Single-robot Topological Mapping and Map Merging for Sensing-impaired Robots

Kris Beevers
Rensselaer Polytechnic Institute
Algorithmic Robotics Laboratory
`beevek@cs.rpi.edu`

February 13, 2003

Motivation

- Team of cheap, “disposable” robots
  - only a few short-range sensors (e.g. side, front, 45^o)
  - error in movement, odometry measurements
- Objective: create a map that is useful for navigation
- Applications: search & rescue, reconnaissance, etc.
- Video
- This talk:
  - Single-robot topological mapping algorithm
  - Topological map merging algorithm (briefly)

Single-robot mapping

- Problem: release a single robot somewhere in an enclosed, static environment
- For now, assume the environment is polygonal in nature (possibly with “holes”/“islands”)
- Approach: create a topological map
  - simple graph representation: good for storage, communication
  - captures the connectivity of the environment
  - essentially, encodes only information that is necessary for navigation

Mapping strategy

- Three-phase mapping algorithm:

  1. Create a “basic map” by following walls
  2. Add “refinements” to the basic map to improve its usefulness in navigation
  3. Use the map for navigation

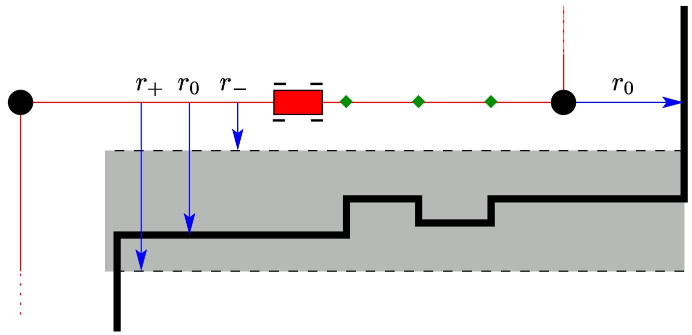
- Hardest problem: “closing the loop” when creating the basic map

- Another hard problem: adding refinements that require further exploration

Features

- Because of sensing limitations, environmental features that we use to create the map must be easy to detect

- Use *discontinuities* in walls of environment (i.e. corners) as features



- Robot follows walls at offset r_0; discontinuities that fall outside $[r_-, r_+]$ are *well-defined* features
  - $r > r_+$: exterior corner
  - $r < r_-$: interior corner

Basic mapping

1. Release the robot

2. Robot moves forward until it encounters some wall

3. Follow the wall at r_0 until a well-defined feature is encountered: this feature is v_0, the start node

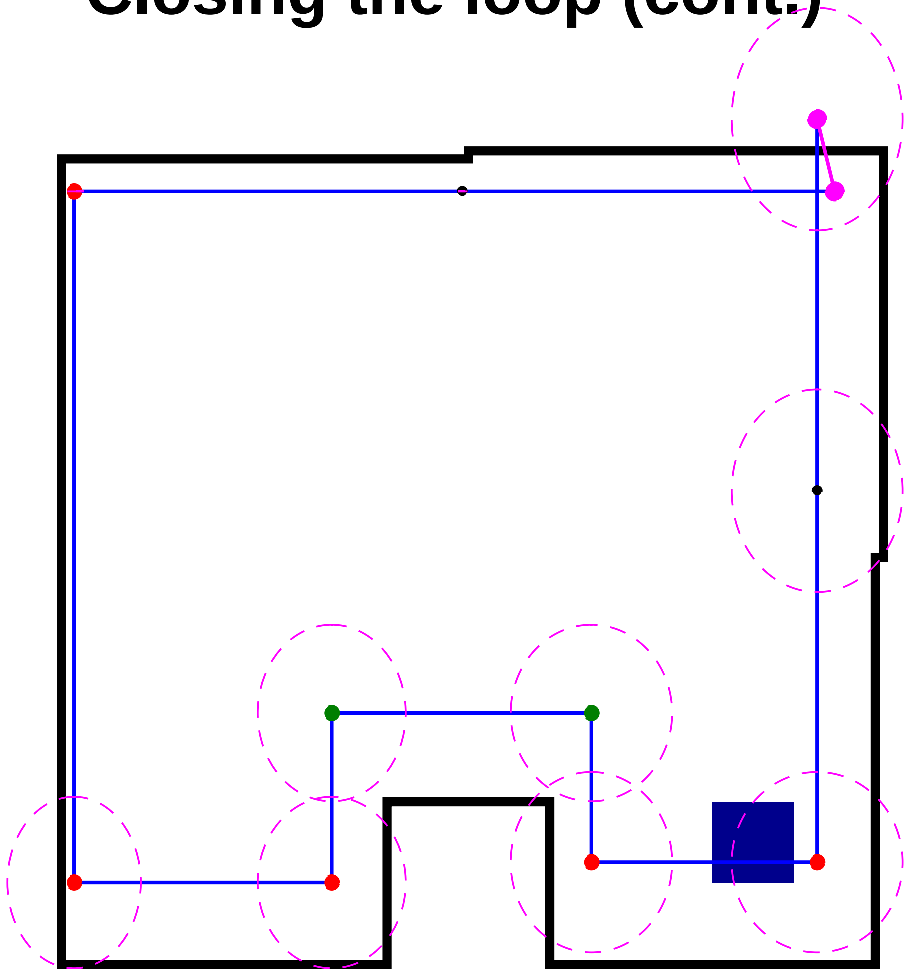
4. Turn and follow the next wall (incident to v_0) until another feature is found

5. Repeat until we return to v_0 (this must happen if the environment is enclosed — but how do we recognize it?)

Closing the loop

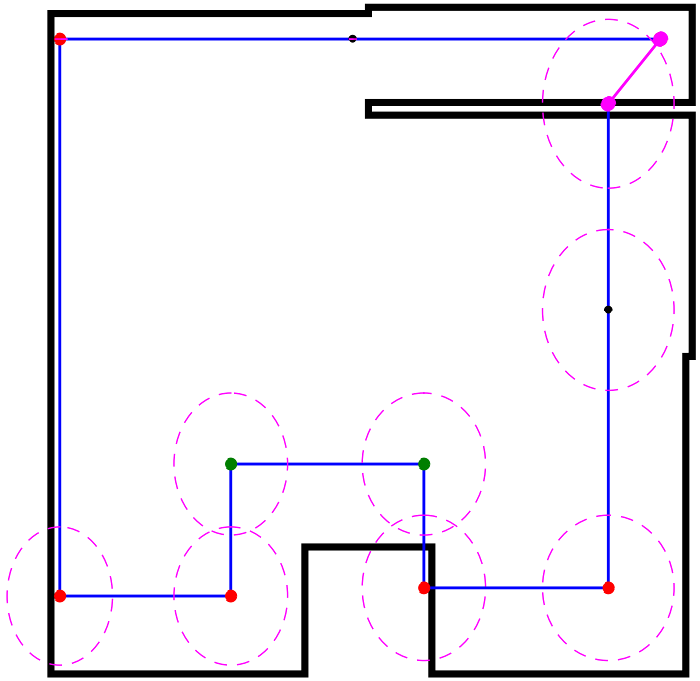
- Need to recognize when we've returned to v_0
- Take a "hypothesis"-based approach (i.e., hypothesize that we have returned to v_0, and attempt to prove or disprove the hypothesis); similar to approaches of Kuipers [5], Tomatis [7], Choset [1]
- When do we make such a hypothesis?
  - Node must be same type as v_0 (interior/exterior)
  - If we have some error model for the robot, and some confidence bound threshold, this bound must overlap v_0
  - If we have information about the "orientation" of nodes, orientations must match

Closing the loop (cont.)



Closing the loop (cont.)

- With enough error or sufficiently difficult worlds, we sometimes generate incorrect hypotheses



Closing the loop (cont.)

- Approaches to choosing the correct hypothesis:
  - Continue traversing walls, matching nodes structurally and geometrically; if subsequent pairs don't match, the hypothesis is incorrect (problem: how far is far enough?)
  - Assume the first hypothesis is correct; disprove it with later exploration and navigation if it isn't

- For now, we are using the second method; major issues:
  - in some cases, recognizing that the hypothesis is incorrect seems to *require* that we traverse the basic map past the hypothesized match (as with the first approach)
  - if we discover that the hypothesis is wrong, how do we "revert" to a valid map?

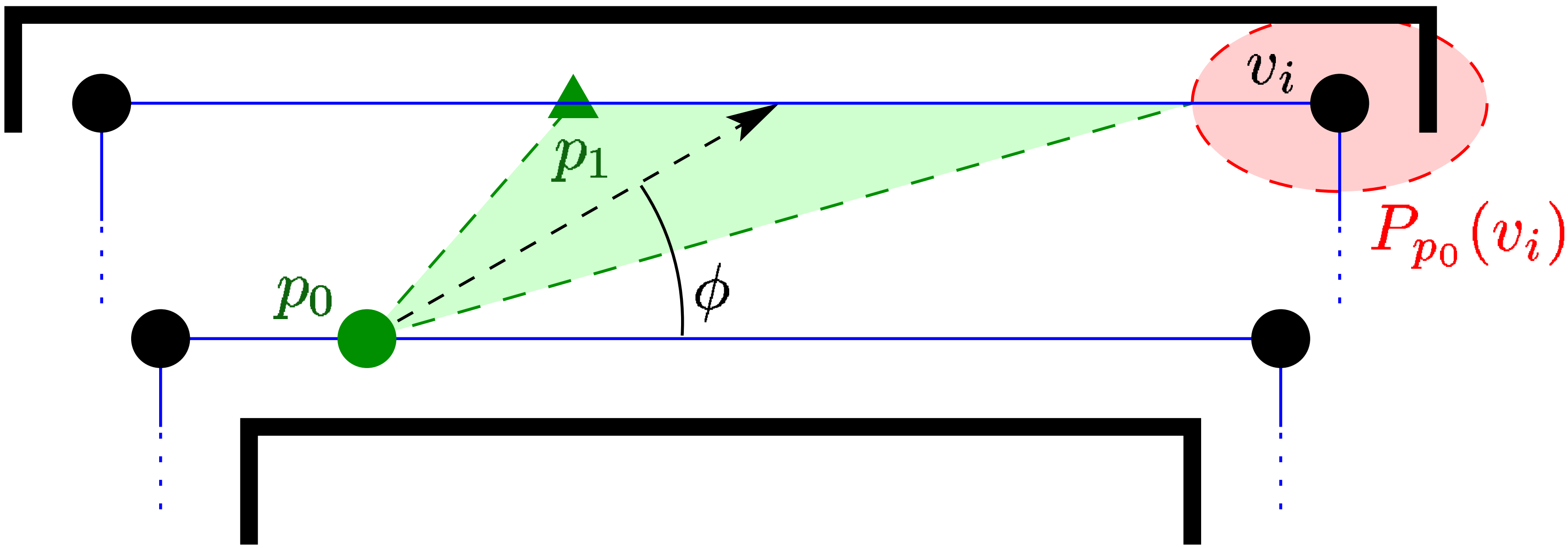
Embedding

- After we've closed the loop, we need to make our map embeddable in the plane
- Approach: treat each edge in the map as a spring
- Solve for edge lengths that allow map to be embedded
- Literature: Duckett *et al.* [2], Lu & Milios [6], Golfarelli *et al.* [3]

Refinements

- Basic map is useful — we can get anywhere we've explored — but:
  - we need to circumnavigate even to cross a hallway!
  - there may be "islands" we don't know about
- Refinement idea: try to add more paths between nodes in the map
- Biggest problem: we can't follow these paths by wall-following
  - Turn to some angle away from a wall
  - "Foray" until we encounter a new wall
- To keep from getting lost, we need to make guarantees about which wall we "land" on, despite rotational and translational uncertainties

Refinements (cont.)



Refinements (cont.)

- Passive refinements:
  - pass entirely through "known space" already swept out by the robot's sensors during basic mapping
  - require no further exploration

- "Exploration targets":
  - refinements that pass through unknown space
  - need to actually explore these refinements — they may run into an island, for example
  - only allow active refinements for which, if we run into something, we can "safely" get back to a known location in the map

Refinements (cont.)

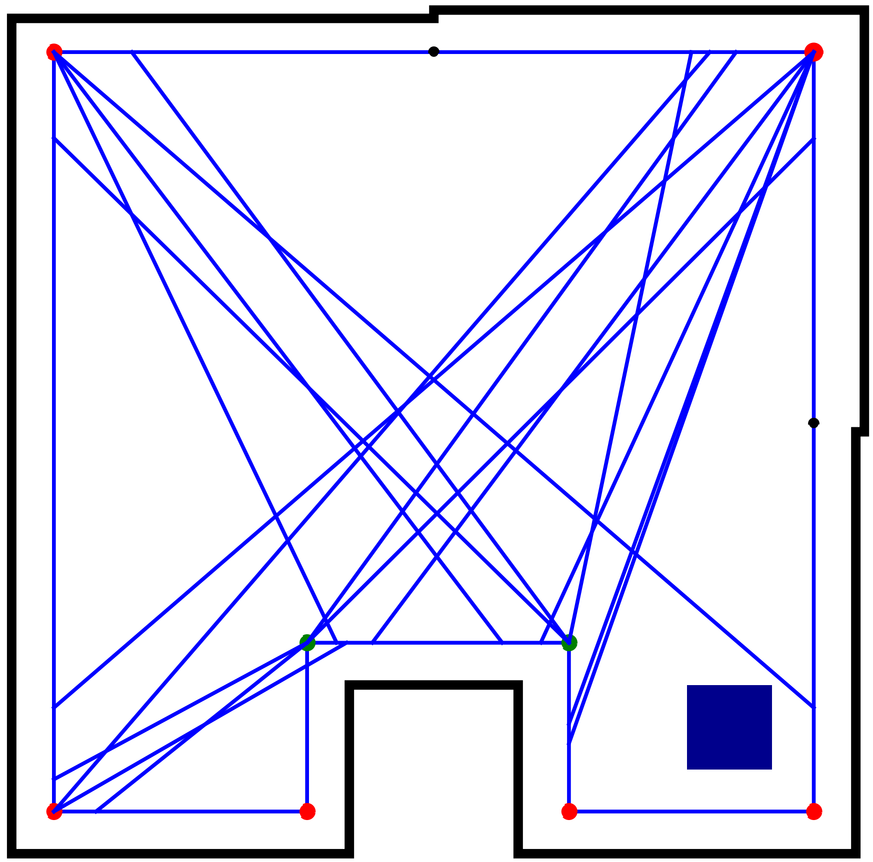
- After basic mapping and closing the loop, generate a list of all potential refinements

- Use traveling-salesman type planning to determine the sequence of exploration targets to visit

- When exploring, if we run into an island before getting to target wall:
  - we may have some “leeway” to explore (using basic mapping methods), depending on error model/accumulation
  - must not allow error to accumulate to the extent that we can’t ensure return to a known location

- If we don’t encounter our target wall at all (we go far beyond its estimated location): this *disproves* our loop-closing hypothesis!

Refinements (cont.)

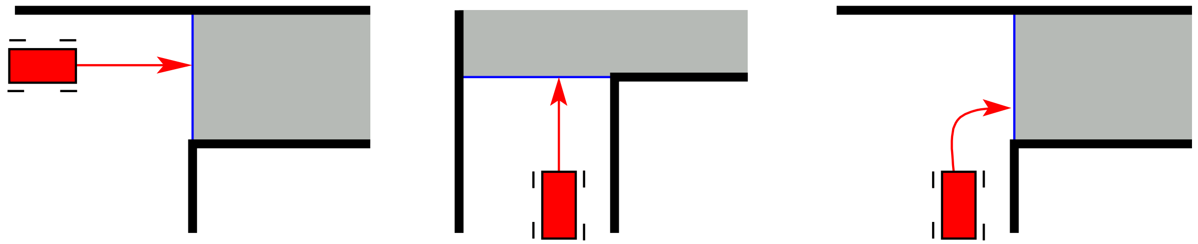


Navigation

- If our map remains consistent after refinements, we enter the navigation phase
- Use the map to navigate between known locations
- Essentially just shortest-path graph search
  - note that edges between nodes have associated behaviors (wall-follow, or turn & move-to-wall, etc.)
- In some cases, a refinement (even if we've previously explored it) might fail
  - should be ok — we can get back to a known location
  - discard the refinement if this happens

Another enhancement: “portals”

- Detected when:
  1. wall-following sensor detects an exterior corner
  2. opposite sensor detects a wall



- We can use portals to divide the world into “subregions” (i.e. treat a portal as a “virtual wall”)

- Explore each subregion using basic mapping and refinement methods, and connect the subregions using portals between them

- Main advantage: smaller loops to close

Results/Future work

- Simulation: mostly implemented, works as expected for the most part
  - easy to come up with situations where initial hypothesis is incorrect, as long as robot experiences enough error
  - need to work on detecting incorrect hypotheses, etc.

- Real robots:
  - previously implemented basic mapping on Magellan
  - main issue: wall-following methods (straight-line wall-following vs. “normal” wall-following)
  - working on implementing with new little robots

- Need to resolve hypothesis issues, fine-tune refinement exploration

Topological map merging (quickly)

- Problem: given consistent topological maps created by two robots with different reference frames, find correspondences between them and merge them into a single map
- With no metric information: pure subgraph isomorphism
- We assume some metric information is available (edge lengths), but it is noisy
- Approach:

1. "grow" match "hypotheses" using only structural information
2. estimate geometric transformations for these hypotheses
3. cluster the hypotheses into consistent groups based on their locations in transformation space

Growing matches

- Assumption: by visiting a vertex in the map, the robot knows its degree
- Start with initial pairing of "compatible" vertices (one from each map)
- Exactly-known vertex attributes (such as degree) must match exactly; inexactly-known attributes must be compared with a similarity function
- "Grow" by testing corresponding pairs of edges and neighboring vertices leaving from the initial pairing
  - if compatible, add to the match
  - if not, reject the entire match

Estimating geometric transformations

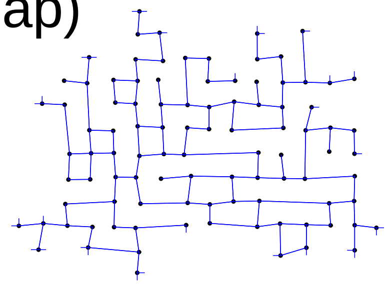
- First, we must embed the vertices of the maps in the plane (just like with single-robot mapping)
- Use least squares estimation to find transform implied by a hypothesis
- Closed-form SVD-based method (from image registration) lets us do this in one step

Clustering of hypotheses

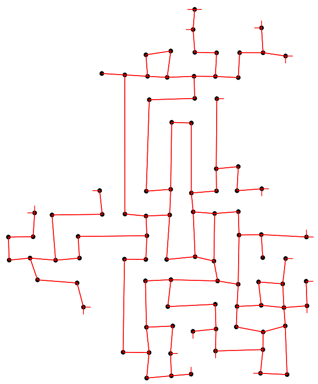
- Cluster based on closeness of transformations
- A cluster cannot have multiple correspondences for a vertex in either map (this is inconsistent)
- After clustering, order clusters by “quality”
  - number of vertex correspondences
  - total squared error under cluster transform
  - number and sizes of hypotheses in the cluster
- Always a tradeoff between size and quality (a single-node match is perfect!)

Results

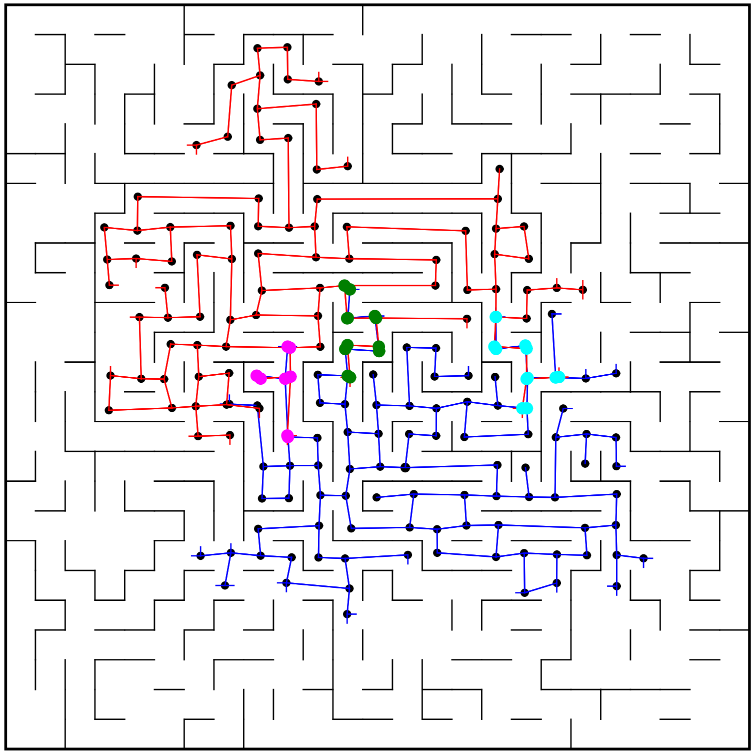
- Algorithm works well and is *fast* (even for large maps with small overlap)



Map $\mathcal{A}$



Map $\mathcal{B}$



References

[1] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization. *IEEE Trans. on Robotics & Automation*, 17(2), 2001.

[2] T. Duckett, S. Marsland and Jonathan Shapiro. Learning globally consistent maps by relaxation. In *Proc. 2000 IEEE Intl. Conf. on Robotics & Automation*, pages 3841–3846, 2000.

[3] M. Golfarelli, D. Maio and S. Rizzi. Elastic correction of dead-reckoning errors in map building. In *Proc. 1998 Intl. Conf. on Intelligent Robots and Systems*, pages 905–911, 1998.

[4] W. Huang and K. Beevers. Topological map merging. Submitted to

the 7th Intl. Symposium on Distributed Autonomous Robotic Systems, 2004.

[5] B.J. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics & Autonomous Systems*, 8:47–63, 1991.

[6] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4(4):333–349, 1997.

[7] N. Tomatis, I. Nourbakhsh, and R. Siegward. Hybrid simultaneous localization and map building: closing the loop with multi-hypothesis tracking. In *Proc. 2002 IEEE Intl. Conf. on Robotics & Automation*, 2002.