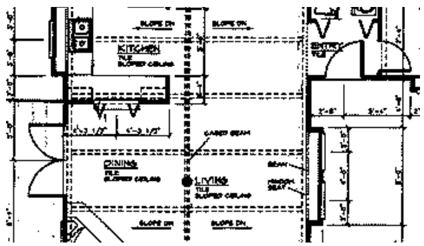
Track an omnivision camera given a coarse 3D structure of the environment.

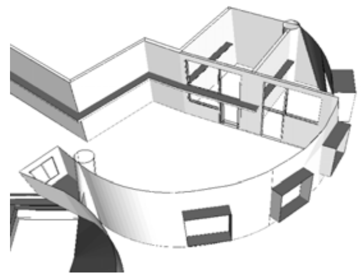


- in 3D (rotation + translation)
- wide-scale
- robust
- accurate (a few inches)

- Application 1: Mobile Robotics

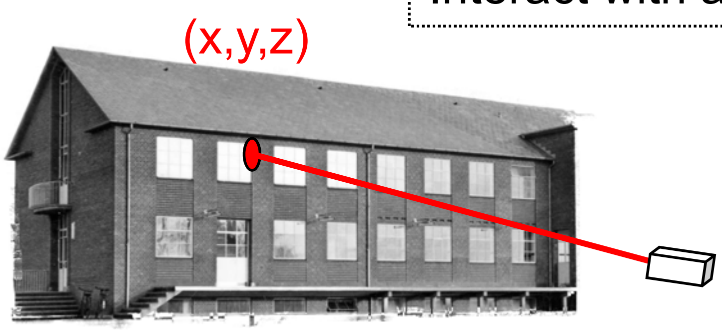
Localize a mobile robot given a 3D map or a 2D blue print.





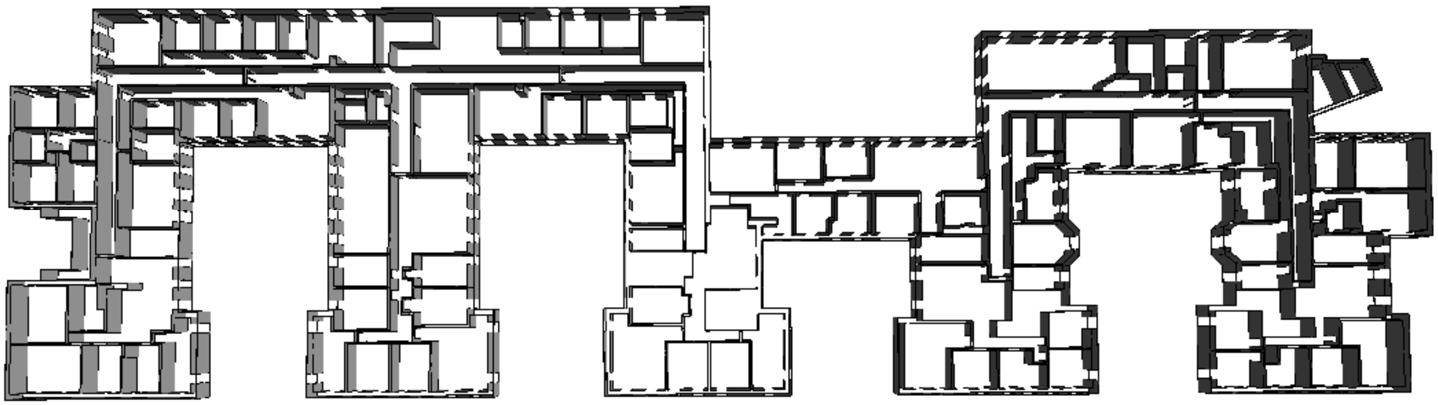
- Application 2: Pervasive Computing *(the science of tiny mobile, embedded devices)*
  Augmented Reality

Interact with a wide 3D structure (architects, engineers, etc.)

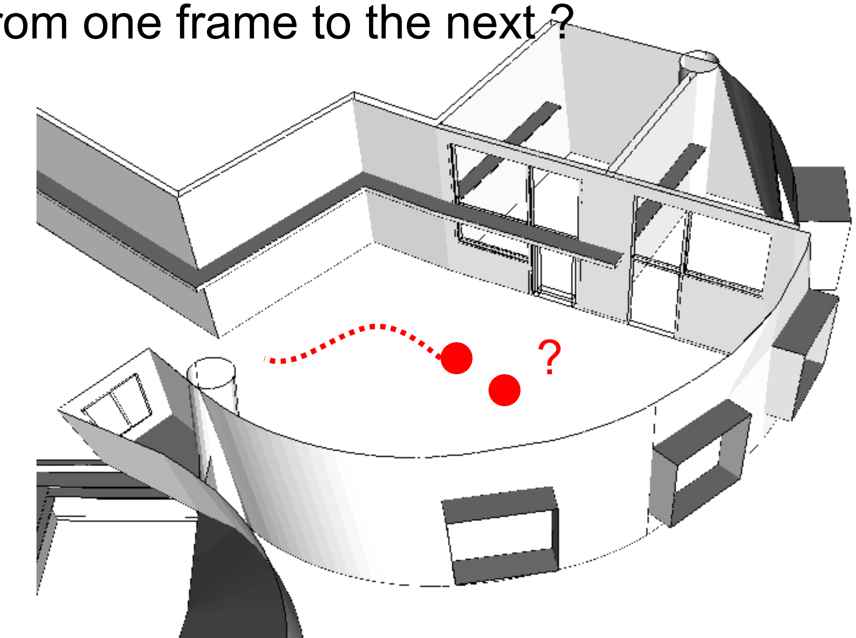


- Detect hidden structures (pipes, etc.)
- Measure volumes and surfaces
- Operate lights, windows, doors

INIT : where is the camera in the global map?



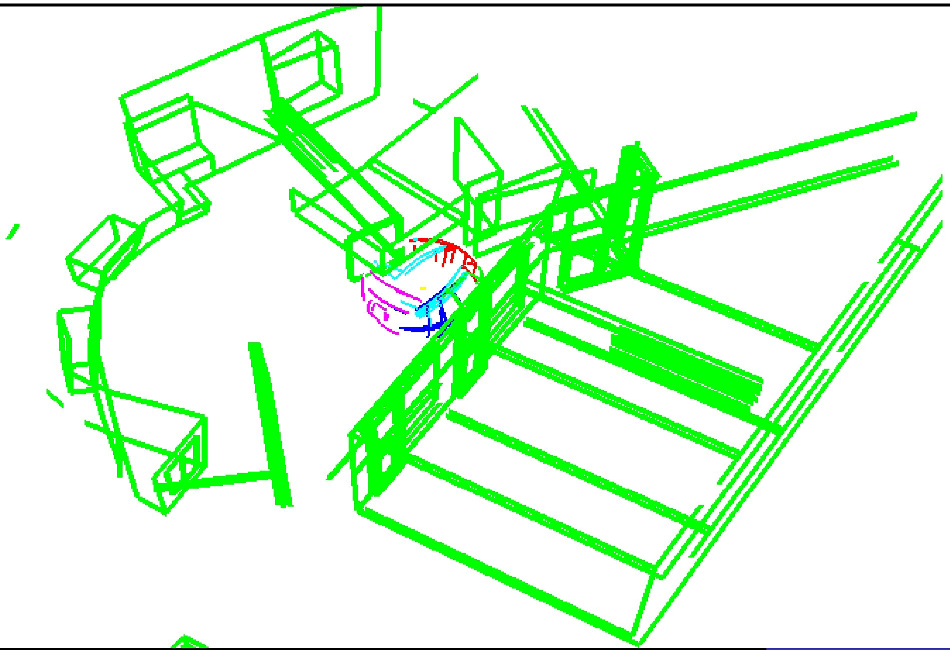
MAINTENANCE: how do I track the camera from one frame to the next ?



Approach: compute correspondences between image lines and model lines.

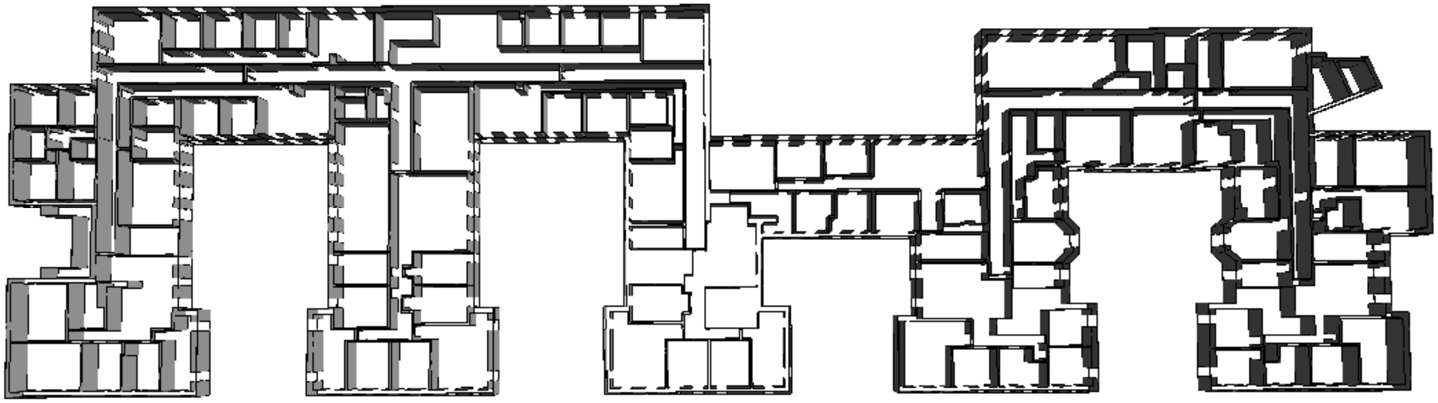
Image coordinate frame

Camera coordinate frame

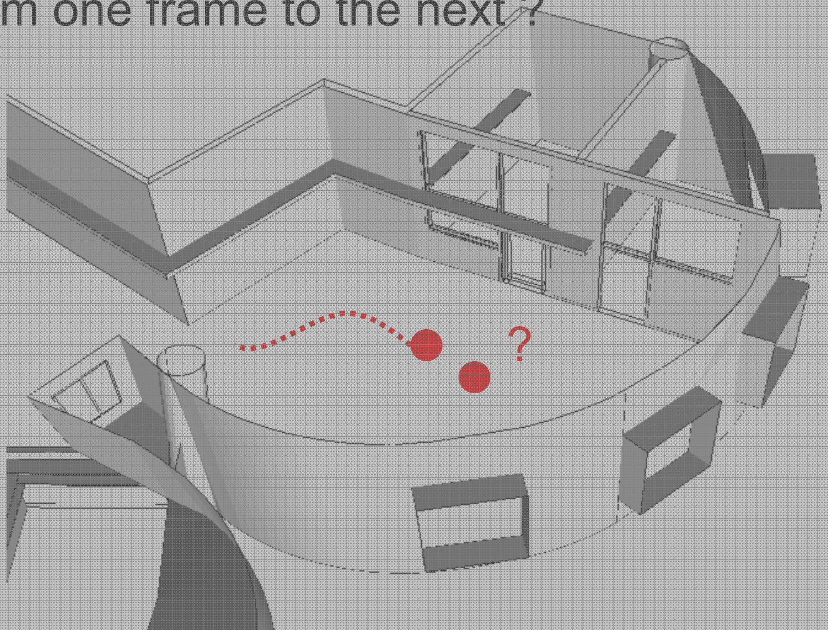


From 4 correspondences, we can uniquely determine the camera pose (rotation and translation).

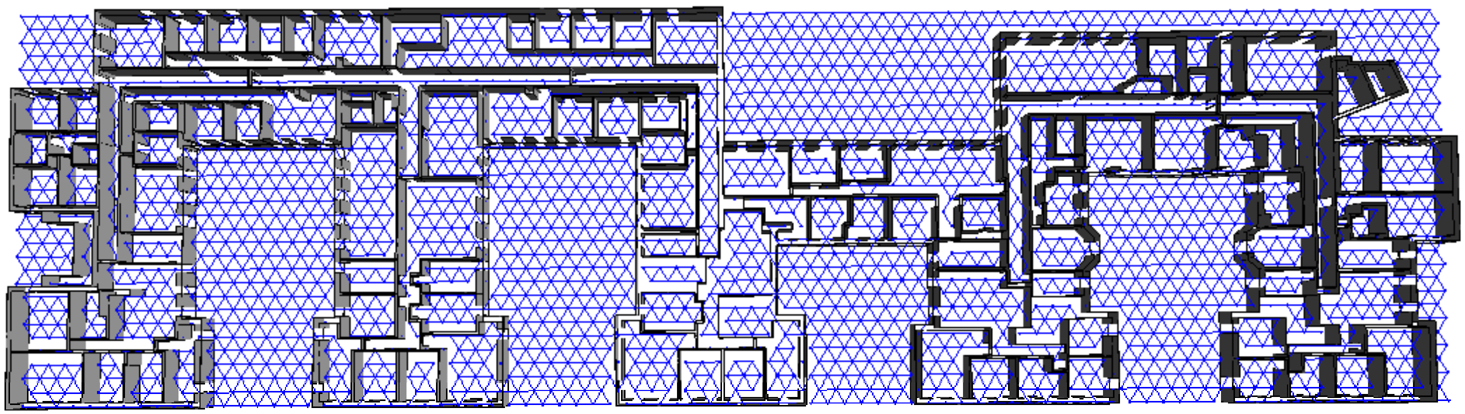
INIT : where is the camera in the global map?



MAINTENANCE: how do I track the camera from one frame to the next ?



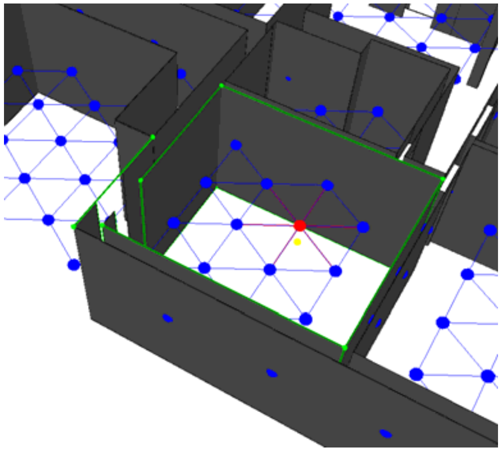
INIT : where is the camera in the global map?



At each node:

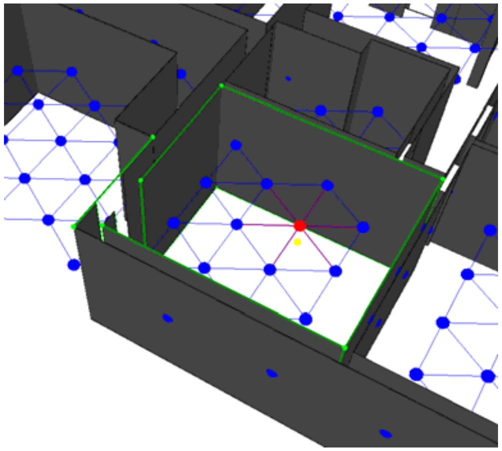
- Compute a correlation function between the observed lines and the expected lines.
- Keep the top-K nodes and run the next steps on them.

INIT : where is the camera in the global map?



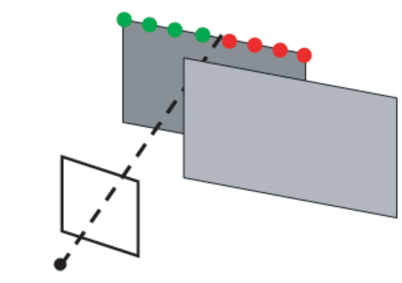
- Consider the set of visible model lines (in green)
- Match them with the observed edges
- Compute the “best” camera pose

INIT : where is the camera in the global map?

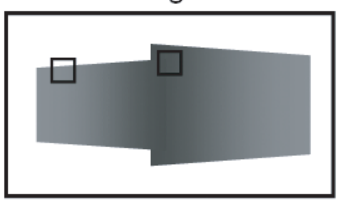


- <u>Consider the set of visible model lines (in green)</u>
- Match them with the observed edges
- Compute the "best" camera pose

- Render all model faces using openGL
- Each face is rendered with a unique RGB color
- At each pixel, determine which face is visible
- Use depth to determine which lines are visible

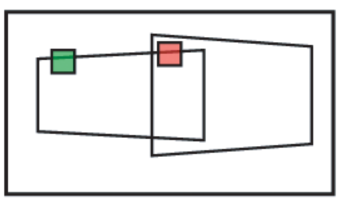


Z-buffer image



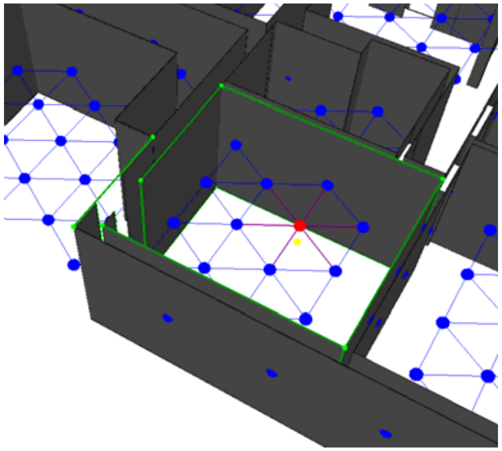
Each pixel contains the image depth.

Feedback buffer



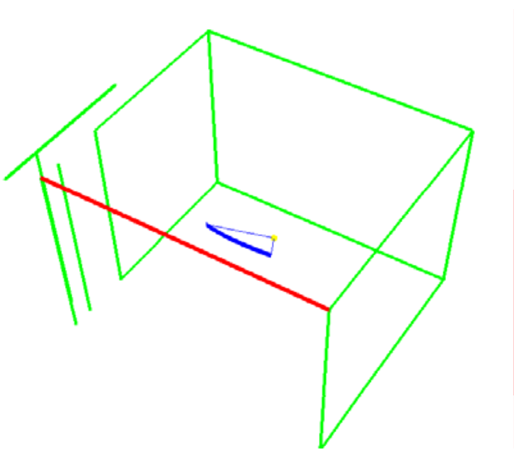
Each pixel contains:
- 1 if the pixel belongs to a line
- the depth if the line were to be displayed

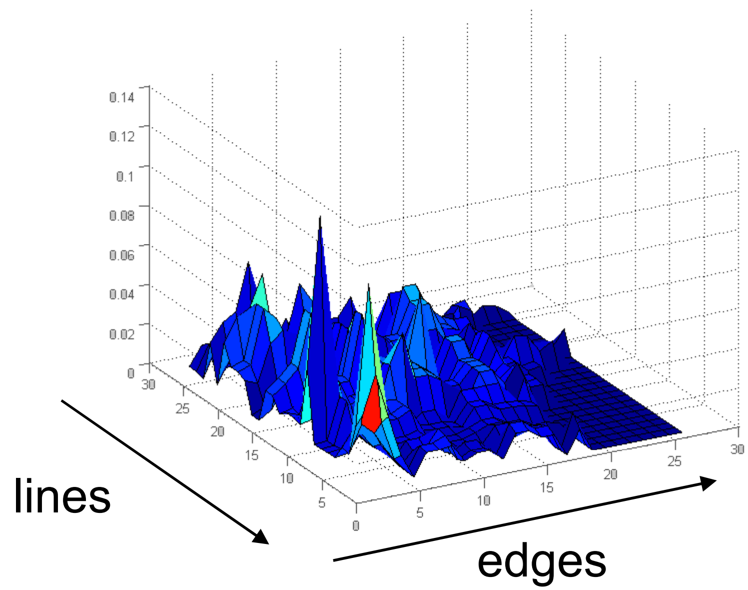
INIT : where is the camera in the global map?



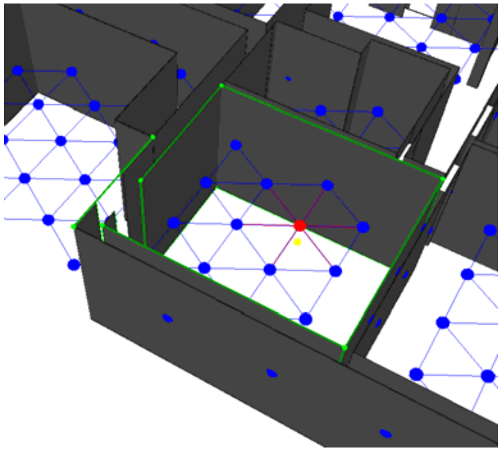
- Consider the set of visible model lines (in green)
- Match them with the observed edges
- Compute the "best" camera pose

1. Specify an error ellipse on the camera position
2. For each <line,edge> pair, compute a correspondence likelihood
3. For each line, consider the best image edge candidates and compute the camera pose



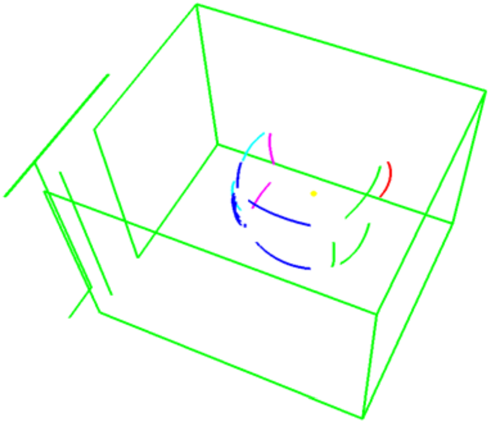


INIT : where is the camera in the global map?

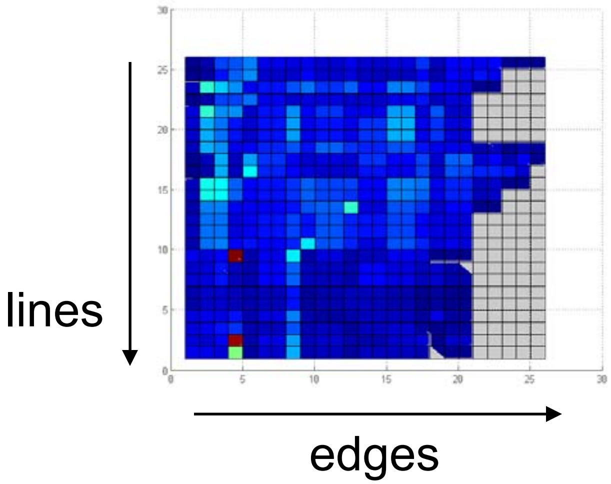


- Consider the set of visible model lines (in green)
- Match them with the observed edges
- Compute the "best" camera pose

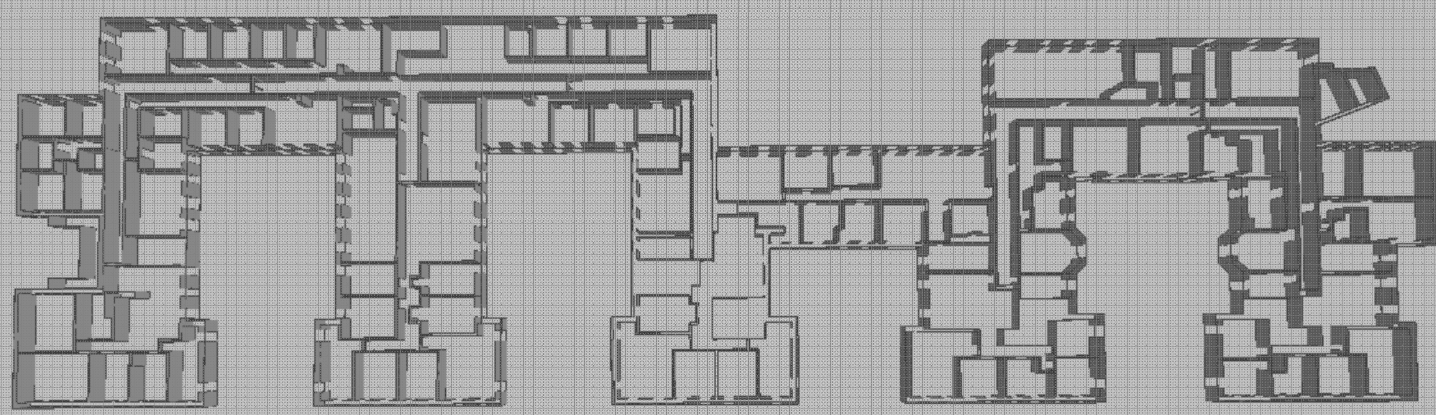
1. Compare each image edge with each model line
2. Accumulate the scores in a table
3. Compute a global score from the table



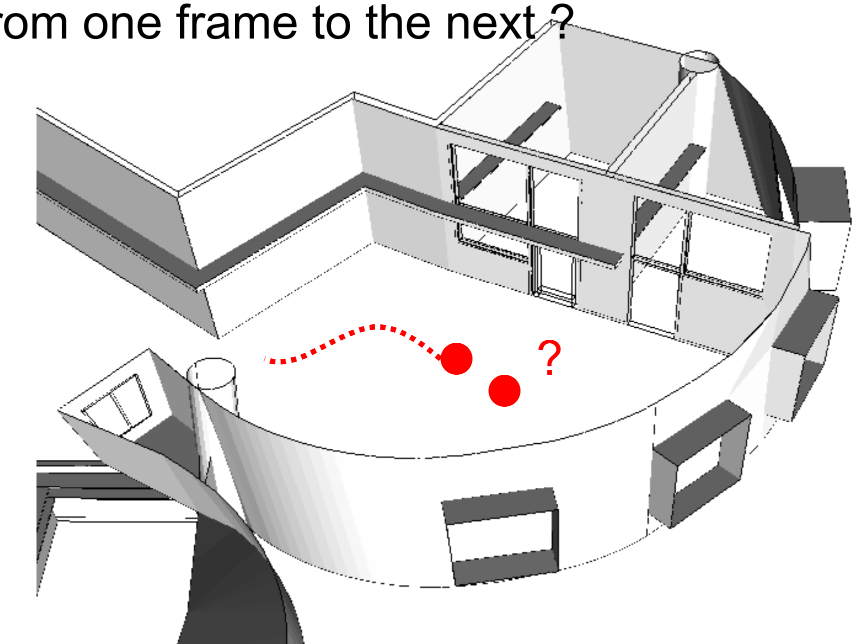
$$S = \sum \max s_{i,j} - \sum N_{\text{edges no match}}$$



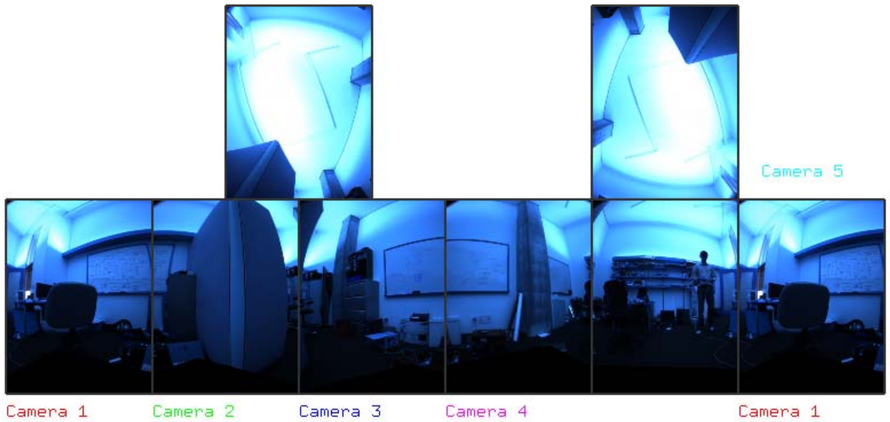
INIT : where is the camera in the global map?

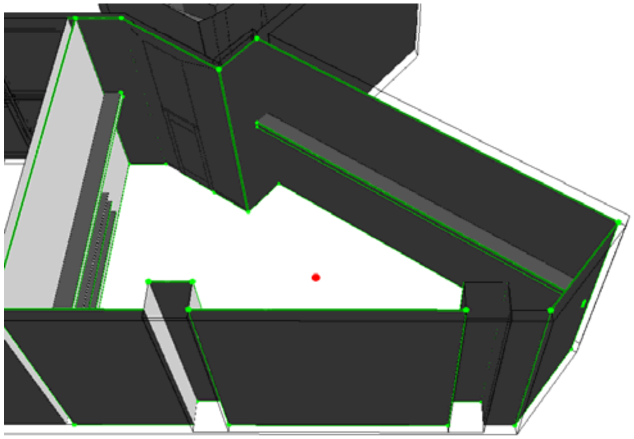


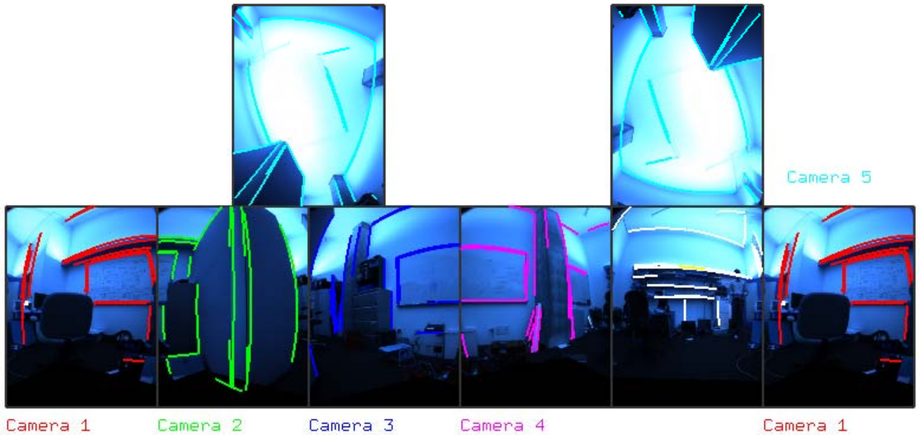
MAINTENANCE: how do I track the camera from one frame to the next ?



MAINTENANCE: how do I track the camera from one frame to the next ?

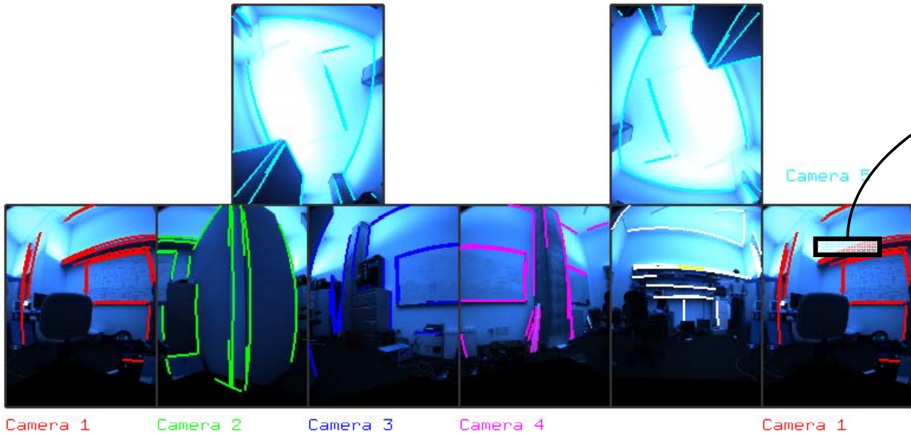


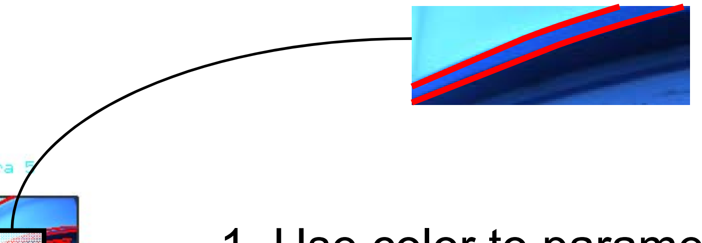




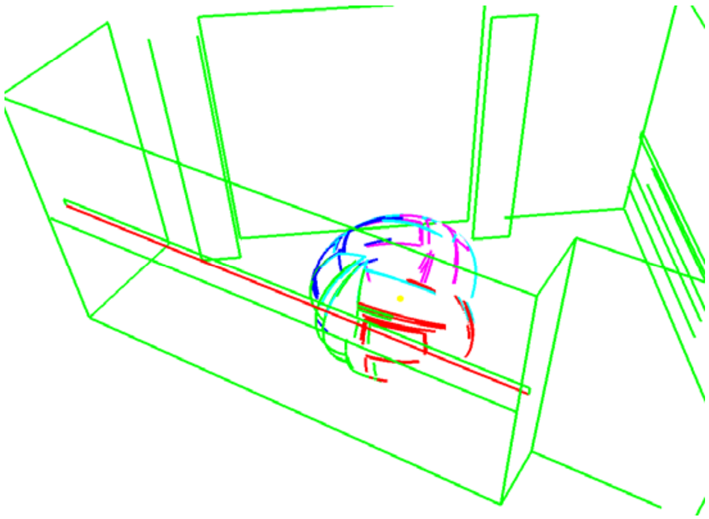
- Update each correspondence with the closest-neighbor edge
- Compute the new camera position from correspondences
- … doesn't work!!!
- Snaps on the wrong model lines

MAINTENANCE: how do I track the camera from one frame to the next ?





1. Use color to parameterize edges.



2. Run a multi-hypothesis model where each model line has several candidates matches on the image.

At each frame, keep the hypothesis with highest score.