Navigation: Mapping

RSS Lecture 13

Wednesday 19 March 2014

Prof. Teller

Text: Siegwart and Nourbakhsh Ch. 5, 6

Dudek and Jenkin Ch. 8

Navigation Overview

- Where am I?
  - Localization (Lecture 12)
  - Assumes perfect map, imperfect sensing
- How can I get there from here?
  - Planning (Lectures 8-10)
  - Assumes perfect map, sensing, and actuation
- What did I observe during my excursion?
  - Mapping (Today)
  - Assumes perfect localization, noisy sensing
- Can I build a map *and* localize in it, on-line?
  - Yes; using SLAM
  - Assumes no prior knowledge of the world

Lecture Overview

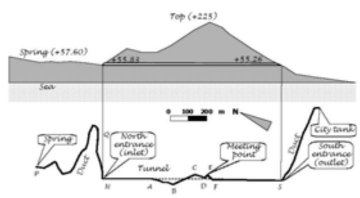
- What are maps?
- Map representations
- Fusing observations
- Uncertainty: noise and outliers
- Feature and free-space complexity

What are maps?

- Collection of elements or features at some scale of interest, and a representation of the geometric and/or topological relationships among them
- Also *semantic information (metadata)*
  - Segmentation, place/object naming, function, etc.
- We will focus on *geometry* and *topology*
  - But *semantics* are also critical in real-world applications!

History

- Early surveying, mapping methods:
  - Egyptians (c. 1400 B.C.): Nile floods, taxation
    - Plumb bobs, sighting instruments, area measurement
  - Greeks (c. 550 B.C.): Trade, warfare, engineering
    - Coastal, nautical maps for marine navigation
    - Dug Eupalinos tunnel from both ends, 1036m long!
  - Europeans (16th century onward): foundational computational methods
    - Gauss, method of least squares (1809)



Demetris Koutsoyiannis

ams.org

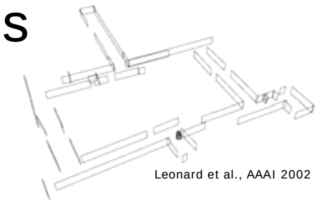
Triangulation of Hanover, 1820-1850

Why maps? From where?

- Essential for a wide variety of human, robotic activities (localization, planning)
- Maps are highly labor-intensive to create:
  - Exploration (global coverage)
  - Measurement (local coverage)
  - Validity (correctness, error bounds)
  - Currency (freshness)
  - As-planned vs. as-built building models
  - On top of all that: metadata/semantics …
- Map creation is an ideal robotics task!
  - Achieving a robust, sustained, large-area, fully autonomous mapping capability has been an “open” (i.e., unsolved) problem for decades

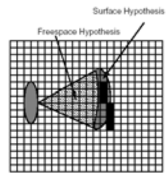
Some robot map types

- Continuous / “vector” format
  - Points, linear or curved segments, surface patches

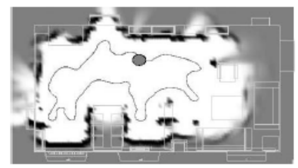


Leonard et al., AAAI 2002

- Discrete / “raster” format
  - Occupancy grids

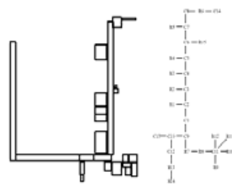


Konolige

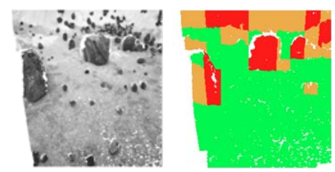


Chatila, SSS 2004

- Metrical / Topological
- Global / Local



Metrical / Topological



Chatila, SSS 2004

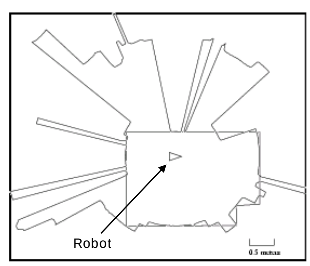
Local, Metrical, Qualitative

- Hybrid

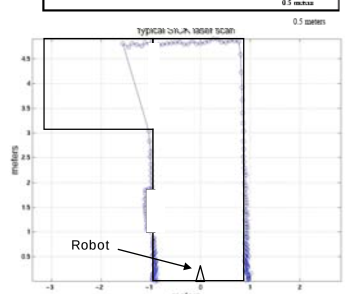
Commonly used range sensors

Polaroid sonar ring
12 range returns, one per 30 degrees, at ~4 Hz

(+ servoed rotation)



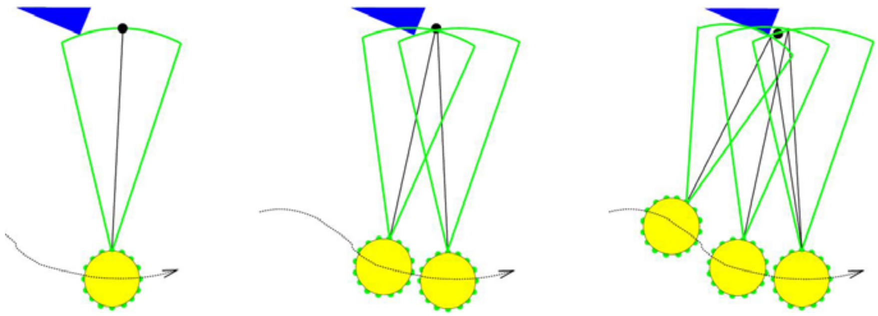
SICK laser scanner
180 range returns, one per degree, at 5-75 Hz



Other possibilities: Stereo/monocular vision; Robot body (e.g. bump/stall sensing)

Fusing multiple returns

- Crucial assumption: pose estimation (e.g., odometry, dead reckoning) is accurate over short times and distances

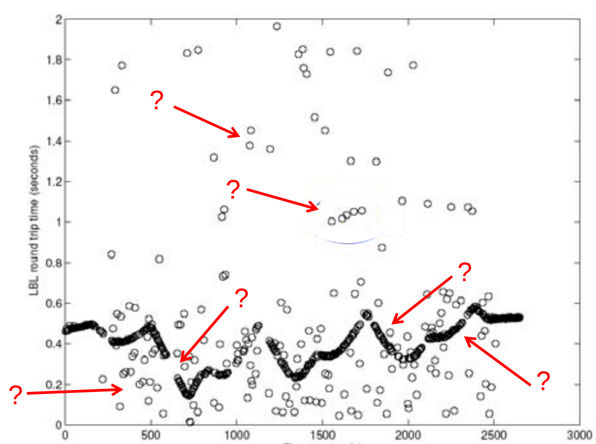


Wijk 2001

- Can then localize features using conventional triangulation (sonar beam width complicates things)

Digression: sensing challenges

- Time series of round-trip-time to one acoustic beacon for an underwater autonomous vehicle

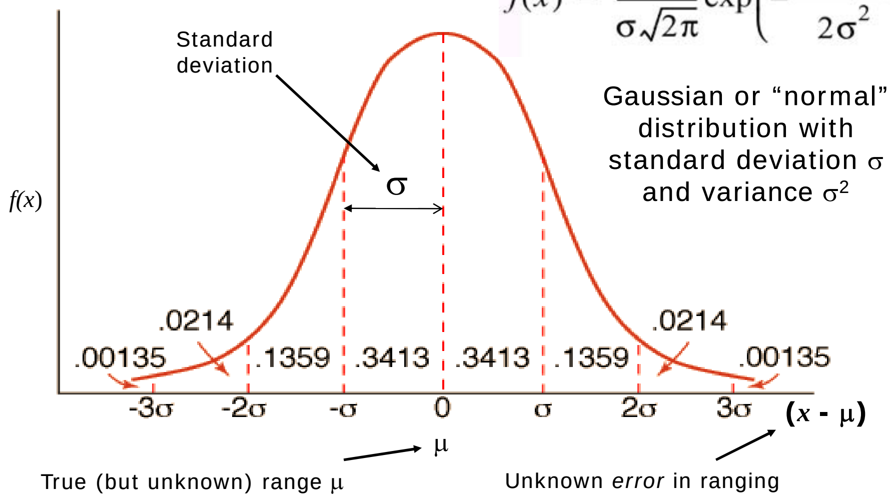


(Olson, Leonard, Teller, Robust Range-Only Beacon Localization, IEEE AUV, June 2004)

Gaussian noise model

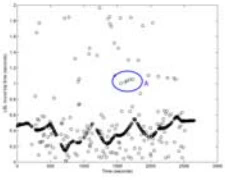
- Measurement returns a corrupted value

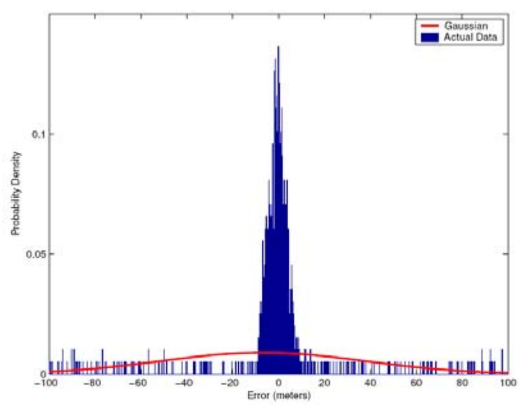
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$



Outliers

- Many measurements are *outliers*; their frequency is not well-modeled by a Gaussian distribution

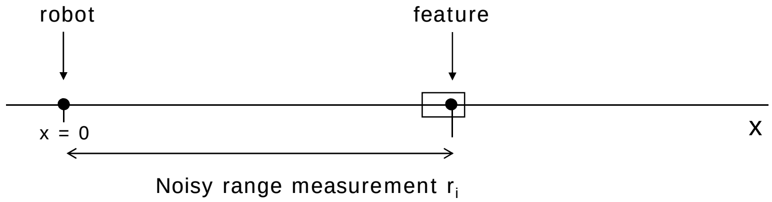




... what to do?

Filtering

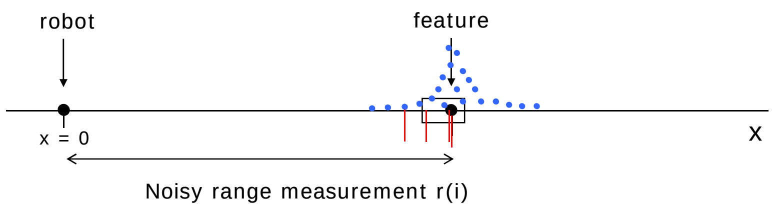
- Consider one-dimensional localization:
  - Robot measures range r(i) at ith time step
  - Ranges *corrupted* by Gaussian noise, outliers



- *Filter* measurements; combine over time
  - Incorporate each measurement as it arrives
  - *Recursive* (*on-line*) filtering (contrast *batch*)

Filtering with no outliers

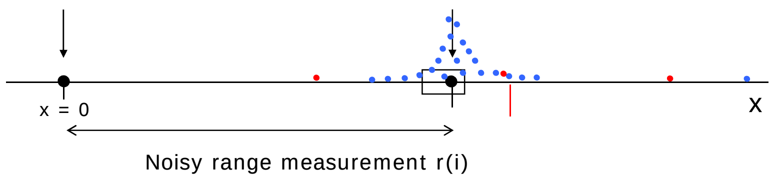
- Suppose neither robot nor feature moves
  - What should our filtering strategy be?
  - Call x(t) our *estimate* of x after t time steps



- Compute the mean (arithmetic average)
  - x(i) = (r(1) + r(2) + ... + r(i)) / i (batch)
  - x(i) = [x(i-1) * (i-1) / i] + [r(i) / i] (on-line or "recursive")
  - ... if no outliers, no change over time, filter is optimal
- Computational complexity of each update?

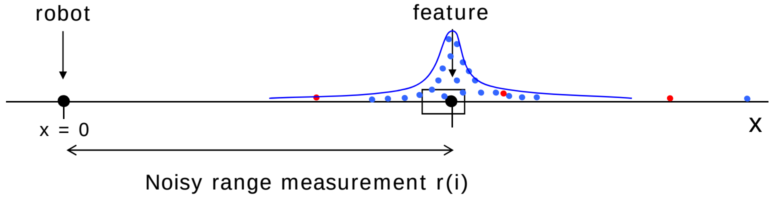
Handling outliers

- Suppose a fraction of r(i) are wildly wrong
  - Classify r(i) as *inliers* ● or *outliers* ●
  - How to do this?



Modeling measurement noise

- Estimate sample *variance* as well as mean



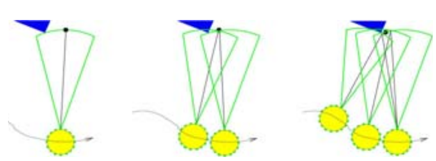
- Reject unlikely samples (e.g., $p < 1\%$)
  - Filter only inliers, by averaging as before
- ... But where does variance come from?
  - Determine it *a priori* (e.g. from bench tests)
  - Or, estimate it *on-line*, in addition to mean
    - Chicken-and-egg problem (could be unlucky)
    - If "outliers" become frequent, what can you do?

Estimating variance

- Define $\sigma^2(i)$ as variance after i steps
- Batch computation:
  - As before, $x(i)$ is the mean after i steps
  - Then variance $\sigma^2(i)$ is $[\sum(r(i)-x(i))^2] / i$
- Recursive (on-line) computation:
  - Estimate $x(i)$ recursively as before
  - Define $\sigma^2(1) = 0$; then for $i > 1$:

$$\sigma^2(i) = (i-1)/i * \sigma^2(i-1) + 1/(i-1) * (r(i) - x(i))^2$$

Local vs. global data fusion



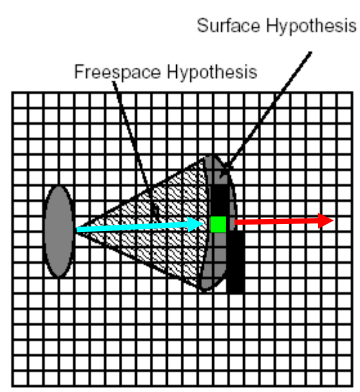
- Crucial assumption: that robot can solve strong localization (global pose estimation) throughout
- This is a very difficult problem without a map! (It's difficult even *with* a map or a partial map.)
- SLAM: Simultaneous Localization and Mapping
- For now, we assume localization; o/wise, need SLAM

Representation considerations

- We want our robot to be able to plan and execute high-level motions among obstacles
- What do we want from our map?
  - Consistent global, or locally metrical, coordinate system
  - Identification and localization of substantial *features,* e.g., obstacles that may hinder or damage the robot
  - All of this should be well-defined and computationally accessible (data model, data structure, API)
  - Scalability (reasonable search, access times as exploration continues, and map gets really large)
- ... Is that all we need/want from a map?

Alternative 1: Discretize

- *Occupancy grid* of *cells*
  - Regular subdivision of region
  - Models free & occupied space
- Cells accumulate *evidence* of presence of obstacle surface
- Grid is updated on-line with recent measurements
- Range return from obstacle implies three grid intervals:
  - From robot to obstacle (FS)
  - At (quantized) obstacle depth
  - Beyond obstacle (from robot's point of view)



Konolige

Many occupancy grid methods

- Example: sonar data, varying update rules
  - White: freespace; black: obstacle; grey: unknown

Wijk 2001

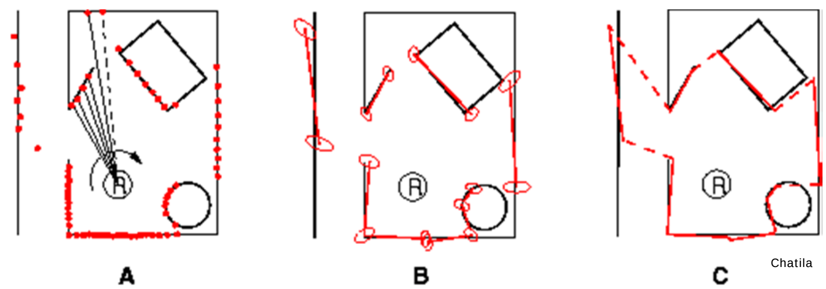
Bor: Histogramic (Borenstein 1991); accumulates hits
Fuz: Fuzzy (Zadeh 1973; Ribo and Pinz 1999); with weights
TBF: Triangulation-Based Fusion (Wijk 2000); local triangulation
Bay: Bayesian (Elfes 1988); probabilistic occupancy/emptiness
DS: Dempster-Shafer (Shafer 1976; Pagac 1996); with "ignorance"

Pitfalls of occupancy grids

- Quantization error
  - Cells too large: not faithful to environment or robot task
  - Cells too small: too numerous (expensive) to process efficiently
  - Task-dependent: grid size can be simultaneously too small *and* too large!
- Blurring
  - Caused by pose estimation error, sensor uncertainty, grid quantization

Alternative 2: Line Features

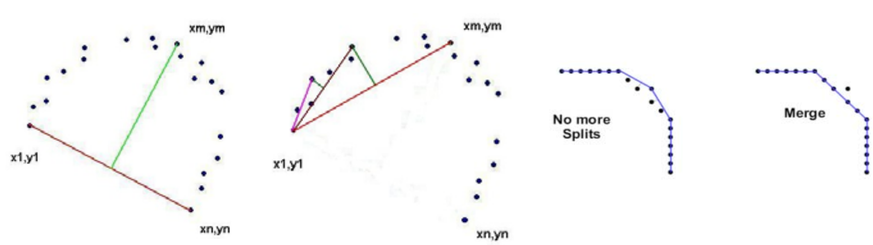
- Piecewise linear approximation of sequence of point features (i.e., ranges)



- How are individual ranges, point features grouped into useable line segments?
- How to counteract noise inherent in data?

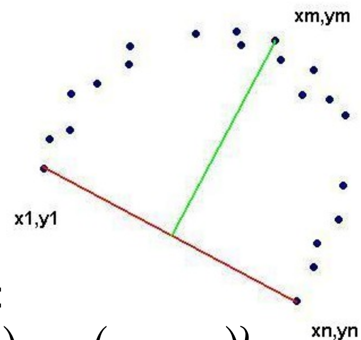
Split, Merge, Fit algorithm

- Used for *ordered sets* of laser or sonar returns
- Takes two thresholds: split distance, merge angle
- Split phase:
  - Recursively split until (max) distance criterion is met
- Merge phase:
  - Merge adjacent segments until (min) angle criterion is met
- Fit phase (perhaps with explicit outlier handling):
  - Fit line segments to resulting (noisy) point sequences



Split phase

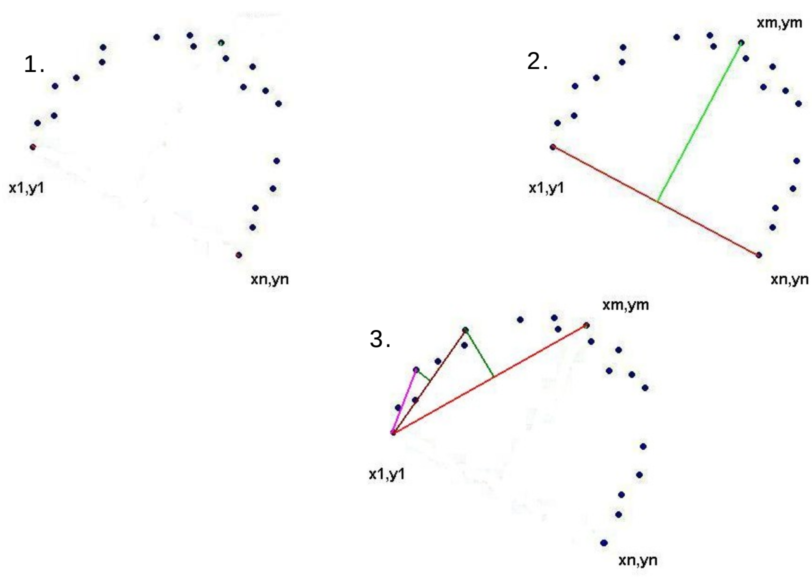
- Given points $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- Find (x_m, y_m): point with maximum distance to line $L = \{(x_1, y_1), (x_n, y_n)\}$



- Split into two subsets:

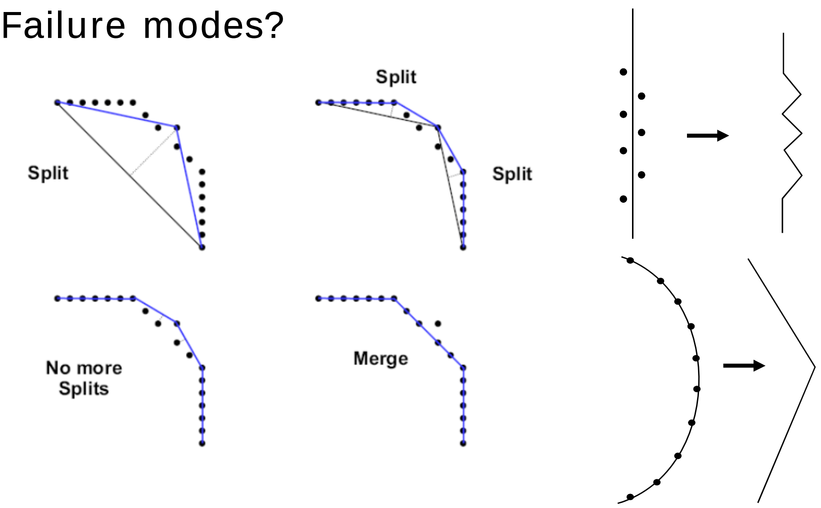
$$P' = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$$
$$P'' = \{(x_m, y_m), (x_{m+1}, y_{m+1}), \ldots, (x_n, y_n)\}$$

Splitting is recursive



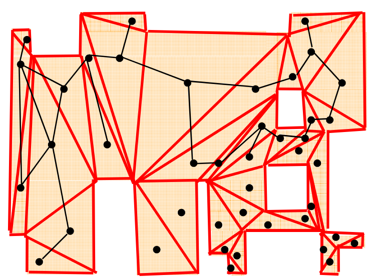
Segment merging phase

- Merge adjacent segments if nearly collinear
- Failure modes?



Storing extracted features

- Store as linear list
  - Advantage: very simple. Drawbacks: ?
- Or, store in *proximity data structure*
  - E.g., constrained Delaunay triangulation



- CDT has many nice properties:
  - Linear size; logarithmic search; temporal coherence; maximum minimum angle; dual to Voronoi diagram; etc.

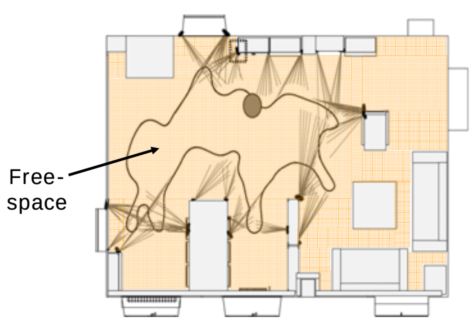
Alternative 3: Free-space Map

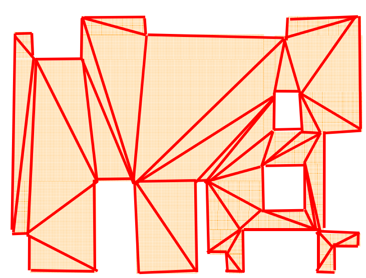
- Robot spends most time well away from obstacles

- Call this area “free-space,” *i.e., the region through which the robot can expect to be free to move*
- The *complement* of the *union* of all *obstacles*

Free-space complexity

- It’s empty, but that doesn’t mean its representation is compact! What’s the *descriptive complexity* of FS?





- Free-space is *more complex* than obstacle union *n*
  - 2D simple polygon (no holes):
  - 2D segments:
  - 3D polyhedron:

Mapping summary

- Maps are critical to many tasks
- Assumed localization for now
- Saw several map representations, data fusion algorithms
- Considered scaling requirements

Coming Up In RSS

- Today in lab:
  - Final Lab briefings
  - Teamwork on CDO's (due **Friday 1pm**)
- Friday
  - No forum
- Next week
  - Spring Break! Get some R&R.