

Topological Mapping with Sensing-Limited Robots

Wes Huang & Kris Beevers
Algorithmic Robotics Laboratory
Department of Computer Science
Rensselaer Polytechnic Institute
{whuang, beevek}@cs.rpi.edu

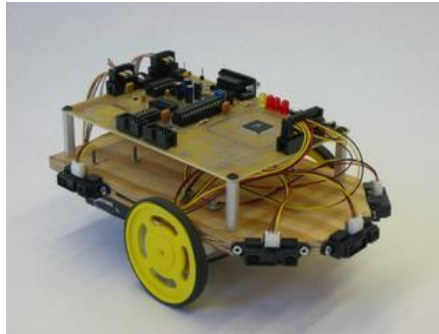
February 20, 2004

Motivation

- Use teams of small, inexpensive, “disposable” robots
- Applications:
 - Search & rescue
 - Urban reconnaissance
 - Hazard assessment
- Basic capabilities:
 - Map & explore unknown environments
 - Navigate using the resulting map

Sensing-limited robots

- To make inexpensive robots, use a limited array of sensors
- Our prototype hardware:



Uses only five infrared range sensors!

- Sensing is limited in both:
 - Range (approx 80 cm)
 - Density (five samples, not a “scan”)

Questions

- Can sensing limited robots map and explore an unknown world?
- How are mapping ability and map quality affected by:
 - limited sensing
 - sensor error
 - world complexity
- What do assumptions about the world (e.g. rectilinear) provide?

Assumptions

- Rectilinear enclosed world (for now)
No assumptions as to “smoothness” of the world.
- Errors in sensing with known probabilistic model
- Errors in odometry with known probabilistic model
(Rectilinear assumption assures no orientation error.)

Outline

- Single-robot topological mapping
 - Basic mapping approach
 - Closing the loop
 - Portals
 - Refinements
- Topological map merging

Single-robot mapping

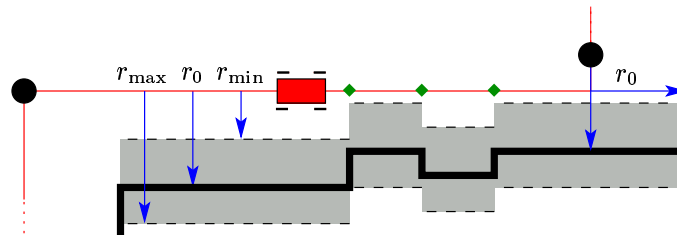
- Problem: release a single robot somewhere in an enclosed, static environment
- For now, assume the environment is polygonal in nature (possibly with “holes”/“islands”)
- Approach: create a topological map
 - simple graph representation: good for storage, communication
 - captures the connectivity of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

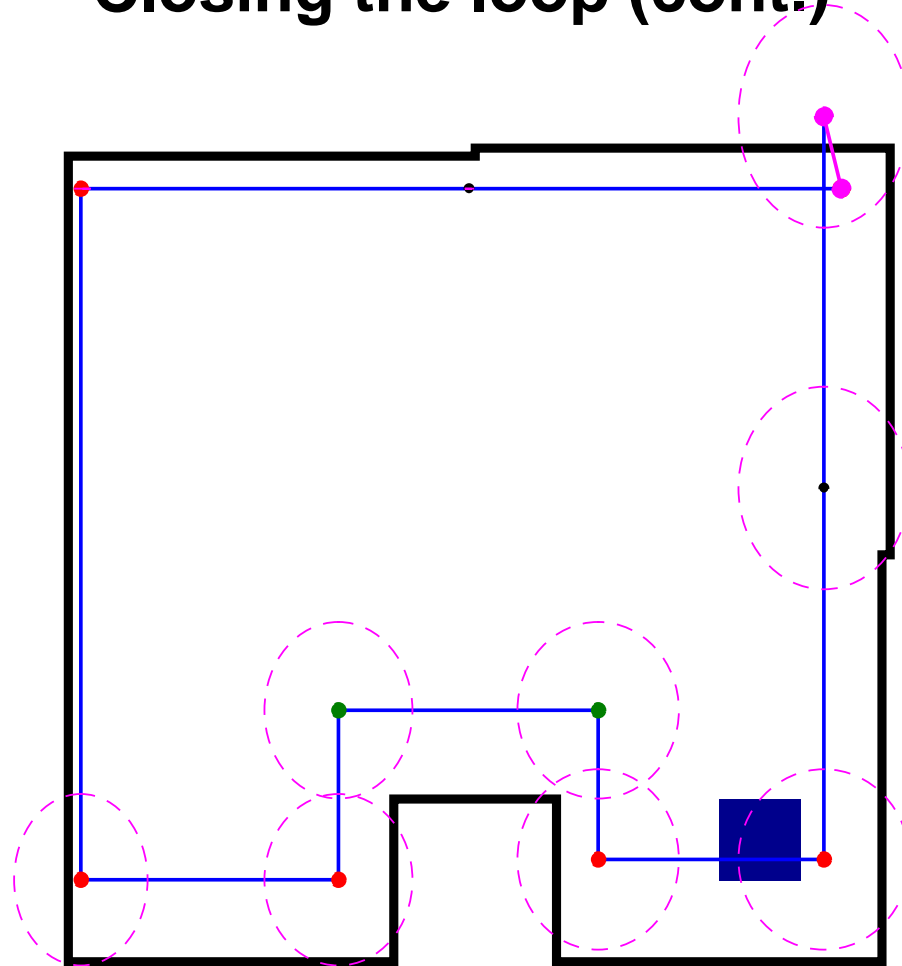
Basic mapping

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)
- 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.)



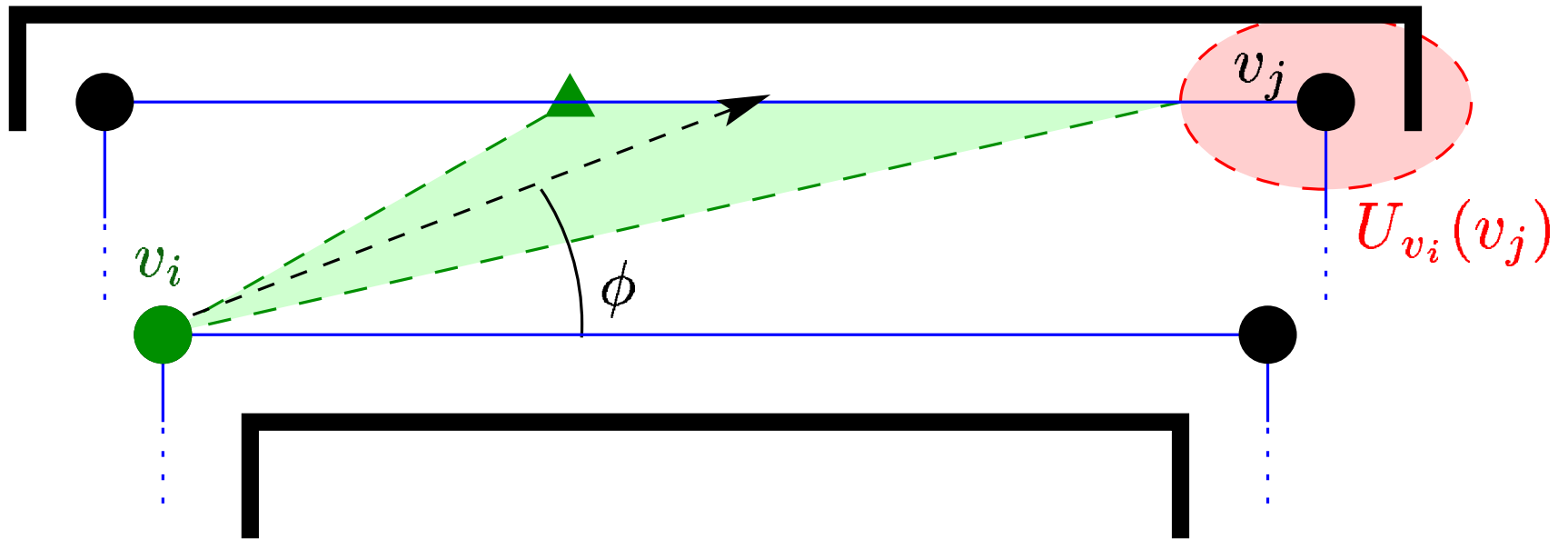
Closing the loop (cont.)

- We take an “evidential” approach to closing the loop
- Attempt to prove or disprove a hypothesis by gathering “evidence”
- Continue forward traversal of walls
 - If the hypothesis is correct, edge length measurements (and feature types) should match
 - Significantly different measurements are strong evidence against the hypothesis
 - Very close measurements are strong evidence in support of the hypothesis
 - Use the Dempster-Shafer theory of evidence to build evidence based on multiple measurements

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.)



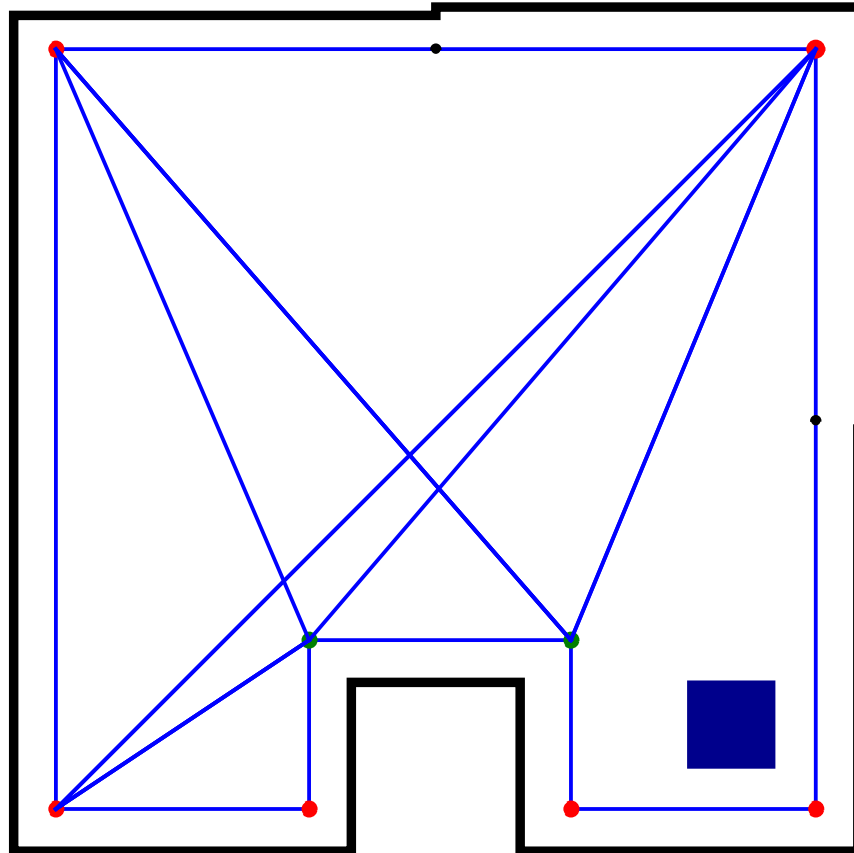
Refinements (cont.)

- 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

Refinements (cont.)

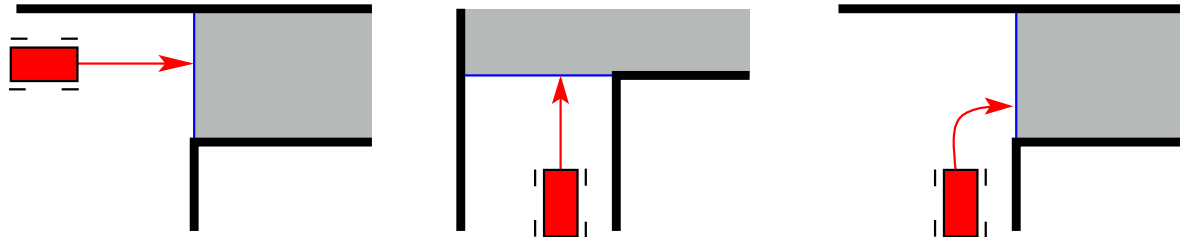


Navigation

- 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”); also indicate switch to hall-following
- Explore each subregion using basic mapping and refinement methods, and connect the subregions using portals between them
- Main advantage: smaller loops to close

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

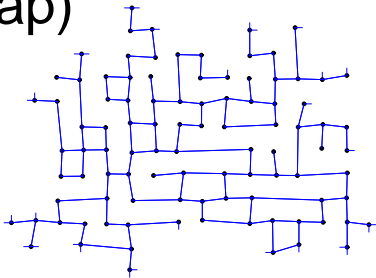
- First, we must embed the vertices of the maps in the plane (edge length measurements may not be “consistent”)
- Embedding essentially just involves solving a system of linear equations
- Use least squares estimation to find transform implied by a hypothesis
- Closed-form SVD-based method (from image registration) lets us do this in one step

Clustering of hypotheses

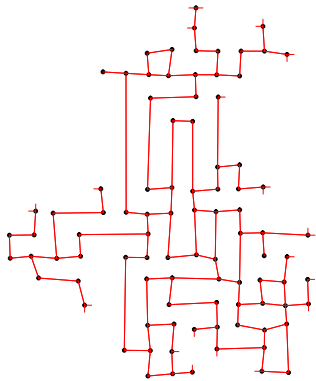
- Cluster based on closeness of transformations
- A cluster cannot have multiple correspondences for a vertex in either map (this is inconsistent)
- After clustering, order clusters by “quality”
 - number of vertex correspondences
 - total squared error under cluster transform
 - number and sizes of 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}

