Mapping With Limited Sensing Capabilities

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

October 15, 2003

Problem

- Need a map that is useful for navigation, and is *consistent*
- Our robots are limited in their sensing capabilities (and maybe their computational abilities as well), so:
  - Occupancy grid maps are not feasible!
  - Instead we will create a topological map (or a variation thereof)
  - This map can be navigated with a small set of simple predefined behaviors that are independent of sensing ability

What is a Map?

- We want to make maps that are useful to *robots* for *navigation*

- We *do not* require our maps to be useful to humans in any way!

- In particular, we would like a representation of the environment, and places within it, that:
  - Is easily searched for shortest paths
  - Can represent paths as a sequence of simple behaviors that will get the robot from one place to another
  - Is efficient to create, even with limited sensing ability
  - Is memory-efficient

Assumptions (for now)

- Approach: find something that works with a number of simplifying assumptions, and then work on relaxing these

- Environment: enclosed, rectilinear, static, “smooth”

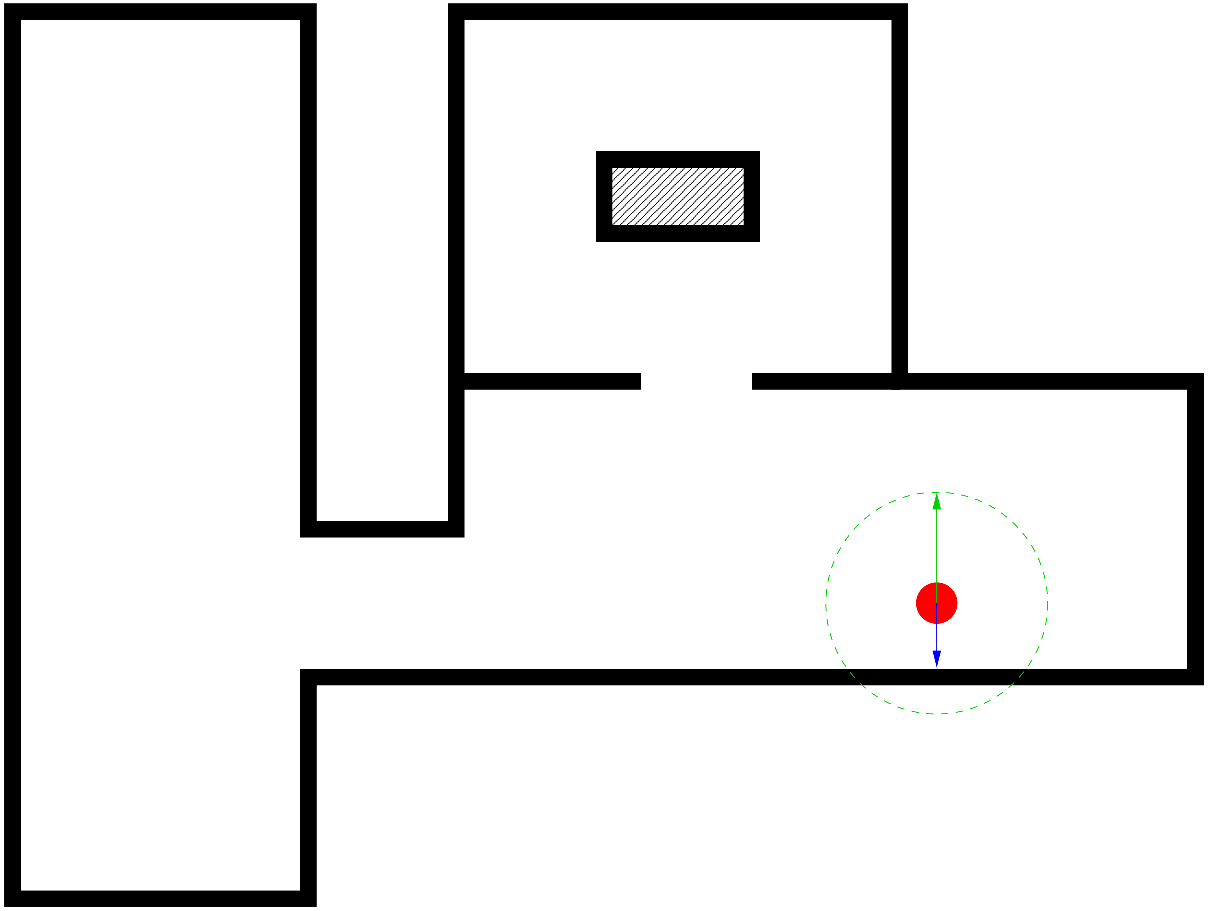
- Sensors/actuators:
  - Range sensors: known range
  - All sensors/actuators: *no error* (this is the biggest assumption by far!)

- Available behaviors:
  1. Straight-line wall-follow (with an optional distance argument)
  2. “Guarded move” (drive until we encounter a wall)
  3. Turn 90 degrees (left or right)

Three-phase Algorithm

- Algorithm phases:
- Phase 1: Create initial, very simple map, using only our available behaviors
- Phase 2: Refine the map to make it more useful for navigation
- Phase 3: Navigate using the refined map (hopefully just with graph search that outputs a sequence of behaviors to execute)

Example World

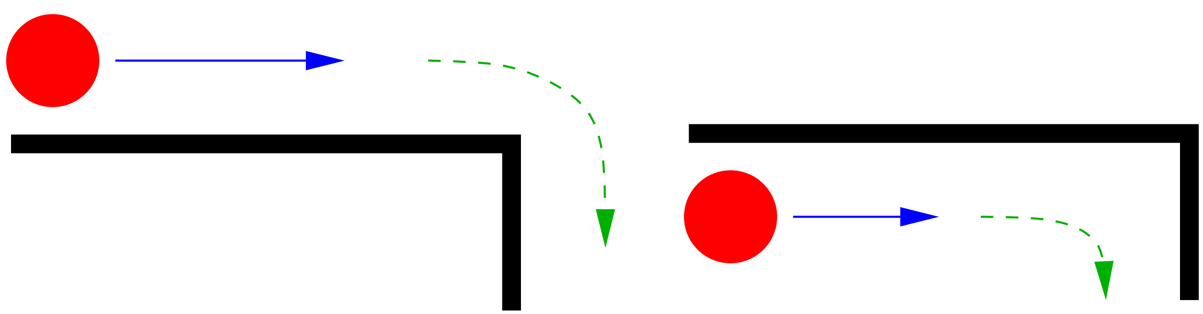


Phase 1: Create Initial Map

1. Idea: use "discontinuities" in walls that we are following as features in a topological map

2. *Discontinuity*: a location at which the robot must turn to continue following the wall (i.e. corners)

3. Two types in a rectilinear world: *exterior* and *interior* corners, at which the robot must turn 90 degrees



Phase 1: Create Initial Map (cont.)

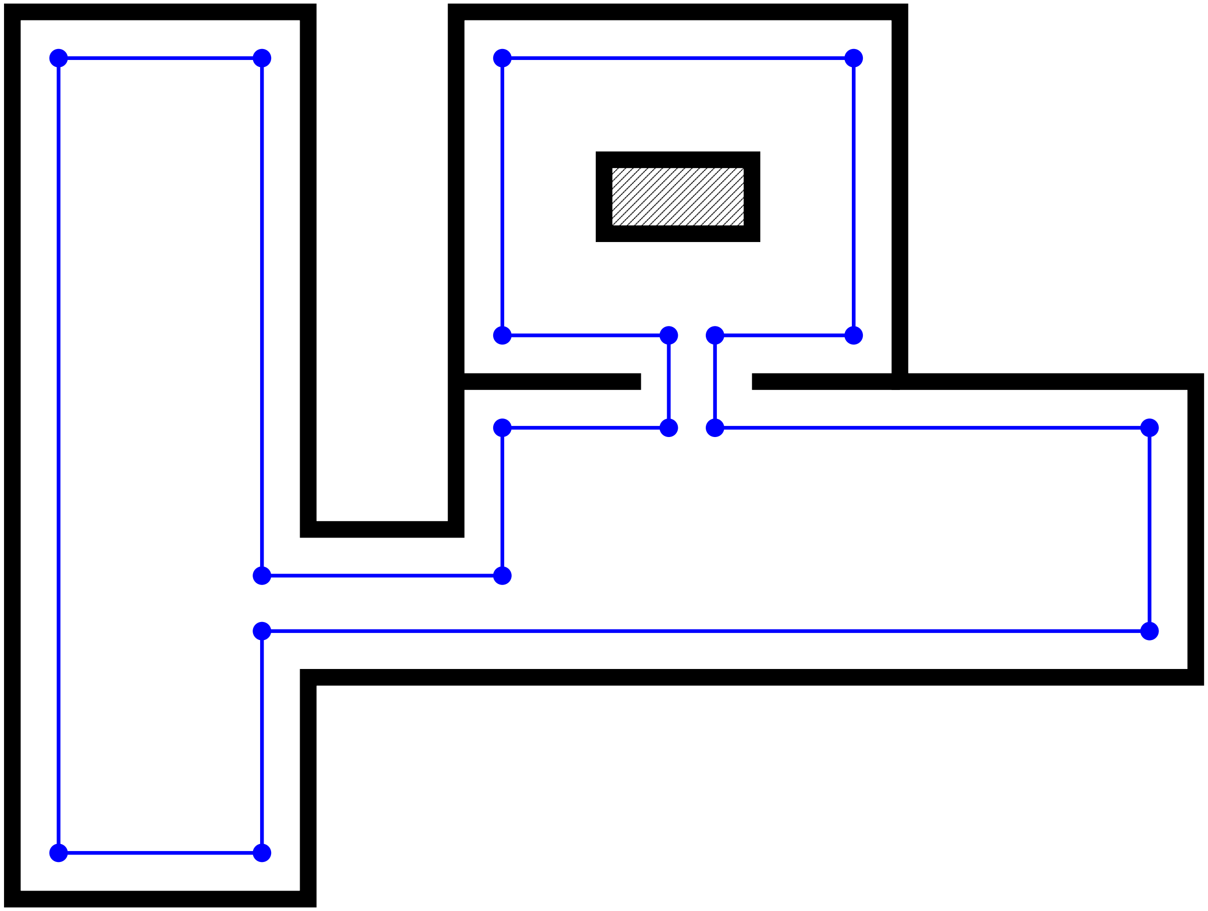
- Algorithm for creating the initial map:
  1. Initiate "guarded move" behavior until we encounter a wall; add a node to our map labeled with the configuration of the robot (e.g. its (x, y, θ)), and call this the *start node*
  2. Repeat: follow the wall (in either direction) until a discontinuity is encountered; when one is found:
     (a) Add a new node to our map, labeled with the configuration of the robot at the time it detected the discontinuity
     (b) Connect the new node to the previously encountered node with an undirected edge
  3. Stop when we return to the start node

- Call this initial map G_W (since we created it by wall-following)

Assumptions Make This Easy

- Enclosed environment: algorithm will terminate
- Static environment: we don't have to worry about, for example, opening/closing doors
- Known sensor range: we know that we've "seen" a specific amount of area around the robot's actual path
- No sensor/actuator error: we can reliably detect discontinuities, and we can reliably detect our location in the map (so, for example, we know when we return to the start node)
- Later, we'll have to overcome most of these issues!

Example World After Phase 1



Phase 2: Refinement

- The map we have now is already useful for navigation
- Any wall-following robot can find its way between two places that have been seen by the sensors during the map-making phase
- But, the path we take between the two places might be arbitrarily long, even if the places are very close!
- We'd like to "enhance" the map with, for example, knowledge that we can cross a hallway safely, and not lose ourselves in the map

Phase 2: Refinement (cont.)

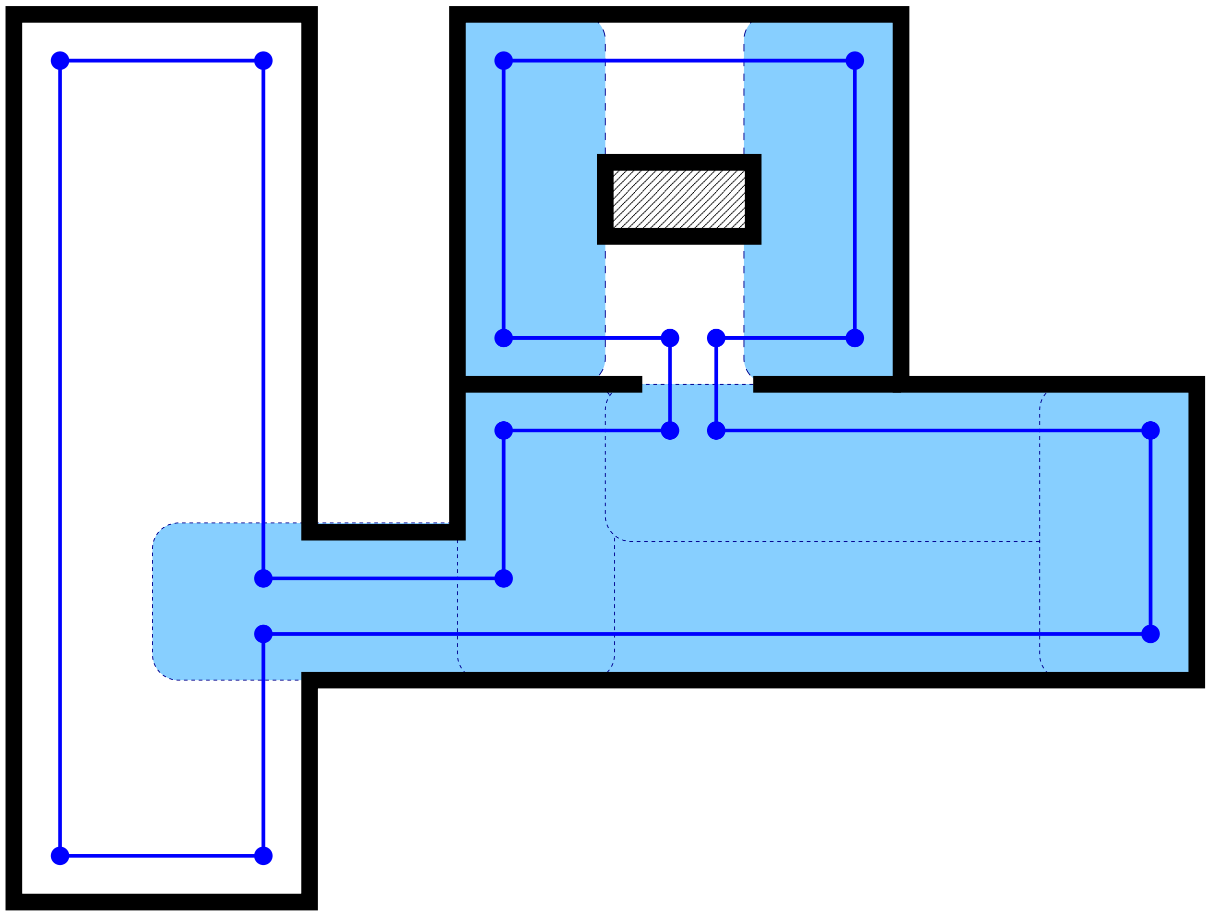
- Idea: use the fact that we know the range of the robot's sensors in order to infer the existence of a path between two parts of the map

- If two "path segments" of the map (edges of G_W) can "see" each other inside the area known after the initial mapping phase, we can infer that a path exists between them

- Use the rectilinear assumption, along with our turn and move-to-wall behaviors

- Show that, for some pairs of path segments, a specific sequence of actions guarantees that we can move from one edge to the other

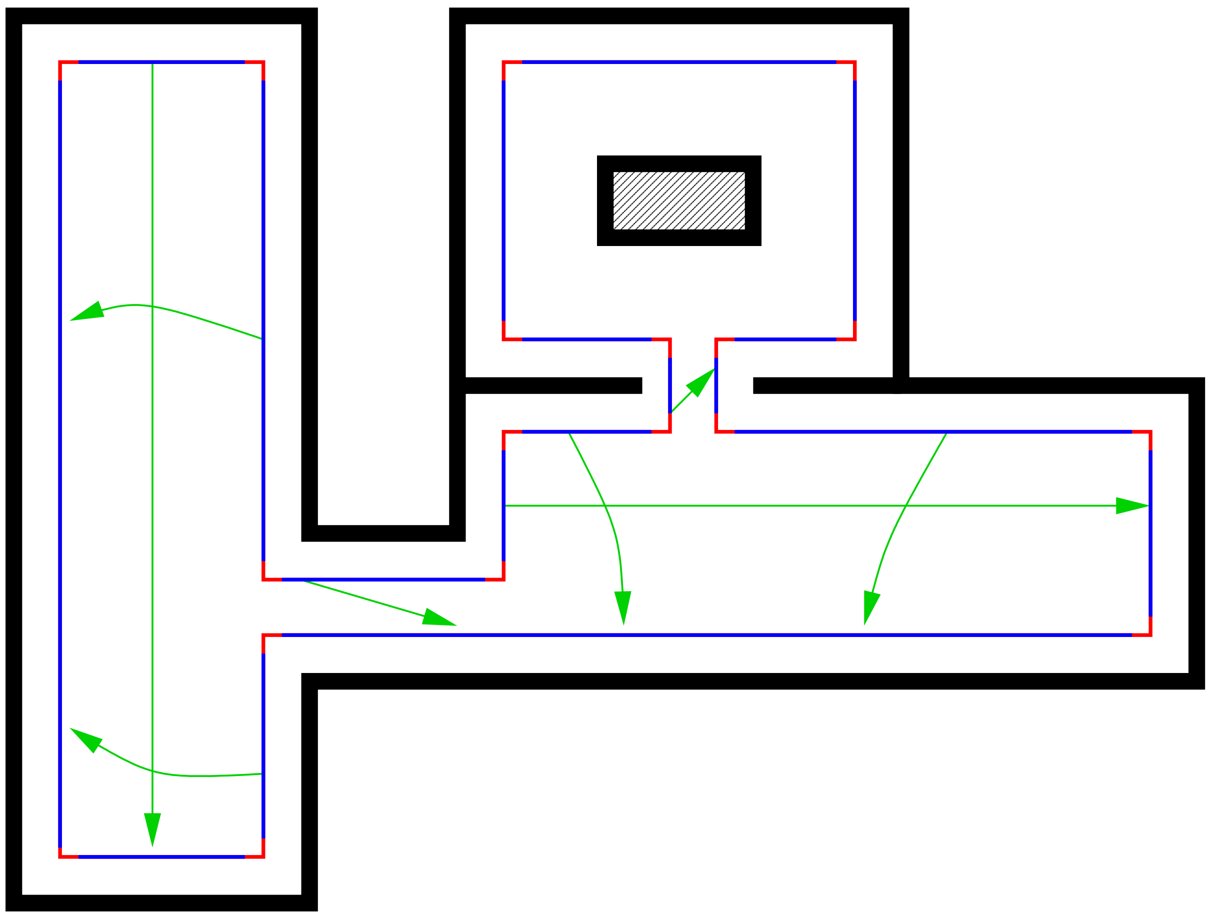
What the Robot Knows (partially highlighted)



Phase 2: Refinement (cont.)

- Simple algorithm:
  1. Create a new graph G_P, where each node represents an *edge* (path segment) from the original map G_W
  2. For all *ordered pairs* of path segments (p_1, p_2) in G_P such that p_1 and p_2 are *geometrically parallel*, and such that, when p_1 is projected onto the line on which p_2 lies, $p_1 \subseteq p_2$: add a *directed edge* from p_1 to p_2 if and only if a sweep of p_1 across the known free space, towards p_2, remains enclosed in the free space until it reaches p_2
- This results in G_P containing a directed edge from a path segment p_1 to another path segment, p_2, only if, at any point in p_1, a robot can:
  1. Turn 90 degrees away from the wall it is following
  2. Initiate its move-to-wall behavior
  3. Know that it lands on p_2 when it encounters a wall

G_P for the Example World



Phase 2: Refinement—Further Considerations

- How do we combine G_W and G_P into a representation that is useful for navigation?
  - One possibility: split p_2 into multiple segments; add new nodes to correspond to the ends of p_1, and connect the ends of p_1 to these new nodes
  - Or: find some other way of representing the fact that we can get from *any* point in p_1 to *some* point in p_2 (though with perfect sensing we know this point exactly)
  - Or: maybe we don't need to combine G_W and G_P?
- Are there better ways to represent our knowledge of the free space? Certainly we can do better given more complicated behaviors (e.g. move diagonally)
- We're still working on this!

Phase 3: Navigation

- Clearly this depends on what we finally end up with after the refinement stage

- Ideally, *given start and goal locations noted during the mapping stage*, we would like to:

  1. Connect the start node to some node (or path segment?) in our map; we might pick the node to move to based on some sort of planning, or maybe we just pick the closest node
  2. Connect the goal node to some node in our map with an undirected edge
  3. Use a graph search to find the shortest distance path through the graph from start to goal
  4. Traverse this path, which should be possible *using only the set of simple behaviors we assumed earlier*

Phase 3: Navigation (cont.)

- Note that we don't really need metric information for any part of the navigation except the shortest path search (assuming we know our initial location)!

- So, traversing the path depends entirely on our ability to detect discontinuities

- This is nice because. . .

Introducing Error

- As long as we can reliably detect discontinuities, it doesn't matter what our *exact metric location* is while we are traversing a path

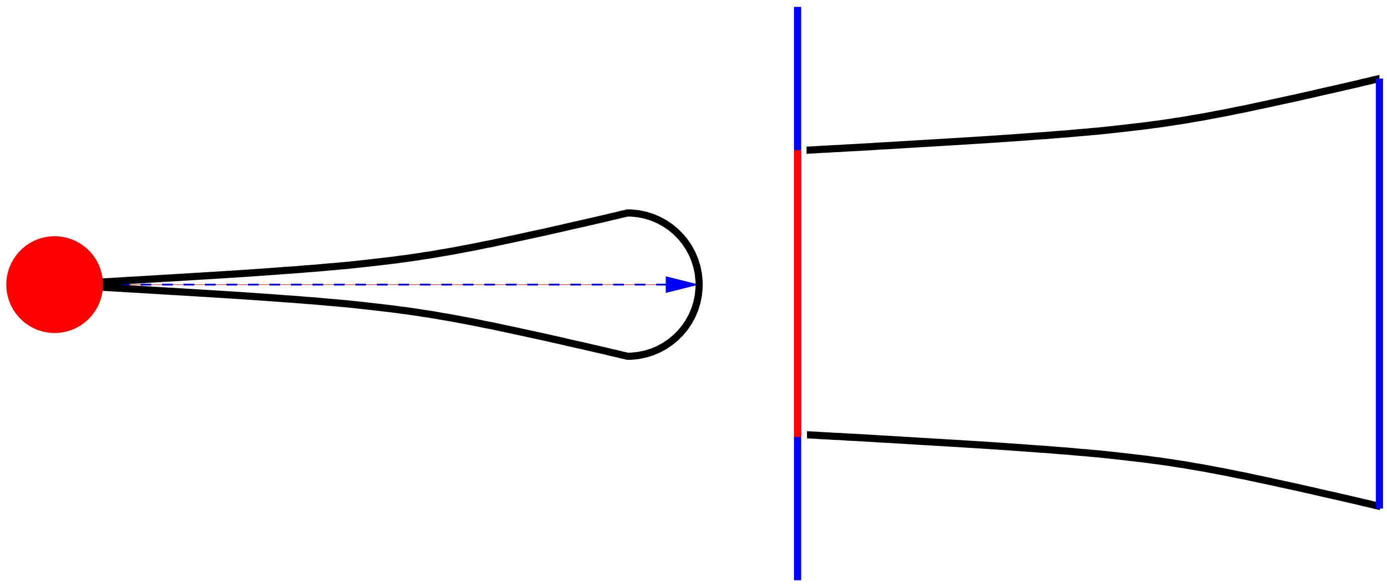
- So, at least for navigation, we just need to worry about error in detecting discontinuities

- What about during the map-making phase? Metric error (in measuring distance between discontinuities) will affect:
  - Knowing when we've returned to the start location
  - Shortest paths (we won't get lost, but our paths might be less optimal)
  - Refinement: we might make mistakes when we decide which path segments can "see" which other path segments

Refinement—Dealing With Distance Error

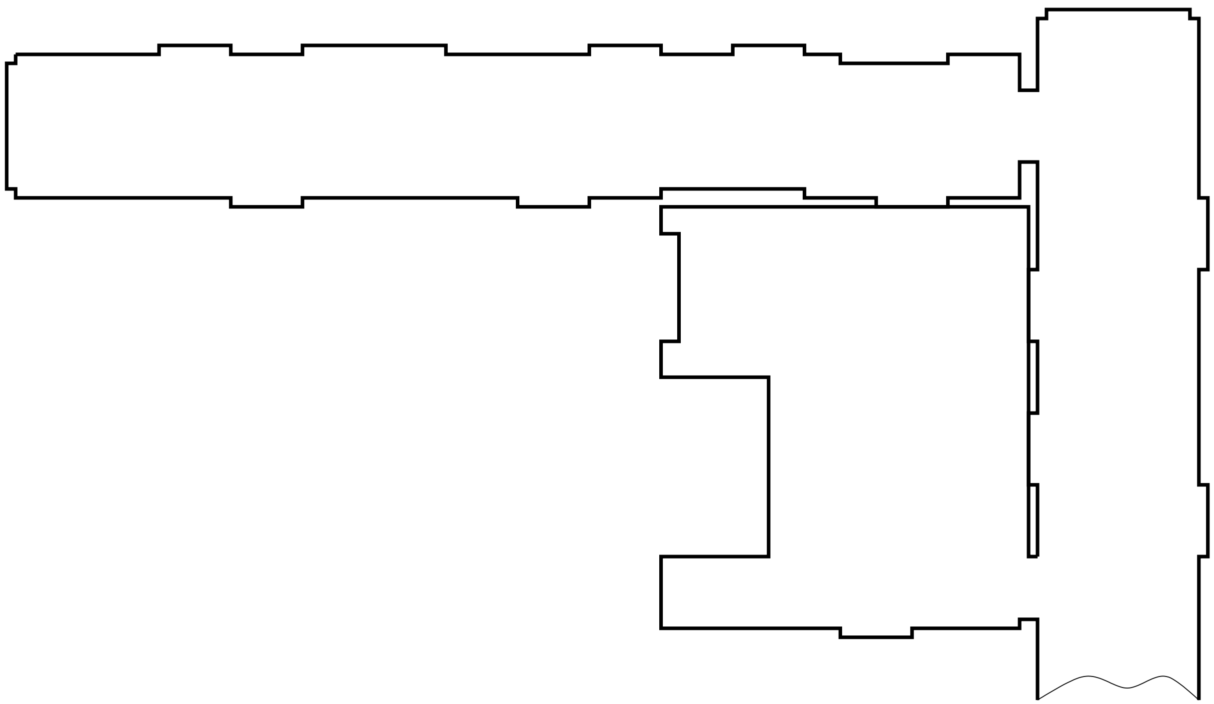
- Suppose we have some model of the odometry error (see below)

- We might be able to restrict the places on p_1 from which we leave, and still guarantee that we end up on p_2:



The “Smooth World” Assumption

- What about worlds like the one below?
- Much more likely to see error in detection of discontinuities



Dynamic Environment

- What happens when discontinuities change?
  - Someone moves a big box
  - Someone (gasp!) opens a door, presenting us with a whole new region
- We might try to recognize that discontinuities might be dynamic when we detect them. (But how? We'd probably be buried in special cases)
- In general, this problem seems hard (i.e. we haven't thought very much about it yet)

What's Next?

- Finalize the refinement phase
- Tackle some of our assumptions:
  - Add sensor/odometry/motion error
  - Generalize to polygonal environments
  - Think about dynamic environments
- Literature review
- Develop more complicated (/useful?) examples
- Implement on some robots with limited sensing capabilities, and try it out!