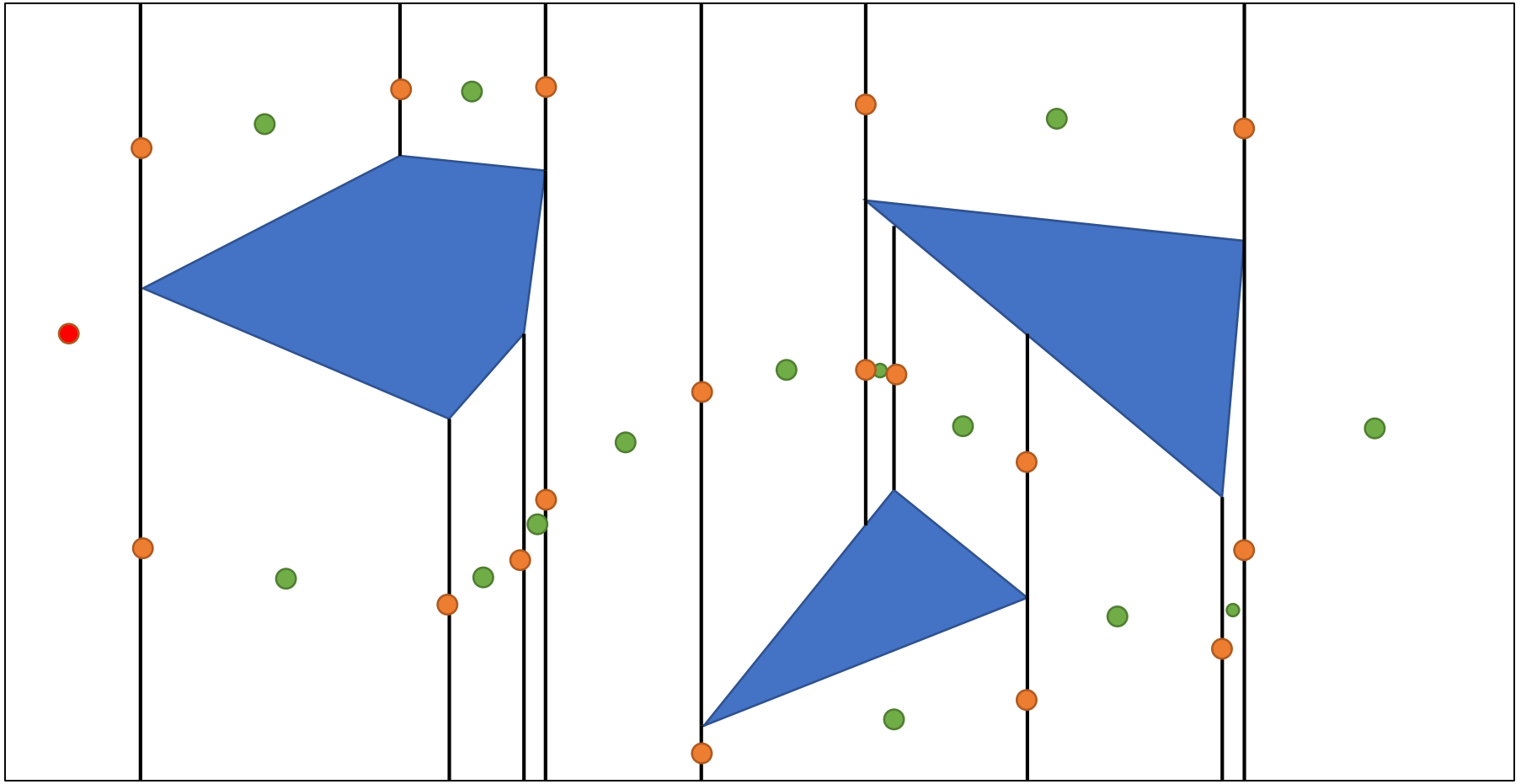
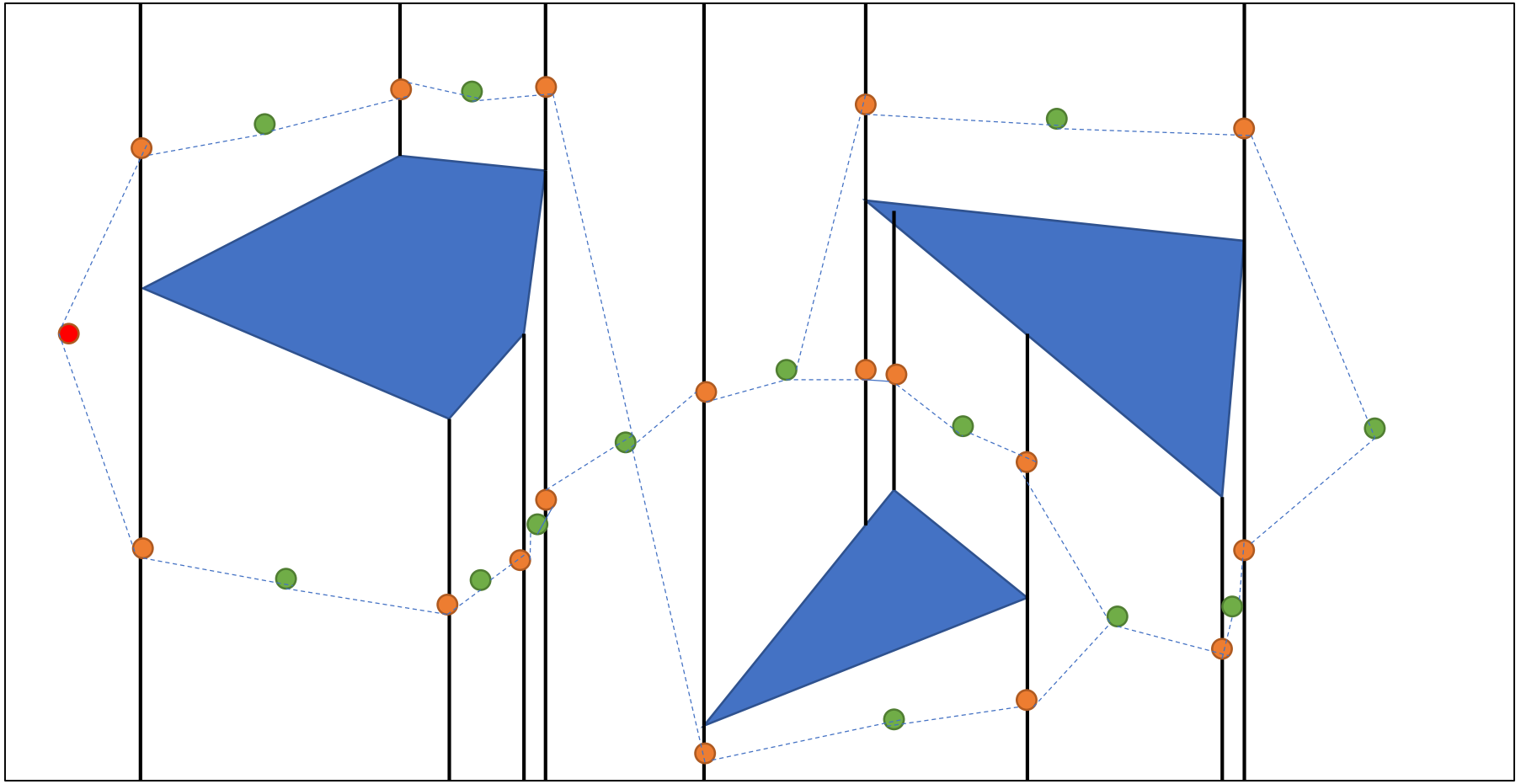
CMSC828T
Vision, Planning And Control In Aerial Robotics

Sampling-Based Planning Techniques

Roadmaps



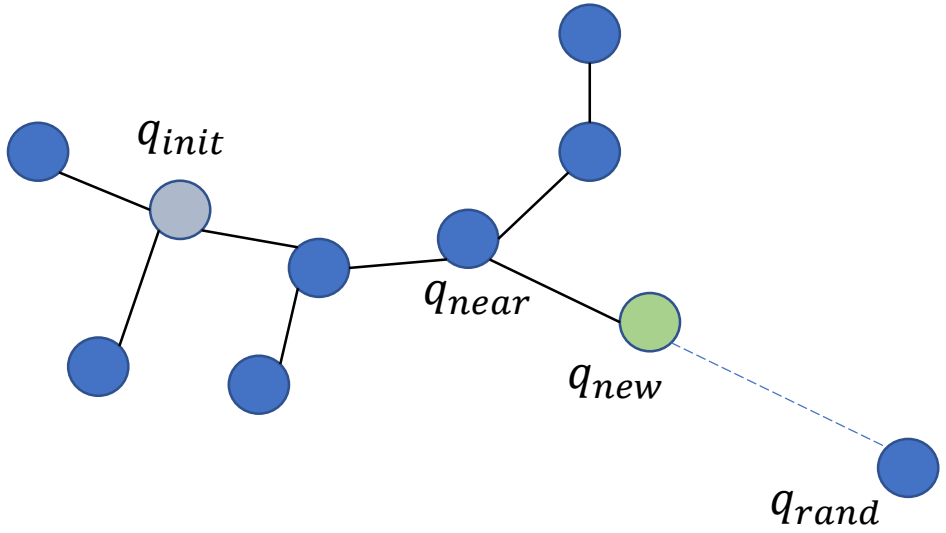
Roadmaps



Rapidly-Exploring Random Trees (RRT)

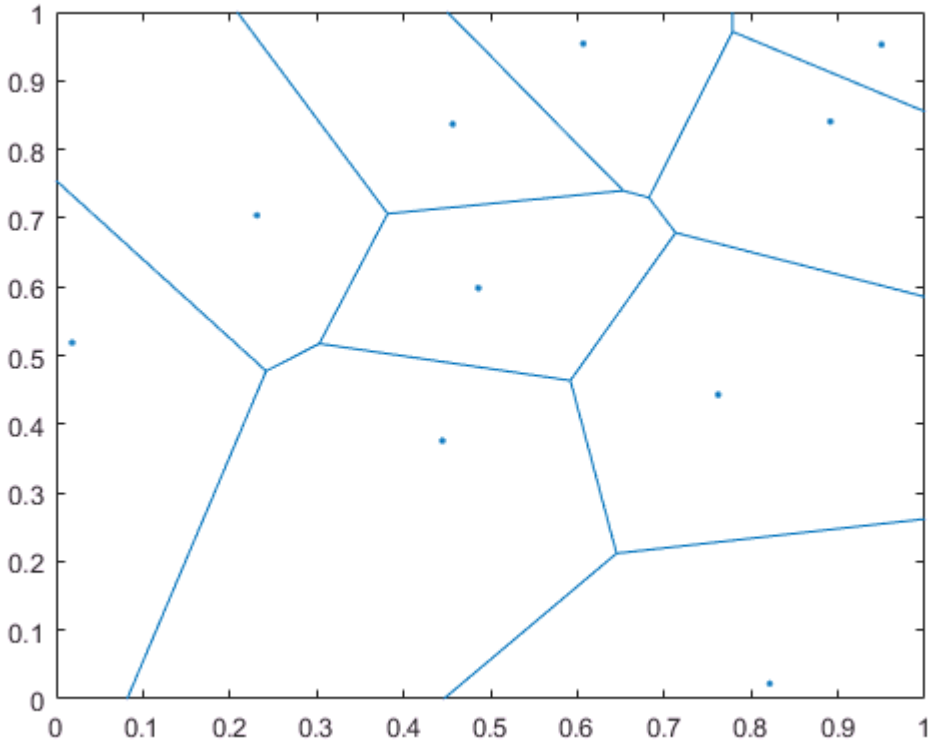
- Kuffner, Lavalle
- Vanilla RRT
  - Single Tree
  - Bidirectional
  - Multiple Trees (aka “forests")
- RRT + Differential constraints
  - Non-holonomic robots
  - Kinodynamic constraints

Rapidly-Exploring Random Trees (RRT)



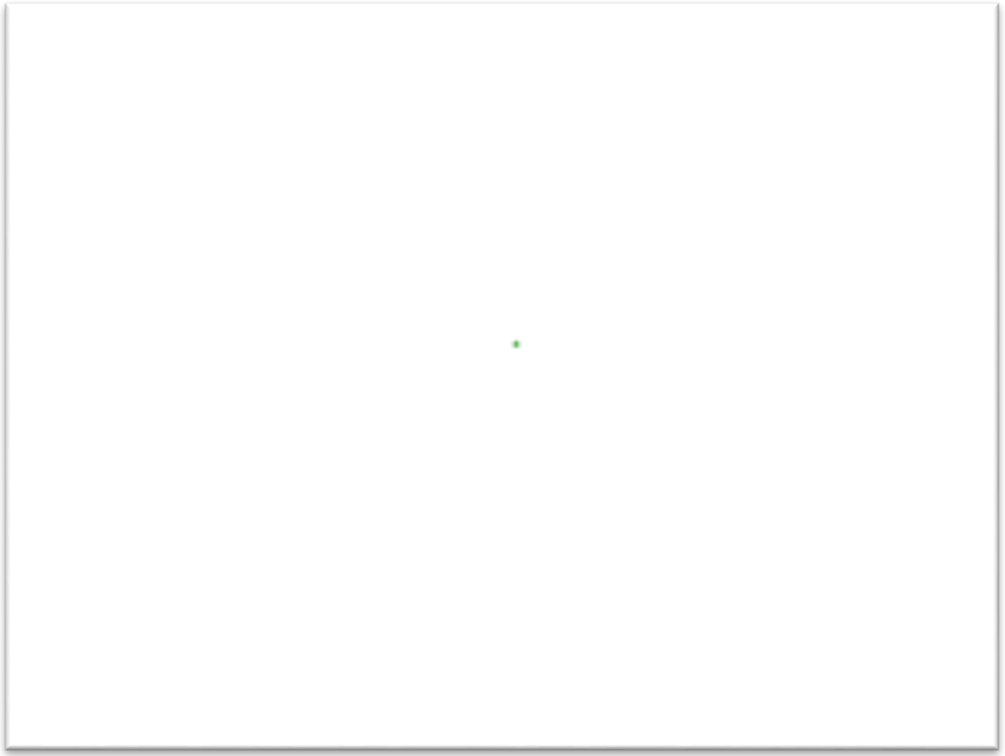
- Pick a random point in space
- Try to connect it to nearest "node" in the tree
- If no obstacles AND all constraints satisfied, attach to tree
- Else, try again.

Voronoi Diagram



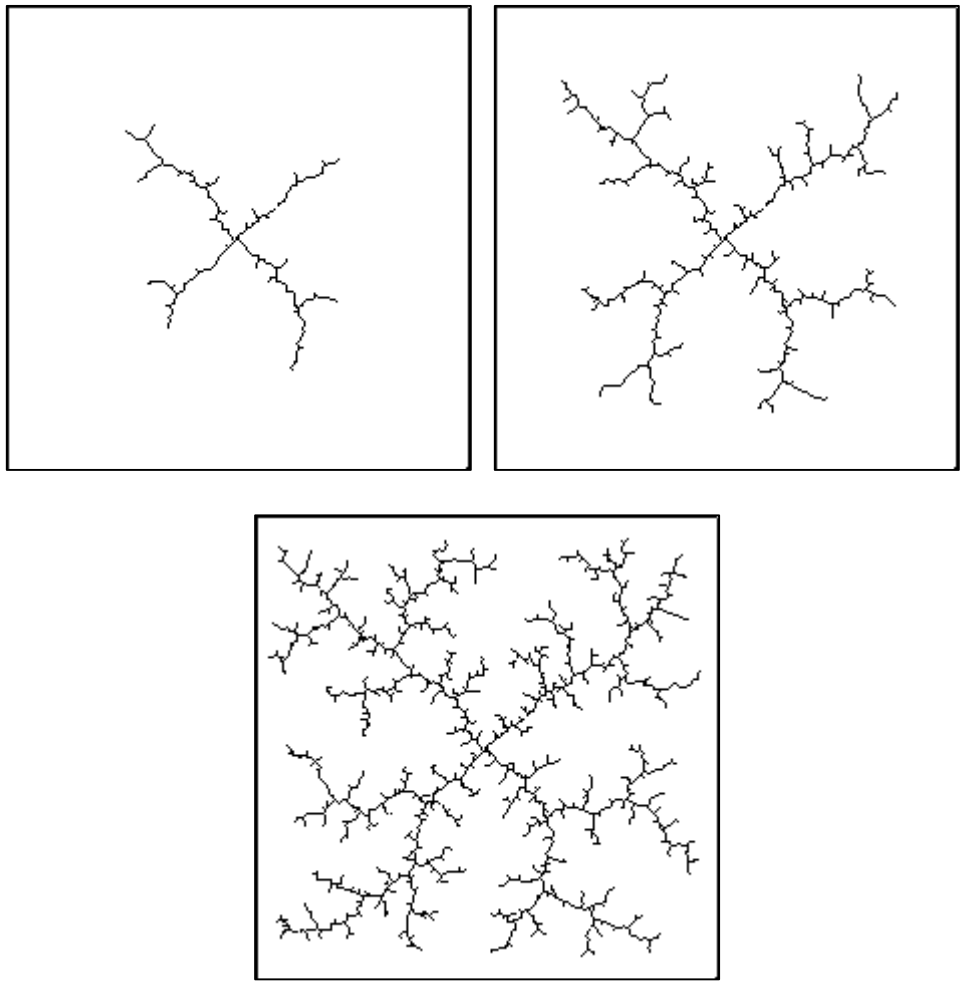
- Voronoi diagram of discrete set of points X
- For each point X_i decompose space around it into some region of influence R_i
- Property:
  - Any point P in this region is closer to X_i than any other point

RRTs and Voronoi Biasing



- Given uniform sampling, probability of expanding an existing state is proportional to size of the Voronoi region of that point.
- Since the largest Voronoi region will belong to states on the search frontier, inherent preference to move towards unexplored regions.
- Hence, Rapidly Exploring
- Read:
  - http://msl.cs.uiuc.edu/~lavalle/papers/LinLav04.pdf

RRTs and Voronoi Biasing



- Naïve random tree:
  - All new points clustered around the start vertex.
  - Little to no “motive” for exploration, keeps visiting explored areas.
- Random Walk, same bias towards explored areas.
- RRT depends on nearest neighbours, largest Voronoi regions more likely selected. Large Voronoi regions occur at search “frontier”.
- Can think of as breaking up, iteratively, large Voronoi regions into smaller ones.

RRTs and Biasing

- Larger spaces biasing (as discussed), causing more exploration.
- Biasing also causes the search to be “focused" towards the goal.
  - When a new point is picked, from time to time, substitute the goal point instead of the random point and see if conditions are satisfied.
- Another option is bidirectional RRT:
  - Start one tree from start node, other from goal node
  - See if they meet

RRT Summary and Notes

- Single parameter controls output
- Greedy-exploration balanced
- Simple, easy to implement

- Metric sensitivity
- Nearest-neighbor decides efficiency
- Unknown rate of convergence

- Example of probabilistic complete, i.e. chances of convergence reaches 100% as iterations reaches ∞

CMSC828T
Vision, Planning And Control In Aerial Robotics

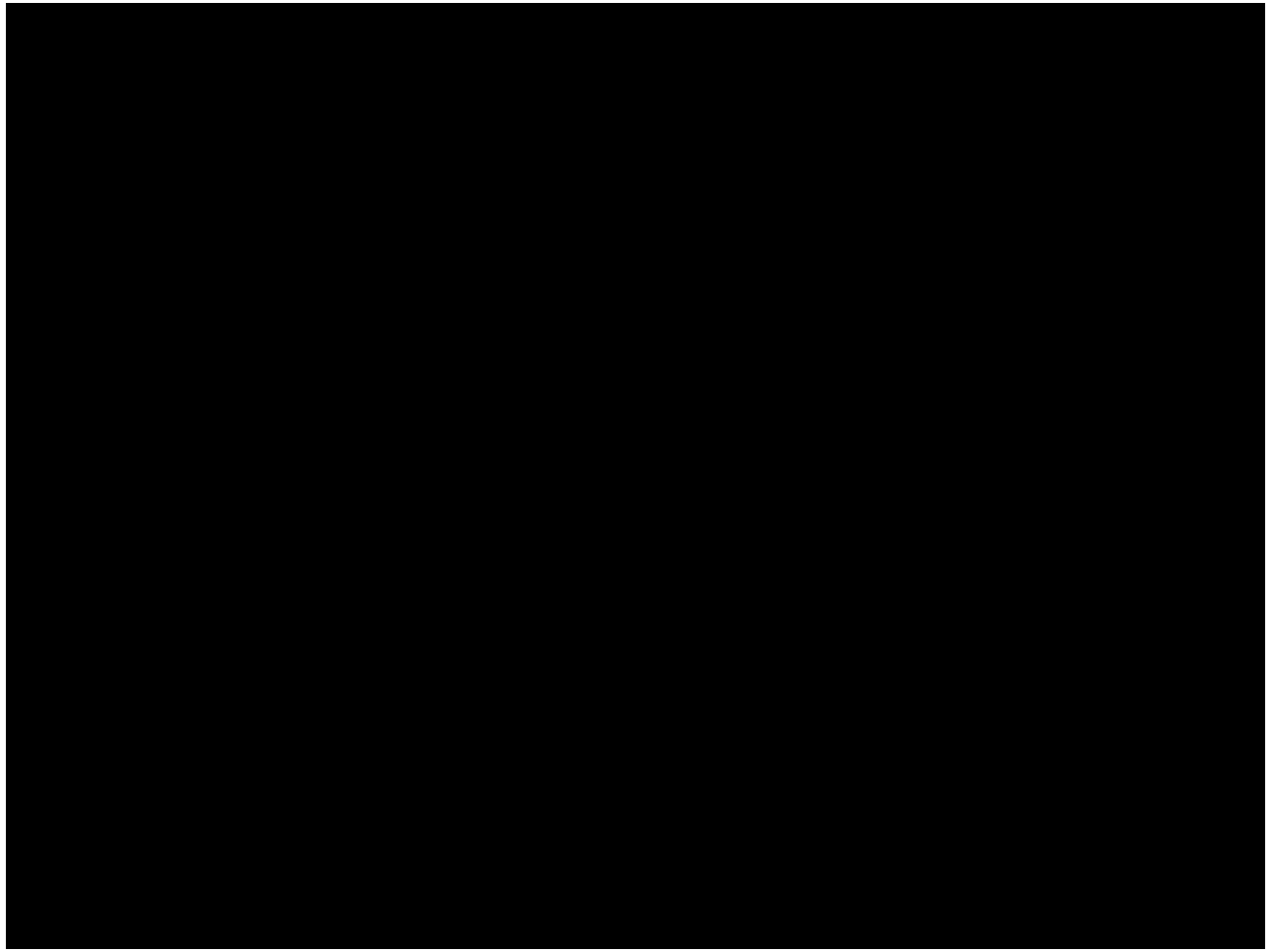
Adaptive Search Techniques

D* Algorithm

- Anthony Stentz – informed, incremental search
- D* → Dynamic A*
  - Works just like A*, but edge weights change during runtime.
- A* throws out information gathered during planning, since replanning is not a factor.
  - D* keeps and uses that information to plan efficiently.
- Check out Focused D* and D* Lite, better algorithms for real-world uses.

D* Lite Video

https://www.youtube.com/watch?v=X5a149nSE9s



D* Algorithm

- Data Structures:
  - NEW list → node has never been in OPEN list
  - OPEN list → node is currently OPEN
  - CLOSED list → node is no longer OPEN
  - RAISE list → node cost is higher than last time it was OPEN
  - LOWER list → node cost is lower than last time it was OPEN

D* Algorithm: Expansion

- Pick node from OPEN and evaluate values.
- Propagate information change to all neighbouring nodes, place them in OPEN.
- D* goes backward, from goal to start.
- Each expanded node has backpointer, aka the next node.
- Cost to target is known exactly.
- When start node is encountered, algorithm ends.
- To find path, we follow backpointers to goal.

D* Algorithm: Obstacles

- When obstacle detected, all affected points put back on OPEN list, with RAISE flag.
- Before cost RAISED, check neighbours to try and reduce cost.
- Else, RAISE flag propagated to all nodes with backpointers.
- If cost can be reduced, place LOWER flag and propagate via backpointers.

D* Algorithm: Example

- (X,Y)
  - states of robot
- b(X) = Y
  - backpointer from X to Y
- c(X,Y)
  - Arc cost from Y to X
- r(X,Y)
  - Arc cost from Y to X from sensor
- t(X)
  - Flag (OPEN, CLOSE, CLOSE)
- h(X)
  - Path cost
- k(X)
  - Smallest h(X) since node was in OPEN

X9	X2	X3
X8	X1	X5
X7	X6	X7

Set c(X) costs for obstacles prohibitively high

D* Algorithm: Example

- Read example from
  - http://www.cs.cmu.edu/~./motionplanning/lecture/AppH-astar-dstar_howie.pdf
  - 38 - 59