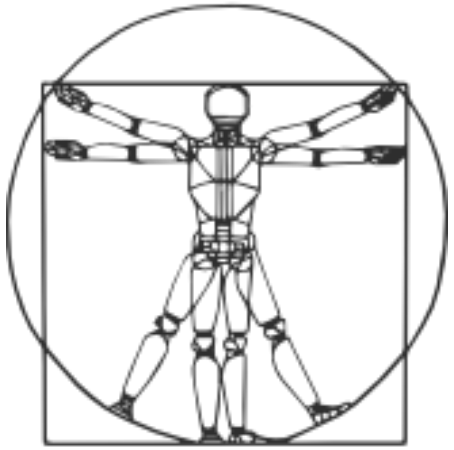
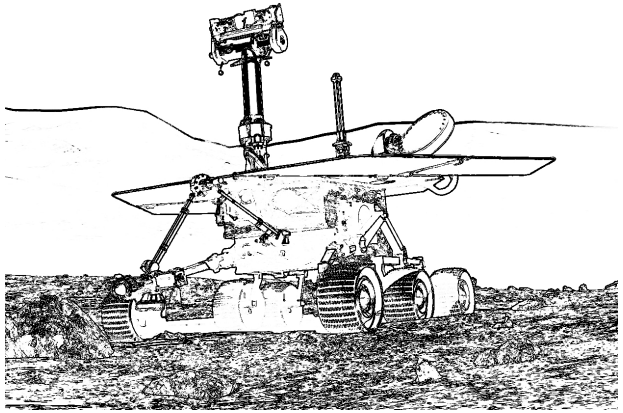




COMP417
Introduction to Robotics and Intelligent Systems

Lecture 19: Applying the EKF on Examples
Visual SLAM

David Meger

Nov 26, 2019

McGill

MRL Mobile Robotics Lab at McGill University

KF vs EKF

Initial Gaussian Belief

$$bel(x_0) \sim \mathcal{N}(\mu_{0|0}, \Sigma_{0|0})$$

Linear Motion w/Gaussian Noise

$$x_{t+1} = Ax_t + Bu_t + Gw_t$$
$$w_t \sim \mathcal{N}(0, Q)$$

Linear Measurement w/Gaussian Noise

$$z_t = Hx_t + n_t$$
$$n_t \sim \mathcal{N}(0, R)$$

Prediction Step

$$\mu_{t+1|t} = A\mu_{t|t} + Bu_t$$
$$\Sigma_{t+1|t} = A\Sigma_{t|t}A^T + GQG^T$$

Update Step

Received measurement $\bar{z}_{t+1}$ but expected $\mu_{z_{t+1}} = H\mu_{t+1|t}$

Prediction residual is a Gaussian random variable $\delta z \sim \mathcal{N}(\bar{z}_{t+1} - \mu_{z_{t+1}}, S_{t+1})$

with covariance: $S_{t+1} = H\Sigma_{t+1|t}H^T + R$

Kalman Gain (optimal correction factor): $K_{t+1} = \Sigma_{t+1|t}H^T S_{t+1}^{-1}$

We update the mean and covariance by:

$$\mu_{t+1|t+1} = \mu_{t+1|t} + K(\bar{z}_{t+1} - \mu_{z_{t+1}})$$
$$\Sigma_{t+1|t+1} = \Sigma_{t+1|t} - KH\Sigma_{t+1|t}$$

KF vs EKF

Initial Gaussian Belief

$$bel(x_0) \sim \mathcal{N}(\mu_{0|0}, \Sigma_{0|0})$$

Non-linear Motion w/Gaussian Noise

$$x_{t+1} = f(x_t, u_t) + Gw_t$$
$$w_t \sim \mathcal{N}(0, Q)$$

Non-linear Measurement w/Gaussian Noise

$$z_t = h(x_t) + n_t$$
$$n_t \sim \mathcal{N}(0, R)$$

Linearize:

$$x_{t+1} \equiv F_t x_t + \bar{u}_t + w_t$$

$$z_{t+1} = H_{t+1} x_{t+1} + \bar{c}_{t+1} + n_{t+1}$$

Prediction Step

$$\mu_{t+1|t} = f(\mu_{t|t}, u_t)$$
$$\Sigma_{t+1|t} = F_t \Sigma_{t|t} F_t^T + GQG^T$$

Update Step

Received measurement $\bar{z}_{t+1}$ but expected $\mu_{z_{t+1}} = h(\mu_{t+1|t})$

Prediction residual is a Gaussian random variable $\delta z \sim \mathcal{N}(\bar{z}_{t+1} - \mu_{z_{t+1}}, S_{t+1})$

with covariance: $S_{t+1} = H\Sigma_{t+1|t}H^T + R$

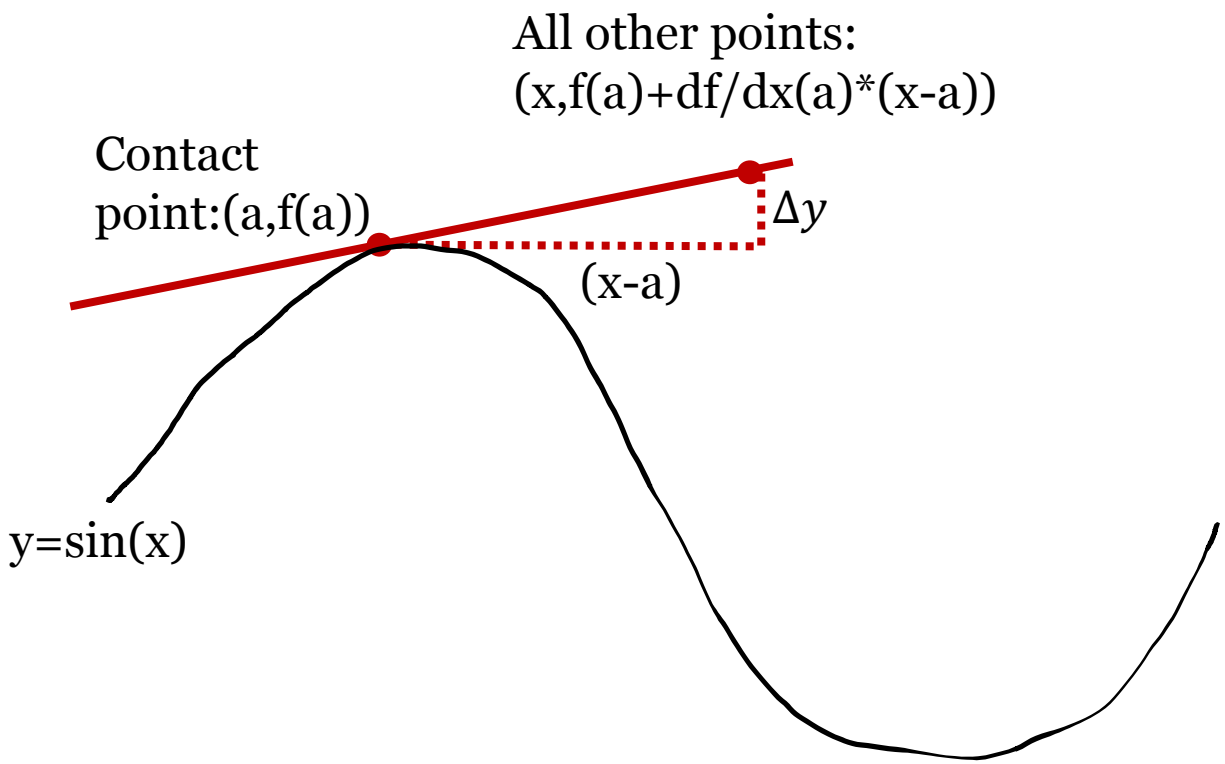
Kalman Gain (optimal correction factor): $K_{t+1} = \Sigma_{t+1|t} H^T S_{t+1}^{-1}$

We update the mean and covariance by:

$$\mu_{t+1|t+1} = \mu_{t+1|t} + K(\bar{z}_{t+1} - \mu_{z_{t+1}})$$
$$\Sigma_{t+1|t+1} = \Sigma_{t+1|t} - KH\Sigma_{t+1|t}$$

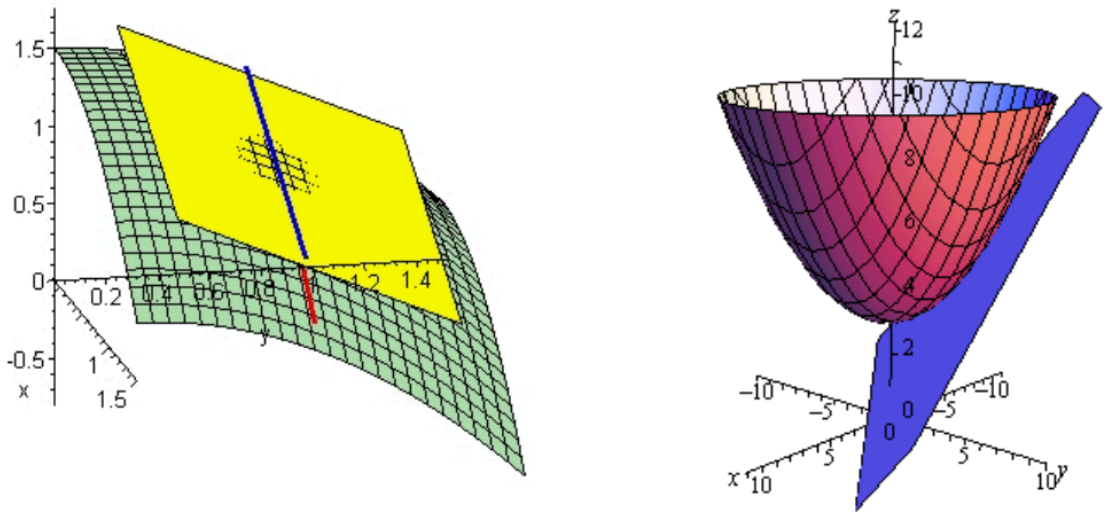
Recall: Linearization in 1D

- The red line approximates the non-linear curve near x=a
- Here df/dx = cos(x), still a function of x
- To get the constant slope of the red line, we substitute x with its value at the approximation point, a:
  - The line shown has slope cos(a)
  - Approximated y value is f(a)+cos(a)*(x-a). This equals f(a) at a, and is "near" sin(x) around a
- Far from a, this is a terrible approximation, so we want to recompute often



Reminder: Jacobian Matrix

- It is the orientation of the tangent plane to the vector-valued function at a given point



- Generalizes the gradient of a scalar valued function

Reminder: Jacobian Matrix

- It is a **non-square matrix** $m \times n$ in general
- Given a vector-valued function

$$g(x) = \begin{pmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_m(x) \end{pmatrix}$$

- The **Jacobian matrix** is defined as

$$G_x = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \frac{\partial g_m}{\partial x_2} & \cdots & \frac{\partial g_m}{\partial x_n} \end{pmatrix}$$

Number of rows equals the number of vector components in g(x).

Number of columns equals the number of "w.r.t." dimensions.

Computing the Jacobian

- Write f as a vector function: $f(x_1, x_2, u) = [x_1 + x_2^2,\ x_2 + 3u,\ x_1^4 - u^2] \in \mathbb{R}^3$
- Apply the gradient definition:
  - Partials of each component of f

$$\frac{\partial f}{\partial x_{1:2}}(\mu_1, \mu_2, u_1) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} \end{bmatrix} (\mu_1, \mu_2, u_1)$$

- This can still be a function of x and u:
  - $\frac{\partial f_1}{\partial x_1} = 1$
  - $\frac{\partial f_1}{\partial x_2} = 2x$ etc..
- To get a constant slope substitute
  - The value of the approximation point: (μ_1, μ_2, u)

$$\frac{\partial f}{\partial x_{1:2}}(\mu_1, \mu_2, u_1) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} \end{bmatrix} (\mu_1, \mu_2, u_1)$$

$$= \begin{bmatrix} 1 & 2\mu_2 \\ 0 & 1 \\ 4\mu_1^3 & 0 \end{bmatrix}$$

Note in this example we have no “u” in the result. In general it can appear here, such as if an entry in f is x1*u.

How do we linearize for the EKF?

- Using the first order Taylor expansion around the mean of the previous update step's state estimate:

$$
\begin{aligned}
x_{t+1} &= f(x_t, u_t) + w_t \\
&\approx f(\mu_{t|t}, u_t) + \frac{\partial f}{\partial x}(\mu_{t|t}, u_t)(x_t - \mu_{t|t}) + w_t \\
&= f(\mu_{t|t}, u_t) + F_t(x_t - \mu_{t|t}) + w_t \\
&= F_t x_t + \boxed{f(\mu_{t|t}, u_t) - F_t \mu_{t|t}} + w_t \\
&= F_t x_t + \bar{u}_t + w_t
\end{aligned}
$$

Constant term with respect to the state

Recall how to compute the Jacobian matrix. For example, if

$$f(x_1, x_2, u) = [x_1 + x_2^2,\ x_2 + 3u,\ x_1^4 - u^2] \in \mathbb{R}^3$$

then the Jacobian of f with respect to (x_1, x_2) at (μ_1, μ_2, u) is

$$
\begin{aligned}
\frac{\partial f}{\partial x_{1:2}}(\mu_1, \mu_2, u_1) &= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} \end{bmatrix} (\mu_1, \mu_2, u_1) \\
&= \begin{bmatrix} 1 & 2\mu_2 \\ 0 & 1 \\ 4\mu_1^3 & 0 \end{bmatrix}
\end{aligned}
$$

How do we linearize for the EKF?

- Using the first-order Taylor expansion around the mean of the previous prediction step's state estimate:

$$
\begin{aligned}
z_{t+1} &= h(x_{t+1}) + n_{t+1} \\
&\approx h(\mu_{t+1|t}) + \frac{\partial h}{\partial x}(\mu_{t+1|t})(x_t - \mu_{t+1|t}) + n_{t+1} \\
&= h(\mu_{t+1|t}) + H_{t+1}(x_{t+1} - \mu_{t+1|t}) + n_{t+1} \\
&= H_{t+1}x_{t+1} + \boxed{h(\mu_{t+1|t}) - H_{t+1}\mu_{t+1|t}} + n_{t+1} \\
&= H_{t+1}x_{t+1} + \bar{c}_{t+1} + n_{t+1}
\end{aligned}
$$

Constant term with respect to the state

Recall how to compute the Jacobian matrix. For example, if

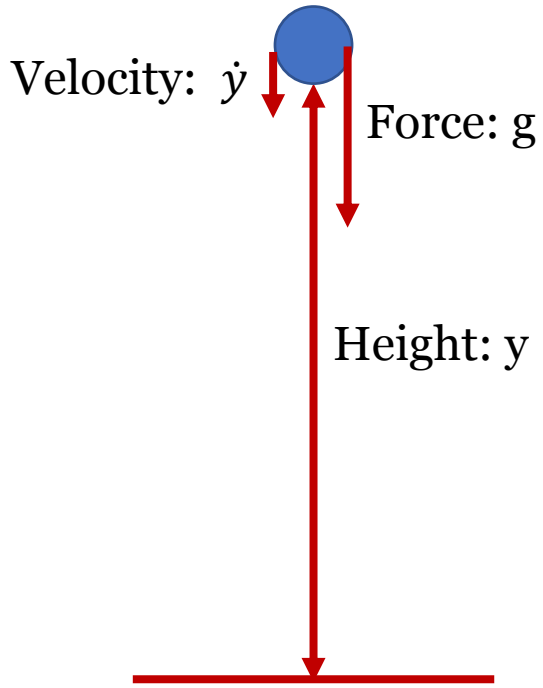
$$h(x_1, x_2) = [x_1 + x_2^2,\ x_2,\ x_1^4] \in \mathbb{R}^3$$

then the Jacobian of f with respect to (x_1, x_2) at (μ_1, μ_2) is

$$
\begin{aligned}
\frac{\partial h}{\partial x_{1:2}}(\mu_1, \mu_2) &= \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} \\ \frac{\partial h_3}{\partial x_1} & \frac{\partial h_3}{\partial x_2} \end{bmatrix} (\mu_1, \mu_2) \\
&= \begin{bmatrix} 1 & 2\mu_2 \\ 0 & 1 \\ 4\mu_1^3 & 0 \end{bmatrix}
\end{aligned}
$$

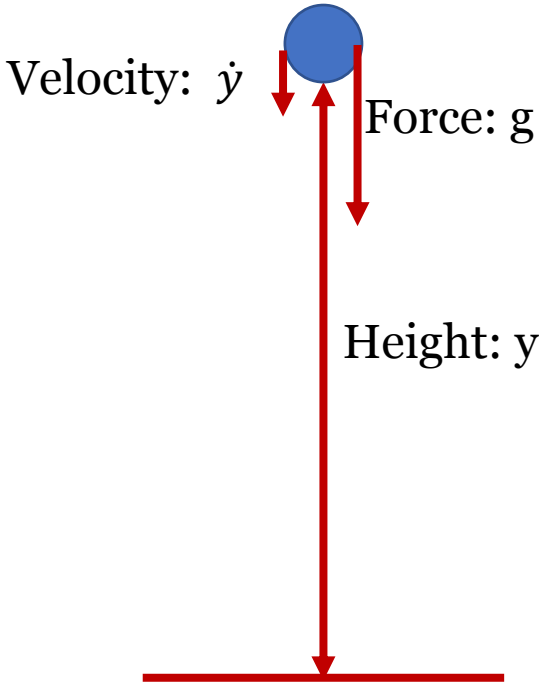
Example 1: Linear Motion

- y(t+1) = y(0) + $\dot{y}$(t)*dt – 0.5*g*(dt)^2
- $\dot{y}$(t+1) = $\dot{y}$(t) – g*dt
- For state x = $[y, \dot{y}]'$, write the vector valued transition function:
  - Assume dt = 1.0
  - $x_{t+1} = \begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix} x_t + \begin{bmatrix} b1 \\ b2 \end{bmatrix} u_t$
- Likewise the vector-valued measurement, given we measure z=y:
  - $z_t = \begin{bmatrix} h1 & h2 \end{bmatrix} x_t$



Example 1: Linear Motion

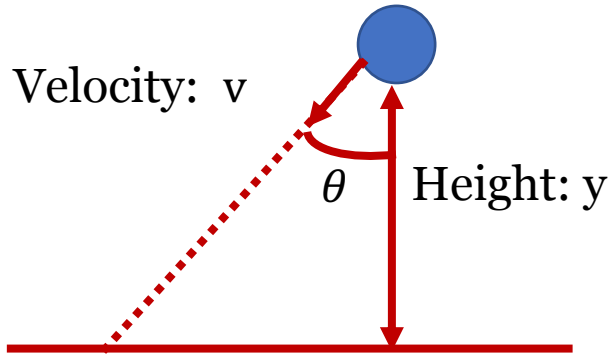
- $y(t+1) = y(0) + \dot{y}(t)*dt - 0.5*g*(dt)^2$
- $\dot{y}(t+1) = \dot{y}(t) - g*dt$
- For state $x = [y, \dot{y}]'$, write the vector valued transition function:
  - Assume dt = 1.0
  - $x_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} -0.5 \\ -1 \end{bmatrix} u_t$
- Likewise the vector-valued measurement, given we measure z=y:
  - $z_t = \begin{bmatrix} 1 & 0 \end{bmatrix} x_t$



NOTE: This is already linear, no need for EKF.

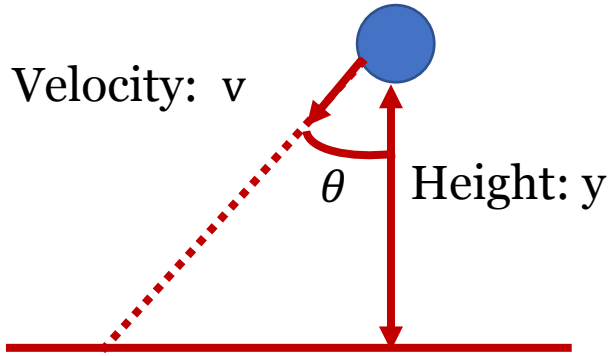
Example 2: Diagonal Motion

- Assume we still move straight, but now on a non-zero angle, theta w.r.t. gravity
- We have a fixed velocity v (there is no control)
- We measure the vertical height
- With state x = $[y, \theta, v]'$, write the motions and measurement equations and derive their Jacobians.



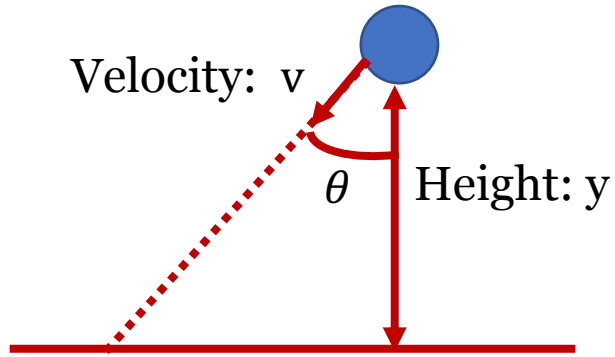
Example 2: Diagonal Motion

- $\mathrm{x} = [y, \theta, v]'$
- $x_{t+1} = \begin{bmatrix} y + v\cos(\theta) \\ \theta \\ v \end{bmatrix}$
- $z_t = [y]$



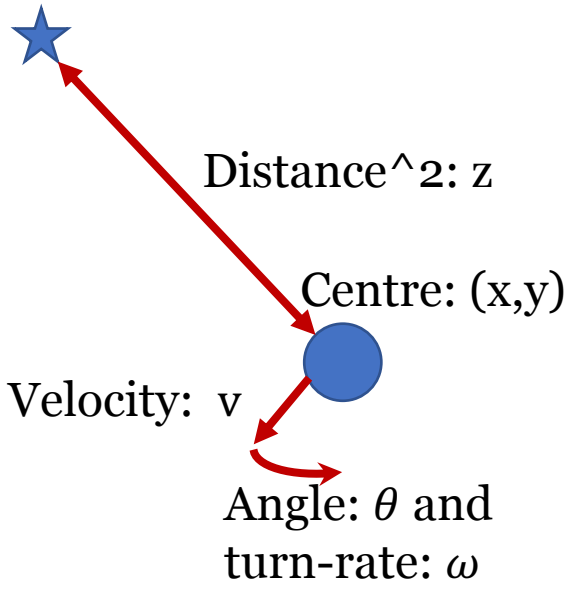
Example 2: Diagonal Motion

- $\mathrm{x} = [y, \theta, v]'$
- $x_{t+1} = \begin{bmatrix} y + v\cos(\theta) \\ \theta \\ v \end{bmatrix}$
- $z_t = [y]$
- $F_t = \begin{bmatrix} 1 & -v\sin(\theta) & \cos(\theta) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- $H_t = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$



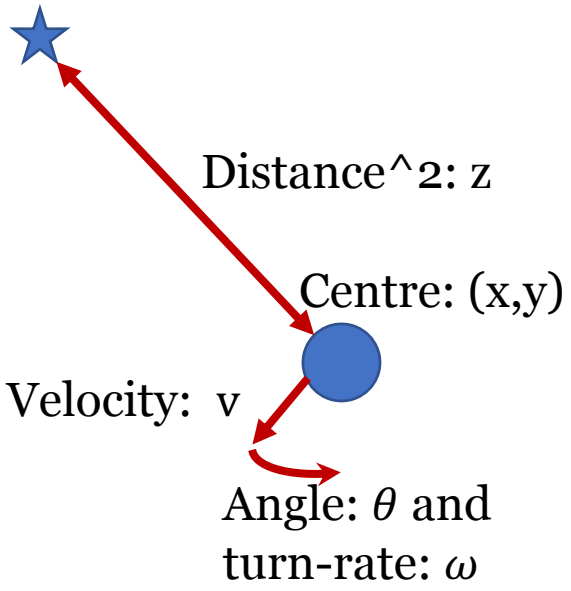
Example 3: Dubin's

- State is x,y,theta.
- Measurement is squared distance to a landmark, l
- Write the motion and measurement equations and take their Jacobians.



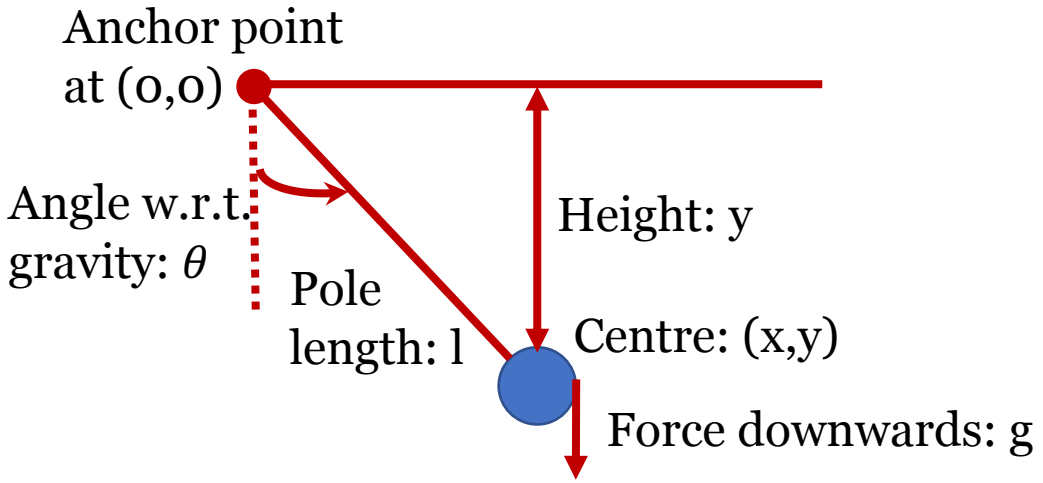
Example 3: Dubin's

- $x_{t+1} = \begin{bmatrix} x_t + v\cos(\theta_t) \\ y_t + v\sin(\theta_t) \\ \theta_t + \omega_t \end{bmatrix}$
- $z_t = [(lx - x)^2 + (ly - y)^2]$
- $F_t = \begin{bmatrix} 1 & 0 & -v\sin(\theta) \\ 0 & 1 & -v\cos(\theta) \\ 0 & 0 & 1 \end{bmatrix}$
- $H_t = [-2(lx - x) \quad -2(ly - y) \quad 0]$



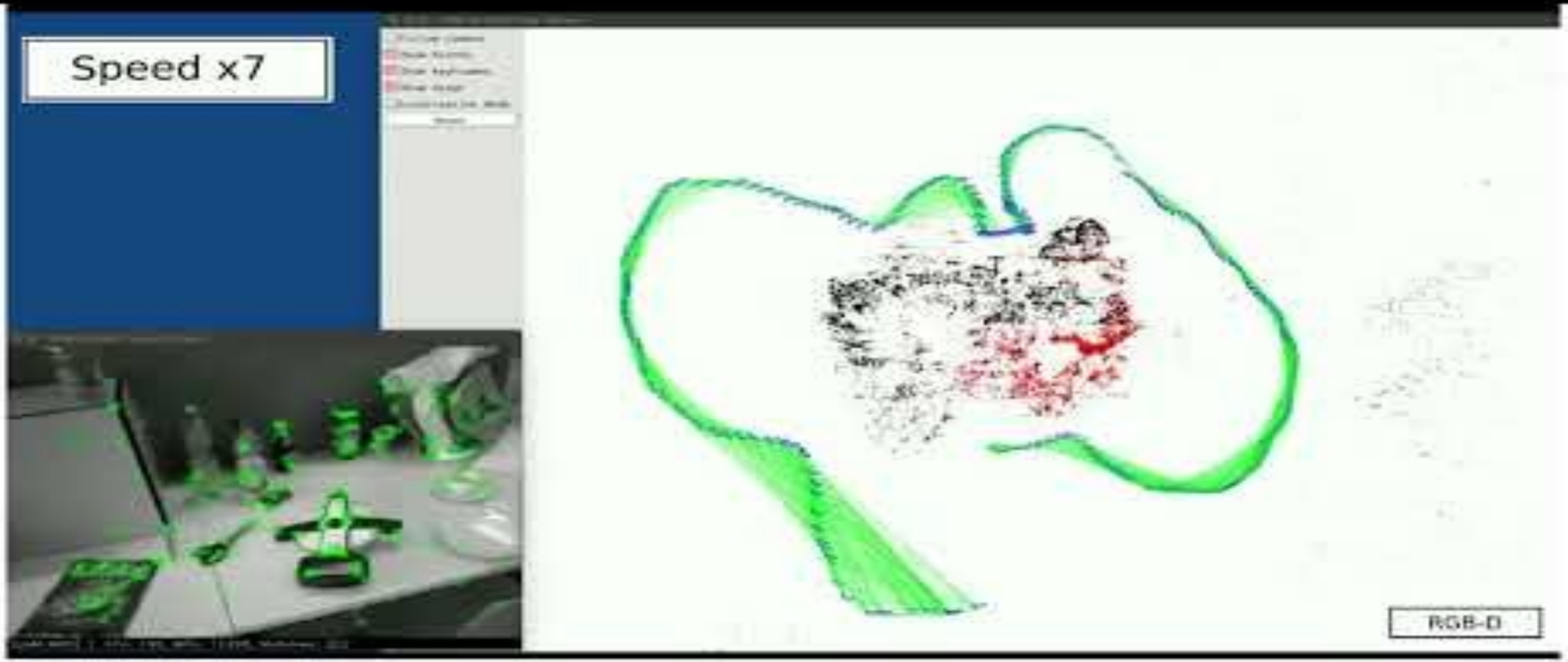
Example 4: Pendulum

- State is $[\theta, \dot{\theta}]$
- Rotational force is m*g*l*sin(θ), opposing our rotation away from 0
- Assuming rotational moment is 1.0, $\ddot{\theta}$=-m*g*l*sin(θ)
- We measure y=l*cos(θ)



Part 2: Visual Navigation for Robots

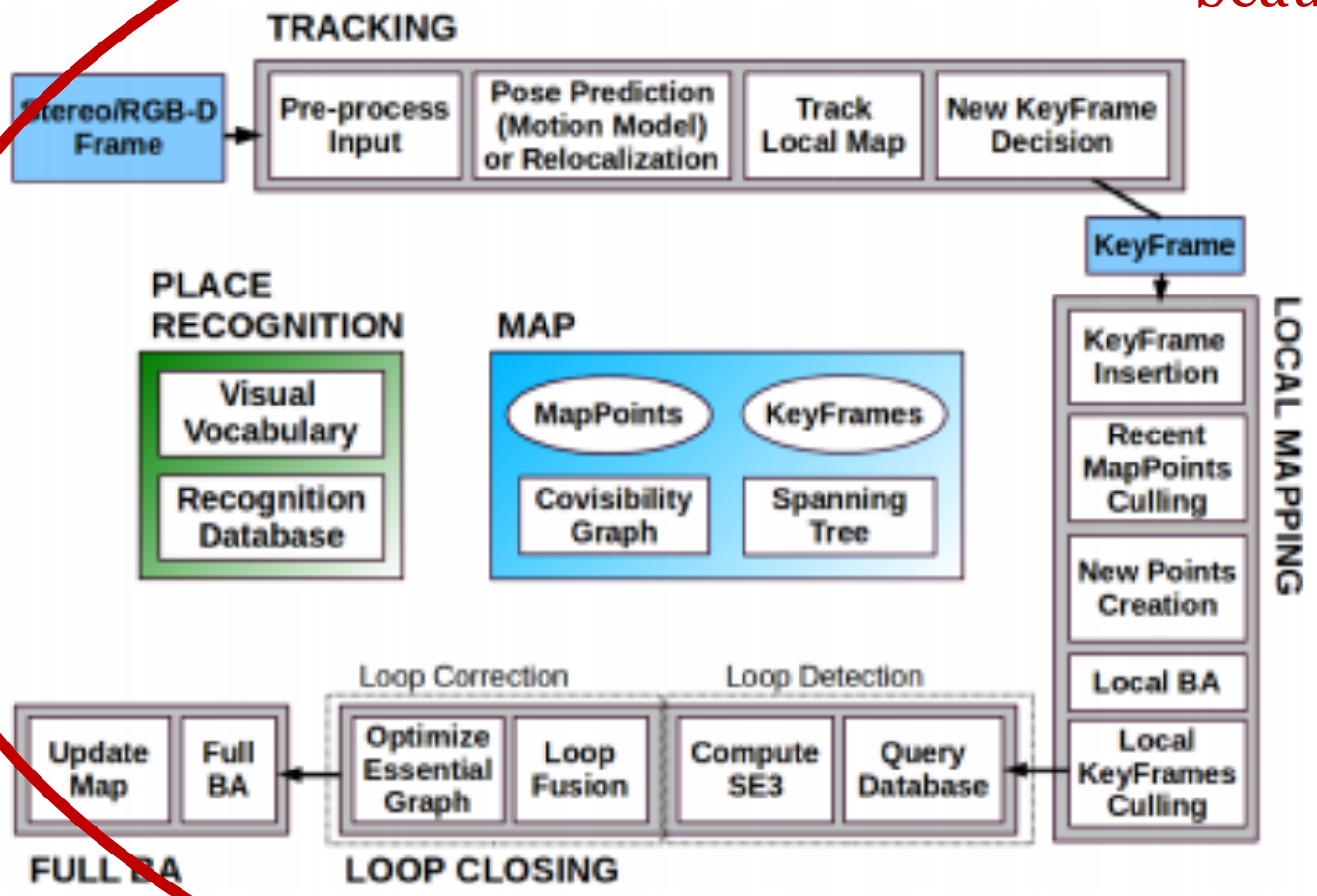
- I'll describe one of the best visual navigation packages called ORBSLAM 2.0. More info can be found here: https://webdiis.unizar.es/~raulmur/orbslam/
  - You can download and try this code (it's not so hard compared to installing ROS)…
  - The paper's PDF is expected to be above COMP 417 level, but you would be able to understand many elements, and it's a good source to dig into for learning and future research projects
  - Many fun videos on really challenging video sequences


Speed x7
RGB-D

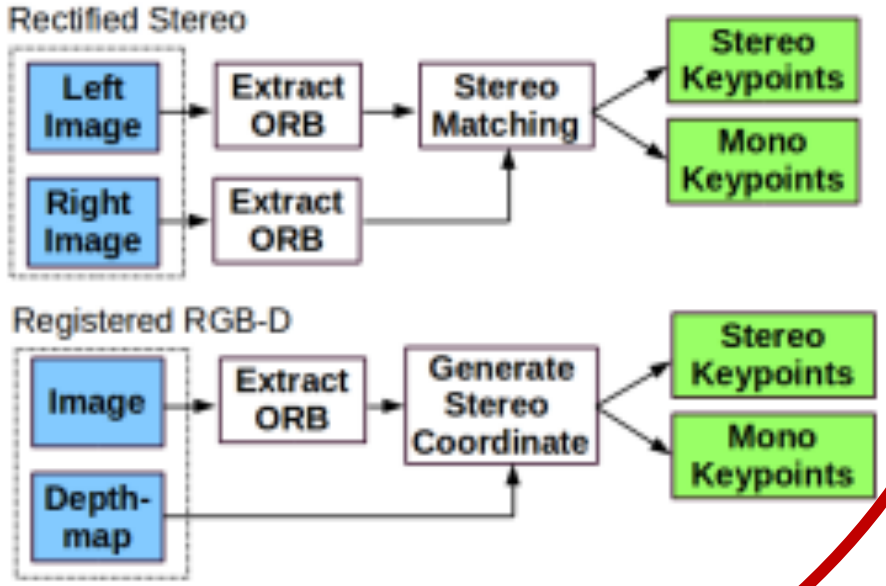
ORB-SLAM

- One of the most popular current visual mapping systems
  - The first open source tool that "really works" (this is a sketchy comment)
- 3D maps of points and camera trajectories are constructed based on matching "Oriented FAST and Rotated BRIEF" visual features
- Several layers of processing:
  - Local feature tracking
  - Localizing keyframes
  - Loop closure
- Back-end involves sparse bundle adjustment

The overview: A beautiful mess!



(a) System Threads and Modules.



(b) Input pre-processing

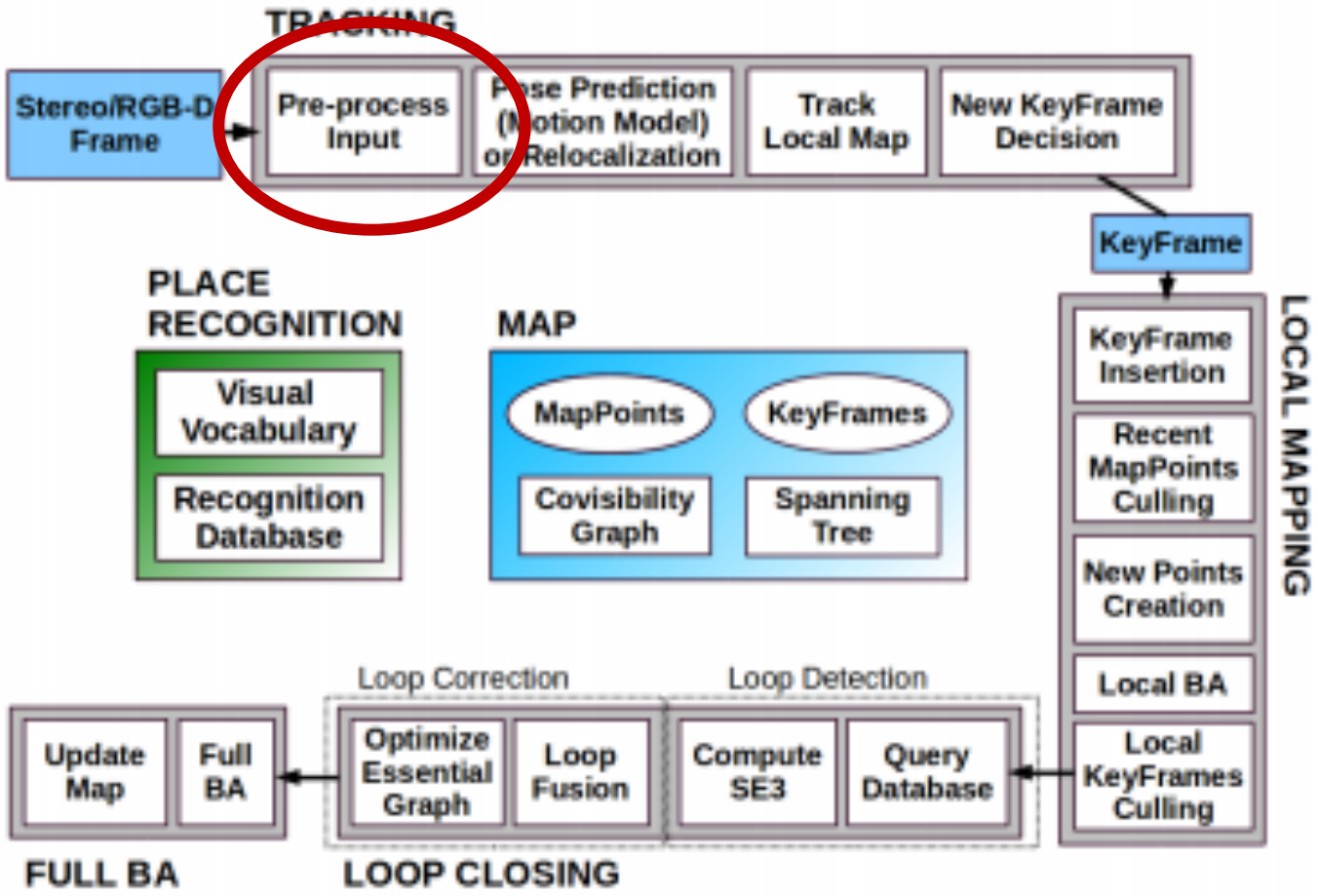
Big Picture

- Fast(ish) ***feature-based image processing*** lets us relate current image to map with minimal compute

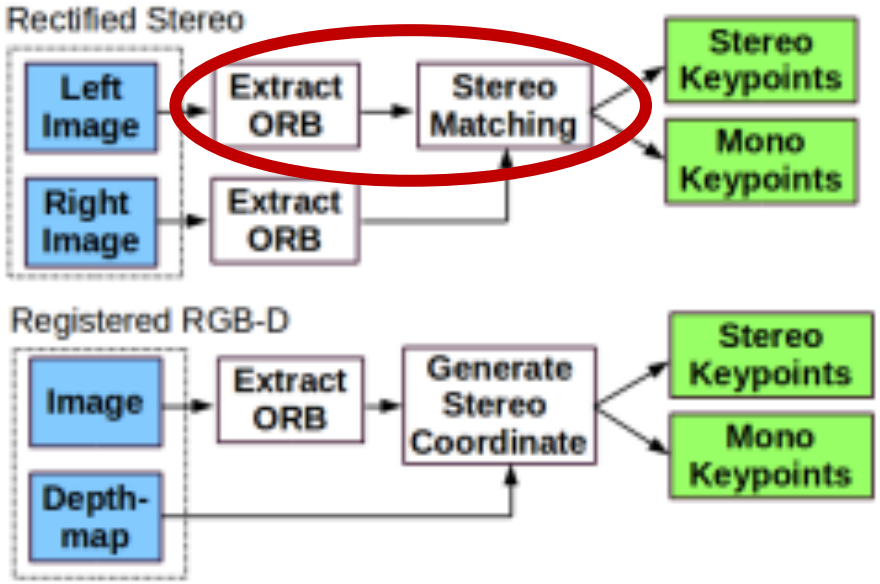
- A bit slower local mapping "adjusts" (through BA) how the local features and cameras fit with the ***keyframe***. Still fairly fast because only a few features are considered.

- Very occassional ***loop closure*** is the hardest optimization attempted. It uses co-visibility to optimize an essential graph, which is lighter than the full camera-to-camera graph, which we never touch.

First step, compute and match features



(a) System Threads and Modules.

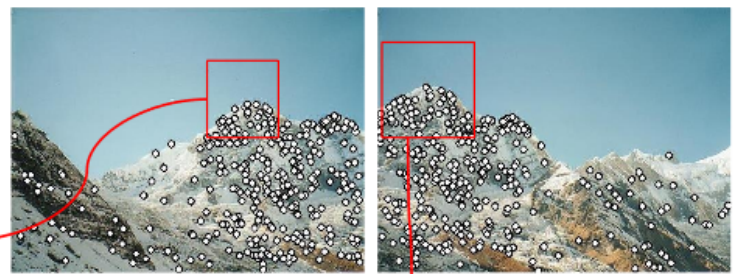


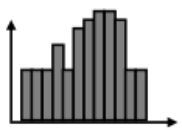
(b) Input pre-processing

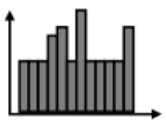
Local features: main components

1) **Detection:**
Find a set of distinctive key points.

2) **Description:**
Extract feature descriptor around each interest point as vector.



$\mathbf{x}_1$  $\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$

$\mathbf{x}_2$  $\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$

3) **Matching:**
Compute distance between feature vectors to find correspondence.

$$d(\mathbf{x}_1, \mathbf{x}_2) < T$$

Feature matching

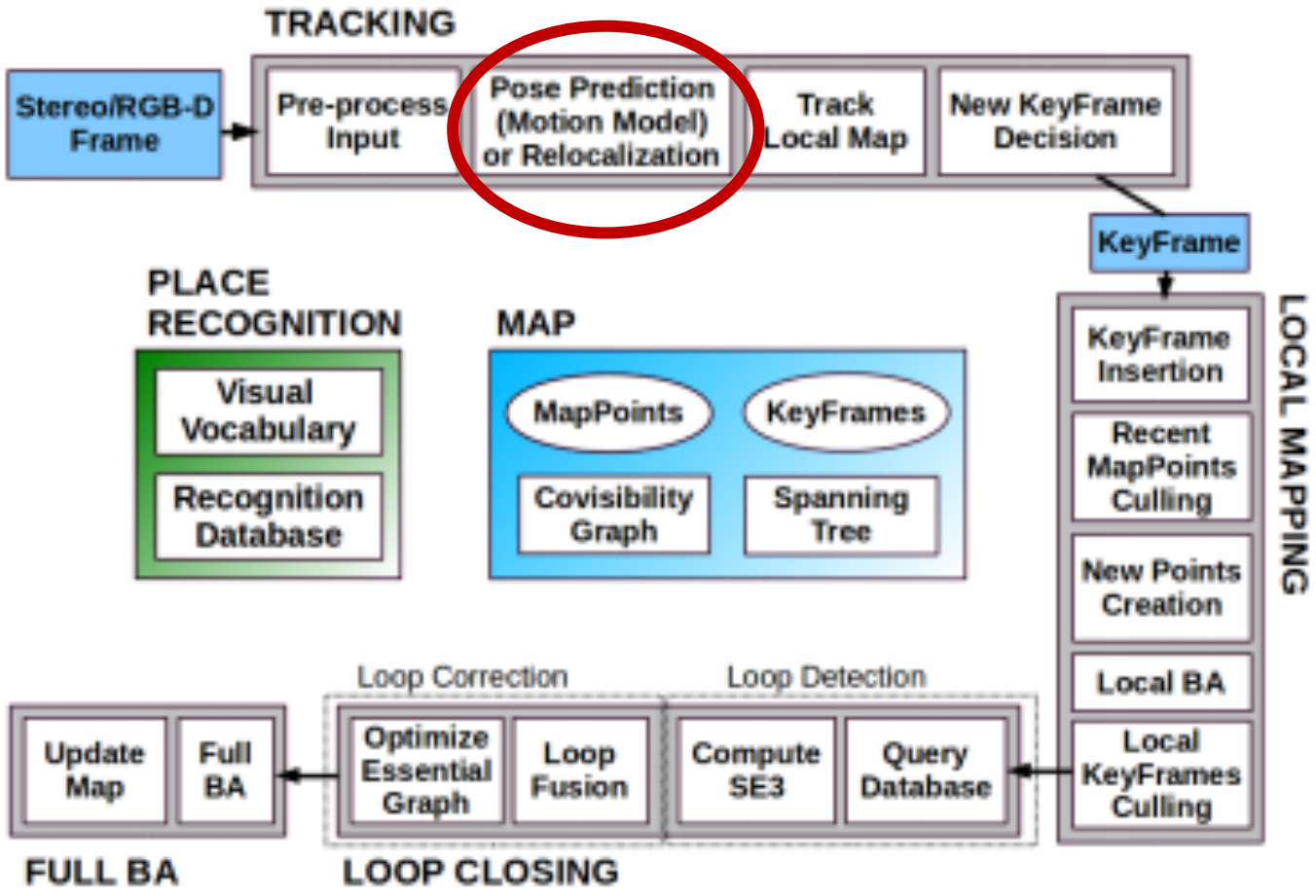
Why ORB Features?

- These are fast to compute and reliable compared with alternatives (that were available at the time)

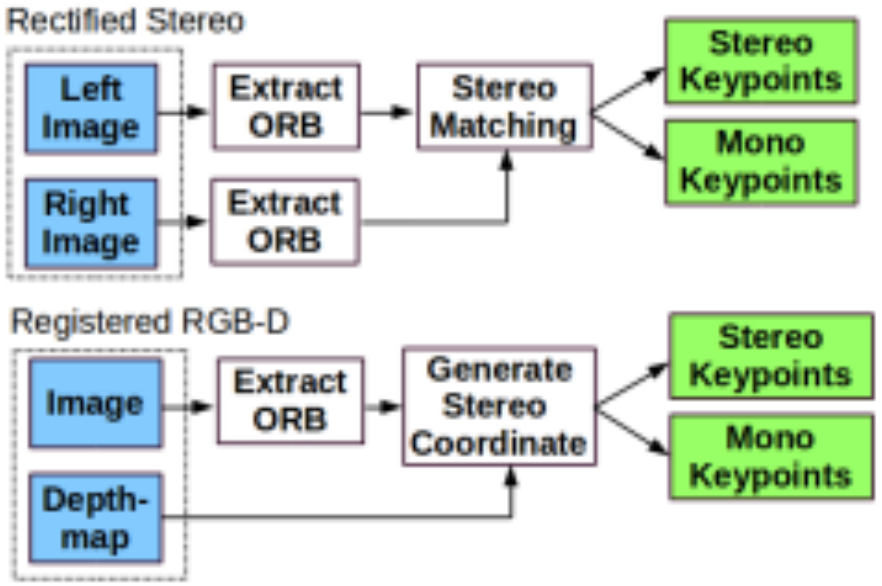
- Binary texture descriptor ***BRIEF: Binary Robust Independent Elementary Features*** is stored as a bit-string. Comparison can be done very rapidly using bit-wise operations

- Key concept: Images are big, it's too costly to store and search all of the pictures we'd take of a city. ORBs effectively compress these and allow fast lookups.

Next, find local pose



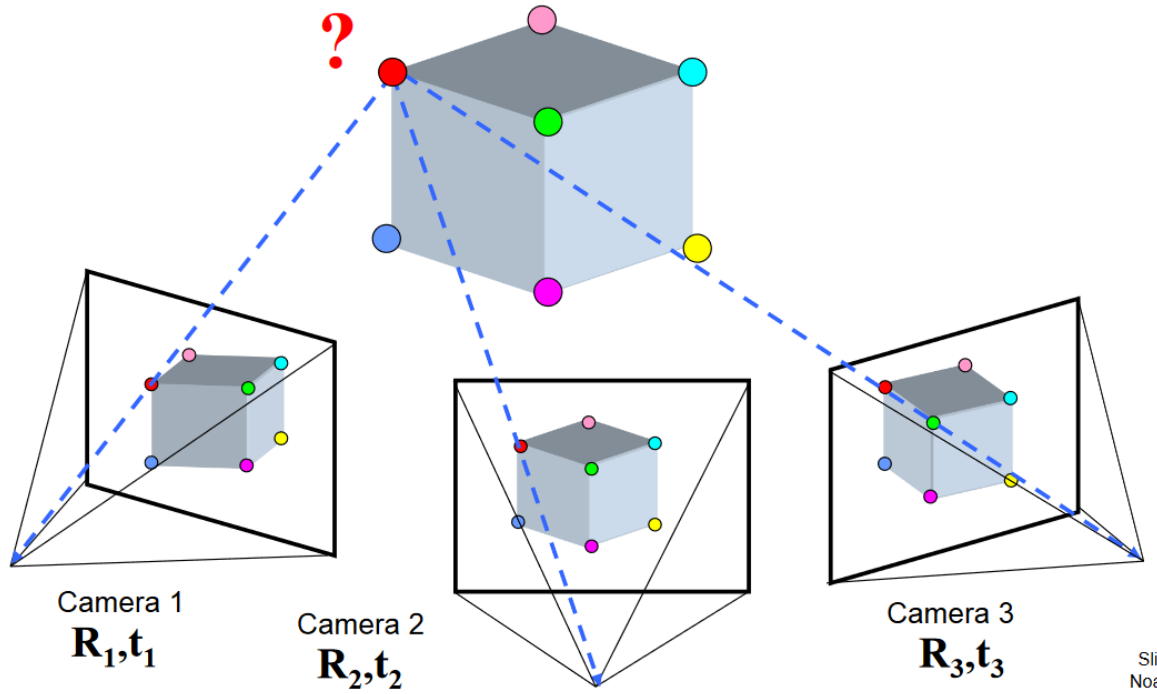
(a) System Threads and Modules.



(b) Input pre-processing

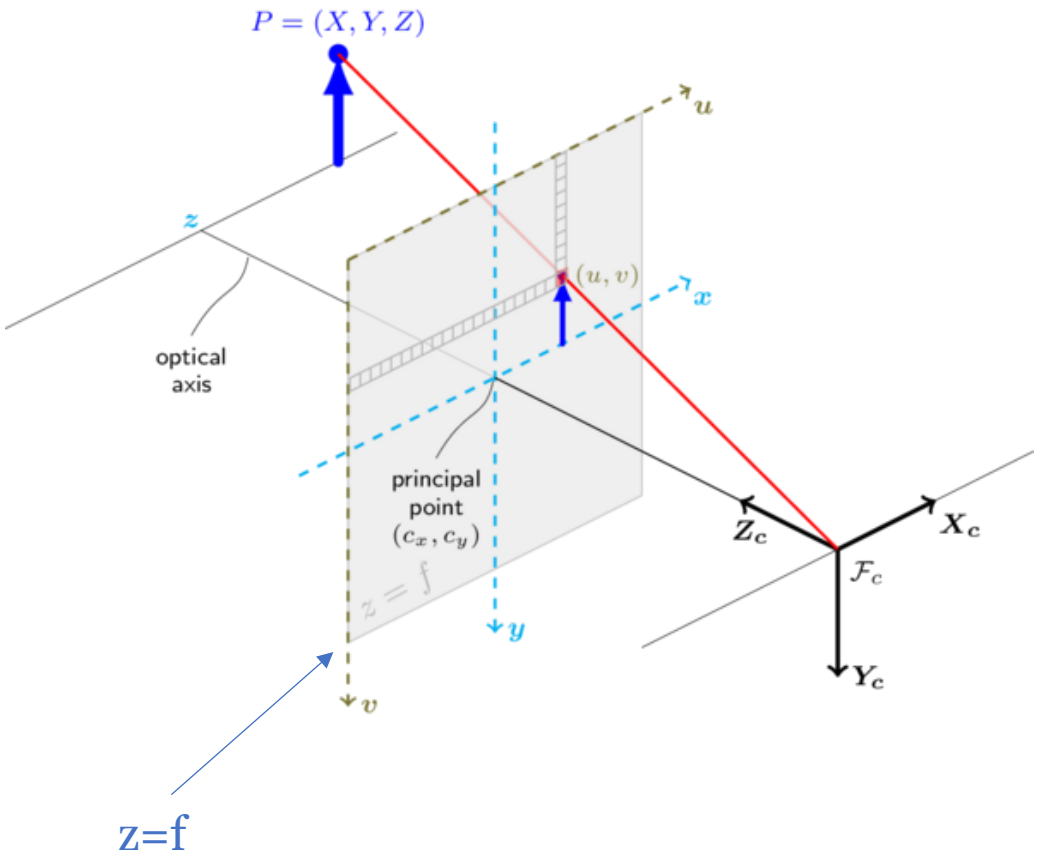
Quick background: Visual geometry

- Images of the same 3D point ***project*** to different 2D image space locations, under camera's geometry (we assume known)
- With sufficient correct matches (e.g. all corners on the cube), we can solve for:
  - All 3D point locations
  - R and t of each camera
  - All in the same frame
  - BUT with a ***scale ambiguity***



Slide credit: Noah Snavely

From 3D points to pixels: pinhole camera



(1) Perspective projection $\begin{bmatrix} x \\ y \end{bmatrix} = \pi(X, Y, Z)$

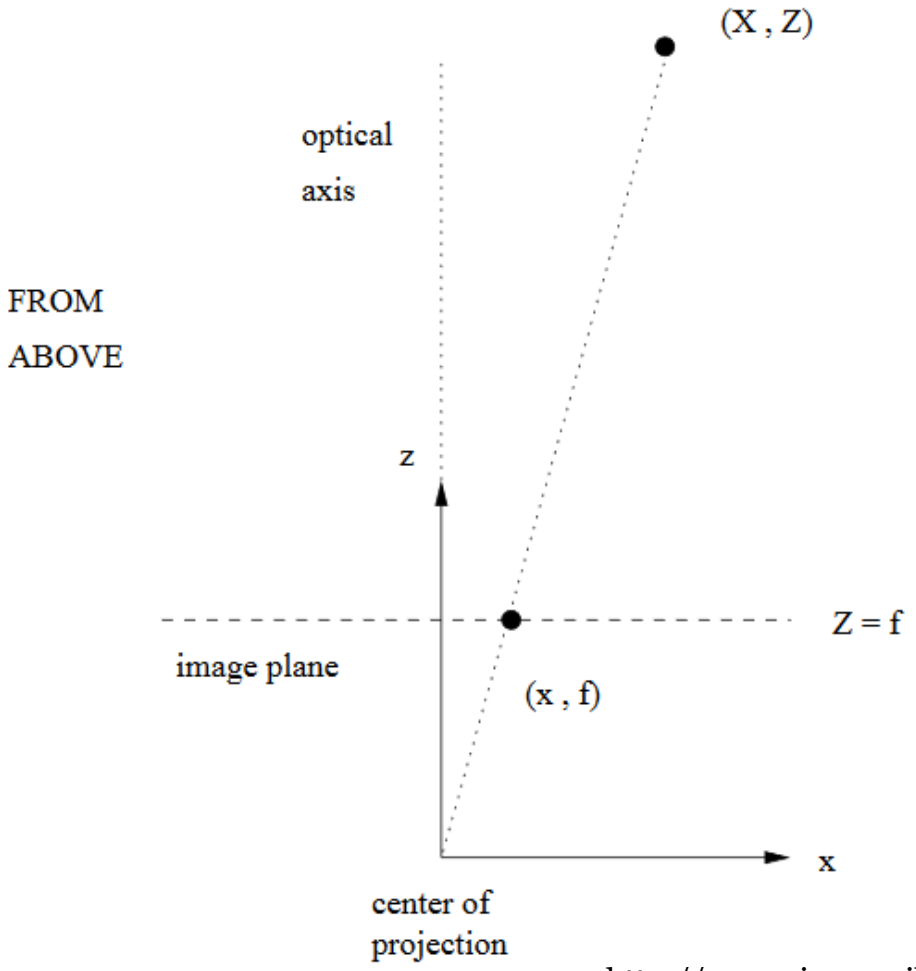
(2) Conversion from metric to pixel coordinates

$$u = m_x x + c_x$$

$$v = m_y y + c_y$$

m_x, m_y represent number of pixels per mm for the two axes

Perspective projection
$[x,y] = \pi(X,Y,Z)$



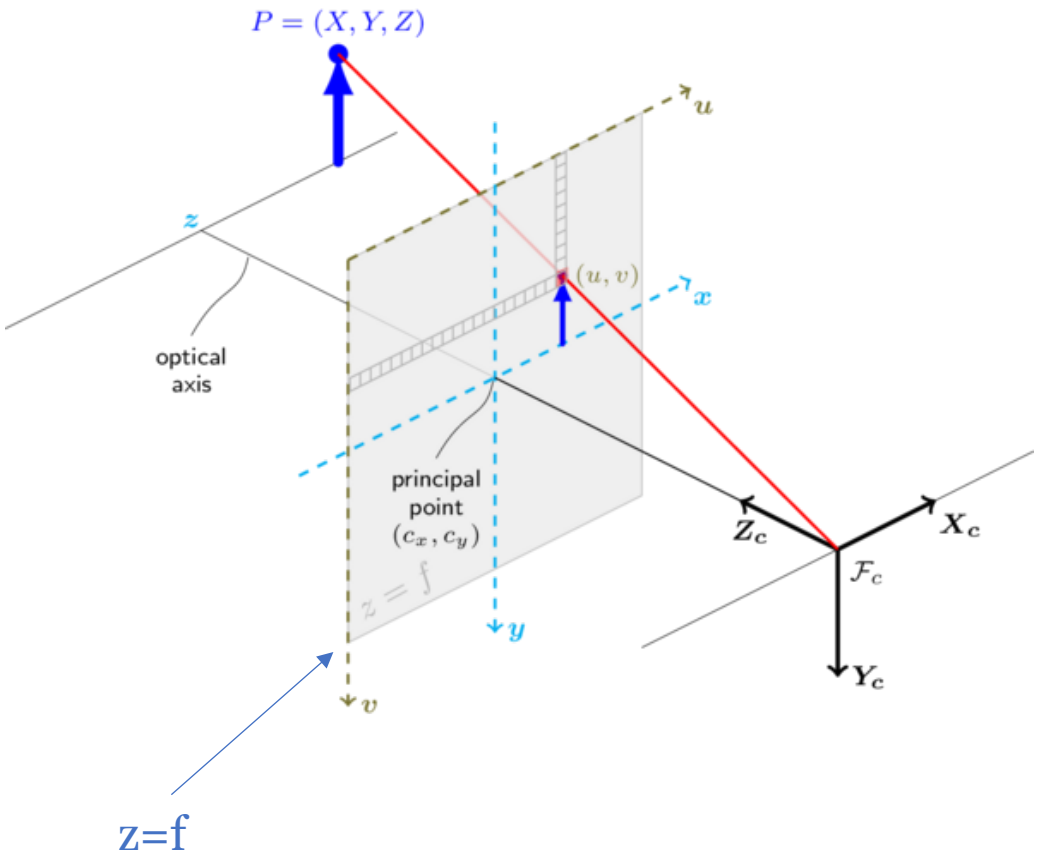
By similar triangles: $x/f = X/Z$

So, $x = f * X/Z$ and similarly $y = f * Y/Z$

Problem: we just lost depth (Z) information by doing this projection, i.e. depth is now uncertain.

http://www.cim.mcgill.ca/%7Elanger/558.html

From 3D points to pixels: pinhole camera



(1) Perspective projection $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \end{bmatrix} = \pi(X, Y, Z)$

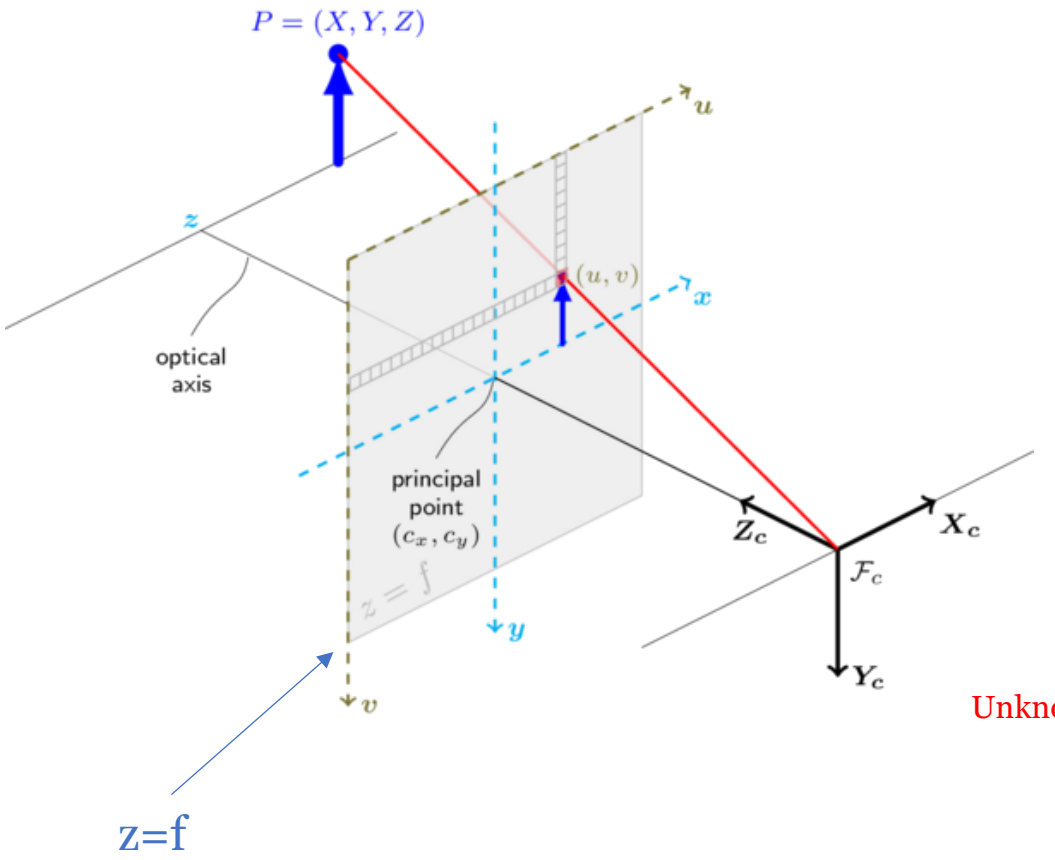
(2) Conversion from metric to pixel coordinates

$$u = m_x x + c_x$$

$$v = m_y y + c_y$$

$$h_{\text{pinhole}}(X, Y, Z) = \begin{bmatrix} \frac{f m_x X}{Z} + c_x \\ \frac{f m_y Y}{Z} + c_y \end{bmatrix} + \text{noise in pixels}$$

Background: From 3D points to pixels: pinhole camera



(1) Perspective projection $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \end{bmatrix} = \pi(X, Y, Z)$

(2) Conversion from metric to pixel coordinates

$$u = m_x x + c_x$$

$$v = m_y y + c_y$$

Unknown depth/scale

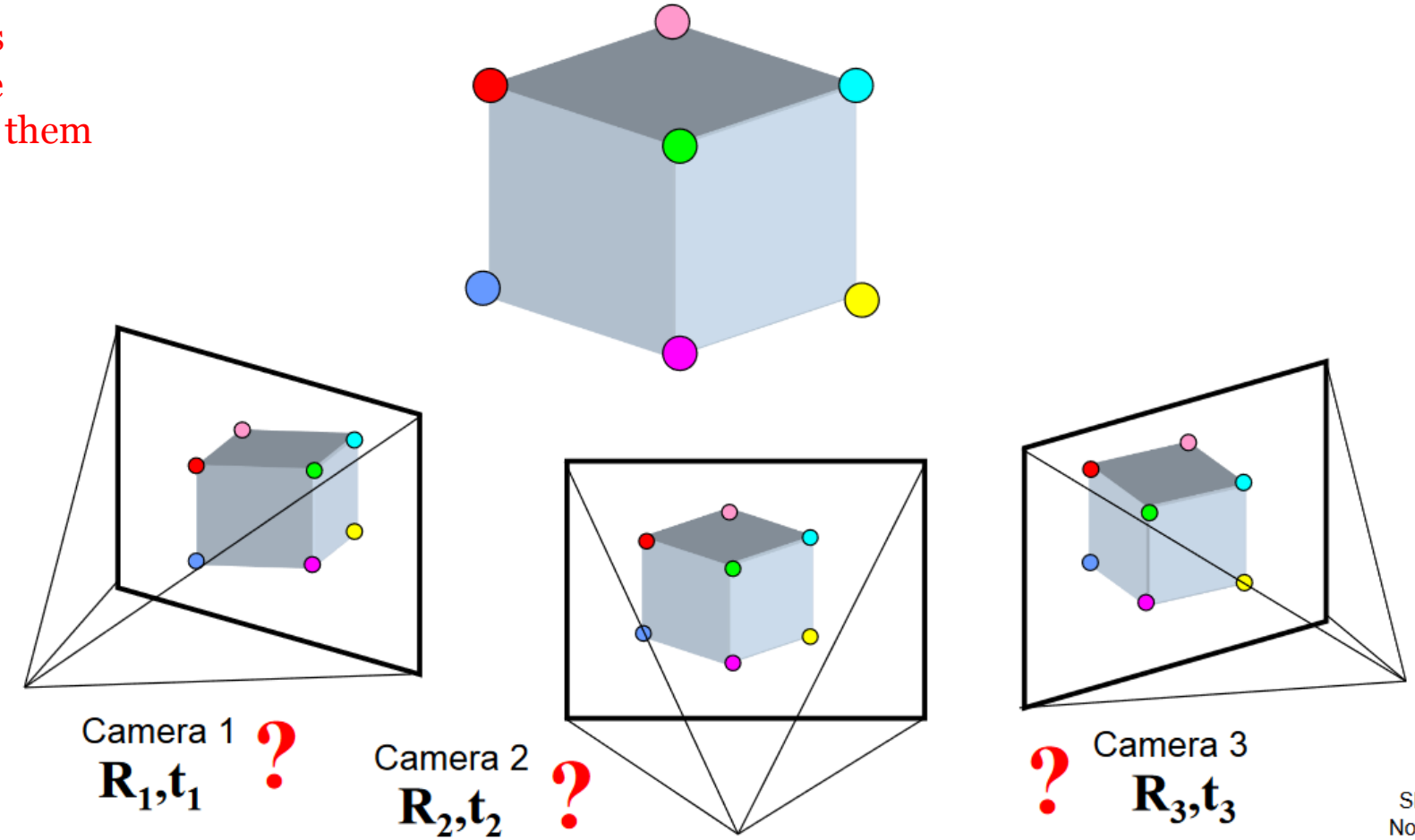
Camera calibration matrix

Usually presented as $s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fm_x & 0 & c_x \\ 0 & fm_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$

Multi-view geometry problems

- **Motion:** Given a set of corresponding points in two or more images, compute the camera parameters

3D point coordinates are unknown, but we won't try to estimate them



Camera frame transformations are unknown and to be estimated

Slide credit: Noah Snavely

Camera localization as a least squares problem?

$$ {}^{k}_{w}R^{*}, {}^{k}t^{*}_{kw} = \underset{{}^{k}_{w}R, {}^{k}t_{kw}}{\operatorname{argmin}} \sum_{k=1}^{K} ||\bar{z}^{(k)} - \mathbb{E}[\, h_{\text{pinhole}}({}^{k}_{w}R\, {}^{w}p + {}^{k}t_{kw})\,]\, ||^2 $$

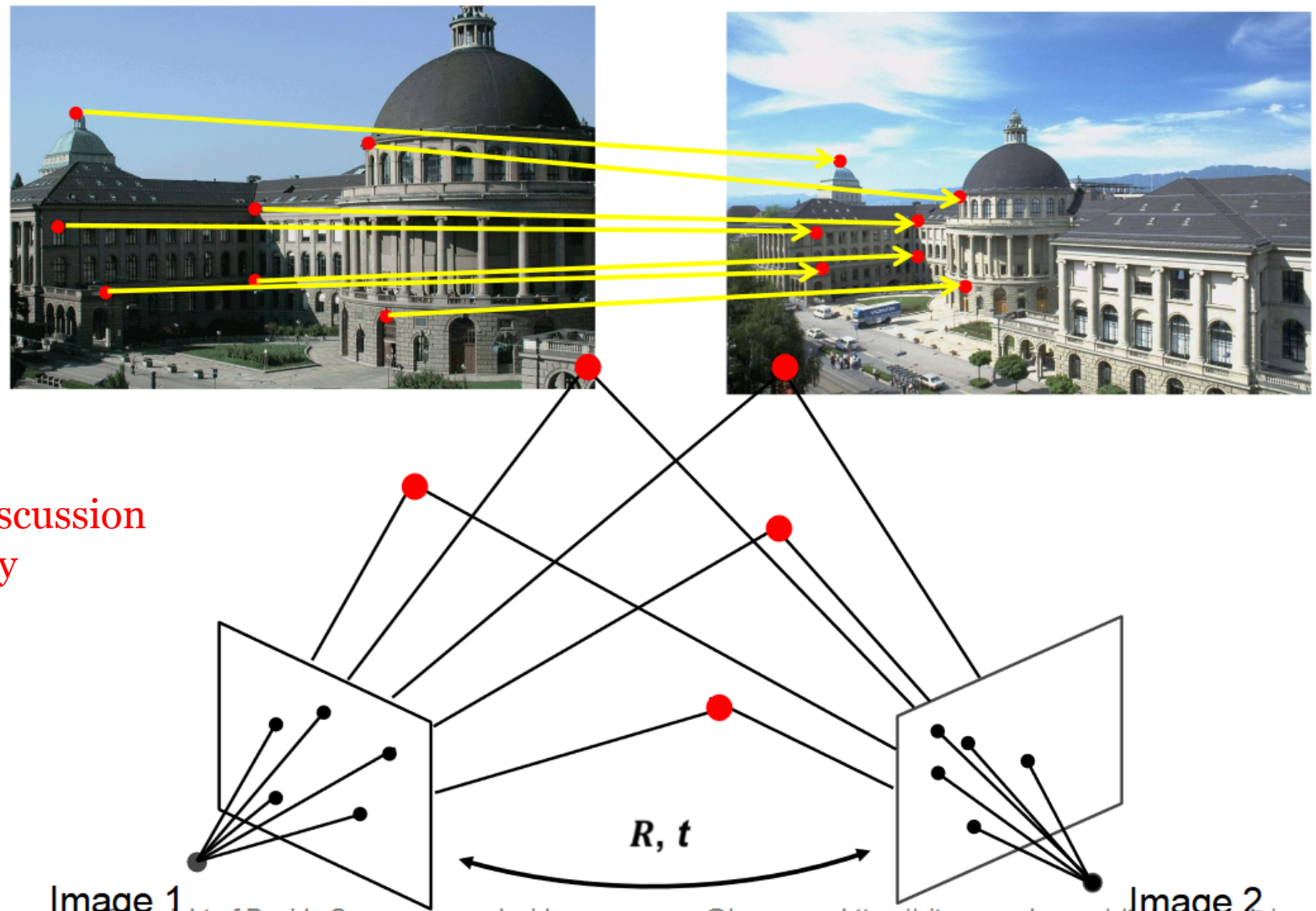
The only terms to be optimized.

But, 3D position is unknown!

So, we cannot solve the problem using the reprojection error until we know the 3D position corresponding to the keypoint.

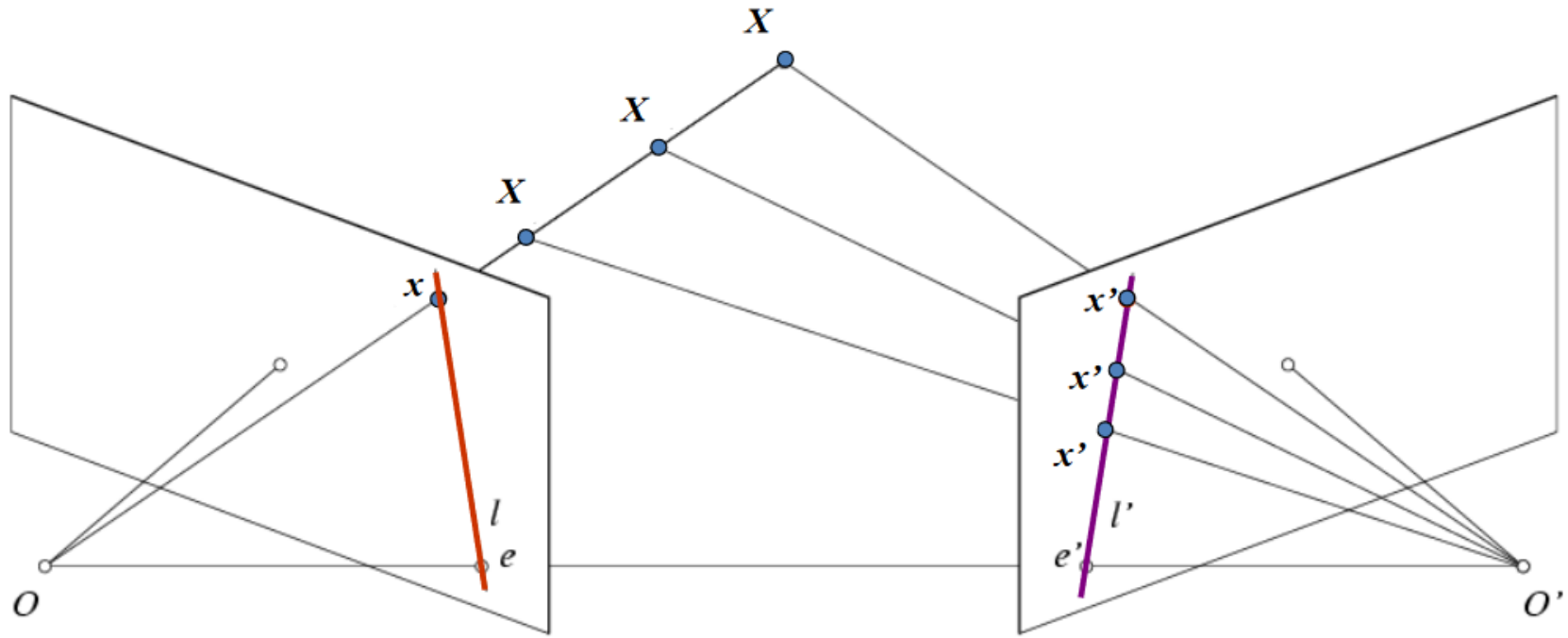
$$ h_{\text{pinhole}}(X, Y, Z) = \begin{bmatrix} \frac{f m_x X}{Z} + c_x \\ \\ \frac{f m_y Y}{Z} + c_y \end{bmatrix} + n, \quad n \sim \mathcal{N}(0, R) \quad \text{noise (in pixels)} $$

Working Principle



Let's restrict the discussion to two cameras only

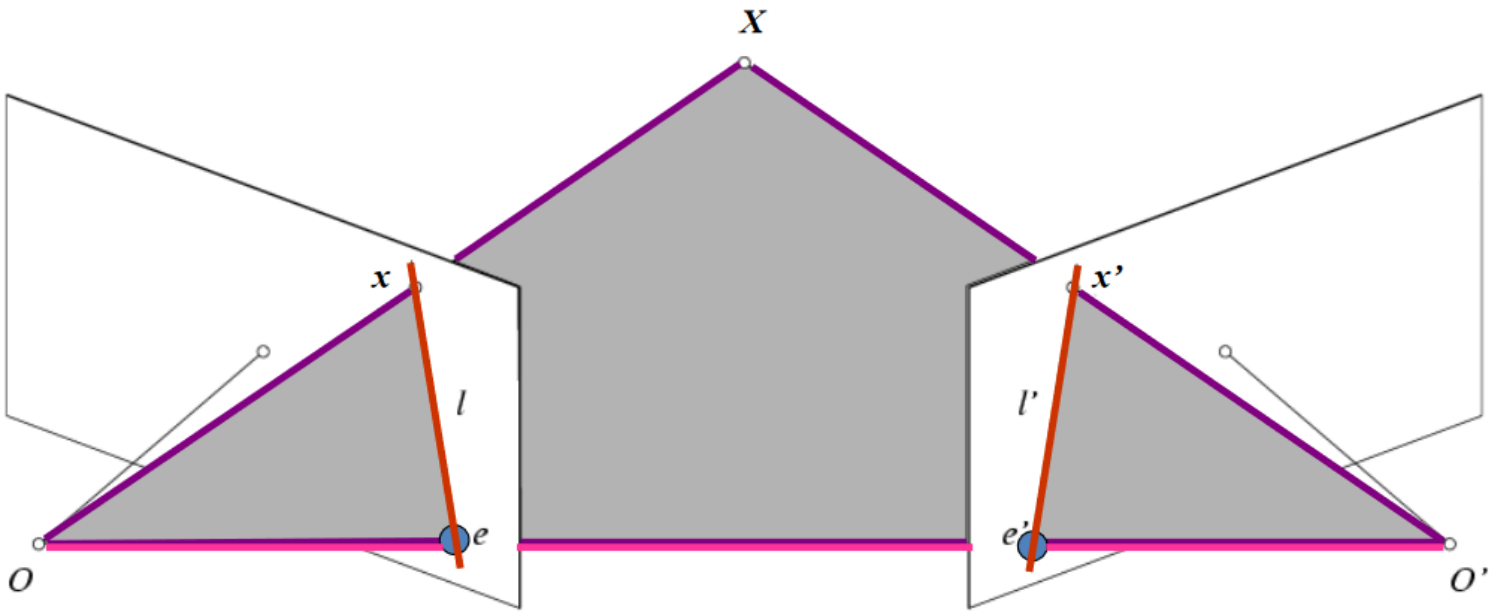
Key idea: Epipolar constraint



Potential matches for *x'* have to lie on the corresponding line *l*.

Potential matches for *x* have to lie on the corresponding line *l'*.

Epipolar geometry: notation



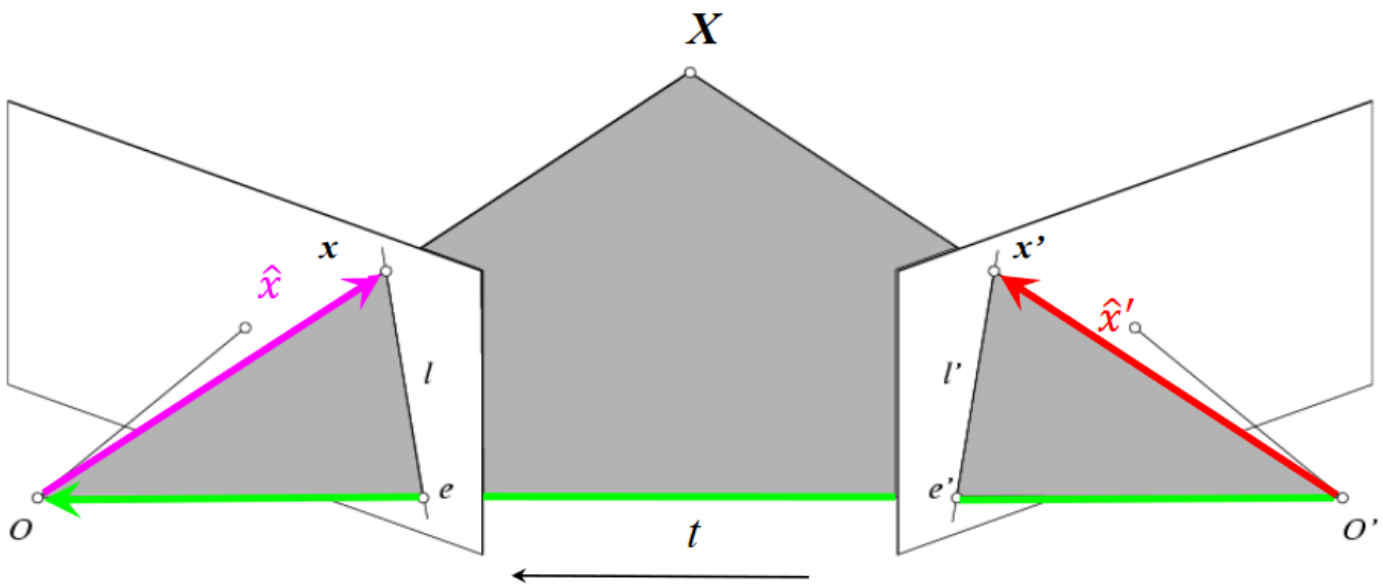
- **Baseline** – line connecting the two camera centers
- **Epipoles**

= intersections of baseline with image planes

= projections of the other camera center

- **Epipolar Plane** – plane containing baseline (1D family)
- **Epipolar Lines** - intersections of epipolar plane with image planes (always come in corresponding pairs)

Epipolar constraint: Calibrated case



$$\hat{x} = K^{-1}x = X$$

Homogeneous 2d point (3D ray towards X)

2D pixel coordinate (homogeneous)

3D scene point

$$\hat{x}' = K'^{-1}x' = X'$$

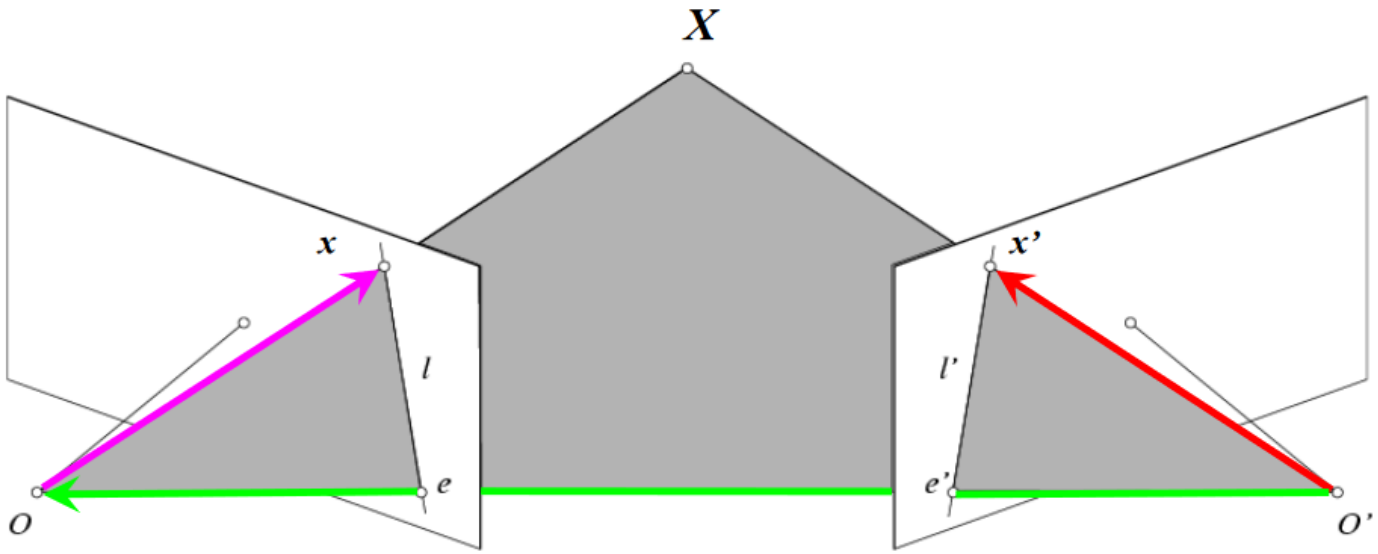
3D scene point in 2nd camera's 3D coordinates

$$\hat{x} \cdot [t \times (R\hat{x}')] = 0$$

(because $\hat{x}, R\hat{x}'$, and t are co-planar)

Essential matrix

"5-point algorithm" by David Nister computes essential matrix and then decomposes it into rotation and translation.



$$\hat{x} \cdot [t \times (R\hat{x}')] = 0 \quad \Rightarrow \quad \hat{x}^T E \hat{x}' = 0 \quad \text{with} \quad E = [t]_\times R$$

E is a 3x3 matrix which relates corresponding pairs of normalized homogeneous image points across pairs of images – for *K* calibrated cameras.

Estimates relative position/orientation.

Essential Matrix
(Longuet-Higgins, 1981)

Note: $[t]_\times$ is matrix representation of cross product

$$[t]_\times = \begin{bmatrix} 0 & -t_2 & t_1 \\ t_2 & 0 & -t_0 \\ -t_1 & t_0 & 0 \end{bmatrix}$$

After estimating the essential matrix, we extract t, R.

However, the translation t, is only estimated up to a multiplicative scale.

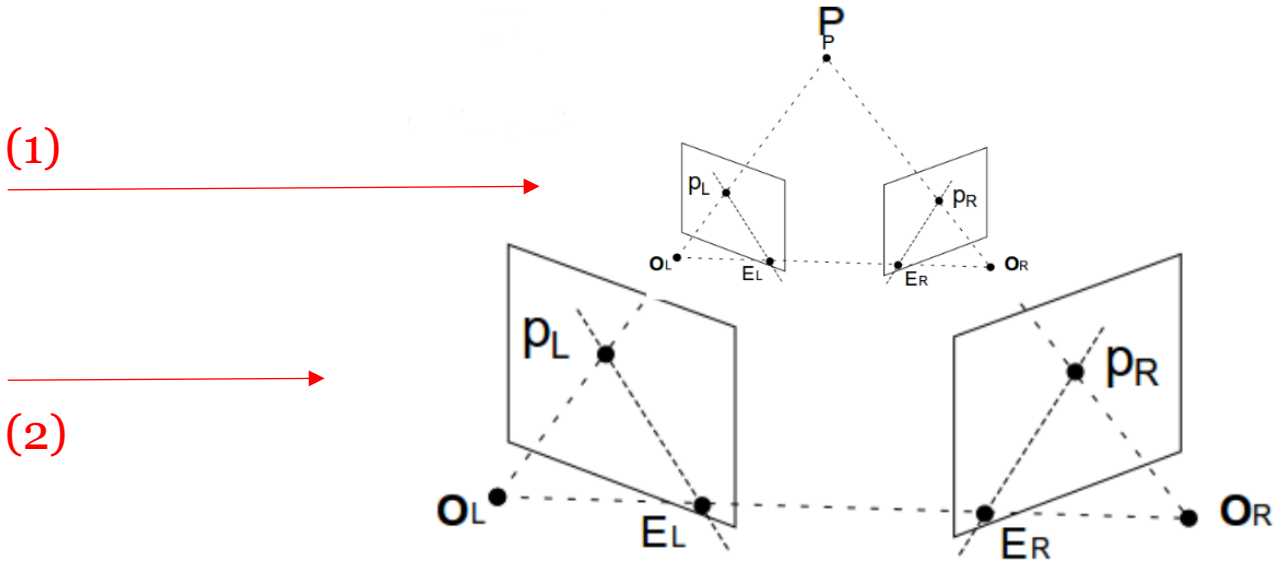
- Translation is not fully observable with a single camera.

- To make it observable we need stereo

Visual odometry with a single camera: translation is recovered only up to a scale

- Scale = relationship between real-world metric distance units and estimated map distance units

Camera placements (1) and (2) generate the same observation of P. In fact, infinitely many possible placements of the two camera frames along their projection rays could have generated the same measurement.

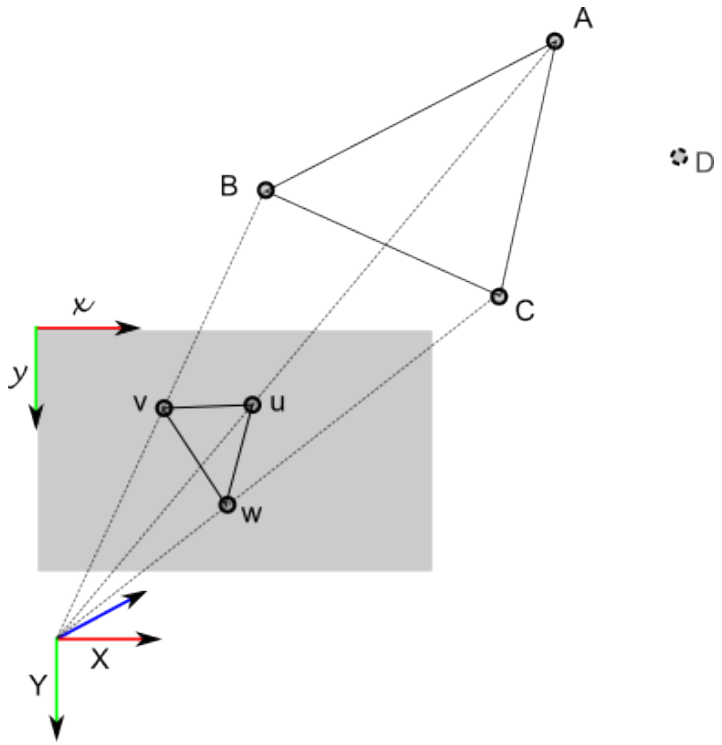


Visual odometry with a single camera: translation is recovered only up to a scale

- Scale = relationship between real-world metric distance units and estimated map distance units

Q: Is there a way to obtain true metric distances only with a single camera?

A: The only way is to have an object of known metric dimensions in the observed scene. For example if you know distances AB, BC, CA then you can recover true translation. This is commonly referred to as the Perspective-3-Point (P3P), or in General, the Perspective-n-Point (PnP) problem.

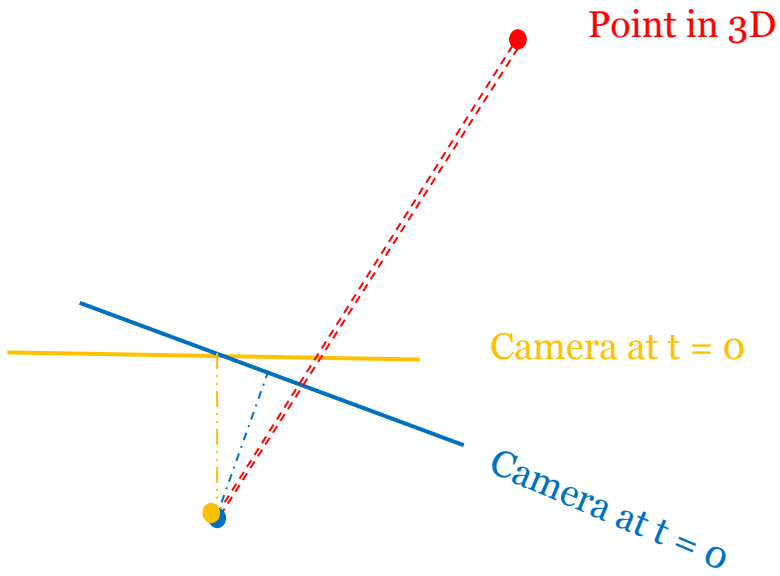


Visual odometry with a single camera: translation is recovered only up to a scale

- Scale = relationship between real-world metric distance units and estimated map distance units

Q: Does scale remain constant throughout the trajectory of a single camera?

A: No, there is **scale drift,** which is most apparent during in-place rotations (i.e. pure rotation, no translation), because depth estimation for 3D points is unconstrained, so it is easily misestimated.



2D-to-2D Algorithm

Motion estimation		
2D-2D	3D-3D	3D-2D

Algorithm 1: VO from 2D-to-2D correspondeces

1 Capture new frame I_k
2 Extract and match features between I_{k-1} and I_k,
3 Compute essential matrix for image pair I_{k-1}, I_k
4 Decompose essential matrix into R_k and t_k, and form T_k
5 Compute relative scale and rescale t_k accordingly
6 Concatenate transformation by computing $C_k = C_{k-1}T_k$
7 Repeat from 1

Take-home message on pose prediction: given corresponding features, basic math gives us the camera motion.

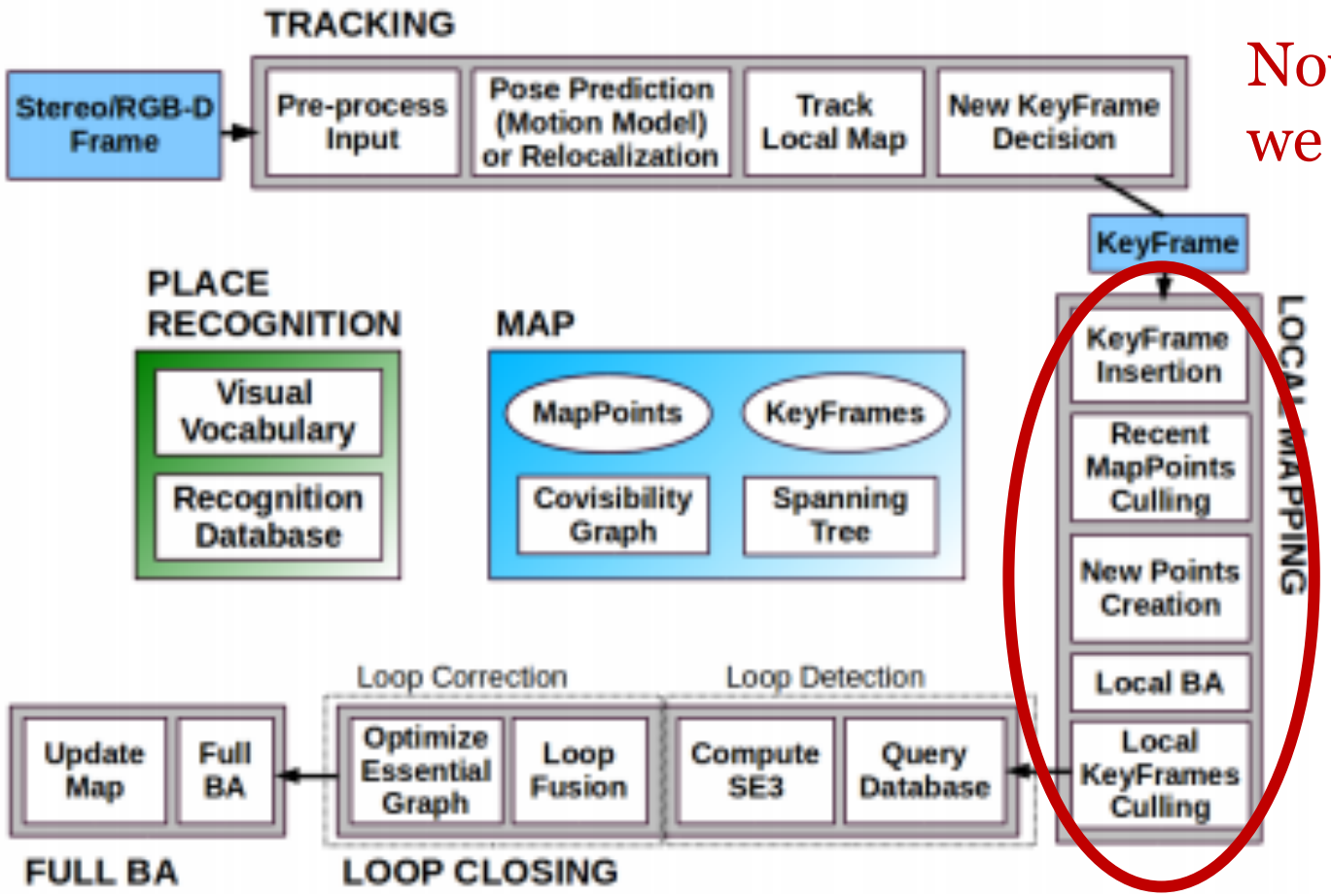
Robust Feature matching

- Pose prediction works with good matches, but errors would actually pull our solution off of "truth"!
- This is one of the most important, and difficult aspects of visual navigation
- Several tricks allow us to remove false matches

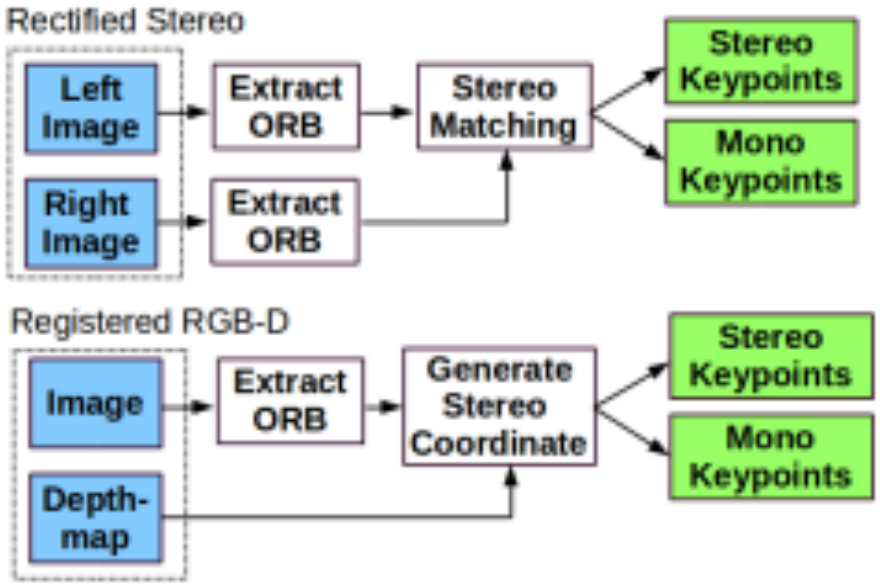
Finding correct matches

- Idea #1: Randomized Sampling and Consensus (RANSAC: Fischler and Bolles 1981)
  - Iterate:
    - Pick a set of matches at random just large enough to solve for pose
    - Compute the solution
    - Apply the solution to all other matches, and find "inliers"
    - Stop when enough inliers are found
  - It is very unlikely we'd accidentally find a "wrong" solution that explains other points. Provable under some assumptions.
- Idea #2: Use constraints we can build up from mapping:
  - Tracking over video frames: features should stay close
  - 3D structure of map should be relatively coherent

Now, maintain a map as we move along



(a) System Threads and Modules.



(b) Input pre-processing

Map representation

- Keyframes attached to features
- Co-visibility tells us which keyframes are related
- Spanning tree and Essential graph allow more rapid global solving

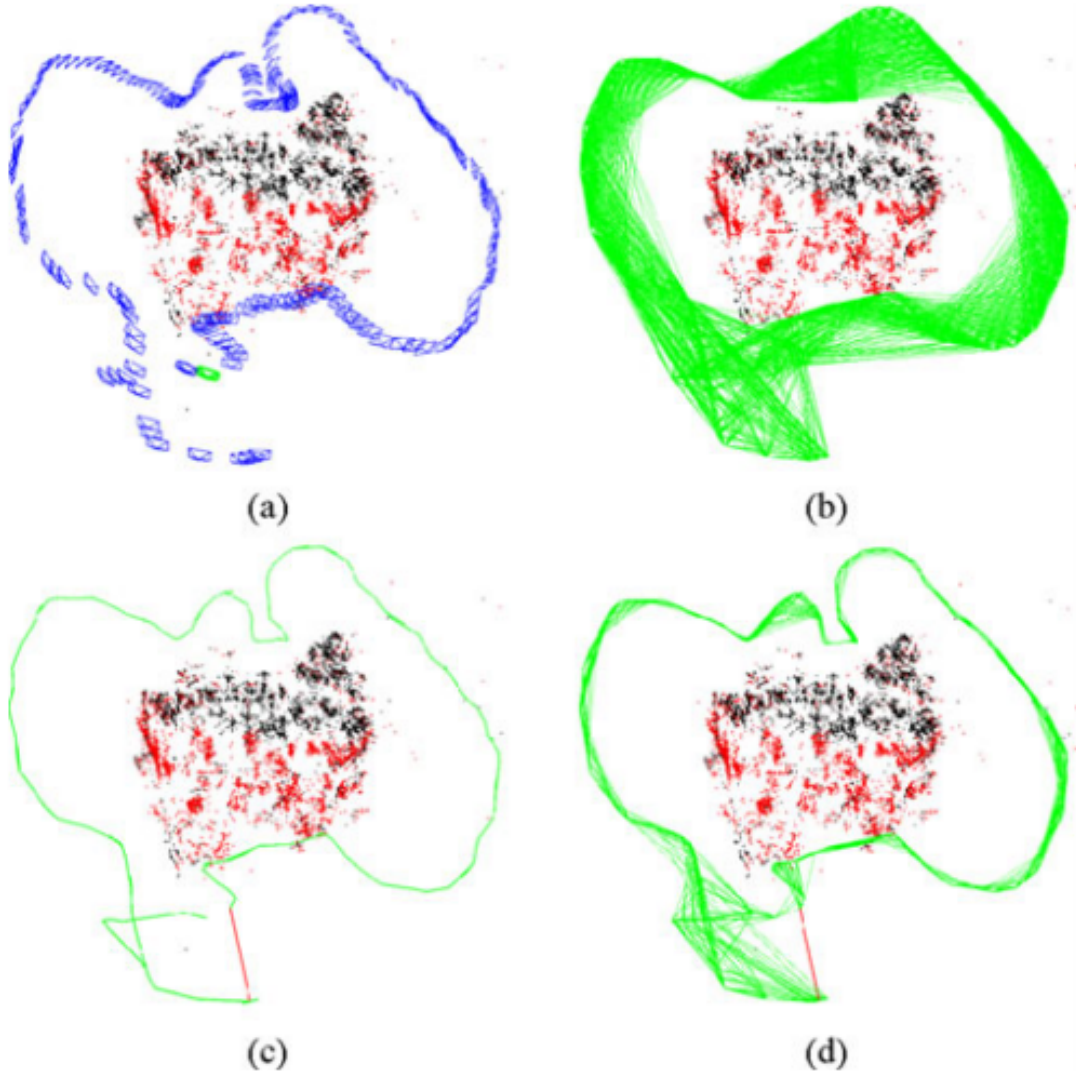
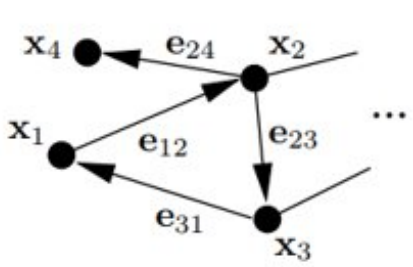


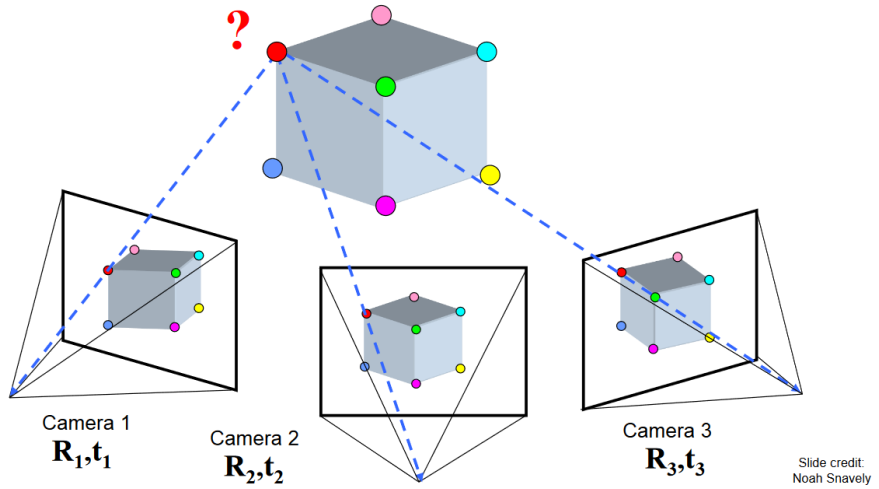
Fig. 2. Reconstruction and graphs in the sequence $fr3\_long\_office\_household$ from the TUM RGB-D Benchmark [38]. (a) Keyframes (blue), current camera (green), map points (black, red), current local map points (red). (b) Covisibility graph. (c) Spanning tree (green) and loop closure (red). (d) Essential graph.

Mapping Algorithm

- Construct a local BA problem with the cameras and features seen by the keyframes that "see" the current set of features
  - These are the ones that can change with the new information
- BA is allowed to adjust camera R and t as well as 3D point locations.
- Error is reprojection

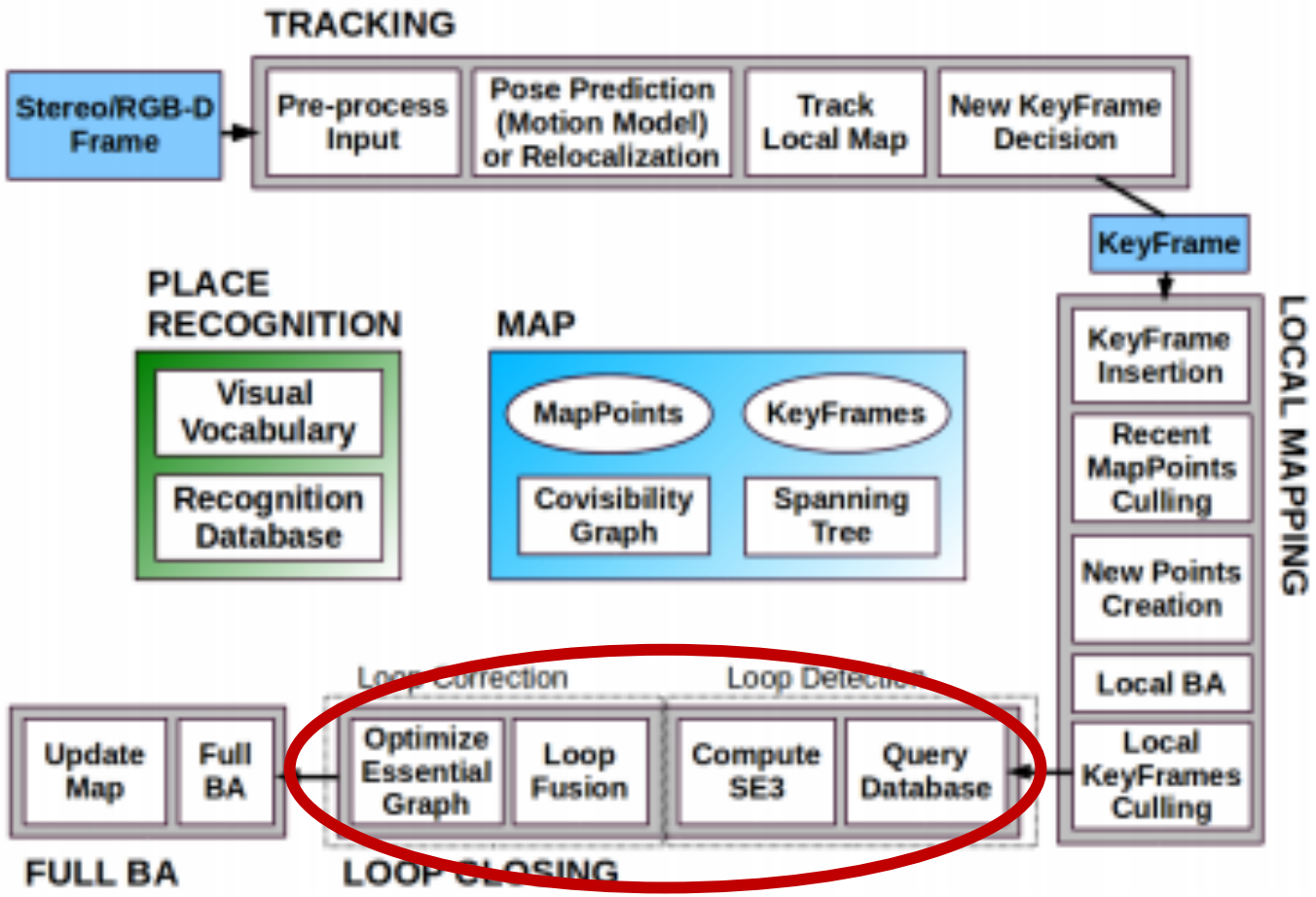


$$\mathbf{F}(\mathbf{x}) = \mathbf{e}_{12}^\top \boldsymbol{\Omega}_{12}\, \mathbf{e}_{12} + \mathbf{e}_{23}^\top \boldsymbol{\Omega}_{23}\, \mathbf{e}_{23} + \mathbf{e}_{31}^\top \boldsymbol{\Omega}_{31}\, \mathbf{e}_{31} + \mathbf{e}_{24}^\top \boldsymbol{\Omega}_{24}\, \mathbf{e}_{24} + \dots$$

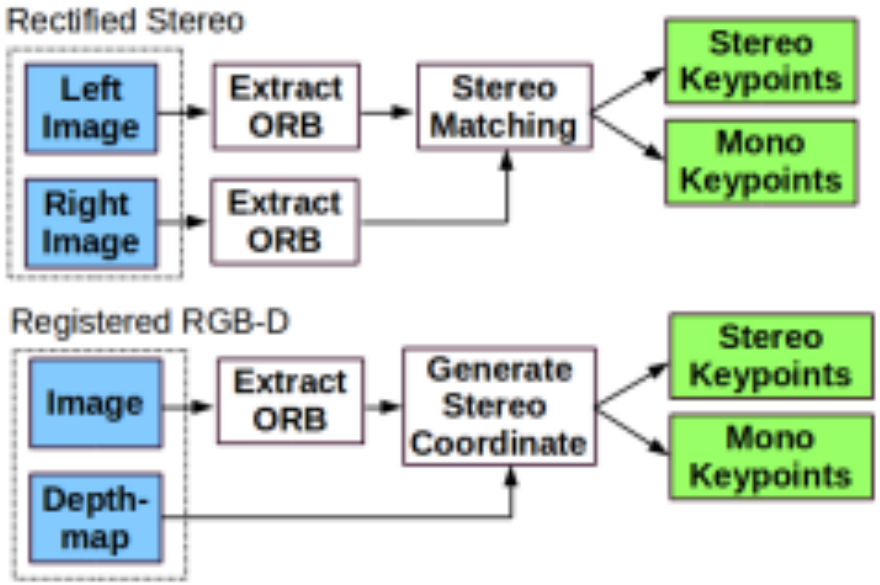


Mapping: Important Questions

- How do the number of keyframes affect…
  - Run-time of the BA algorithm?
  - Memory required to store the map?
  - Difficulty in feature matching between current image and keyframes?



(a) System Threads and Modules.

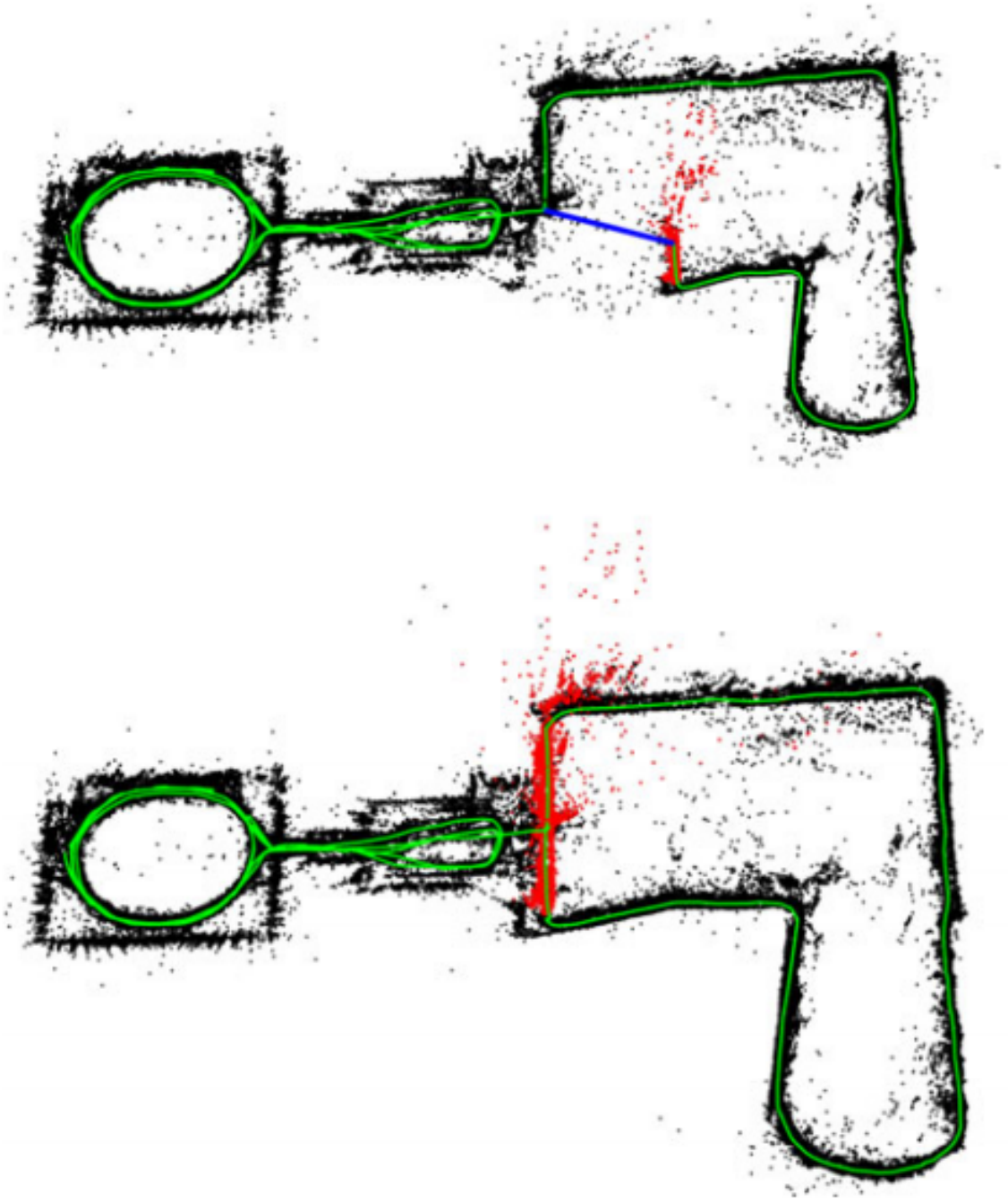


(b) Input pre-processing

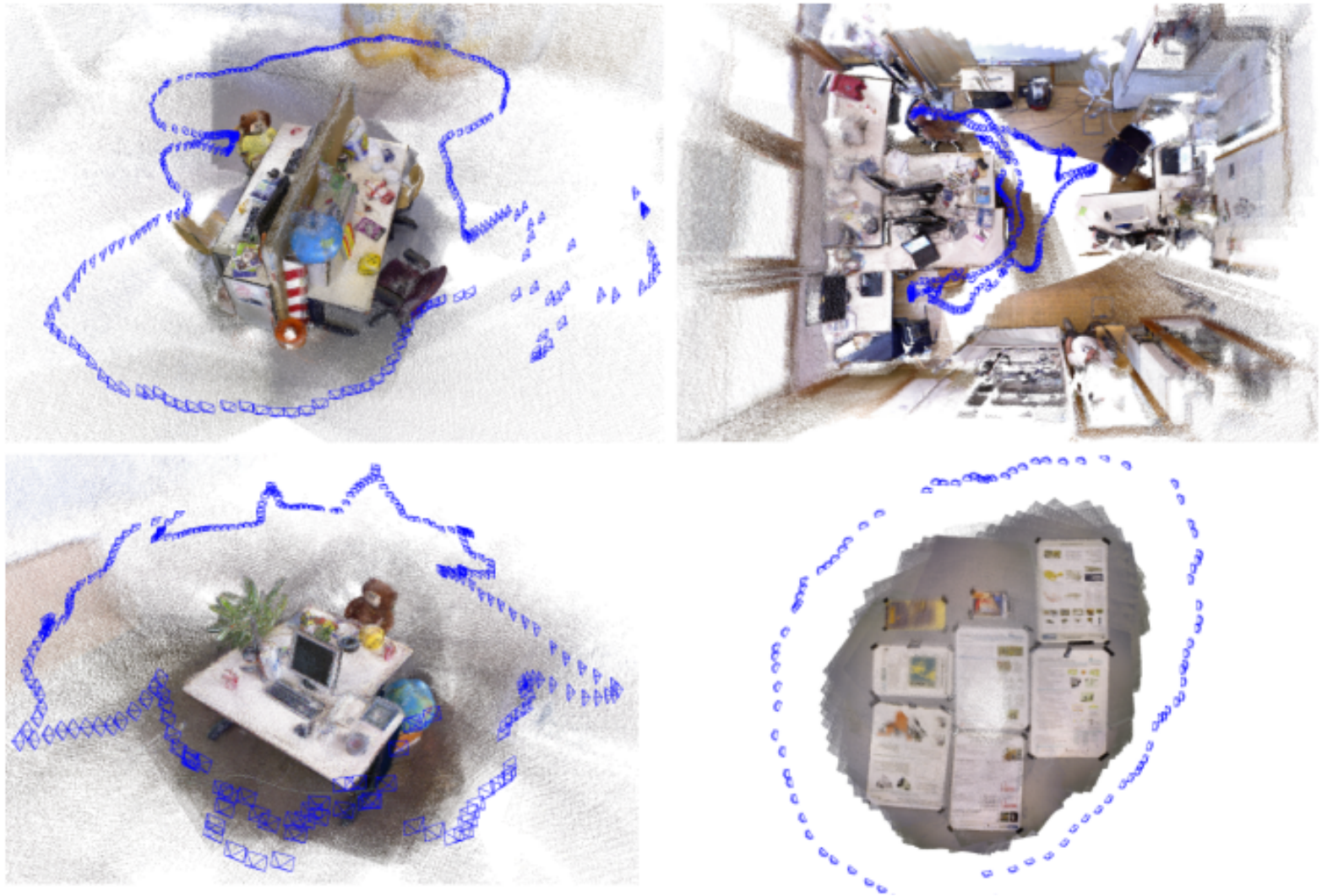
Finally, update estimates when we complete large cycles

Loop Closure

- Want to avoid redundant keyframes if we cover the same area
- Also reduce drift that accumulates if we don’t re-observe previous areas
- Here a full-map BA is run on the keyframes based on the new "loop closing" constraint



Indoor Results w/Kinect



So, what's wrong with this picture?

- Opponent view of the ORBSLAM paper (with help from Travis)
- ORBSLAM works on everything from cellphone cameras to self-driving cars, but it is a bit overly general, so does not exploit robotic sensors or physical constraints
- ORBSLAM is not that efficient computationally

- Initialization requires translation to generate initial MapPoints
  - Assumes translation is on a plane
  - The scale is arbitrary and similar motions can lead to different scales making repeatability difficult
- Stereo cameras allow for single frame initialization
  - Homogeneous coordinates of a point in world fran $\mathbf{X} = [A \;\; B \;\; C \;\; D]^T$
  - Left and Right Projection into camera fran $[u \;\; v \;\; 1]^T = \mathbf{P}_{\text{left}}\mathbf{X}$

$$[u' \;\; v' \;\; 1]^T = \mathbf{P}_{\text{right}}\mathbf{X}$$

  - Solve a linear least squares problem to estimate MapPoint location

$$\begin{aligned} \mathbf{QX} &= \mathbf{0} \\ \text{subject to } \|\mathbf{X}\| &= 1 \end{aligned}$$

$$\mathbf{Q} = \begin{bmatrix} uP^T_{\text{left},3} - P^T_{\text{left},1} \\ vP^T_{\text{left},3} - P^T_{\text{left},2} \\ u'P^T_{\text{right},3} - P^T_{\text{right},1} \\ v'P^T_{\text{right},3} - P^T_{\text{right},2} \end{bmatrix}$$

- Use IMU to make initial estimate of transformation and track the gyro and accelerometer bias in the estimated state:

$$\mathbf{S} = \begin{bmatrix} {}^{LC}_{W}\mathbf{q} & {}^{LC}\mathbf{p}_W & {}^{W}\mathbf{v}_I & \mathbf{b}_g & \mathbf{b}_a \end{bmatrix}$$

- Tight integration of IMU is used to minimize the relative motion error between two frames (S_0 and S_1):

$$\text{RME}(\mathbf{S}_0, \mathbf{S}_1) = \left\| \begin{bmatrix} \bar{\Delta}\mathbf{q}_1^0 \otimes \hat{\Delta}\mathbf{q}_0^1 \\ \bar{\Delta}\mathbf{p}_1^0 - \hat{\Delta}\mathbf{p}_1^0 \\ {}^{W}\bar{\mathbf{v}}_{I_1} - {}^{W}\mathbf{v}_{I_1} \\ \mathbf{b}_{g_1} - \mathbf{b}_{g_0} \\ \mathbf{b}_{a_1} - \mathbf{b}_{a_0} \end{bmatrix} \right\|^2_{\Sigma_{\text{IMU}}}$$

- When merging MapPoints model the position as a Gaussian

$$
\begin{aligned}
\mathbf{K} &= \boldsymbol{\Sigma}_{t-1}\mathbf{J}_g^T(\mathbf{J}_g\boldsymbol{\Sigma}_{t-1}\mathbf{J}_g^T + \boldsymbol{\Sigma}_z)^{-1} \\
{}^W\Delta\mathbf{f} &= \mathbf{K}(\mathbf{z} - \mathbf{g}({}^{LC}_{W}\mathbf{q}, {}^{LC}\mathbf{p}_W, {}^W\mathbf{f}_{t-1})) \\
\boldsymbol{\Sigma}_t &= \boldsymbol{\Sigma}_{t-1} - \mathbf{K}\mathbf{J}_g\boldsymbol{\Sigma}_{t-1}
\end{aligned}
$$

  - During g2o graph optimization, proportionally weight the MapPoints with their respective variance.

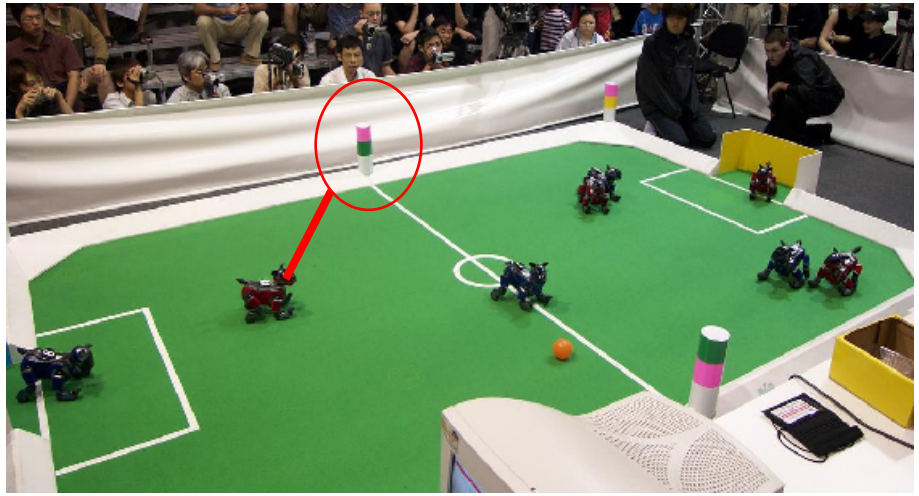
ORBSLAM Compute

- 3 threads required, which are not available on always available on low-powered robots
  - Even if they are, a robot should do many more tasks!
- Representing features and visibility in 3D turns out to be costly. Other methods maintain a 2D matched-point representation only.
- The number of keyframes, number of features and precise error functions used are not as elegant nor efficient as some other methods
- See more recent papers: Romouliotis (project Tango) among others.

Beyond the KF Assumptions

- We used the linear forms of motion and measurement (e.g., z=Hx+noise) at several stages to make the equations easier
- But, this was not a good thing in general. Real sensors almost all make ***nonlinear measurements!***
  - E.g., the robots below sense the distance and angle to the landmarks (l) at the edge of the field
  - To form this equation, need trigonometry and square root of squares
  - These are not linear, so are we stuck?

$$z_t^{(i)} = h_i(x_t) = \begin{bmatrix} \sqrt{(p_x(t) - l_x^{(i)})^2 + (p_y(t) - l_y^{(i)})^2} \\ \text{atan2}(p_y(t) - l_y^{(i)}, \quad p_x(t) - l_x^{(i)}) - \theta(t) \end{bmatrix} + n_t$$



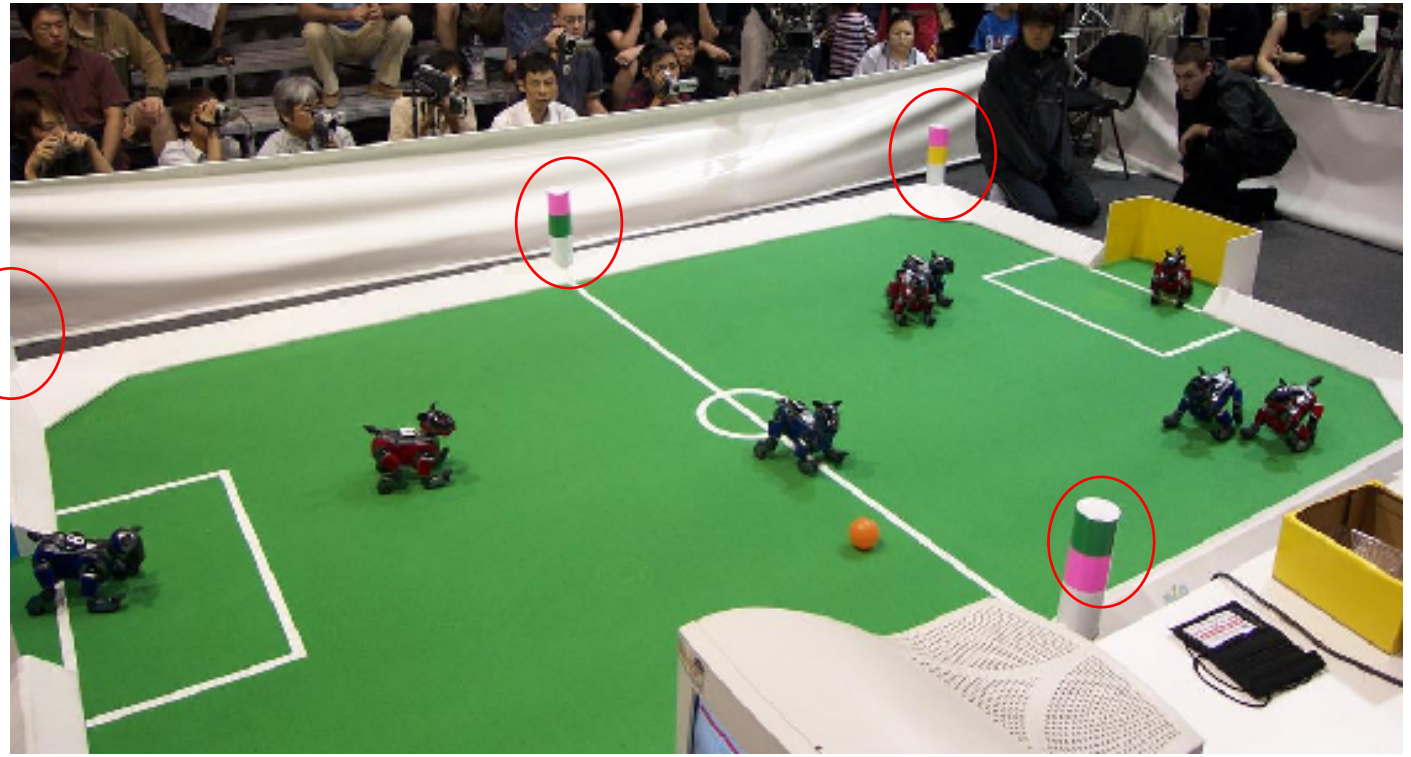
Landmark-based Localization

Landmarks, whose position $(l_x^{(i)}, l_y^{(i)})$ in the world is known.

Each robot measures its range and bearing from each landmark to localize itself.

State of a robot:

$$x_t = \begin{bmatrix} p_x(t) \\ p_y(t) \\ \theta(t) \end{bmatrix}$$



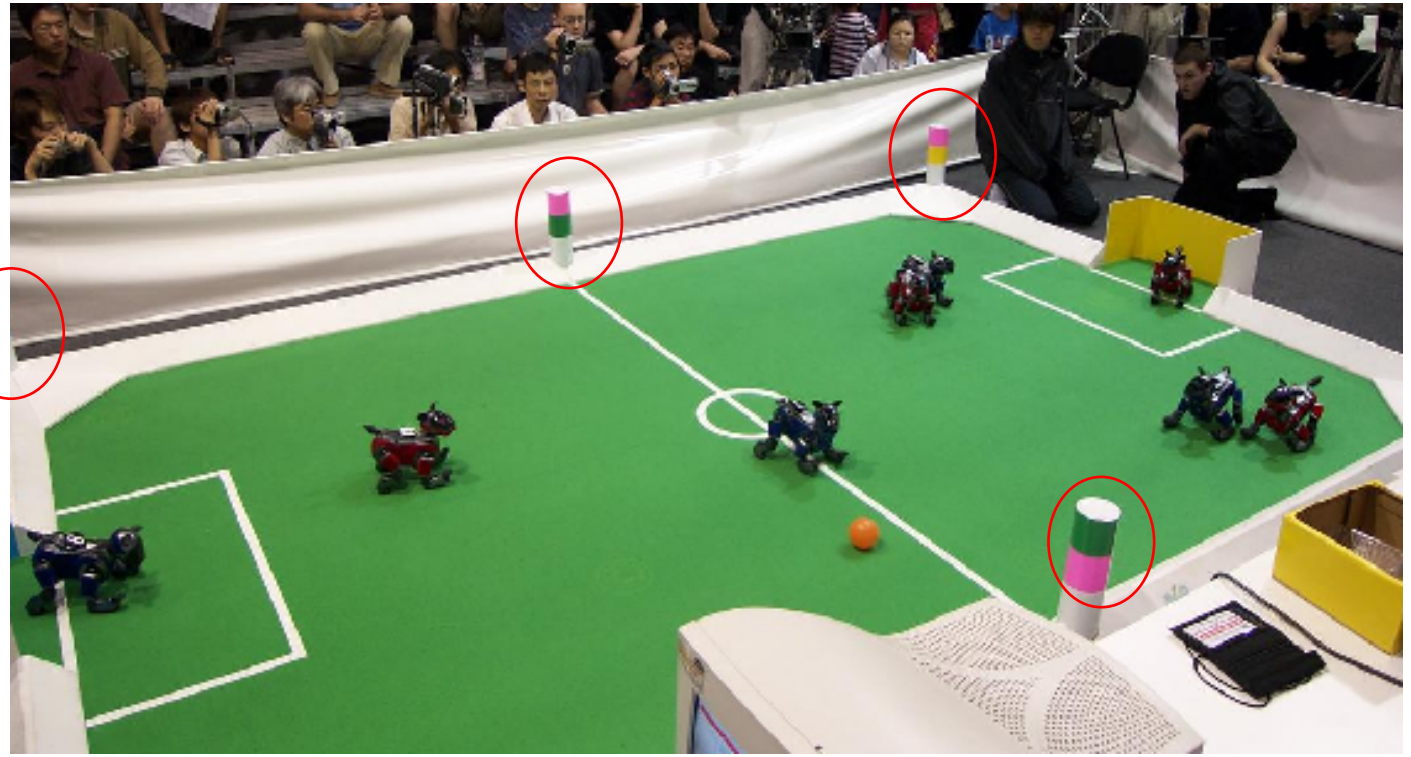
Landmark-based Localization

Measurement at time t,

$$z_t = \begin{bmatrix} \dots \\ z_t^{(i)} \\ \dots \end{bmatrix}$$

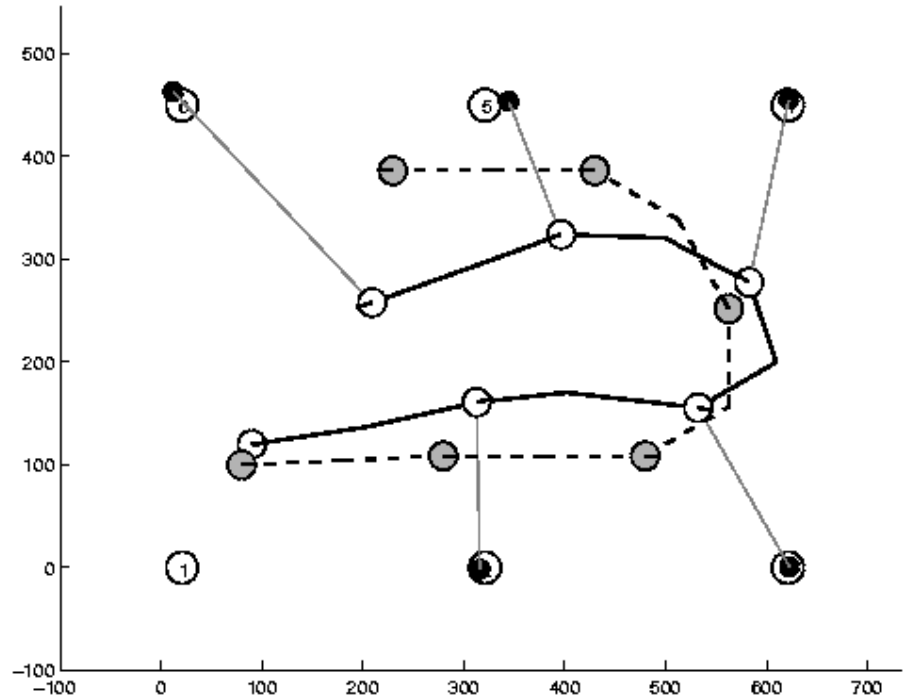
is a variable-sized vector, depending on the landmarks that are visible at time t.

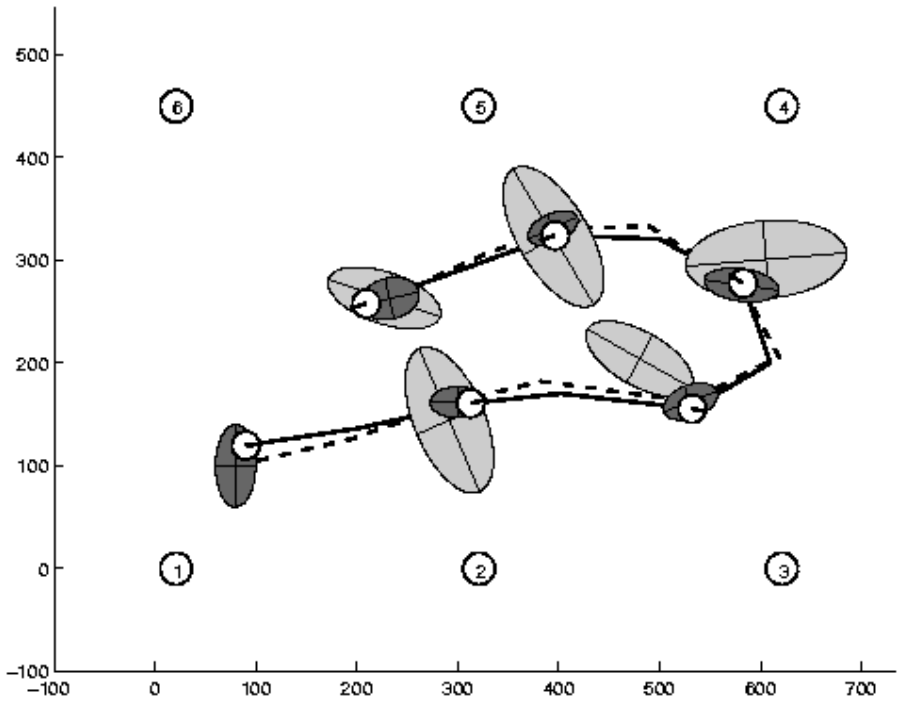
Each measurement is a 2D vector, containing range and bearing from the robot to a landmark.



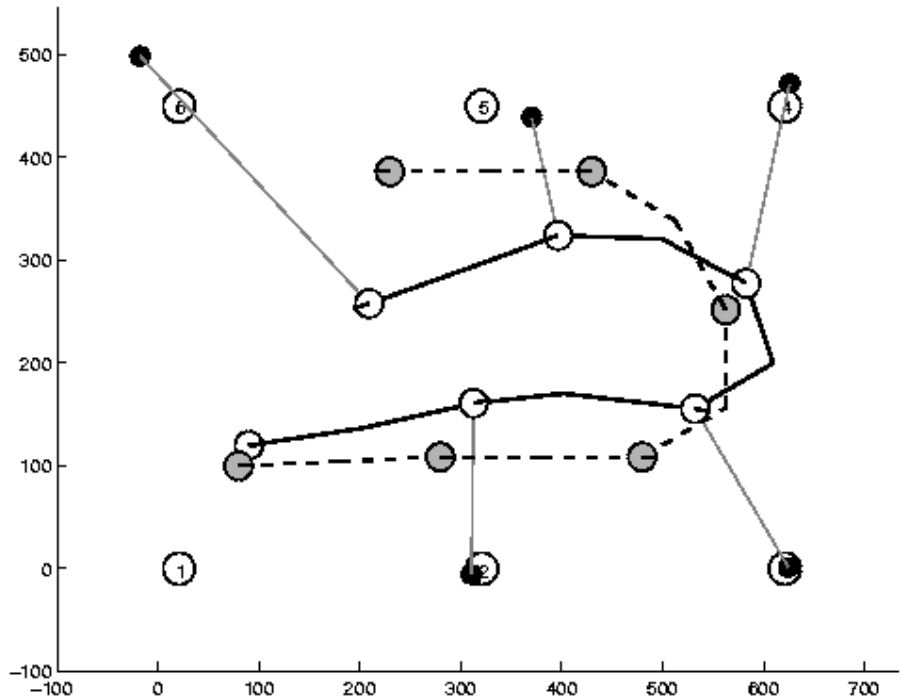
$$z_t^{(i)} = h_i(x_t) = \begin{bmatrix} \sqrt{(p_x(t) - l_x^{(i)})^2 + (p_y(t) - l_y^{(i)})^2} \\ \text{atan2}(p_y(t) - l_y^{(i)}, \quad p_x(t) - l_x^{(i)}) - \theta(t) \end{bmatrix} + n_t$$

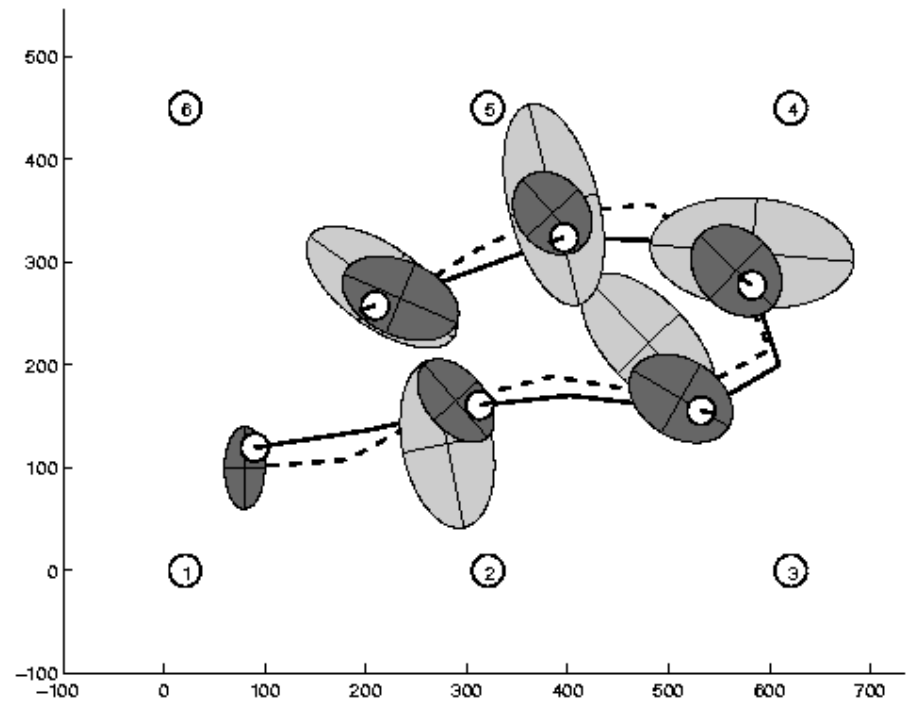
Estimation Sequence (1)





Estimation Sequence (2)





Comparison to true trajectory