Mapping and Navigation

Principles and Shortcuts

January 17th, 2006

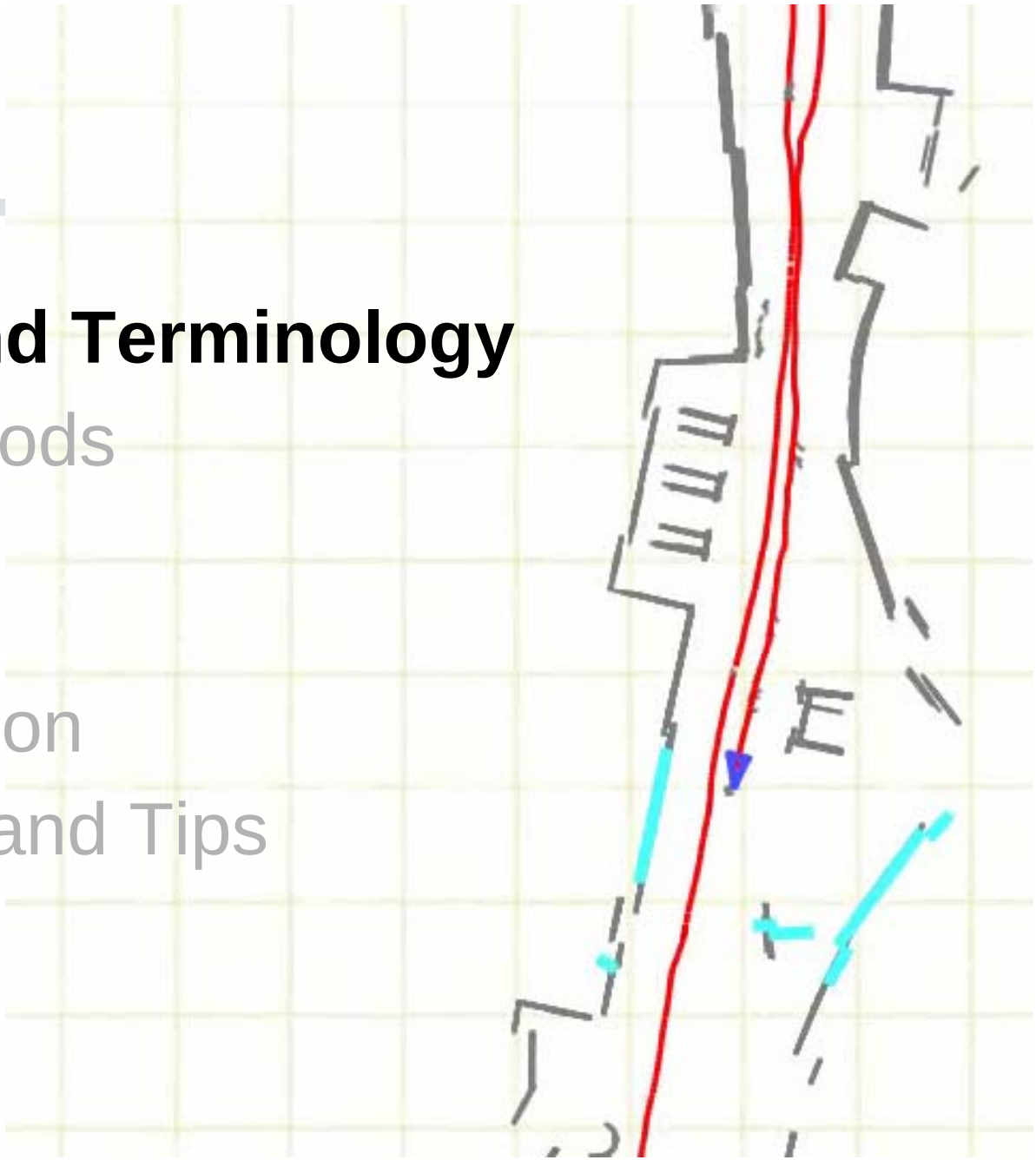
Edwin Olson, eolson@mit.edu

Goals for this talk

- Principles
  - Present fundamental concepts, algorithms
  - Give an idea of how rigorous methods work
- Shortcuts
  - Present some simple but workable approaches

Attack Plan

- **Motivation and Terminology**
- Mapping Methods
  - Topological
  - Metrical
- Data Association
- Sensor Ideas and Tips



Why build a map?

- Time
  - Playing field is big, robot is slow
  - Driving around perimeter takes a minute!
  - Scoring takes time… often ~20 seconds to “line up” to a mouse hole.
- MASLab ’06 scoring nonlinearity
  - Better to put balls in *different* goals, but which goals have been visited?

MASLab ‘06 Mapping Goals

1. Be able to efficiently move to specific locations that we have previously seen
   - I’ve got the bonus ball, now, where was that bonus gate?!
   - Where is the nearest gate?

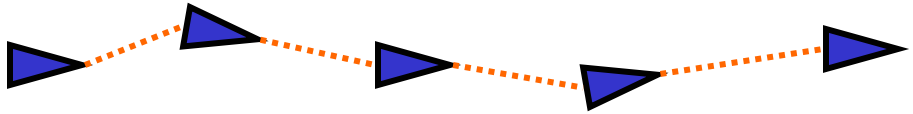
- Be able to efficiently explore unseen areas
  - I just turned on. What do I do?
  - I don’t know where the bonus ball/gate is
  - I’ve put too many balls in this gate; I need to find a new gate.

Note: Producing the map is not a goal in itself, but it might be a good way to win the MASLab Engineering Award!

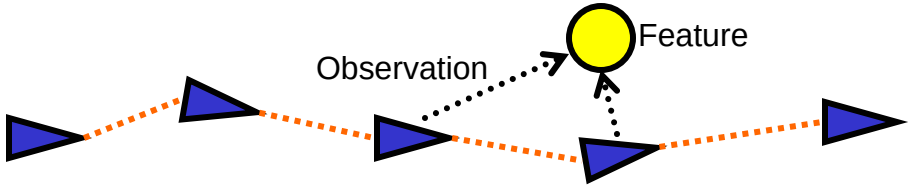
Definitions

- **Pose:** a place where the robot has been

- **Trajectory:** the linked set of poses

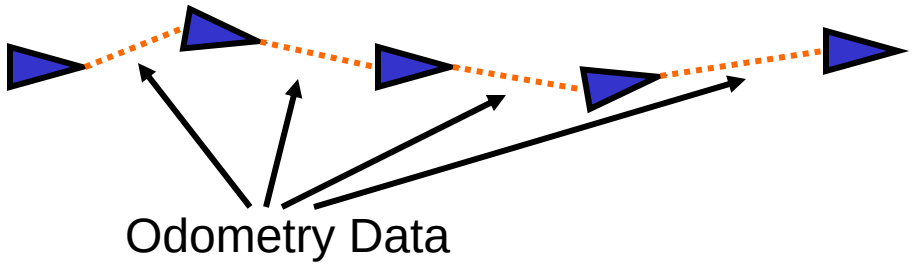


- **Feature:** Something in the world that we represent in our map that we can *observe.*

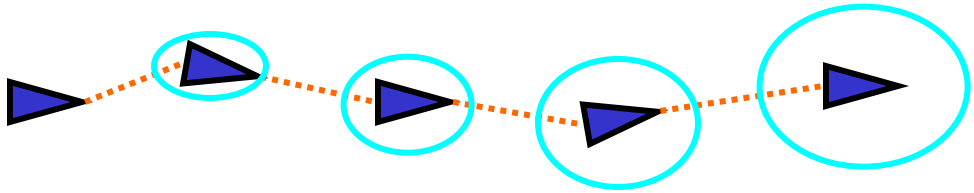


Odometry Trajectory

- Integrating odometry data yields a trajectory

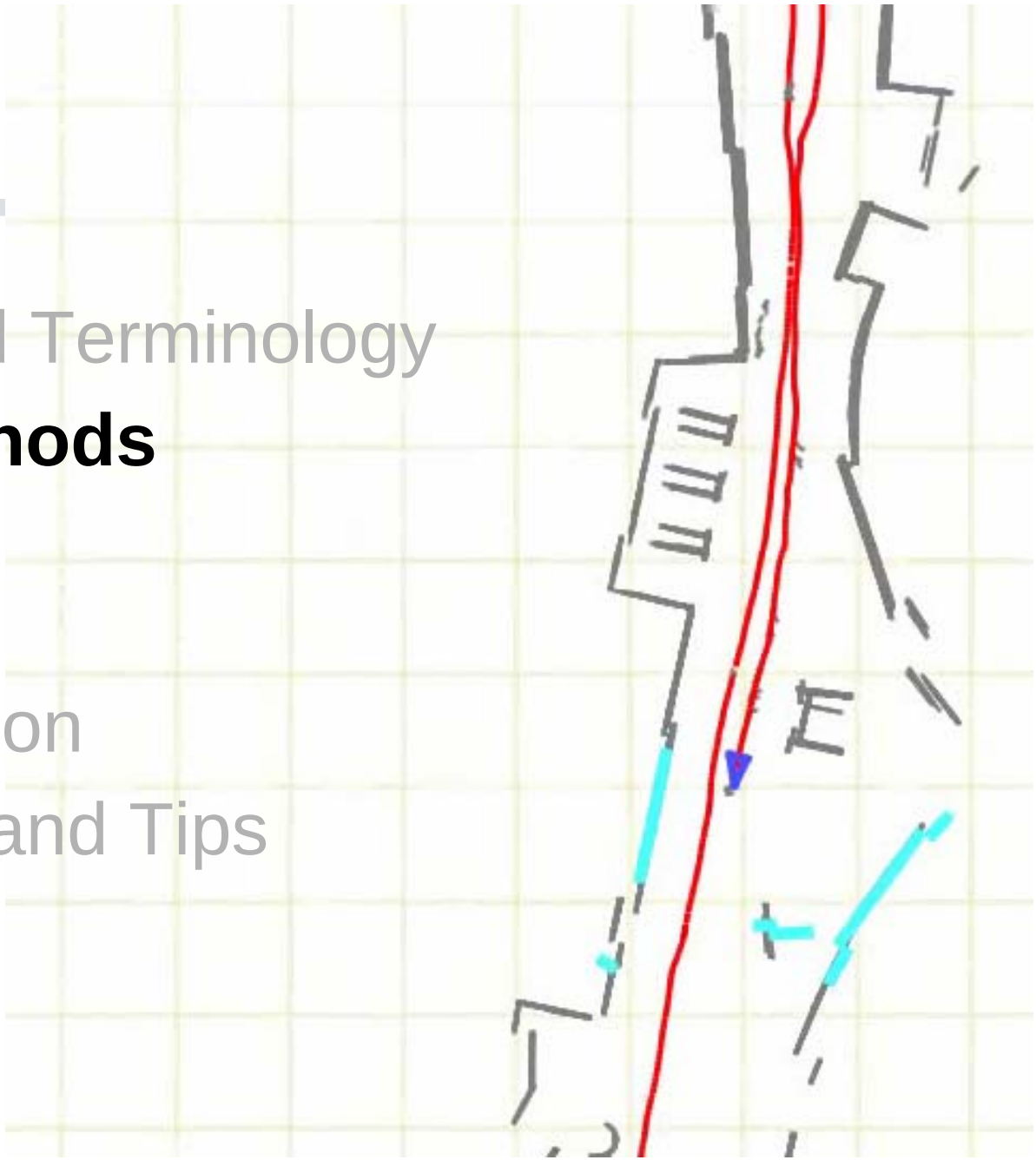


- Uncertainty of pose increases at every step



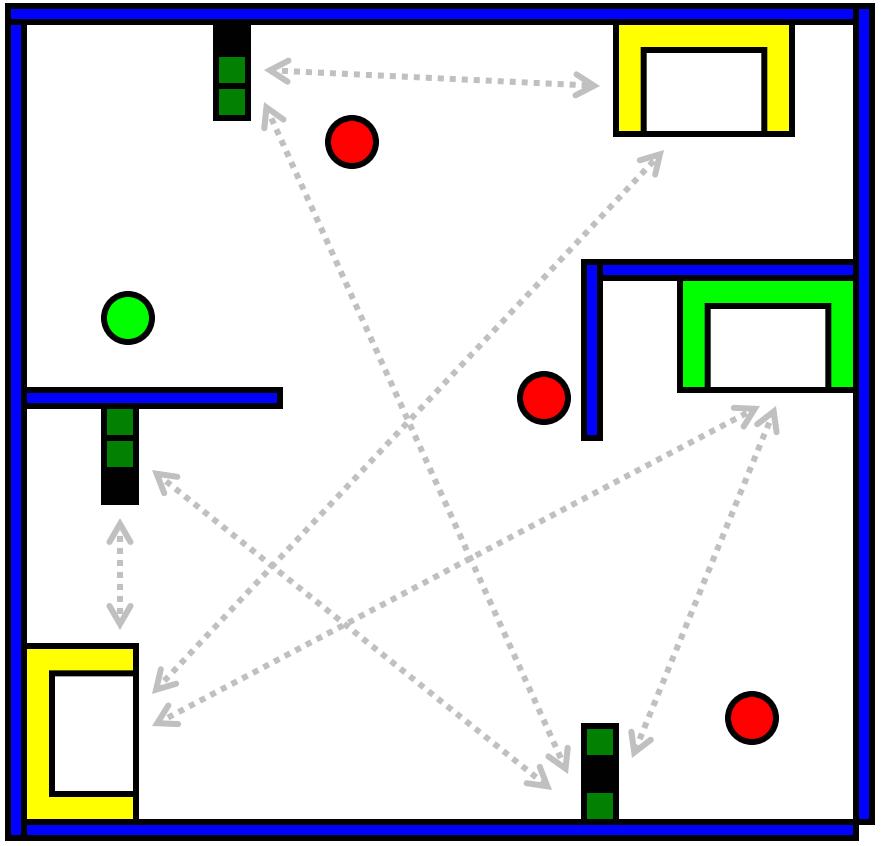
Attack Plan

- Motivation and Terminology
- **Mapping Methods**
  - **Topological**
  - Metrical
- Data Association
- Sensor Ideas and Tips



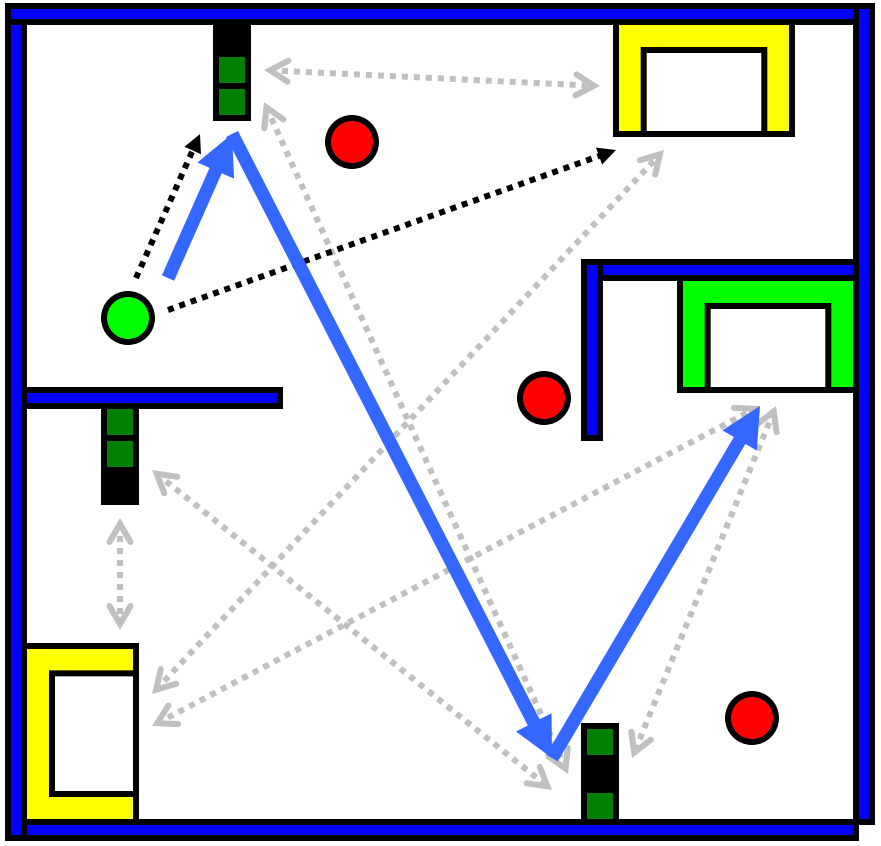
Visibility Graphs

- A type of “topological map”:
  - Which features can see each other?
  - Edges needn’t contain *any* distance/angle information.
- Easy to build/update
  - No math!
- Accumulated odometry error has no impact
- Provides a “highway system” allowing navigation from one feature to another



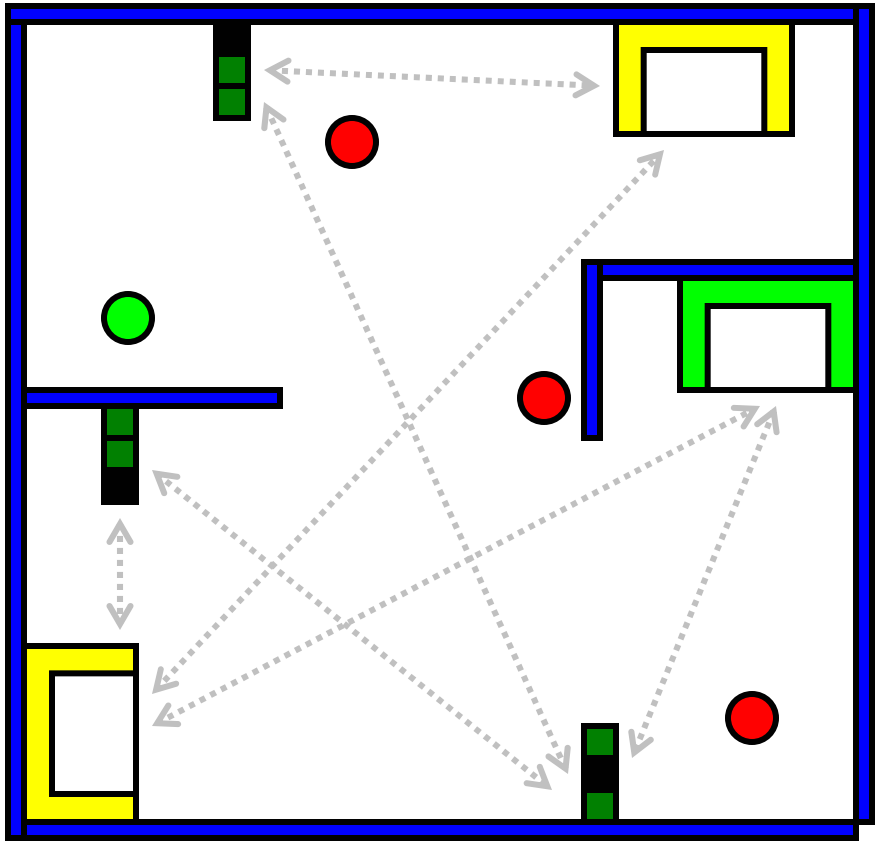
Visibility Graph: Example

- You've just grabbed the green ball
- Compute local visibility (do a 360)
  - You see a two features
- Do a tree search for the shortest path to the green goal
  - Shortest = fewest hops?
  - Edges can contain distance *estimates*
- Drive to the next visible target, search (do a 360) for next target on path…



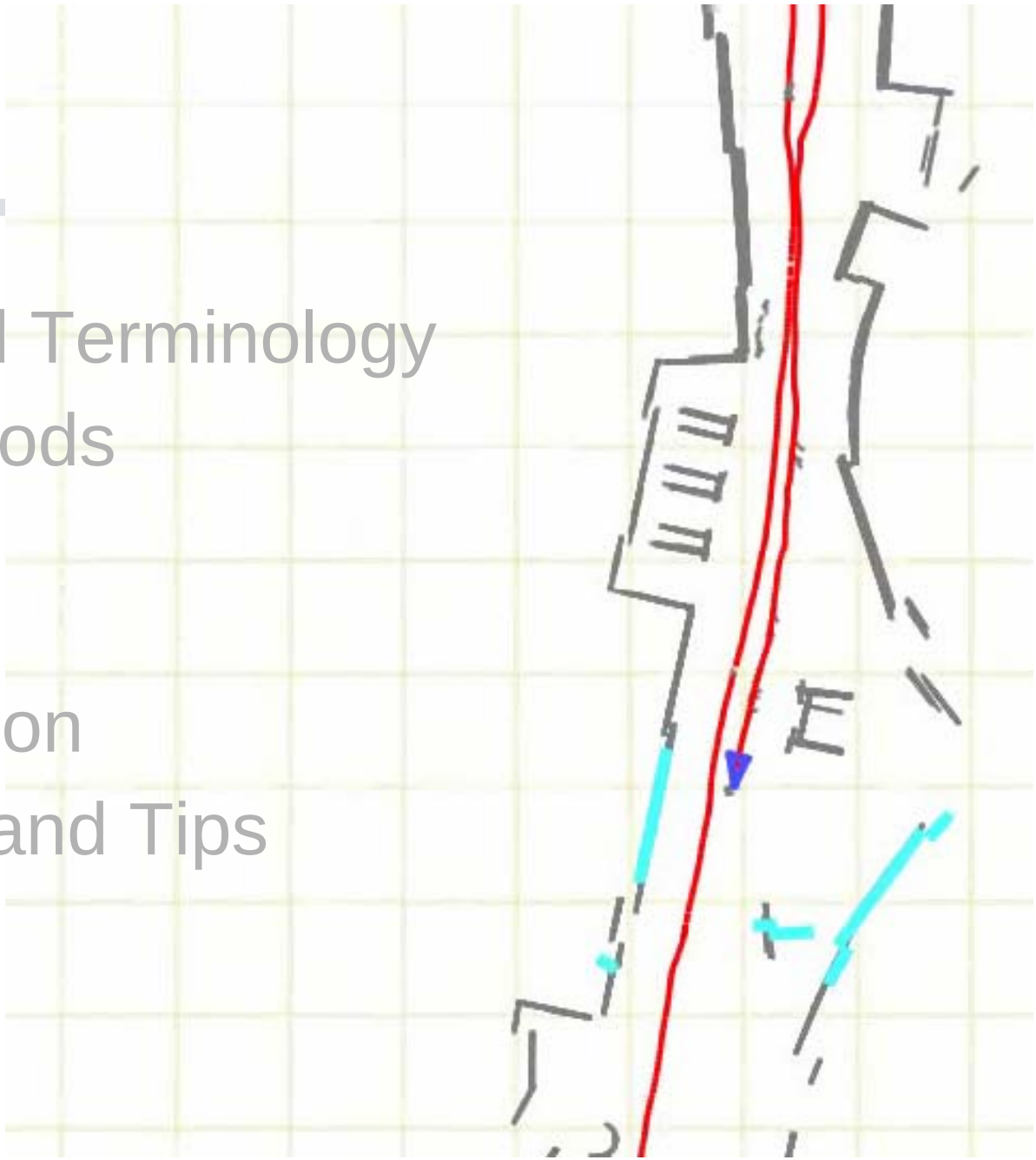
Visibility Graphs: Problems

- How to handle balls?
  - Use hints (“I last saw 3 red balls plus the power ball”)
  - Balls become features?
    - Are they identifiable by their visibility?
- Define “visible” (do I have to do 360s frequently?)
  - Do a 360 only once per feature, remembering the relative angles of other features.
- How do you know where unexplored areas are?
  - Random walk?
- Generated paths are suboptimal
- Data Association: is this goal the same as the one I saw over there?



Attack Plan

- Motivation and Terminology
- Mapping Methods
  - Topological
  - **Metrical**
- Data Association
- Sensor Ideas and Tips



Metrical Maps

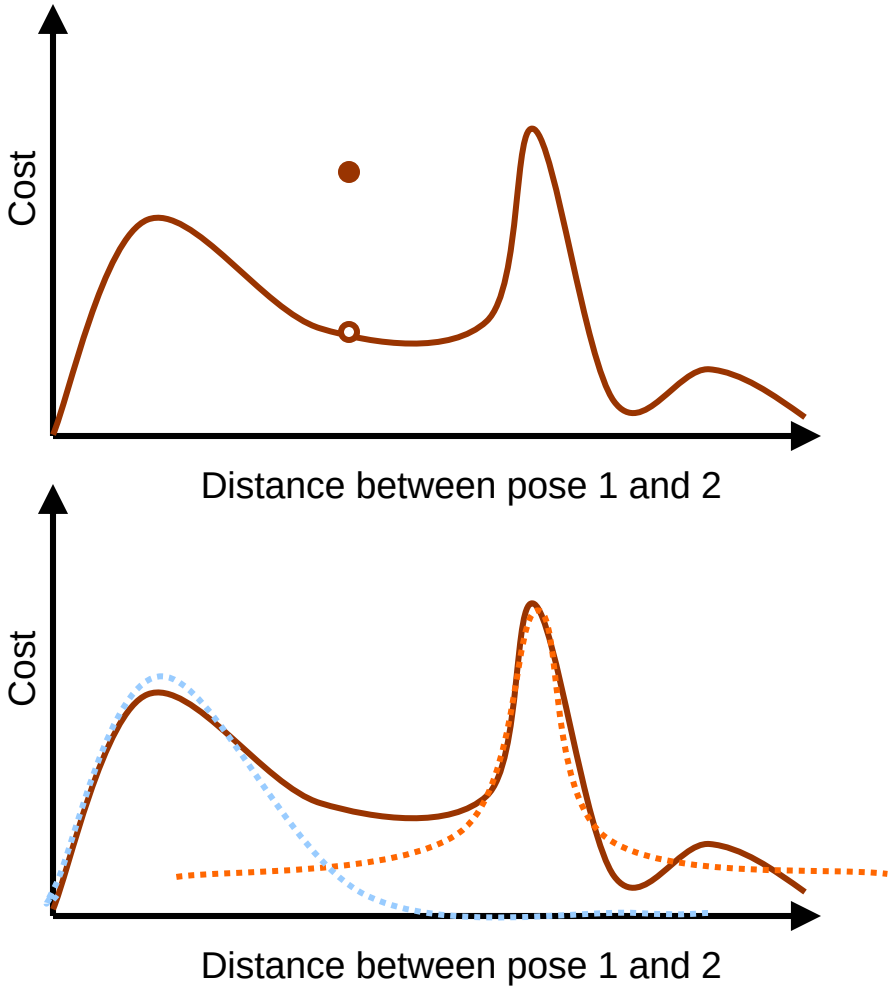
- Try to find actual locations (or parameters) of features
  - Ball is at (x,y)
  - Wall is parameterized by (e.g.): $(x_1,y_1),(x_2,y_2)$
- Advantages
  - Know where unexplored territory is
  - Compute optimal paths
  - Location information makes data association easier
- Disadvantages
  - Accumulating odometry error makes this difficult
  - Math can be more difficult

Metrical Maps

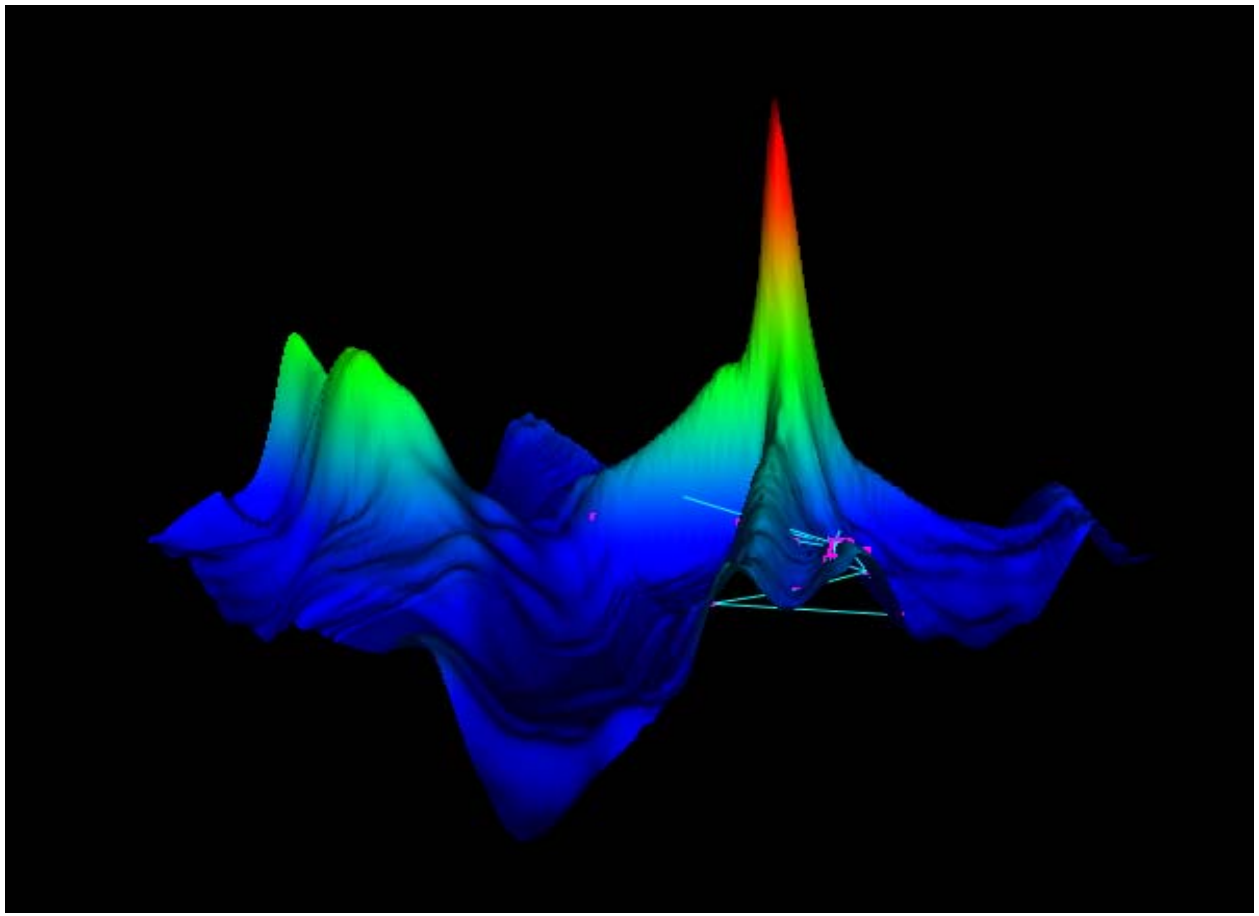
- Basic idea:
  - Make all features and robot poses unknown variables
  - Make a big list of equations constraining the value of those variables
    - These come from observations, like “pose 14 is 3.2 meters from feature 37”
    - Equations compute a cost as a function of the poses
  - Solve for poses, minimizing the total cost

Metrical Map: Cost Function

- Cost function *could* be arbitrarily complicated
  - Optimization of these is intractable
- We can make a local approximation around *the current pose estimates*
  - Resembles the arbitrary cost function in that neighborhood
  - Typically Gaussian



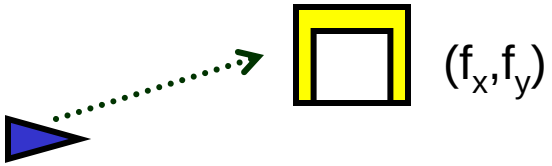
Metrical Map: Real World Cost Function



Cost function arising from aligning two LADAR scans

Metrical Map: Cost Example

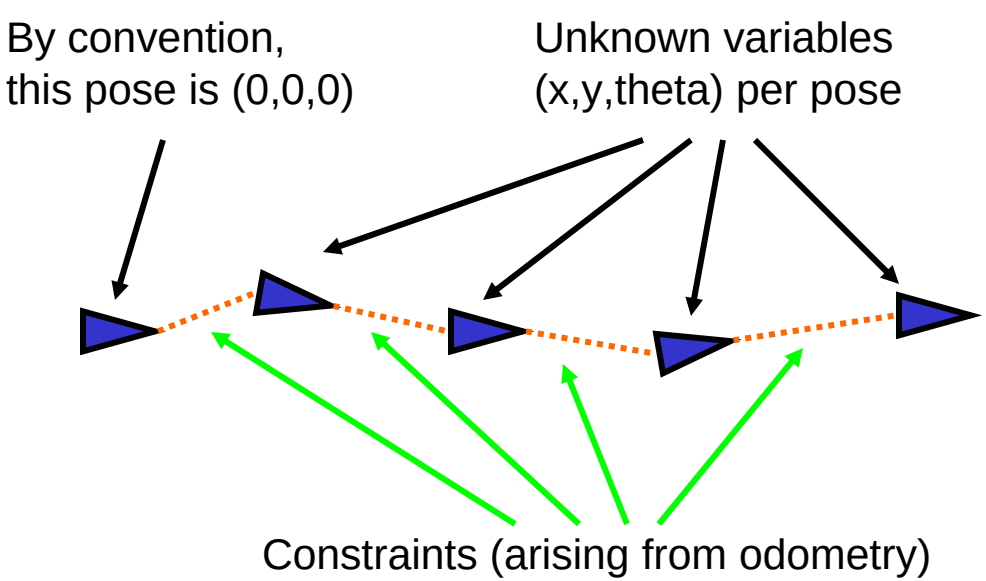
- Suppose we observe the distance to a goal z_0
- Governing equation:
  - $z = [(p_x-f_x)^2+(p_y-f_y)^2]^{1/2}$
  - z = prediction
  - z_0 = observation
- Assume (or approximate) cost:
  - Cost = $W(z-z_0)^2 = (z-z_0)^T W(z-z_0)$
  - W is a "weight" (related to covariance,
    - $W = \Sigma^{-1}$
- Linearize z:
  - $z = Jx + b$, $(z-z_0) = Jx + b - z_0 = Jx - r$
  - J is Jacobian of z with respect to all x
  - Cost = $(Jx-r)^T W(Jx-r)$
- Differentiate cost with respect to unknowns (p_x,p_y,f_x,f_y), set to 0:
  - $J^T WJx = J^T Wz_0$

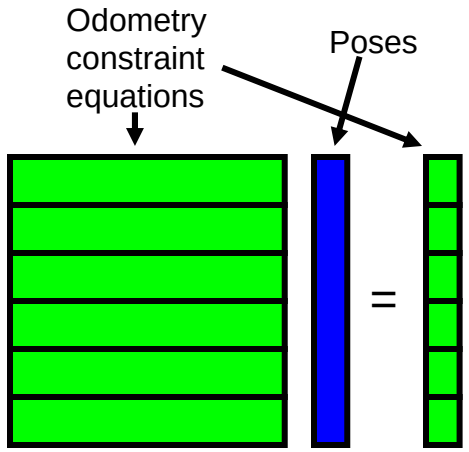


Jacobian Terms:

$dz/dp_x = z^{-1/2}(p_x-f_x)$
$dz/dp_y = z^{-1/2}(p_y-f_y)$
$dz/df_x = - z^{-1/2}(p_x-f_x)$
$dz/df_y = - z^{-1/2}(p_y-f_y)$

Metrical Map example





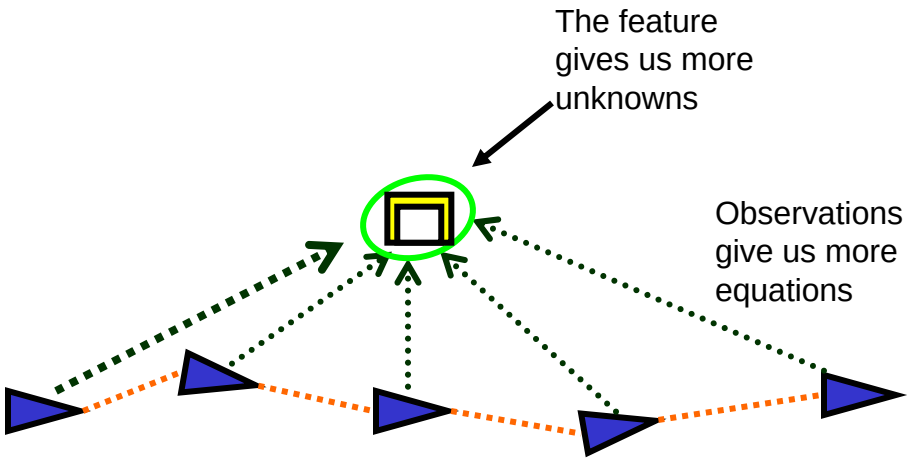
$J^T W J \; x = J^T W z_0$

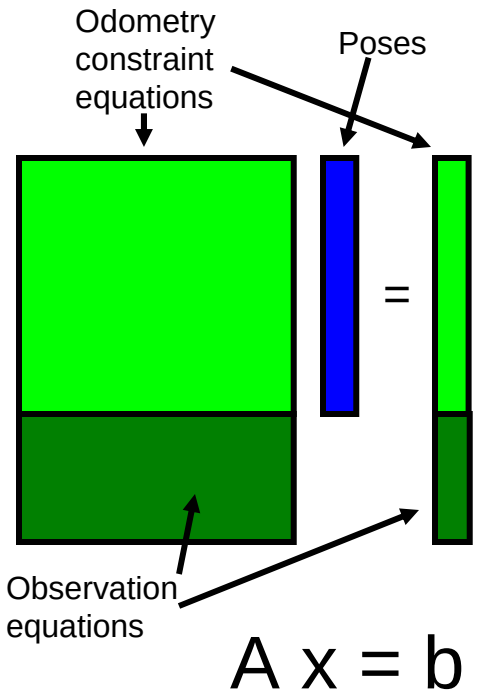
$A x = b$

number unknowns==number of equations, solution is critically determined.

$x = A^{-1} b$

Metrical Map example





number unknowns < number of equations, solution is *over determined.*
Least-squares solution is:

$$x = (A^T A)^{-1} A^T b$$

More equations = better pose estimate

Metrical Map example

1. Original Trajectory with odometry constraints

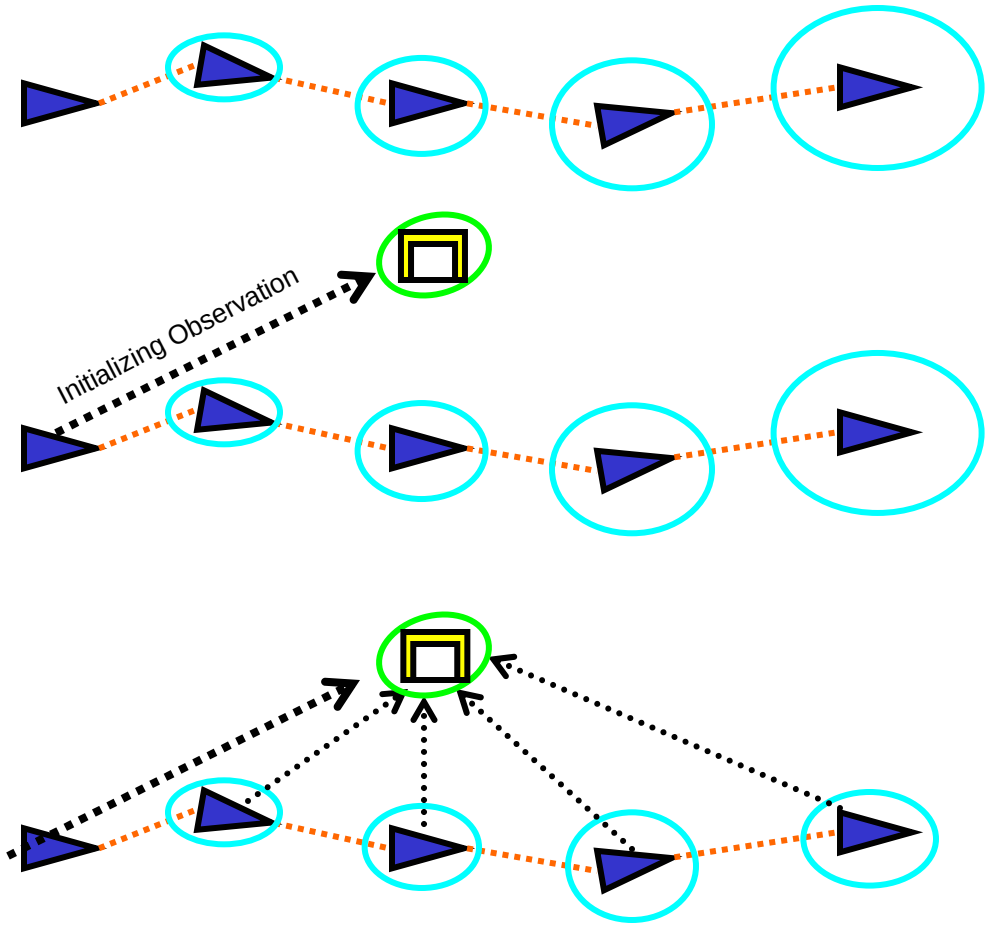
2. Observe external feature

   Initial feature uncertainty =

   pose uncertainty +

   observation uncertainty

3. Reobserving feature helps subsequent pose estimates



Gotcha!

- The least-squares solution to the mapping problem:

$$x = (A^T A)^{-1} A^T b$$

- Must invert a matrix of size 3Nx3N (N = number of poses.) Inverting this matrix costs $O(N^3)$!

- We can choose to "forget" robot trajectory, and use only most recent pose.
  - Reduces computational complexity
  - Lose valuable information?

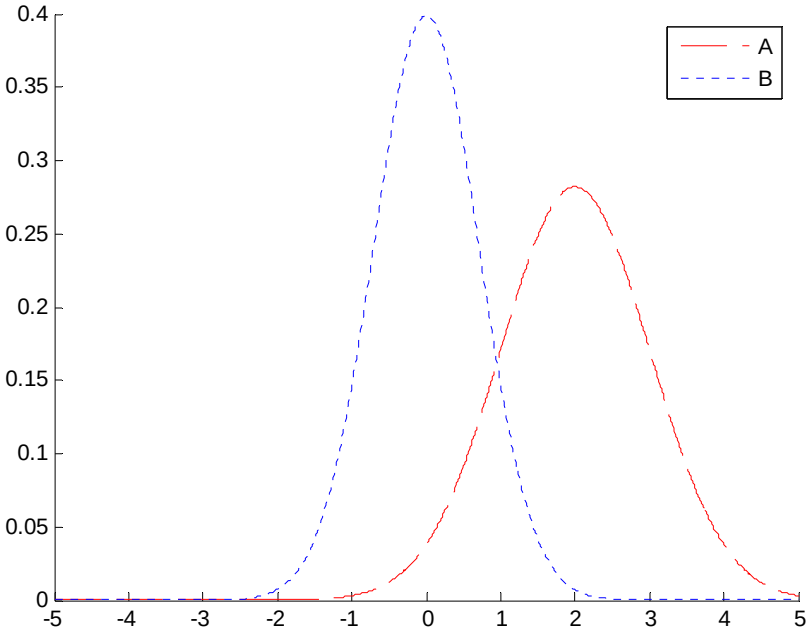
* We'd never actually invert it; that's numerically unstable. Instead, we'd use a Cholesky Decomposition or something similar. But it has the same computational complexity.

Extended Kalman Filter

- If we assume all error is Gaussian, Extended Kalman Filter reduces time complexity to $O(N^2)$.
- EKF allows us to add one observation at a time, rather than resolving the entire system.

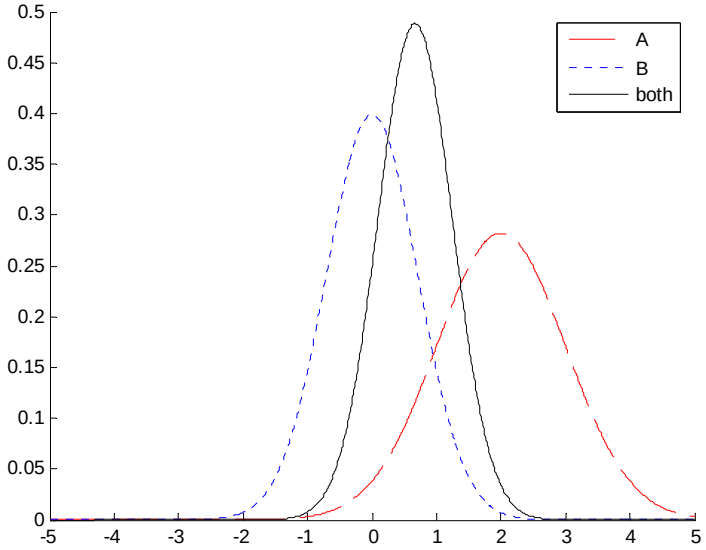
Extended Kalman Filter

- Example: Estimating where Jill is standing:
  - Alice says: x=2
    - We think σ^2 =2; she wears thick glasses
  - Bob says: x=0
    - We think σ^2 =1; he's pretty reliable
- How do we combine these measurements?



Simple Kalman Filter

- Answer: algebra (and a little calculus)!
  - Compute mean by finding maxima of the log probability of the product $P_A P_B$.
  - Variance is messy; consider case when $P_A = P_B = N(0,1)$
- *Try deriving these equations at home!*

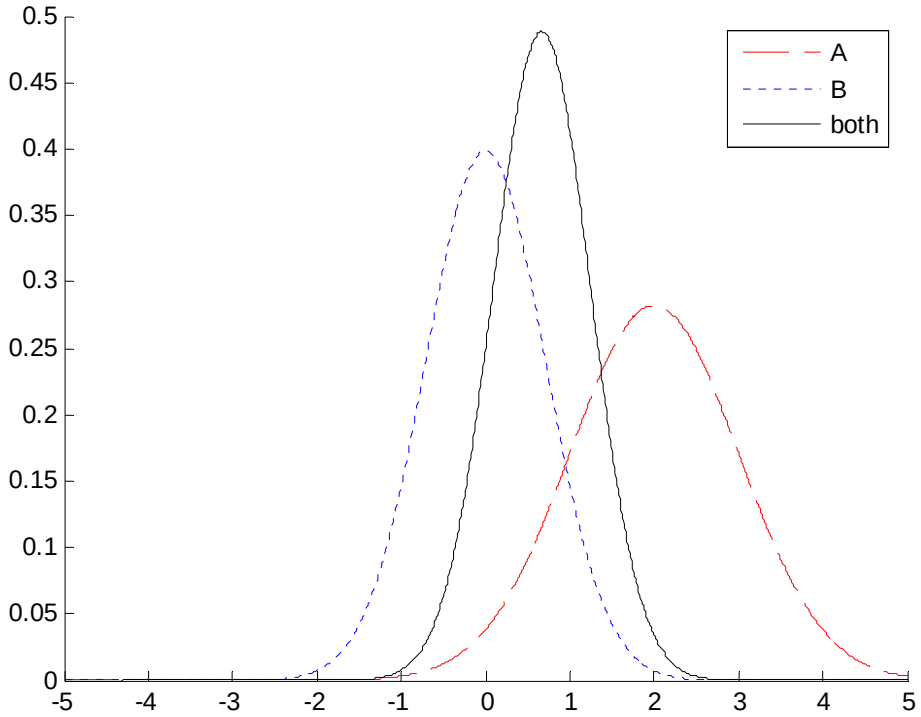


$$\frac{1}{\sigma^2} = \frac{1}{\sigma_A{}^2} + \frac{1}{\sigma_B{}^2}$$

$$\mu = \frac{\mu_A \sigma_B{}^2 + \mu_B \sigma_A{}^2}{\sigma_A{}^2 + \sigma_B{}^2}$$

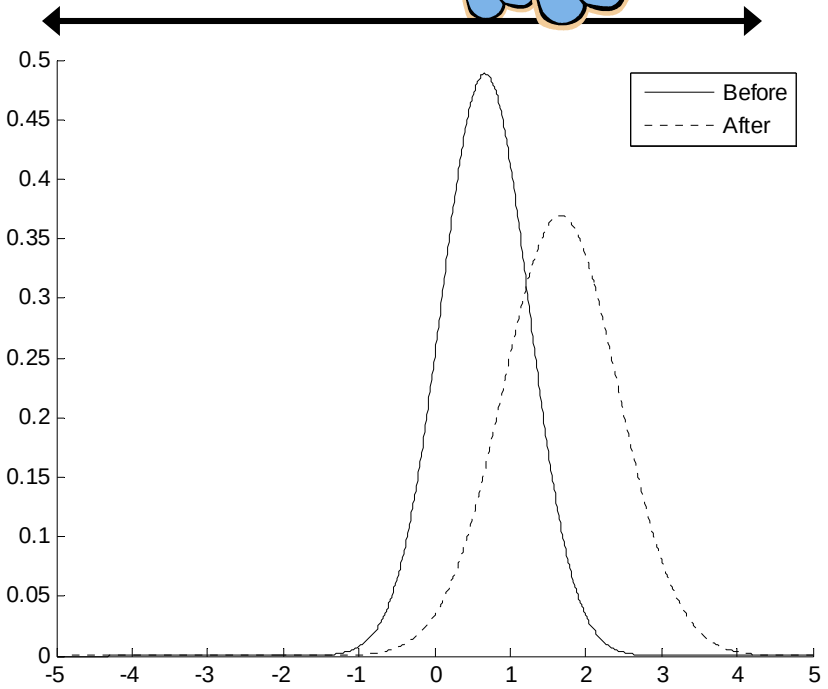
Kalman Filter Example

- We now think Jill is at:
  - x = 0.66
  - σ^2 =0.66
- Note: Observations *always* reduce uncertainty
  - Even in the face of conflicting information, EKF never becomes less certain.



Kalman Filter

- Now Jill steps forward one step
- We think one of Jill's steps is about 1 meter, $\sigma^2 = 0.5$
- We estimate her position:
  - $x = x_{before} + x_{change}$
  - $\sigma^2 = \sigma_{before}^2 + \sigma_{change}^2$
- Uncertainty *increases*



Kalman Filter: Properties

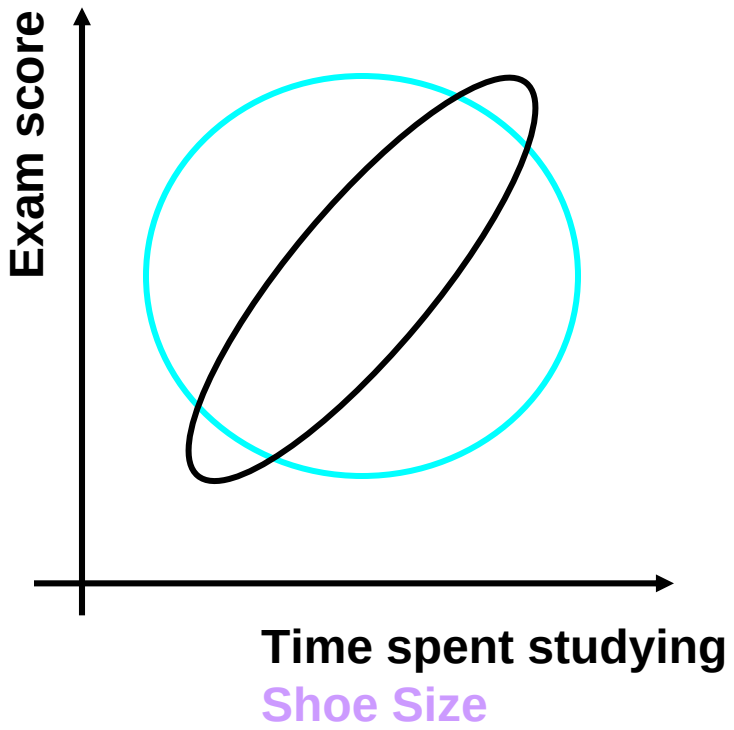
- You incorporate sensor observations one at a time.
- Each successive observation is the same amount of work (in terms of CPU).
- *Yet, the final estimate is the **global** optimal solution.*
  - The same solution we would have gotten using least-squares.

The Kalman Filter is an *optimal,*

recursive estimator.

Correlation/Covariance

- In multidimensional Gaussian problems, equal-probability contours are ellipsoids.
- Shoe size doesn't affect grades:
  P(grade,shoesize)=P(grade)P(shoesize)
- Studying helps grades:
  P(grade,studytime)!=P(grade)P(studytime)
  - We must consider P(x,y) jointly, respecting the correlation!
  - If I tell you the grade, you learn something about study time.

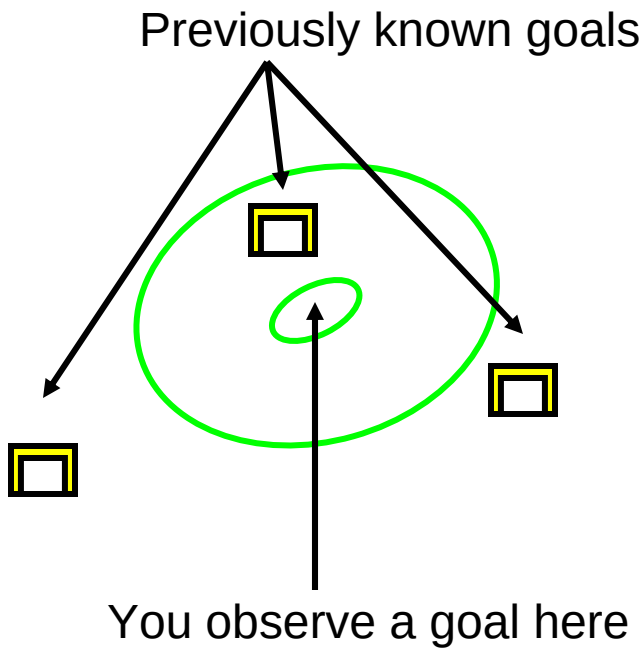


State Correlation/Covariance

- We observe features relative to the robot's current position
  - Therefore, feature location estimates *covary* (or correlate) with robot pose.
- Why do we care?
  - We get the wrong answer if we don't consider correlations
  - Covariance is useful!

Why is covariance useful?

- Loop Closing (and Data Association)
- Suppose you observe a goal (with some uncertainty)
  - Which previously-known goal is it?
  - Or is it a new one?
- Covariance information helps you decide



System Equations (EKF)

- Consider range/bearing measurements, differentially driven robot
- Let $x_k=f(x_{k-1},u_{k-1}, w_{k-1})$ u=control inputs, w=noise
- Let $z_k=h(x_k,v_k)$ v=noise

$$f = \begin{pmatrix} x' = x + (u_d + w_d)\cos(\theta + w_\theta) \\ y' = y + (u_d + w_d)\sin(\theta + w_\theta) \\ \theta' = \theta + u_\theta + w_\theta \end{pmatrix}$$

$$h = \begin{pmatrix} z_d = [(x_f - x_r)^2 + (y_f - y_r)^2]^{1/2} + v_d \\ z_\theta = \arctan 2(y_f - y_r, x_f - x_r) - x_\theta + v_\theta \end{pmatrix}$$

EKF Update Equations

- Time update:
  - x'=f(x,u,0)
  - $P=APA^T+WQW^T$
- Observation
  - $K=PH^T(HPH^T + VRV^T)^{-1}$
  - x'=x+K(z-h(x,0))
  - P=(I-KH)P
- P is your covariance matrix
- They look scary, but once you compute your Jacobians, it just works!
  - A=df/dx W=df/dw H=dh/dx V=dh/dv
  - Staff can help… (It's easy except for the atan!)

$$f = \begin{pmatrix} x' = x + (u_d + w_d)\cos(\theta + w_\theta) \\ y' = y + (u_d + w_d)\sin(\theta + w_\theta) \\ \theta' = \theta + u_\theta + w_\theta \end{pmatrix}$$

$$h = \begin{pmatrix} z_d = [(x_f - x_r)^2 + (y_f - y_r)^2]^{1/2} + v_d \\ z_\theta = \arctan 2(y_f - y_r, x_f - x_r) - x_\theta + v_\theta \end{pmatrix}$$

EKF Jacobians

$$f = \begin{pmatrix} x' = x + (u_d + w_d)\cos(\theta + w_\theta) \\ y' = y + (u_d + w_d)\sin(\theta + w_\theta) \\ \theta' = \theta + u_\theta + w_\theta \\ x_1' = x_1 \\ y_1' = y_1 \end{pmatrix}$$

$$d = [(x_f - x_r)^2 + (y_f - y_r)^2]^{1/2}$$

$$d_x = x_f - x_r$$

$$d_y = y_f - y_r$$

$$A = \begin{vmatrix} 1 & 0 & -u_d \sin(\theta) & 0 & 0 \\ 0 & 1 & u_d \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix}$$

$$W = \begin{vmatrix} \cos(\theta) & -u_d \sin(\theta) \\ \sin(\theta) & u_d \cos(\theta) \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{vmatrix}$$

$$Q = \begin{vmatrix} \sigma_{w_d}^{\ 2} & 0 \\ 0 & \sigma_{w_\theta}^{\ 2} \end{vmatrix}$$

EKF Jacobians

$$h = \begin{pmatrix} z_d = [(x_f - x_r)^2 + (y_f - y_r)^2]^{1/2} + v_d \\ z_\theta = \arctan 2(y_f - y_r, x_f - x_r) - x_\theta + v_\theta \end{pmatrix}$$

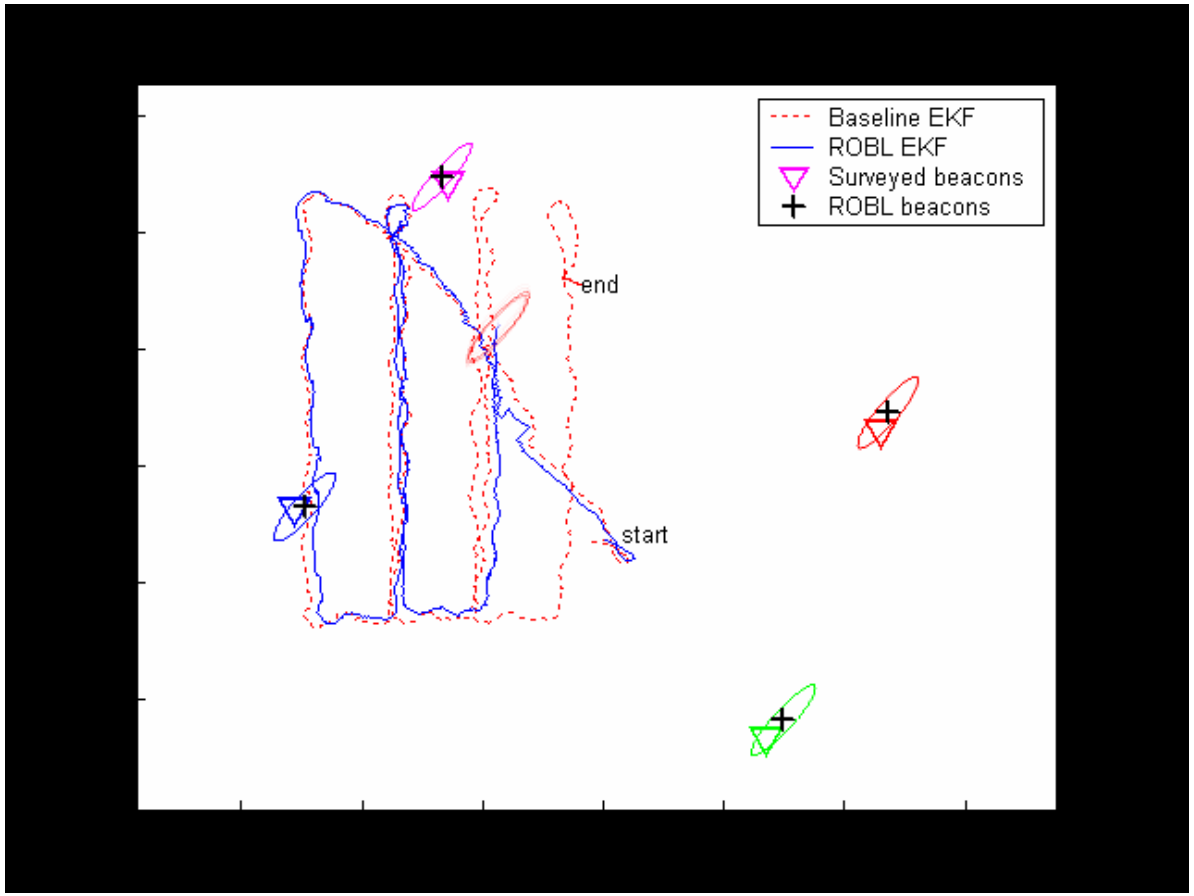
$$\lambda = 1/(1 + (d_y / d_x))^2$$

$$H = \begin{vmatrix} -d_x / d & -d_y / d & 0 & d_x / d & d_y / d \\ \lambda d_y / d_x^{\,2} & -\lambda / d_x & -1 & -\lambda d_y / d_x^{\,2} & \lambda / d_x \end{vmatrix} \qquad VRV^T = \begin{vmatrix} \sigma_{v_d}^{\,2} & 0 \\ 0 & \sigma_{v_\theta}^{\,2} \end{vmatrix}$$

Kalman Filter: Properties

- In the limit, features become highly correlated
  - Because observing one feature gives information about other features
- Kalman filter computes the *posterior pose*, but **not** the posterior *trajectory*.
  - If you want to know the path that the robot traveled, you have to make an extra “backwards” pass.
  - Or you can maintain the entire robot trajectory as state.

Kalman Filter: a movie

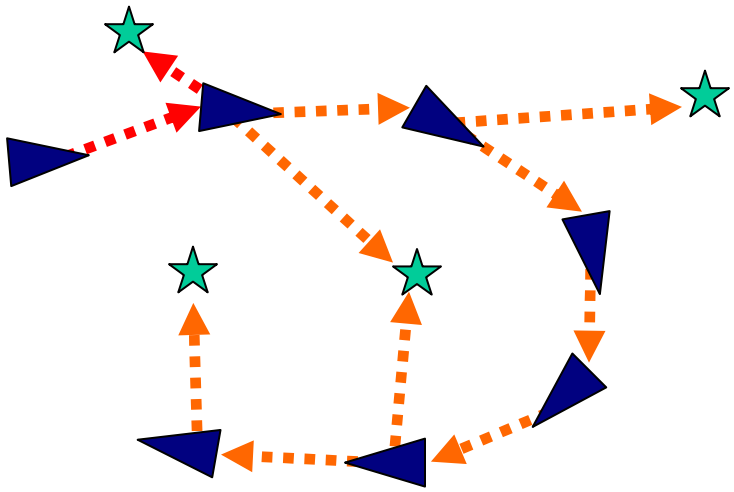


Kalman Filter: Shortcomings

- With N features, update time is still large: $O(N^2)$!
- For Maslab, N is small. Who cares?
- In the “real world”, N can be $>>10^6$.
- Linearization Error
- Current research: lower-cost mapping methods

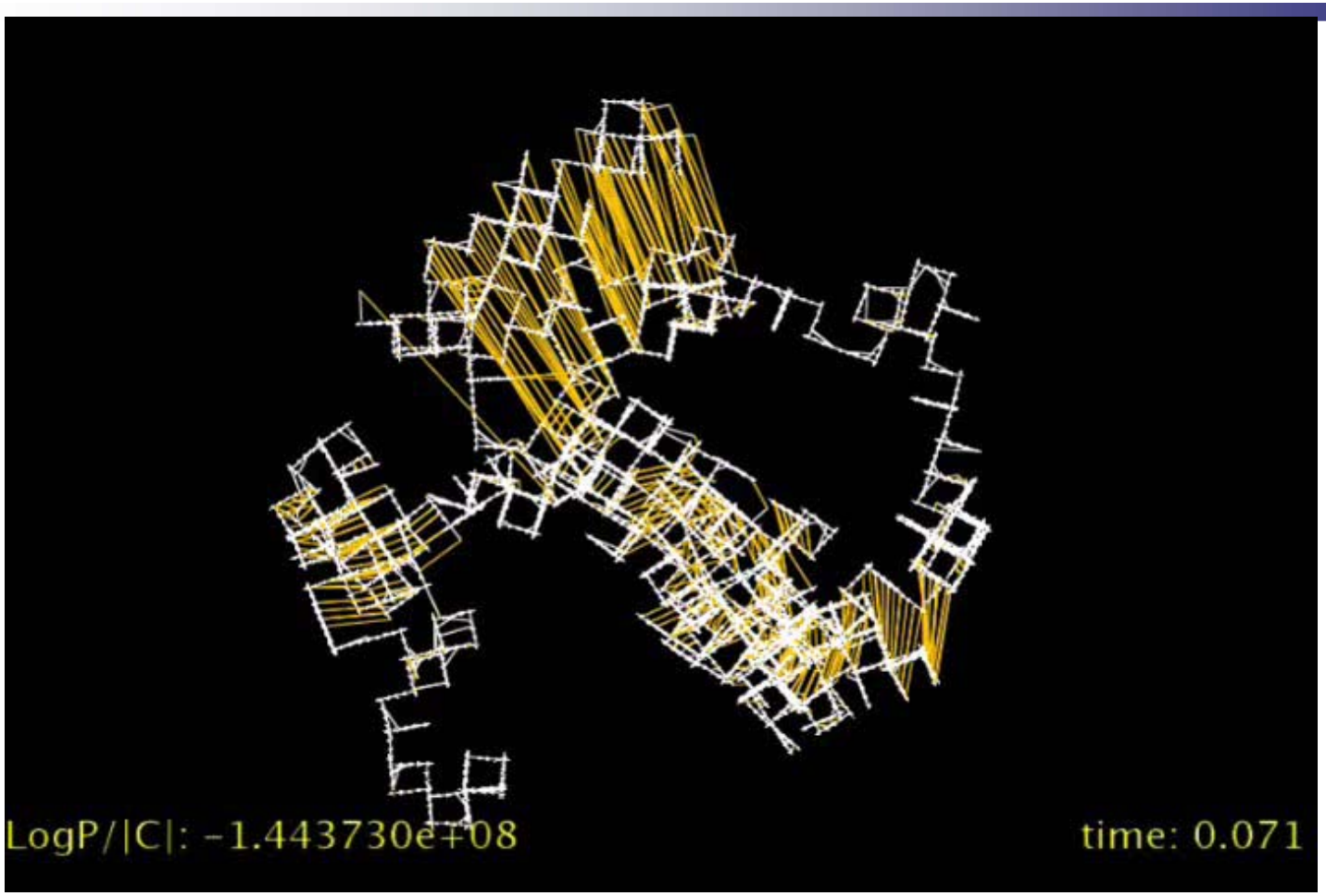
Nonlinear optimization: Relaxation

- Consider each pose/feature:
  - Fix all others features/poses
  - Solve for the position of the unknown pose
- Repeat many times
  - Will converge to minimum
  - Works well on small maps



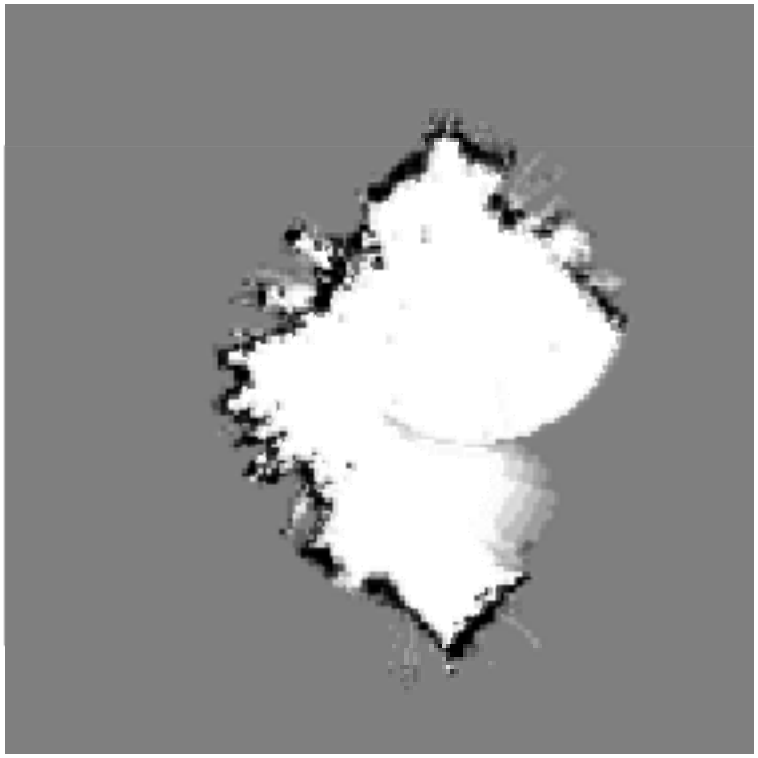
Pose/Feature Graph

Nonlinear Map Optimization



Occupancy Grids

- Divide the world into a grid
  - Each grid records whether there's something there or not
    - Usually as a probability
  - Use current robot position estimate to fill in squares according to sensor observations



Occupancy Grids

- Easy to generate, hard to maintain accuracy
  - Basically impossible to “undo” mistakes
- Convenient for high-quality path planning
- Relatively easy to tell how well you’re doing
  - Do your sensor observations agree with your map?

FastSLAM (Gridmap variant)

- Suppose you maintain a whole bunch of occupancy maps
  - Each assuming a slightly different robot trajectory
- When a map becomes inconsistent, throw it away.
- If you have enough occupancy maps, you'll get a good map at the end.

Gridmap, a la MASLab

- Number of maps you need increases *exponentially* with distance travelled. (Rate constant related to odometry error)
- Build grid maps until odometry error becomes too large, then start a new map.
- Try to find old maps which contain data about your current position
  - Relocalization is usually hard, but you have unambiguous features to help.

Occupancy Grid: Path planning

- Use A* search
  - Finds optimal path (subject to grid resolution)
  - Large search space, but optimum answer is easy to find

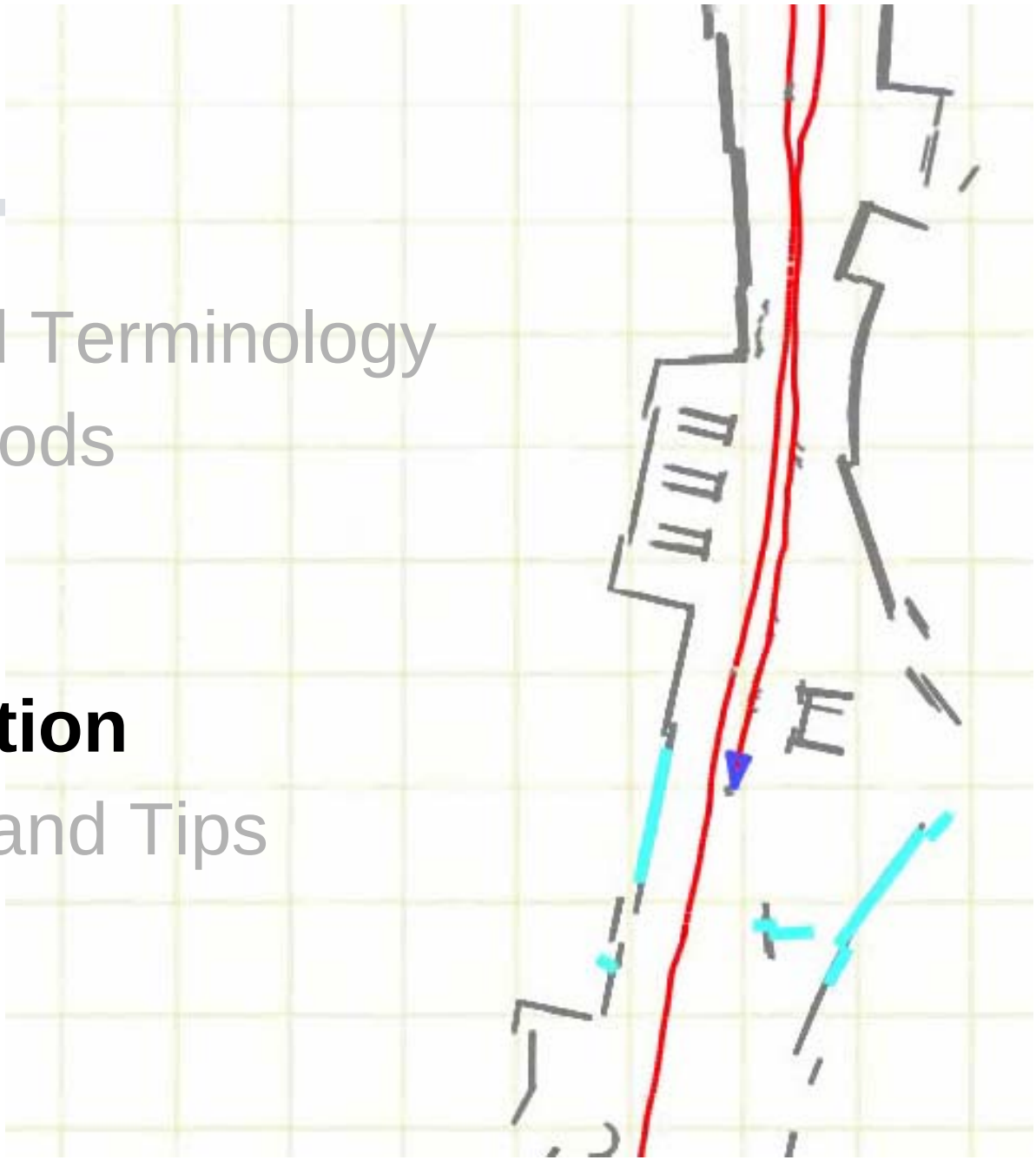
- search(start, end)
  - Initialize **paths** = set of all paths leading out of cell “start”
  - Loop:
    - let **p** be the best path in **paths**
      - Metric = distance of the path + straight-line distance from last cell in path to goal
    - if **p** reaches **end**, return **p**
    - Extend path **p** in all possible directions, adding those paths to **paths**

Occupancy Grid: Path planning

- How do we do path planning with EKFs?
- Easiest way is to rasterize an occupancy grid on demand
  - Either all walls/obstacles must be features themselves, *or*
  - Remember a local occupancy grid of where walls were at each pose.

Attack Plan

- Motivation and Terminology
- Mapping Methods
  - Topological
  - Metrical
- **Data Association**
- Sensor Ideas and Tips



Data Association

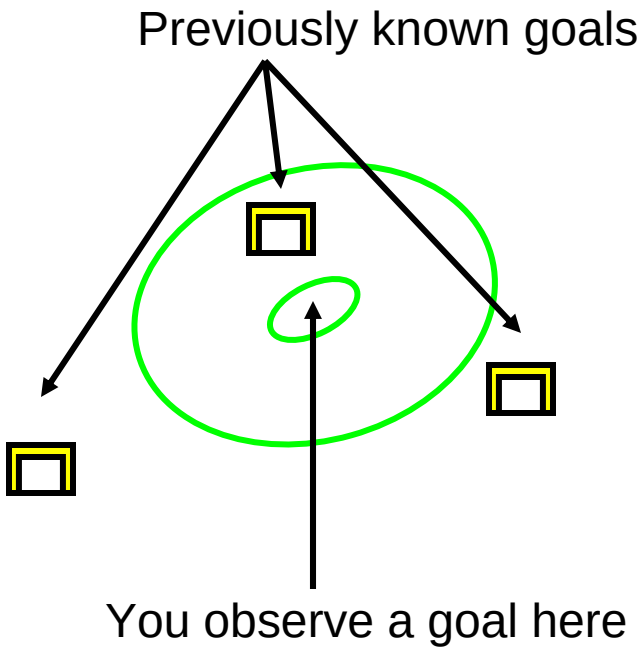
- The problem of recognizing that an object you see now is the same one you saw before
  - Hard for simple features (points, lines)
  - Easy for “high-fidelity” features (barcodes, bunker hill monuments)
- With perfect data association, most mapping problems become “easy”

Data Association

- If we can't tell when we're reobserving a feature, we don't learn anything!
  - We need to observe the same feature *twice* to generate a constraint

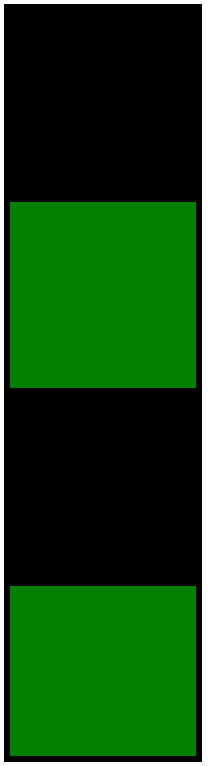
Data Association: Nearest Neighbor

- Nearest Neighbor
  - Simplest data association “algorithm”
  - Only tricky part is determining when you’re seeing a brand-new feature.



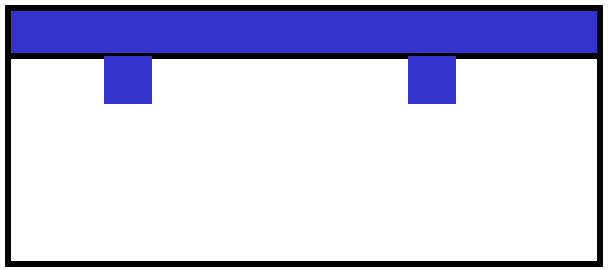
Data Association: Bar Codes

- Trivial!
- The Bar Codes have unique IDs; read the ID.



Data Association: Tick Marks

- The blue tick marks can be used as features too.
  - Probably hard to tell that a particular tick mark is the one you saw 4 minutes ago…
  - You only need to reobserve the same feature *twice* to benefit!
  - If you can track them over short intervals, you can use them to improve your dead-reckoning.
    - Use nearest-neighbor. Your frame-to-frame uncertainty should only be a few pixels.

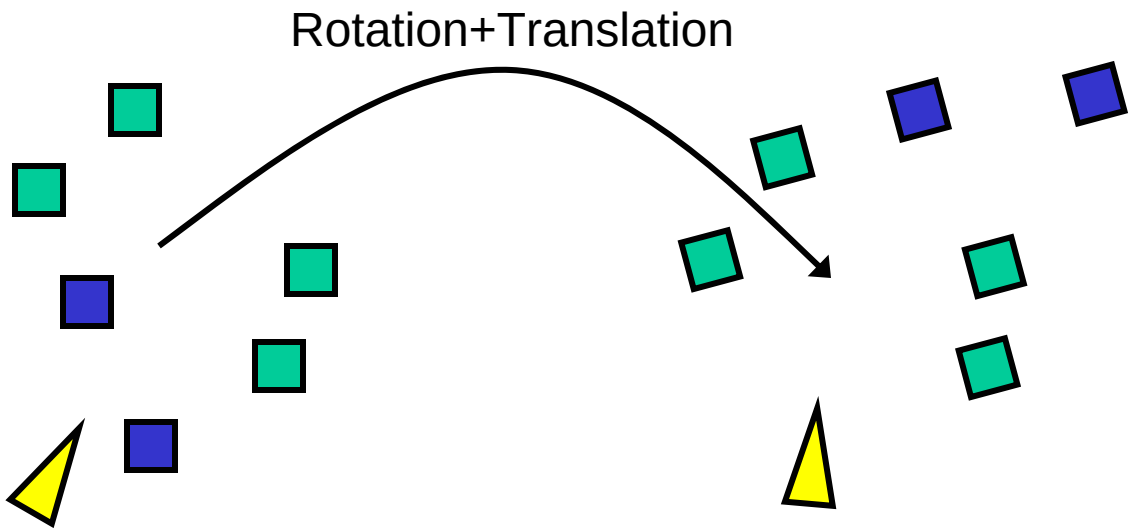


Data Association: Tick Marks

- Ideal situation:
  - Lots of tick marks, randomly arranged
  - Good position estimates on all tick marks
- Then we search for a *rigid-body-transformation* that best aligns the points.

Data Association: Tick Marks

- Find a rotation that aligns the most tick marks…
  - Gives you data association for matched ticks
  - Gives you rigid body transform for the robot!

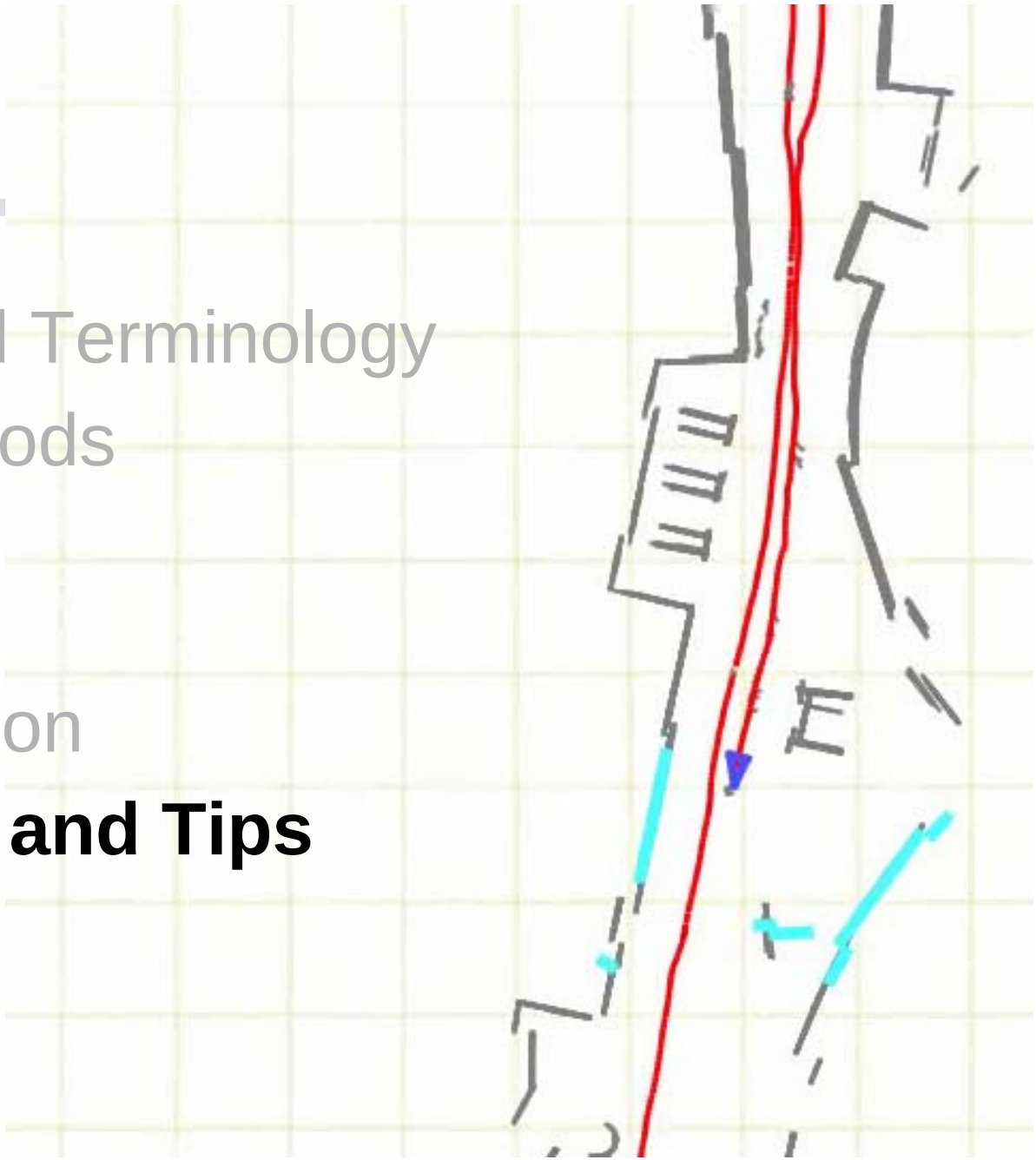


Finding a rigid-body transformation

- Method 1 (silly)
  - Search over all possible rigid-body transformations until you find one that works
    - Compare transformations using some “goodness” metric.
- Method 2 (smarter)
  - Pick two tick marks in both scene A and scene B
  - Compute the implied rigid body transformation, compute some “goodness” metric.
  - Repeat.
    - If there are N tick marks, M of which are in both scenes, how many trials do you need? Minimum: $(M/N)^2$
  - This method is called “RANSAC”, RANdom SAmple Consenus

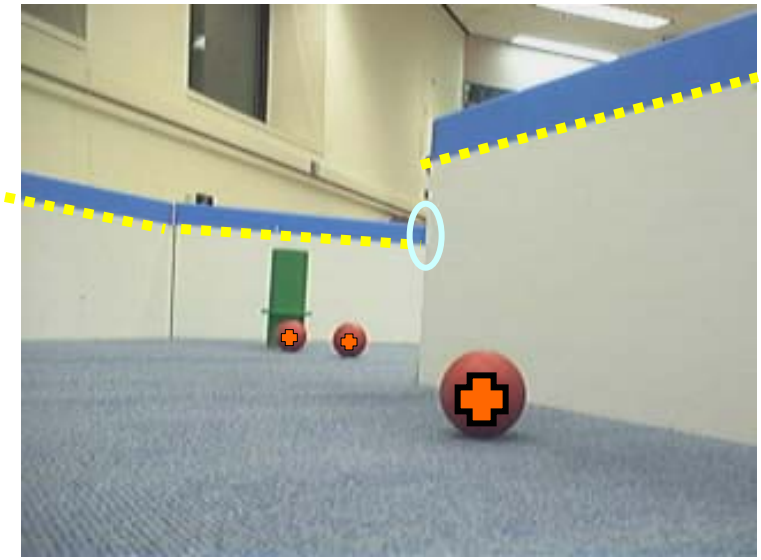
Attack Plan

- Motivation and Terminology
- Mapping Methods
  - Topological
  - Metrical
- Data Association
- **Sensor Ideas and Tips**



Use the Camera, Luke

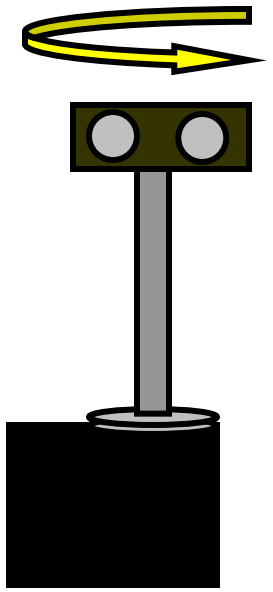
- Other useful features can be extracted!
  - Lines from white/blue boundaries
  - Balls (great point features! Just delete them after you've moved them.)
  - "Accidental features"
- You can estimate bearing *and* distance.
  - Camera mounting angle has effect on distance precision
- Triangulation
  - Make bearing measurement
  - Move robot a bit (keeping odom error small)
  - Make another bearing measurement



More features = better navigation performance

Range finders

- Range finders are most direct way of locating walls/obstacles.
- Build a “LADAR” by putting a range finder on a servo
  - High quality data! Great for mapping!
  - Terribly slow.
    - At least a second per scan.
      - With range of > 1 meter, you don’t have to scan very often.
    - Two range-finders = twice as fast
      - Or alternatively, 360^{o} coverage
    - Hack servo to read analog pot directly
      - Then slew the servo in one command at maximum speed instead of stepping.
    - Add gearbox to get 360^{o} coverage with only one range finder.



Debugging map-building algorithms

- You can’t debug what you can’t see.
- Produce a visualization of the map!
  - Metrical map: easy to draw
  - Topological map: draw the graph (using graphviz/dot?)
  - Display the graph via BotClient
- Write movement/sensor observations to a file to test mapping independently (and off-line)

Today’s Lab Activities

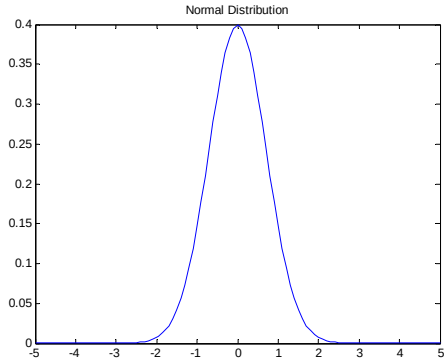
Old Slides

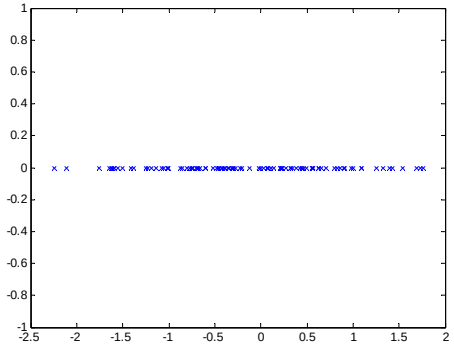
Bayesian Estimation

- Represent unknowns with probability densities
  - Often, we assume the densities are Gaussian

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/\sigma^2}$$

  - Or we represent arbitrary densities with particles
    - We won't cover this today





Metrical Map example

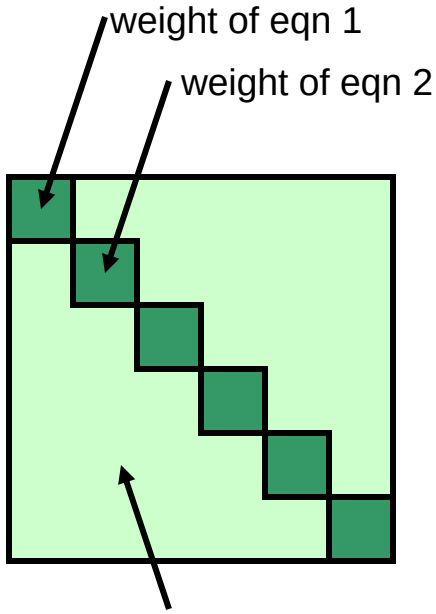
- Some constraints are better than others.
- Incorporate constraint "weights"
  - Weights are closely related to covariance:

$$W = \sum^{-1}$$

  - Covariance of poses is:

$$A^T W A$$

$W =$



In principle, equations might not represent independent constraints. But usually they are, so these terms are zero.

$$x = (A^T W A)^{-1} A^T W b$$

* Of course, "covariance" only makes good sense if we make a Gaussian assumption