Rapidly-exploring **R**andom **T**rees **(RRTs)** for Efficient Motion Planning

RSS Lecture 10

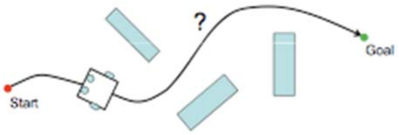
Monday, 10 March 2014

Prof. Seth Teller

(Thanks to Sertac Karaman for animations)

Recap of Previous Lectures:

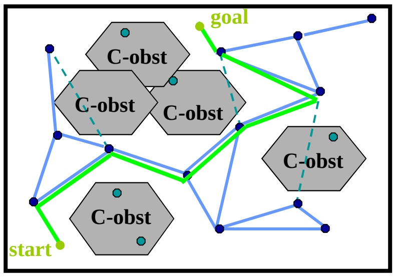
- Recall the motion planning problem:



- We discussed:
  - Cell decomposition
  - Guided search using A*
  - Potential fields
  - Configuration space
  - Probabilistic Road Maps

Recap: PRMs [Kavraki et al. 1996]

C-space



Roadmap Construction (Pre-processing)

1. Randomly generate robot configurations (nodes)
   - Discard invalid nodes (how?)
2. Connect pairs of nodes to form **roadmap edges**
   - Use simple, deterministic *local planner*
   - Discard invalid edges (how?)

Plan Generation (Query processing)

1. Link *start* and *goal* poses into roadmap
2. Find path from *start* to *goal* within roadmap
3. Generate motion plan for each edge used

Primitives Required:

1. Method for sampling C-Space points
2. Method for “validating” C-space points and edges

Today’s Focus

- Retain assumptions:
  - Perfect map
  - Perfect localization
- Incorporate additional elements:
  - Unstable **dynamics**
    - Cars, helicopters, humanoids, …
    - Agile maneuvering aircraft
  - High-dimensional configuration space
  - Real-time and online
    - Trajectory generation & execution

Motion Planning Revisited

- **Given:**
  - Robot's dynamics
  - A map of the environment
    (perfect information, but discovered online)
  - Robot's pose in the map
  - A goal pose in the map
- **Find a sequence of**
  - Actuation commands
    (such as steer, gas/brake, transmission)
  - In real time (requires efficient algorithms)

… that drive system to the goal pose

- Problem is essential in almost all robotics applications irrespective of size, type of actuation, sensor suite, task domain, etc.

Practical Challenges

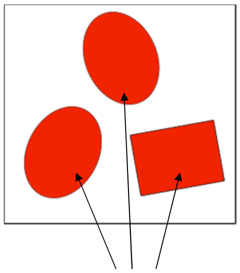
- **Safety:** do not collide with anything;
  ensure that system is stable; etc.
- **Computational effectiveness:** problem is (provably) computationally very challenging
- **Optimize:** fuel, efficiency etc. (alternative framing: not a gross waste of resources)
- **Social acceptability (in human-occupied environments):** motion should seem natural; robot's presence should not be rejected by humans

Different Approaches

- Algebraic Planners
- Cell Decomposition
- Potential Fields
- Sampling-Based Methods

Motion Planning Approaches

- **Algebraic Planners**
  - Explicit (algebraic) representation of obstacles
  - Use algebraic expressions (of visibility computations, projections etc.) to find the path
  - Complete (finds a solution if one exists, otherwise reports failure)
  - Computationally very intensive – impractical

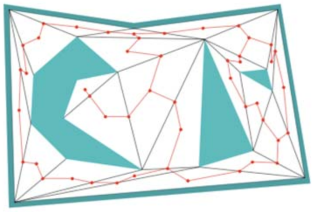


1. Represent with polynomial inequalities
2. Transform inequalities to c-space
3. Solve inequalities in c-space
   to check feasibility and find a plan

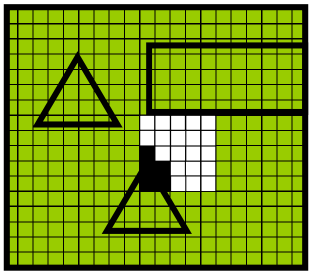
- **Cell Decomposition**
- **Potential Fields**.
- **Sampling-Based Methods**

Motion Planning Approaches

- Algebraic Planners
- **Cell Decomposition**
  - Analytic methods don't scale well with dimension (too many cells in high *d*)
  - Gridding methods are only "resolution complete" (i.e., will find a solution only if the grid resolution is fine enough, and if enough grid cells are inspected)
- Potential Fields.
- Sampling-Based Methods



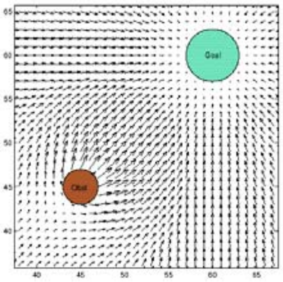
Analytic subdivision

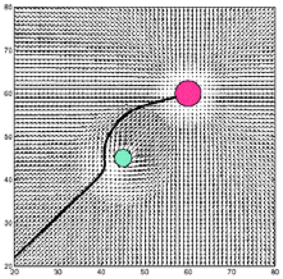


Gridded subdivision

Motion Planning Approaches

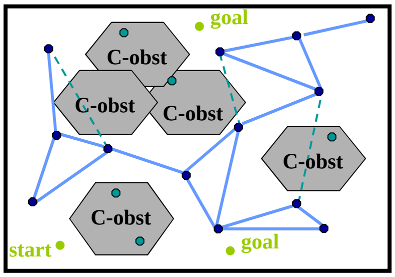
- Algebraic Planners
- Cell Decomposition
- **Potential Fields**
  - No completeness guarantee (can get stuck in local minima)
  - Of intermediate efficiency; don't handle dynamic environments well
- Sampling-Based Methods



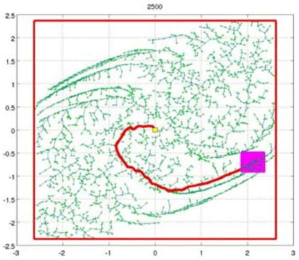


Motion Planning Approaches

- Algebraic Planners
- Cell Decomposition
- Potential Fields

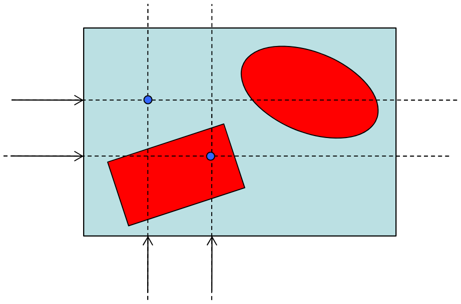


- **Sampling-Based Methods**
  - (Randomly) construct a set of feasible (that is, collision-free) trajectories
  - “Probabilistically complete” (if run long enough, very likely to find a solution)
  - Quite efficient; methods scale well with increasing dimension, # of obstacles



Sampling Strategies

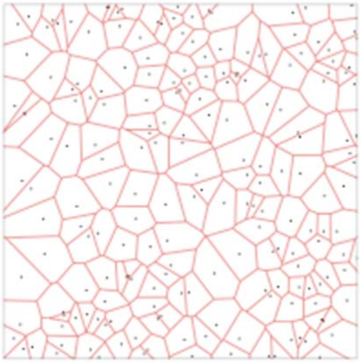
- How can we draw random samples from within c-space?
- Normalize all c-space dimensions to lie inside [0..1]
- Then, simple idea:
  1. Generate a random point in *d*-dimensional space
     - Independently generate *d* random numbers between 0 and 1
     - Aggregate all *d* numbers into a single point in c-space
  2. Check whether sample point (i.e., robot pose) lies within any obstacle



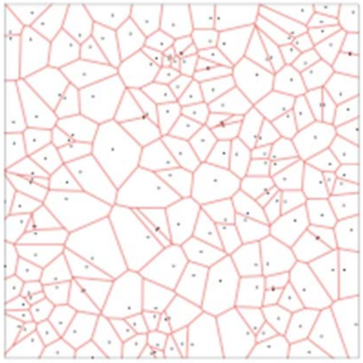
Example Sample Sets

Uniform sampling:

From a given axis, sample each coordinate with equal likelihood



(200 random samples)



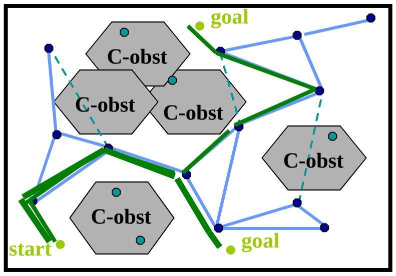
(200 random samples)

Observe:

Significant local variation, but sample *sets* are globally consistent
(Later, we'll see that this yields consistent performance across runs)

Sampling-based Motion Planning

- **Basic idea:**
  - Randomly sample *n* points from c-space
  - Connect them to each other (if no collision with obstacles)
  - Recall the two primitive procedures:
    - Check if a point is in the obstacle-free space
    - Check if a trajectory lies in the obstacle-free space

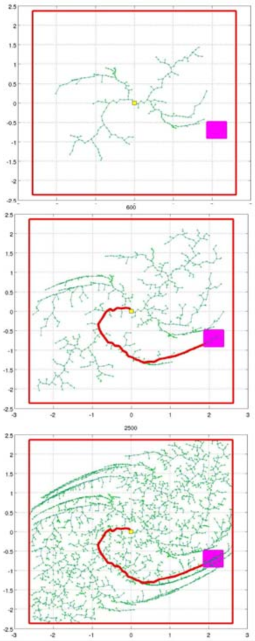


This is the **P**robabilistic **R**oad **M**ap (**PRM**) algorithm

PRM is a **multiple-query** algorithm (can reuse the roadmap for many queries)

Incremental Sampling-based Motion Planning

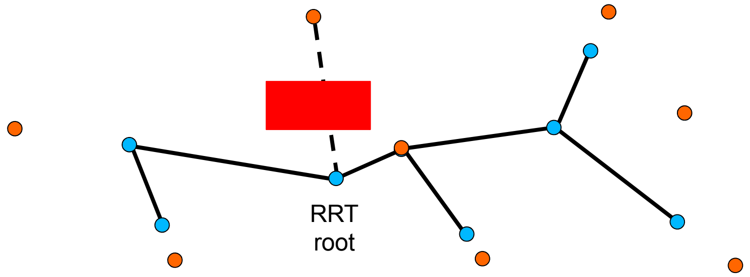
- Sometimes building a roadmap *a priori* might be inefficient (or even impractical)
  - Assumes that all regions of c-space will be utilized during actual motions
- Building a roadmap requires global knowledge
  - But in real settings, obstacles are not known *a priori*; rather, they are discovered *online*
- **We desire an *incremental* method:**
  - Generate motion plans for a single start, goal pose
  - Expending more CPU yields better motion plans
- **The Rapidly-exploring Random Tree (RRT) algorithm meets these requirements**



RRT Data Structure, Algorithm

T = (nodes V, edges E): tree structure

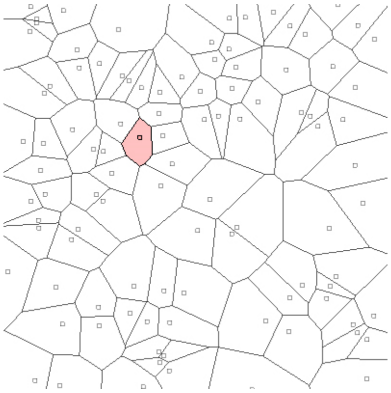
– **Initialized as single root vertex (the robot's current pose)**



```
$x \leftarrow$ Sample()                              // Sample a node x from c-space
$v \leftarrow$ Nearest($T, x$)                         // Find nearest node v in tree
$v' \leftarrow$ Extend($v, x$)                          // Extend nearest node toward sample
If (ObstacleFree($v, v'$)) then                     // If extension is collision-free
    $V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$   // Add new node and edge to tree
```

Digression: Voronoi Diagrams

Given *n sites* in *d* dimensions, the *Voronoi diagram* of the sites is a *partition* of $\boldsymbol{R}^d$ into regions, one region per site, such that all points in the interior of each region lie closer to that region's site than to any other site

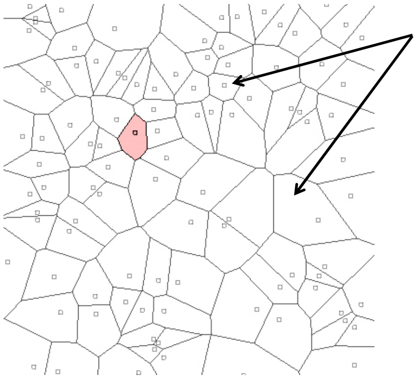


(AKA Dirichlet tesselations, Wigner-Seitz regions, Thiessen polygons, Brillouin zones, …)

Rapidly-exploring Random Trees:

Clearly random! Why rapidly-exploring?

- RRTs tend to grow toward *unexplored* portions of the state-space
  - Unexplored regions are (in some sense) more likely to be sampled
  - This is called a ***Voronoi bias***



For an RRT at a given iteration, some nodes are associated with large Voronoi regions of c-space, some with smaller Voronoi regions

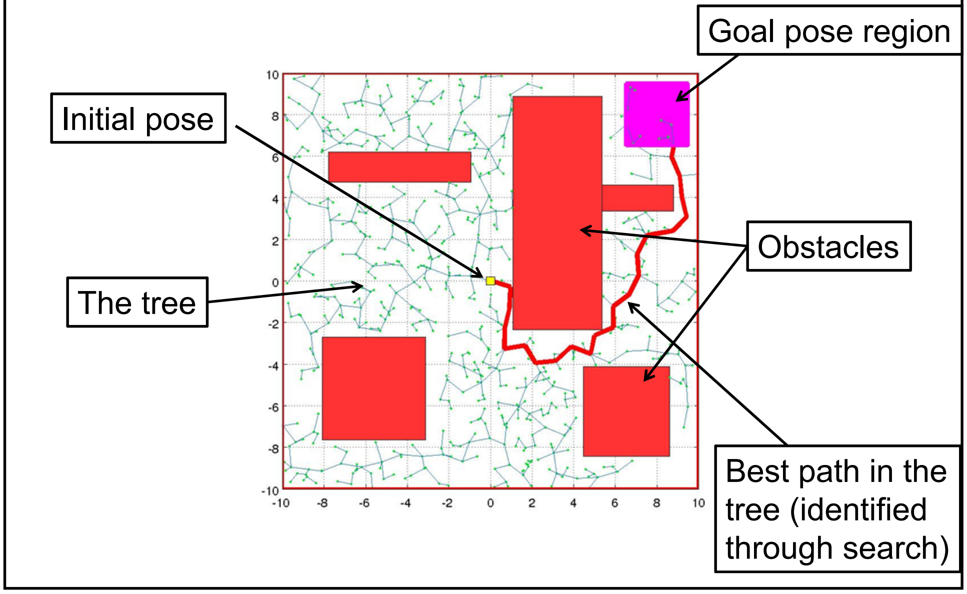
The **unexplored areas** of c-space tend to coincide with the larger Voronoi regions

(Uniform) samples will tend to fall into relatively larger Voronoi regions

Thus unexplored regions will tend to shrink!

Main advantage of RRT: Samples "**grow**" tree toward unexplored regions of c-space!

Rapidly-exploring Random Trees in simulation



Goal pose region

Initial pose

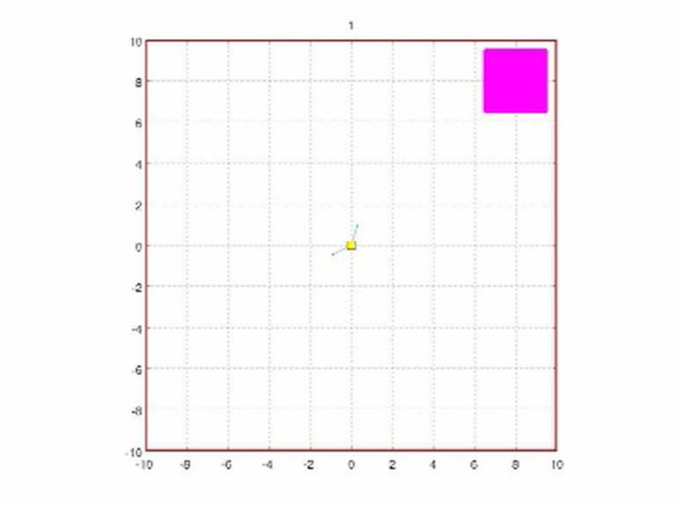
Obstacles

The tree

Best path in the tree (identified through search)

Rapidly-exploring Random Trees in simulation

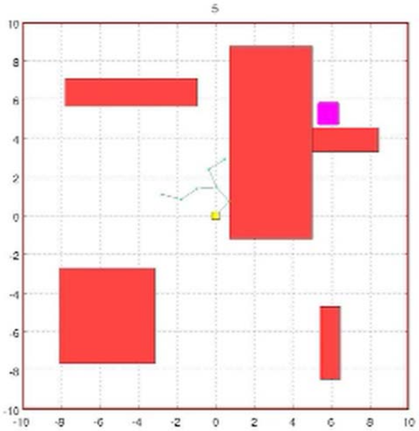
Movie shows the RRT exploring empty c-space



Goal pose region

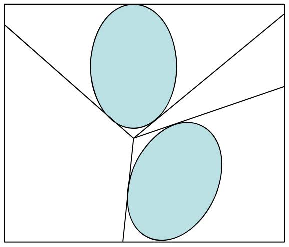
Rapidly-exploring Random Trees in simulation

Exploration amid obstacles, narrow passages:



Performance of Sampling-based Methods

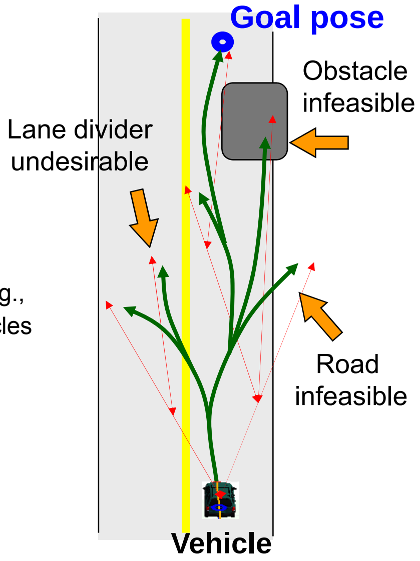
- Why do the PRM and RRT methods work so well?
- **Probabilistic Completeness:**
  - The probability that the RRT will find a path approaches 1 as the number of samples increases — **if** a feasible path exists.
  - The approach rate is exponential — **if** the environment has good "**visibility**'' properties
- **Є-goodness:**
  - A **point** is **ϵ-good** if it "sees" at least an **Є** fraction of the obstacle-free space
  - An **environment** is **ϵ-good** if all freespace points in it are **ϵ-good**



Good performance of PRMs and RRTs has been tied to the fact that, in practice, most applications feature environments with good visibility guarantees ***(Latombe et al., IJRR '06)***.

Example: Unmanned Driving

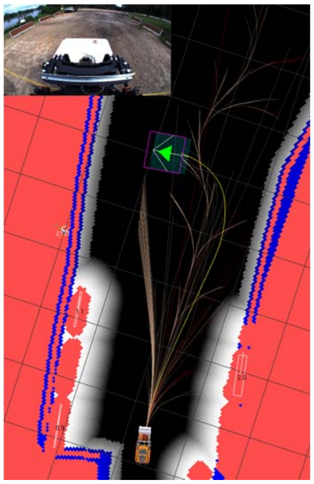
- **Tree of trajectories** is grown by sampling configurations randomly
- **Rapidly explores** several configurations that the robot can reach.
  - Many test trajectories generated (tens of thousands per second)
- **Safety** of any trajectory is “guaranteed”…
  - … as of instantaneous world state at the time of trajectory generation
- **Choose best** one that reaches the goal, e.g.,
  - Maximizes minimum distance to obstacles
  - Minimizes total path length
- **Supports dynamic replanning**; if current trajectory becomes infeasible:
  - Choose another one that is feasible
  - If none remain, then E-stop



Real-world Implementation

A few details:

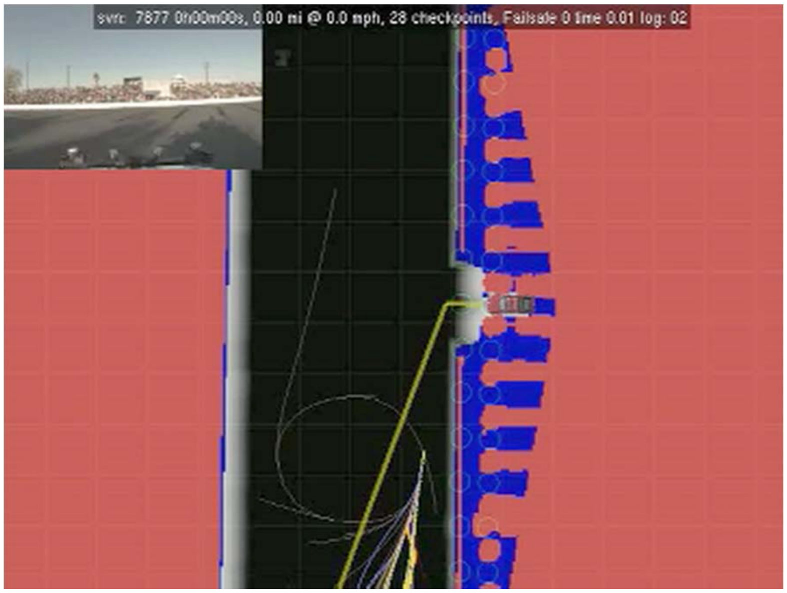
- CPU limitations and sampling method
- Dynamical feasibility constraints
- Grid map with local obstacle awareness
- Stop nodes for safety



Legend for images, videos you'll see next:

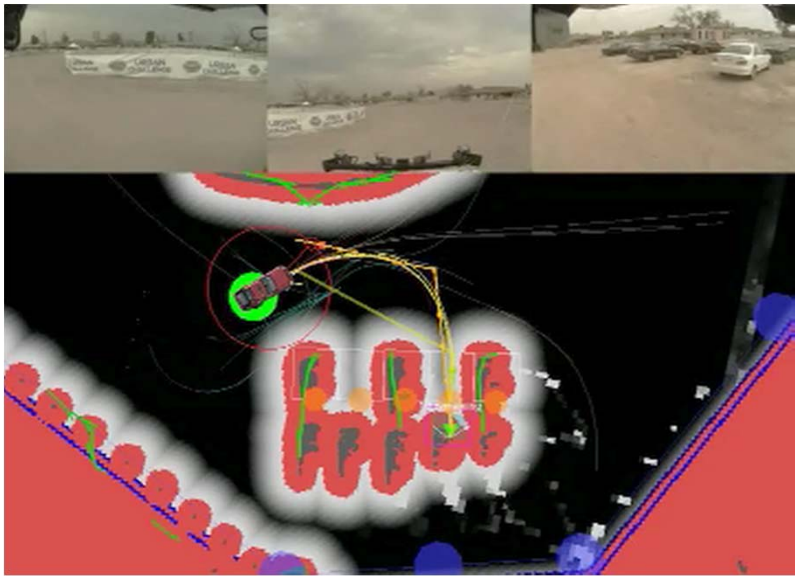
- Instantaneous vehicle pose
- ▲ Goal pose
- Obstacle
- High-cost regions
- Reaching, low cost
- Reaching, high cost
- Non-reaching

RRT at work: Urban Challenge

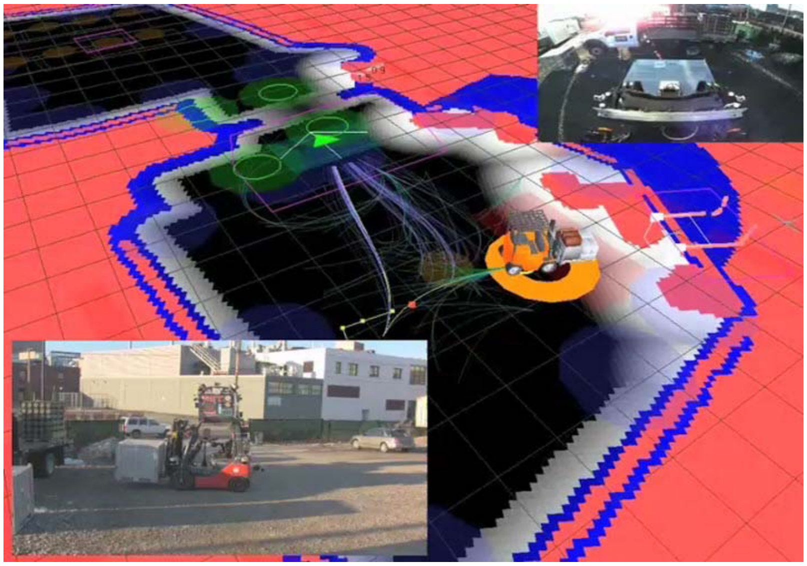


svrc: 7877 0h00m00s, 0.00 mi @ 0.0 mph, 28 checkpoints, Failsafe 0 time 0.01 log: 02

Successful Parking Maneuver



RRT at work: Autonomous Forklift



Summary

- The Rapidly-exploring Random Tree (RRT) algorithm
- Discussed challenges for motion planning methods in real-world applications
- Intuition behind good performance of sampling-based methods
- Two applications:
  - Urban Challenge vehicle, Agile Robotics forklift