COSC 4P78

Navigation, Localization, and Mapping

Week 8

(Work based on Matarić and Murphy)

Brock University

Pre-Introduction

This week, we'll talk about localization and mapping, etc.
However, before we get to that, a quick disclaimer: at this level of discourse, we're going to gloss over certain potential concerns. e.g.:

- Certain wall materials might absorb ultrasonic waves, making walls look like hallways
- We may want independent mapping robots, or communication with additional devices (which introduces range, power, reliability, etc.)

These all *do* matter, but we've already discussed them.

Introduction

This lecture will cover:

- Navigation
  - The method by which a robot finds its way in the environment
- Path planning
  - Planning out the waypoints to achieve the navigation goal
- Localization
  - Determining where you are
- Coverage
  - Ensuring that you have physically covered the entirety of an area
- Mapping
  - Compiling knowlege of the environment for later reference

Navigation

- Navigation is one of the most challenging (and commonly-explored) tasks
- It connects the high-level goal of desiring a position/orientation to the low-level goal of directly interacting with the environment

So, why can navigation be so difficult? Let's look at a sample problem...

Navigation

A *decremental* approach

In this example (from the Robotics Primer), let's look at trying to find (and collect) a puck.

- Suppose the robot has a map of the world, with the locations of itself and the puck marked
    - We just need to work out the sequence of steps to get from point A to point B (with multiple possible intermediate points)
    - This is just a *path planning* problem
    - *If* the map can change (e.g. a hallway getting flooded), this could require a more elaborate mechanism

Navigation
Next decrement

- Suppose the robot has the map, and knows where the puck is, but doesn't know where it falls on that map
  - First, it needs to "find itself"
  - This is a *localization* problem
  - How could it accomplish this?
    - That would depend substantially on the nature of the map
    - This would also affect the required sensor suite of the robot

Navigation

More decrementin'...

- Suppose the robot knows the map, and its location, but doesn't know where the puck is
  - It needs to come up with an *efficient* means of searching the environment
  - It needs to guarantee that it can find a puck anywhere that's accessible
  - This is a *coverage* problem

Navigation

Still decrementin'...

- What if the robot doesn't even have a map at all?
  - It will probably have to *build* a map
  - This is a *mapping* problem
  - Not having a map doesn't mean you don't know where you are
    - If you're beside the Schmon Tower, then you know where you are... beside the Schmon Tower!
    - A robot might use such *landmarks* to identify locations
    - Navigation might then become a matter of connecting a string of landmarks
  - Note that we can still address coverage here; we may need to switch from a heuristic (e.g. spanning tree) to a more practical approach (e.g. start out with a spiral to find boundaries)

Navigation
(Interjection — Example Time!)

Suppose you wanted to get from Thistle 240 to J333.
And further suppose I were to tell you to go from Thistle 240 to J333 by first walking to the library, taking the shortcut to the tower, then to the large staircase, to Tarot Hall, to A, C, D, and then to J-block, and that it's the first door on the left.
None of that involves explicit, distance-based cartography. However, could that conceivably be enough to navigate from the origin to the destination?

Navigation
(almost done!)

- So, we've accepted that we don't have a map — and thus don't know where the puck is on the map
- Would it make sense to fully search the environment, building up a map, and to *then* start looking for the puck?
  - Probably not
  - Alternatively, we could try searching for the puck, and also construct a map as we went along
  - This is called **S***imultaneous* **L***ocalization* **a***nd* **M***apping* (SLAM)
    - (or **C***oncurrent* **M***apping and* **L***ocalization*)

Localization

As previously mentioned, localization is the task of identifying location. There are numerous different ways to do this, and which method is appropriate will largely depend on the type of map being used (more on that in a little while).

- Simply examining your environment may not be enough
  - Seeing that you're in a hallway intersection eliminates several options, but may not be sufficiently unique
- GPS is an obvious option
  - Where the GPS system says you are: that's where you are
  - Is GPS always accurate? *How* accurate?
  - Is there any way to improve on this?
  - What do we do when GPS isn't available at all (e.g. indoors, where there are too many natural barriers, when power is strictly limited)?
- Say you're in a car at Brock. You drive at about 50km/h for about half an hour. Where are you now?
  - You're *approximately* 25km away from Brock
  - Simply using the car's odometer would give an even better idea
  - We can do something similar with robots (*odometry*), by using shaft encoders...

Localization
Odometry

- It's the same premise as dead reckoning and your car's odometer
- The shaft encoder tells us how many times the wheels have turned

If a car's tires aren't properly inflated, does that affect the accuracy of the odometer? What happens if a robot's wheels aren't perfect tori? What happens if they slip slightly in a puddle?

- As distances increase, error accumulates!
- We've already discussed this type of issue as *drift*
- Our estimation of position may be assisted by reading multiple datas (e.g. counting mile markers when they're available), but this still leaves it as that: an estimate
- Sooner or later, we may need to augment with some form of landmark!

Let's take a breather...

So, we've talked about all of the topics we're going to need to address. What's more, of the 'big questions' for navigation:

- Where am I going?
- What's the best way to get there?
- Where have I been?
- Where am I?

We've at least touched on them all enough to understand the concerns we should consider when working out navigation.

Next, we should talk about path planning, but first we need a bit of background information...

Evaluating Path Planners

- Complexity — is the algorithm computationally efficient enough (or efficient enough for space) for the limitations of the robot?
- Sufficiently represents the terrain — Will it always be in a flat lab environment? What about inclines, slippery materials (like sand or mud), weight, etc? It's more complicated that simply marking areas as *navigable* or *not navigable*!
  - Should we explore this a bit?
- Represents the limitations of the robot platform — an algorithm might assume a *holonomic* platform (e.g. the ability to face any direction, without having to use additional space to get turned around).
  - Also, **should we assume the robot's a point?**

Evaluating Path Planners
(cont)

- Compatible with the reactive layer — path planning is, by definition, deliberative. If you're using a hybrid architecture, then it's the reactive layer that'll be handling the actual movement; you'll need to find a way to get the two to cooperate!
- Resilience to changes in map and re-planning — In some cases, it might be imperative to be able to adapt to unexpected changes in the environment

Spatial Memory

We've already touched on the idea of representation; now we need to look more carefully.

- You can't answer the question of, "how do we get there?" without first deciding on a means of representing where you are, and how to direct steps of the path
- *Spatial memory* is the fancy term for how we're going to represent our the world
  - Is everything a well-defined polygonal structure with sets of absolute coordinates?
  - Is the entire world represented by "where you can go from each place you can be"?
  - It should help us map our plans to outputs
    - e.g. if we're supposed to "go down the hall to the third red door on the right.", how do we do that?
    - How do we know what a door is? How are we identifying *red* doors?

So, how do we make use of spatial memory?

Spatial Memory
(cont)

Spatial memory *supports* four basic functions:

- Attention — what features/landmarks to look for next?
- Reasoning — can that surface support my weight?
- Path planning — what is the best way through this space?
- Information collection — What does this place look like? Have I seen it before? Has anything changed since then?

Spatial Memory
(more cont)

Spatial memory takes two forms: route/qualitative, or layout/metric.

- Route representations express connections between landmarks
- Metric provides actual information about layout (e.g. maps) — think *bird's-eye view*

Note that a layout representation can generate a route representation, but the converse may not be as easy.
For example, we talked about directions to J-block, but does that mean you can definitely draw a detailed map?

A final note on spatial memory...

Don't fall for the trap of assuming that more=better.

A reactive paradigm, for example, often won't be well-served by being overloaded with superfluous information. Sometimes *less* really is *more*.

Other considerations when deciding on a representation include whether or not an optimal path is required (these often require metric), whether or not accurate distances are known, availability of landmarks, sources of information about the terrain, how the sensors work (and/or how well they work) in the environment, etc.

Topological Path Planning

(Or Route, or qualitative)

As mentioned before, route path planning determines a path as a sequence of directions. Typically, one of two styles of representation will be employed:

- Relational — akin to “connect the dots”, these are your traditional topological graphs. You can even employ many classic graph-based algorithms!
- Associative — In this case, the spatial memory takes on the form of a graph of associations; extracted features from sensors are coupled with locations

Typically, relational techniques tend to better support path planning; associative is better for retracing known paths.

Landmarks and Gateways

The only way we can travel from point A to point B (in the absence of any notion of absolute coordinates for either point) is if there are some features in each location that unique identify them.

- Landmarks don't need to be singular, self-contained objects (e.g. "red door")
  - We can look for groups of features that collectively identify a location
  - e.g. a tall sign, brightly-coloured building, and lots of traffic might indicate a McDonald's
  - Your landmark could be, "that collection of trees"
- *Gateways* are a special case — a landmark with the opportunity to change direction
  - e.g. if you can uniquely identify a hallway intersection, you can use it to know where you are *and* to decide if you want to switch to the orthogonal hallway
- Either way, landmarks may be natural or artificial
  - In this context, artificial doesn't just mean "man-made"; it means, "created or modified to help it serve as a landmark for navigation"

Landmarks and Gateways
(cont)

Whether natural or artificial, in order for a landmark to be suitable for localization, it must satisfy the following criteria:

- Be readily recognizable — if your robot can't recognize it, it isn't useful
- Support the task-dependent activity — basic orientation cues (like "take the second right after the McDonald's") are easily satisfied, but if precise positioning is required the landmark may need a way to extract additional information (e.g. precise relative distances, etc.).
- Be perceivable from many different viewpoints — a McDonalds is pretty easy to spot from any angle, but this is not true for lots of other stores

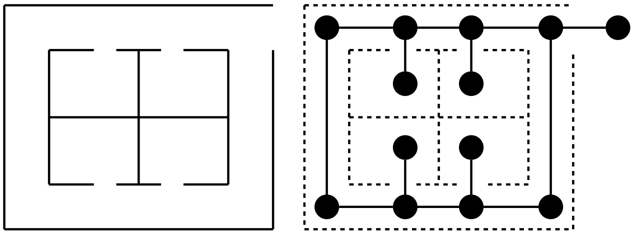
On a related note, there's the issue of uniqueness — depending on the task, a McDonalds might be enough for directions, but obviously the existence of one wouldn't *always* be sufficient for localization, because a city might have more than one McDonalds location in it!

Relational Methods

For topological path planning

As previously indicated, this is a graph-based approach.

- Nodes represent landmarks, gateways, or goals
- Edges represent navigable paths
- Additional information (direction, approximate distance, terrain type, special behaviours for navigation, etc.) *may* be added to the edges



Distinctive Places

It might make sense to use special landmarks (*distinctive places*) that are only detectable within a particular region (or neighbourhood)

- For example, trying to identify hallways and doors
- Note that this is a bit limited, and may require modification of the environment
- It helps eliminate navigational errors at each node (even if the robot drifts, it can still self-correct and localize itself)

To be honest, this is a neat topic, but too esoteric to get into detail here.

Associative Methods

For topological path planning

Quite literally what it sounds like: *associating* an input pattern with a location; if you can rely on experiencing the same perception upon your return, then you can identify that location.

In order for this to work, we usually require:

- Perceptual stability — views that are close together should look similar
- Perceptual distinguishability — views that are far away should look different

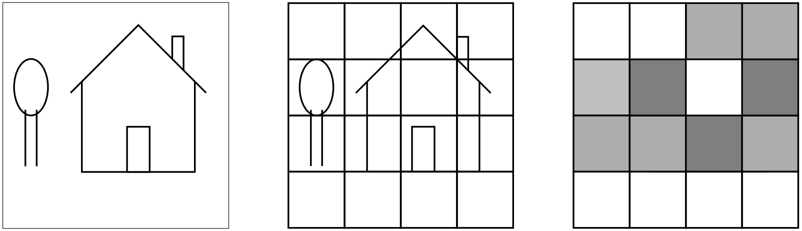
Visual homing

Image signatures might be *substantially* simplified from the captured images (next slides).

- They may be based on the image itself, or they might collect attributes such as edge density, dominant edge orientation, average intensity, etc.
- Multiple signatures for a single location might be used to provide additional orientation help
  - e.g. take a snap of the view when you need to turn right

Visual homing

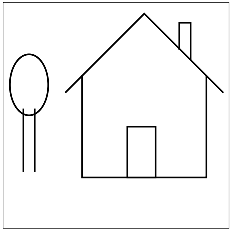
Example

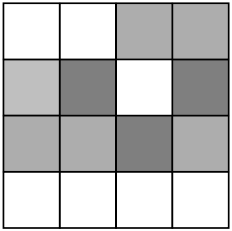


Visual homing

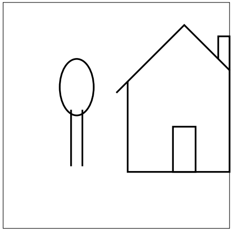
Example (cont)

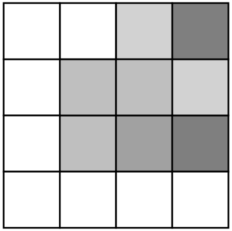
After moving a bit closer:





After turning left a bit:





Hypothetical

Random question.

Suppose you ask Facilities for a floor plan to Brock, and they ask if you want *vector* or *raster*.

- What's the difference?

Metric Path Planning

Quantitative navigation is the opposite of topological navigation.

- They typically favour optimal (according to some metric) paths
- In the absence of landmarks, metric path planning tends to decompose into sequences of waypoints (typically in fixed locations or x/y coordinates)
- Metric path planners require two components:
  - The representation (or data structure)
  - The algorithm
    - By the time it gets to the planning algorithm, it still may treat it as a graph problem
    - Alternatively, it could even treat it as a colouring problem

Most metric approaches involve some form of partitioning.

Configuration Space

Recall that objects in the physical world can have their configuration defined in terms of degrees. When it comes to representation of an environment, it's typically advantageous to choose a *configuration space* (or *Cspace*) that reduces the number of dimensions (and thus shrinks both the search space and the memory requirements).

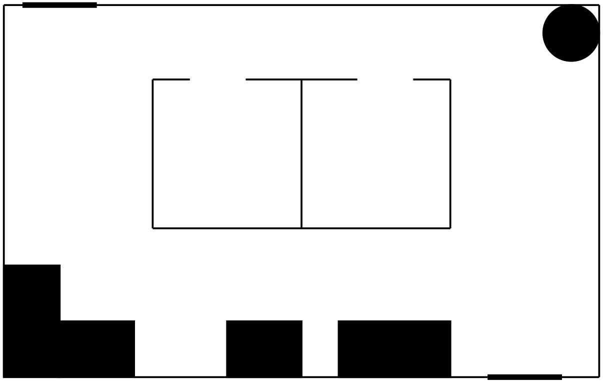
- e.g. a 3D environment might be projected onto a plane, reducing it into 2D coordinates
- Of course, there may be cases where this is a bad idea — airplanes like acknowledging height, tables sometimes have legs, and rescue robots might need to deal with multiple floors

Example

Floor plan

Consider the following room.

Again, *can* we assume the robot is just a point in space?

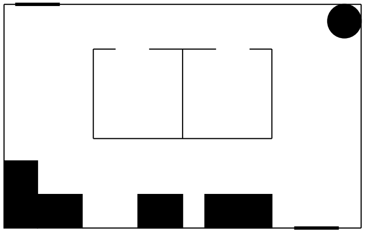


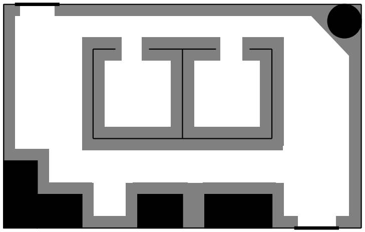
Cspace Representations

- Meadow Maps
  - Divide the area up into convex polygons
  - You can always safely travel along the outer edges, and use them to get to adjacent polygons!
- Generalized Voronoi Graphs
  - Based on partitioning into central points
  - Navigate from Voronoi vertex to vertex
  - (since you're always in the middle of each space, you don't need to account for the size of the robot)
- Regular Grids
  - Obviously the simplest
  - Inefficient in that it either uses very small partitions (costly, both computationally and in terms of memory!), or larger partitions (meaning the coarse representation makes obstacles appear to be larger than they are)
- Quadtrees
  - Minimizes wasted space by partitioning large spaces only when necessary

Example

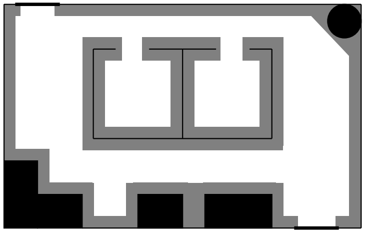
Floor plan and shrunken version

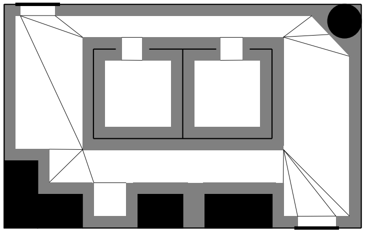




Example

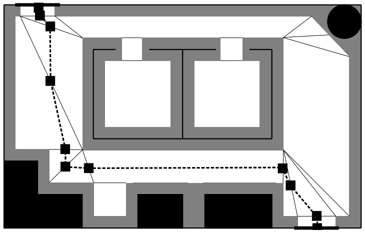
Meadow Map

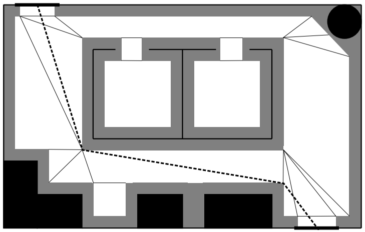




Example

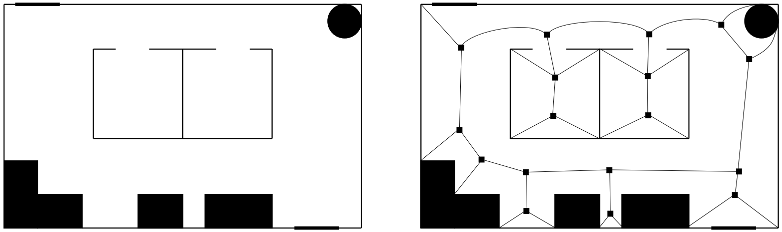
Meadow Map – Path (and with string-tightening)





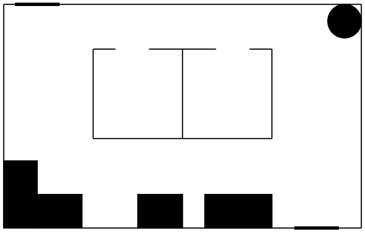
Example

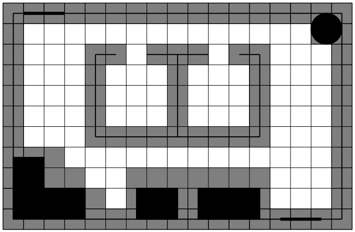
Generalized Voronoi



Example

Regular Grid





Regular Grid? Quadtrees?

If we haven't already discussed it, what's the problem with regular grids?

How could quadtrees help with that?

Additional Considerations

Since we're talking about it...

How else can be represent environments?

- How big of a concern is memory? What can you do when it is?
- When *mapping*, how do you pick the map's *origin*?

Anything else we could discuss about this?

Example
Questions?

Next time, nobody's favourite Avenger!

(aka Vision)