Robot Navigation

- A -> B w/o hitting stuff
- Potential field is a type of map

- Option one:
  Occupancy Grid

- Option two: Navigation Mesh

- In other case, we can form a graph:
  - Vertices/nodes correspond to free space
  - Edges correspond to connectivity

- Graph for Nav. mesh:

- For occ. grids, all open cells would be nodes w/ links connecting adjacent cells.
Final Result

Cost: 48
London → Paris → Vienna → Budapest

Find path by following predecessors

Should not end as soon as you find the goal.

A \rightarrow B, 1000 \quad \text{This would not find the best solution if you ended early while Dijkstra's would find the optimal solution.}
A* very similar to Dijkstra's.

Video of little dog.

Create a matrix of cells represented with a graph. Each foot is in one node. Want to step in better places:
- Low slope, smooth, center of things
- Can create costs for steps
- Just looking at steps instead of 18-dimension full-body problem.
The A* Algorithm

- Have a start and a goal.

The A* Algorithm

- States are any node. Can go to any neighbor.

The A* Algorithm

- Some actions are allowed.
- The successors are not necessarily immediate neighbors.
Assign costs to colours.

- Still have a cost-to-come function \( g(x) \)

List of things to maintain:

- Now we need a heuristic.
- It should be optimistic (best case scenario).
- Our path is guaranteed to be a shortest path IF our heuristic is optimistic.
- Heuristic should be simple to compute: \( h(x) = \max(dx, dy) \)
The A* Algorithm

- Dijkstra's sorted priority queue based on g(x). A* sorts on f(x) = g(x) + h(x).
- f(x) is best guess for each node.

The A* Algorithm

- Final step of A* in action.
- Don't requeue nodes you've already seen when it costs more.

The A* Algorithm

- Find path by following predecessors.
Dijkstra’s vs. A*

- Red is low cost-to-come.
- Search expands in all directions.

- Increases the h(x) as you go in the wrong direction.
- Cause the algorithm to not pursue those paths.
Blue is cheap. High slope, edge, high concavity are expensive (red). Cost is also dependent on the four footsteps' configurations. We don't want awkward stretching.

So cute 😊
Ensuring feasible plans

Feet poses: 8 dimensions.

Need to verify the position in the higher dimension problem.

Need to adjust joints to avoid knee collisions.

We can use the Jacobian to readjust knees, hips, to avoid collisions while keeping feet planted.

Cost Functions

“By far, human engineering is the most common approach to generating cost functions. This is somewhat of a black art, requiring a deep understanding of the features being used, the metric to be optimized, and the behavior of the planner.” [Silver et. al 2004]

A* is good at reasoning with geometry, but it’s just a proxy for the probabilistic functions that we really want.
Want to extract features and turn them into costs.

Heuristic used is a proxy for true function.
Hard to reason absolutely.

Do the easier thing + specify relations - preferences of one step location over another.

Input: relative preferences
Output: Ranking of all places to step
(Ordered Regression)

At best, this is still a weak proxy for true performance.

Questions + Notes:
Order of feet?  
Pre-determined  @4 @1
Avoids too
at an exponentially increasing # of nodes @2 @3
Learned terrain cost
Reflexes and recovery

- Detect execution error
- Stop robot safely
- Kick off new plan
- Replanning speed is critical

Real-time planning

time to compute initial plan