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

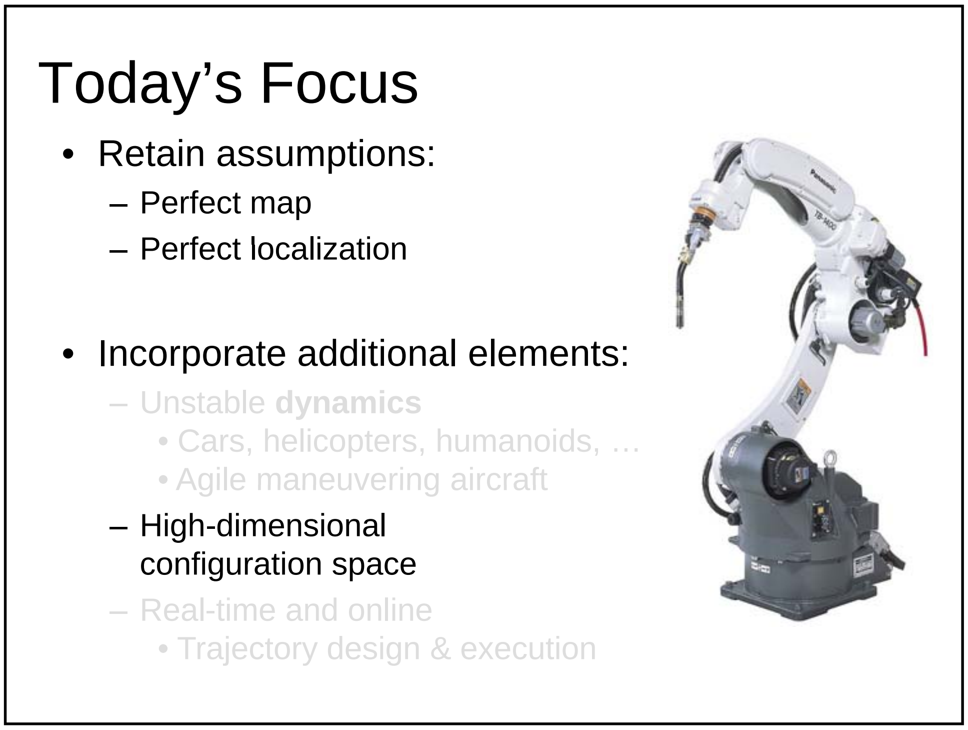
RSS Lecture #11

Monday, 15 March 2010

Guest Lecturer: Sertac Karaman

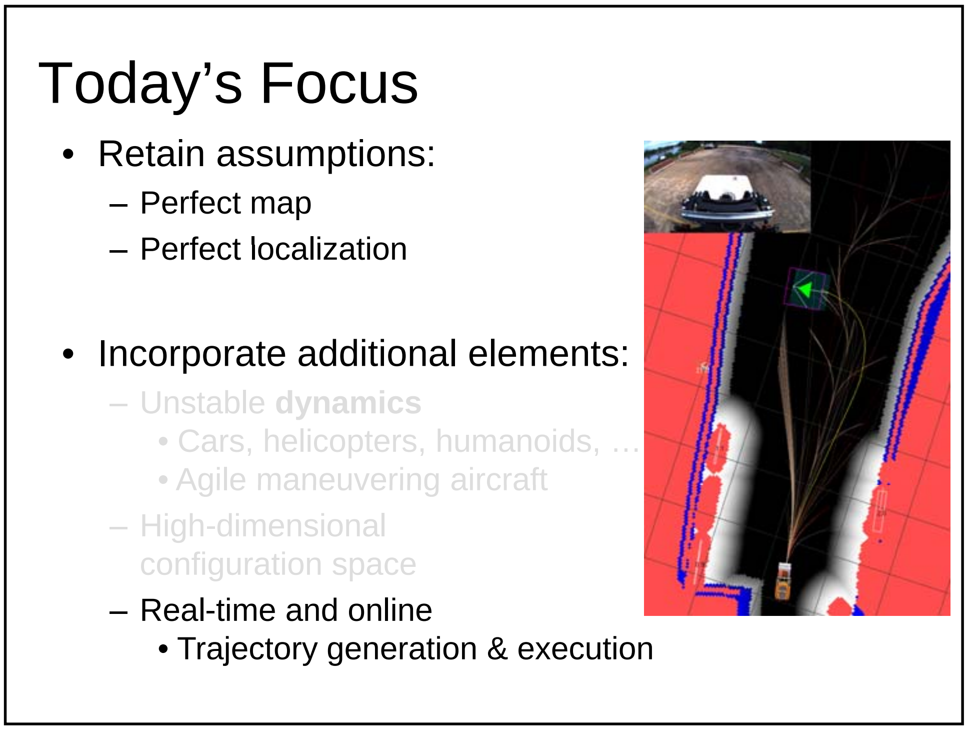
Today’s Focus

- 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 design & execution



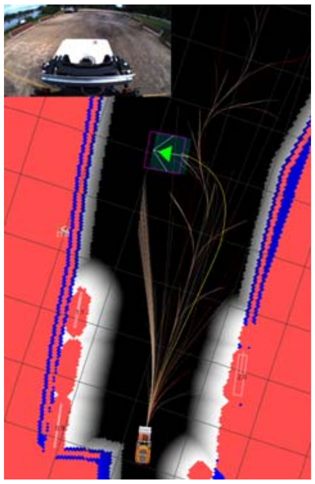
Today’s Focus

- 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 design & execution



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

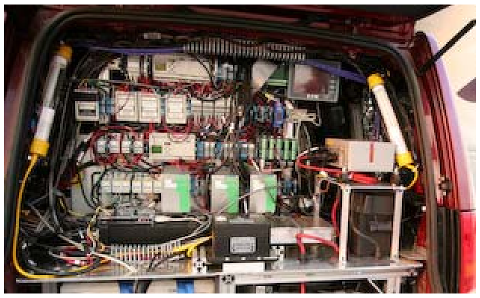
- Problem is essential in almost all robotics applications irrespective of size, type of actuation, sensors, application, etc.

Motion Planning Revisited

- **Challenges in (most) practical applications:**
  - **Safety:** do not collide with anything;
    ensure that system is stable; etc.
  - **Computational effectiveness:** problem is (provably) computationally very challenging
  - **Optimize:** fuel, efficiency etc.
  - **Social challenges (in human-occupied environments):** motion should seem natural; robot should be accepted by humans

Motion Planning Revisited

- **Challenges in (most) practical applications:**
  - **Safety:** do not collide with anything; ensure that system is stable; etc.
  - **Computational effectiveness:** problem is (provably) computationally very challenging
  - **Optimize:** fuel, efficiency etc.
  - **Social challenges (in human-occupied environments):** motion should seem natural; robot should be accepted by humans

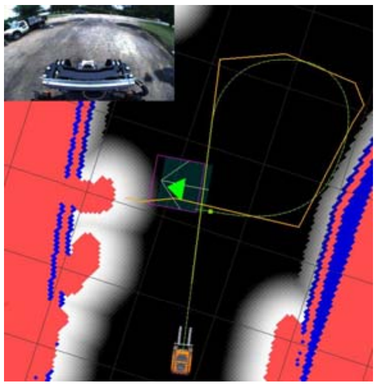


Motion Planning Revisited

- **Challenges in (most) practical applications:**
  - **Safety:** do not collide with anything; ensure that system is stable; etc.
  - **Computational effectiveness:** problem is (provably) computationally very challenging
  - **Optimize:** fuel, efficiency etc.
  - **Social challenges (in human-occupied environments):** motion should seem natural; robot should be accepted by humans

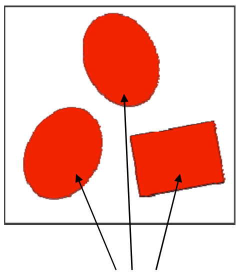
Motion Planning Revisited

- **Challenges in (most) practical applications:**
  - **Safety:** do not collide with anything; ensure that system is stable; etc.
  - **Computational effectiveness:** problem is (provably) computa-tionally very challenging
  - **Optimize:** fuel, efficiency etc.
  - **Social challenges (in human-occupied environments):** motion should seem natural; robot should be accepted by humans



Motion Planning Approaches

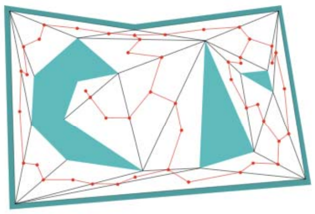
- **Algebraic Planners**
  - Explicit (algebraic) representation of obstacles
  - Use complicated algebra (several visibility computations/projections) to find the path.
  - Complete (finds a solution if one exists, otherwise reports failure)
  - Computationally very intensive – impractical
- **Cell Decomposition**
- **Potential Fields.**
- **Sampling-Based Methods**

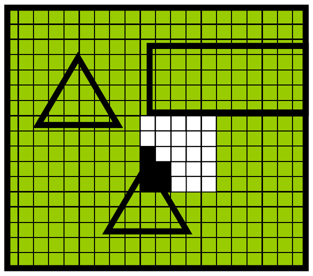


1. Represent with polynomial inequalities
2. Transform inequalities to c-space
3. Solve inequalities in c-space
   to check feasibility and find a plan

Motion Planning Approaches

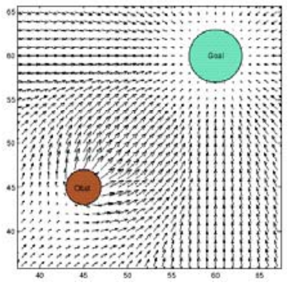
- Algebraic Planners
- **Cell Decomposition**
  - Analytic methods don’t scale well with dimension (too many cells in high *d*)
  - Gridding methods are only “resolution complete” (i.e., will find a solution only if the grid resolution is fine enough)
- Potential Fields.
- Sampling-Based Methods

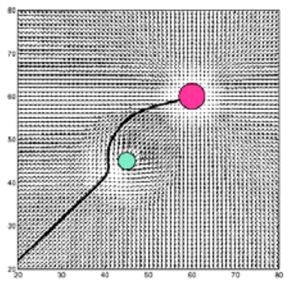

Analytic subdivision

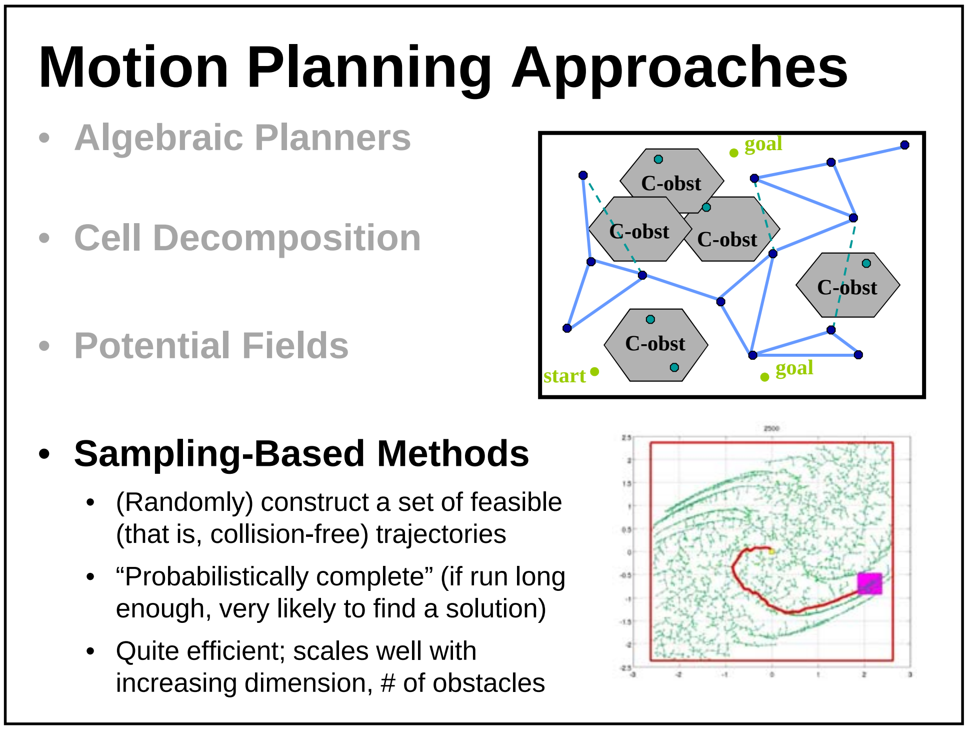

Gridded subdivision

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

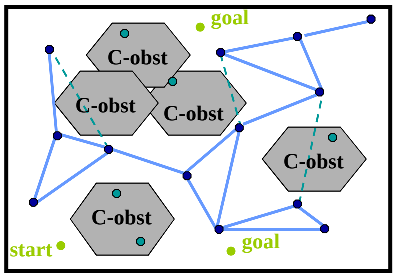




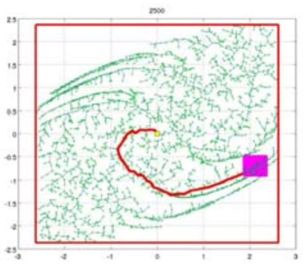


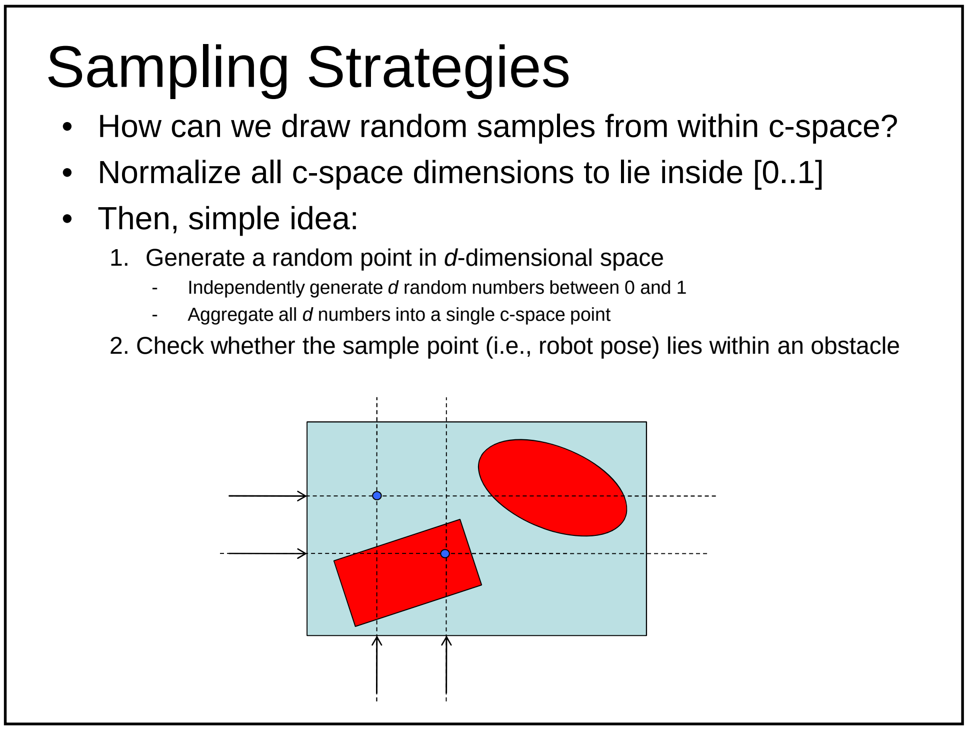
Motion Planning Approaches

- Algebraic Planners
- Cell Decomposition
- Potential Fields



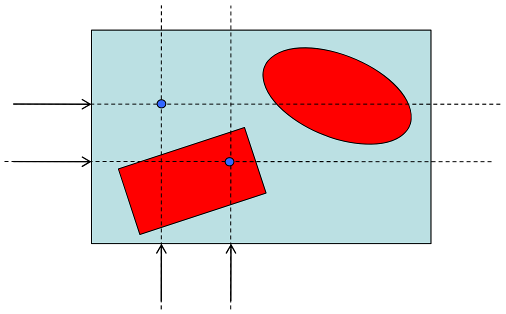
- **Sampling-Based Methods**
  - (Randomly) construct a set of feasible (that is, collision-free) trajectories
  - “Probabilistically complete” (if run long enough, very likely to find a solution)
  - Quite efficient; scales well with increasing dimension, # of obstacles





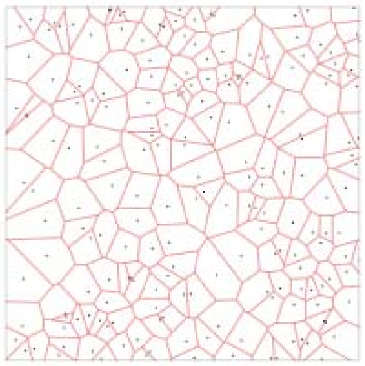
Sampling Strategies

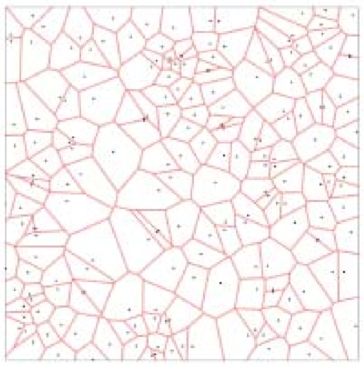
- How can we draw random samples from within c-space?
- Normalize all c-space dimensions to lie inside [0..1]
- Then, simple idea:
  1. Generate a random point in *d*-dimensional space
     - Independently generate *d* random numbers between 0 and 1
     - Aggregate all *d* numbers into a single c-space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



Example Sample Sets

Uniform sampling:
From a given axis, sample each coordinate with equal likelihood

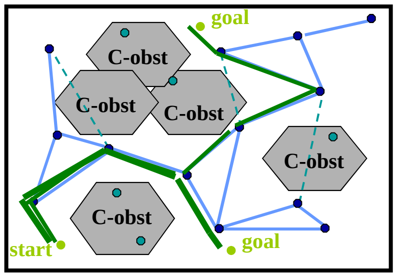

(200 random samples)


(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

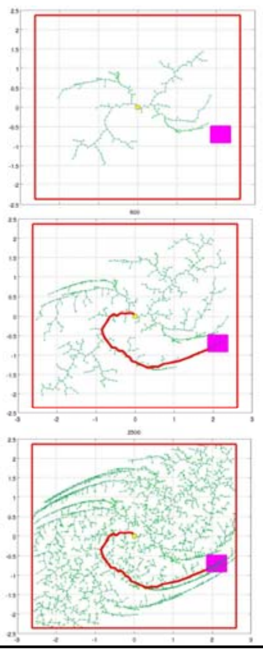


This is the **P**robabilistic **R**oad **M**ap (**PRM**) algorithm

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

Incremental Sampling-based Motion Planning

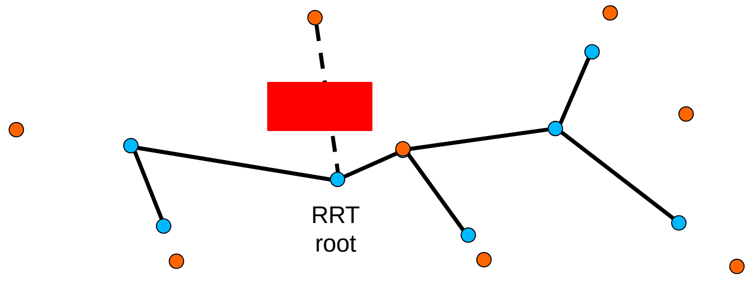
- Sometimes building a roadmap *a priori* might be inefficient (or even impractical)
  - Assumes that all regions of c-space will be utilized during actual motions
- Building a roadmap requires global knowledge
  - Obstacles not known *a priori*, but discovered online
- **We desire an *incremental* method:**
  - 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

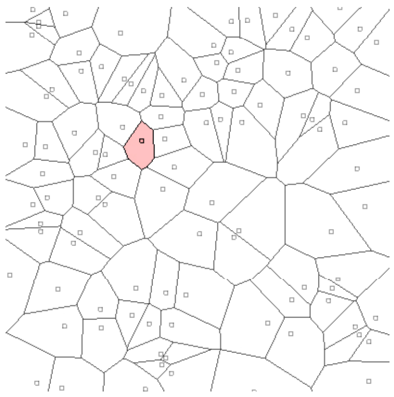
– **Initialized as single root vertex (the robot's current pose)**



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

Digression: Voronoi Diagrams

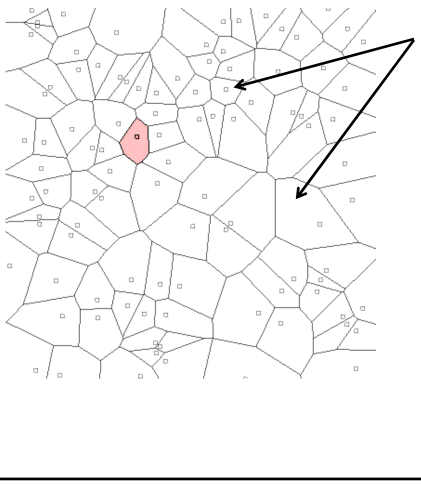
Given *n sites* in *d* dimensions, the *Voronoi diagram* of the sites is a partition of $\boldsymbol{R}^d$ into regions, one region per site, such that all points in the interior of each region lie closer to that region's site than to any other site



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

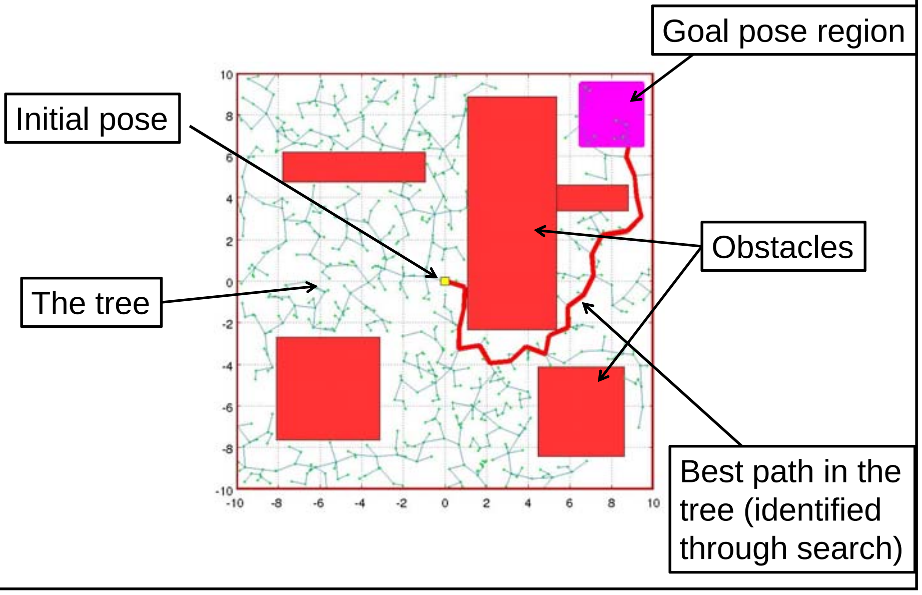
The **unexplored areas** of c-space tend to coincide with the larger Voronoi regions

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

Thus unexplored regions will tend to shrink!

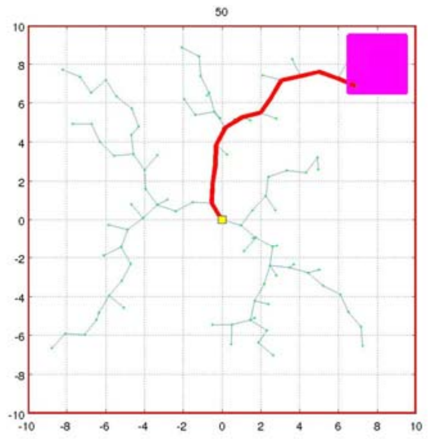
Main advantage of RRT: Samples "**grow**" tree toward unexplored regions of c-space!

Rapidly-exploring Random Trees in simulation



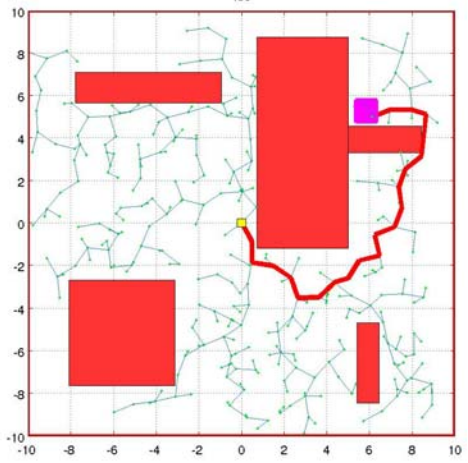
Rapidly-exploring Random Trees in simulation

Movie shows the RRT exploring empty c-space



Rapidly-exploring Random Trees in simulation

Let's look at initial exploration phase more carefully:



(Video rrt_with_obs_beginning)

Performance of Sampling-based Methods

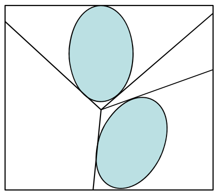
- What is the "*magic*" behind PRMs and RRTs?

- **Probabilistic Completeness:**
  - The probability that the RRT will find a path approaches 1 as the number of samples increases — **if** a feasible path exists.
  - If the environment has good "**visibility**'' properties, the approach rate is exponential

ϵ goodness

A **point** is ϵ-**good** if it "sees" at least an ϵ fraction of the obstacle-free space

A **environment** is ϵ-**good** if all the points in it are ϵ-good

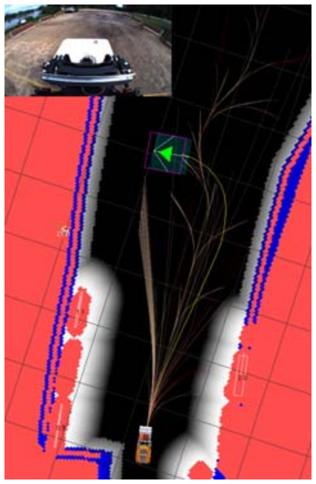


Recently, good performance of RRTs and PRMs has been tied to the fact that most practical applications feature environments with good visibility guarantees ***(Latombe et al., IJRR'06)***.

Real-world Implementation

A few details:

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

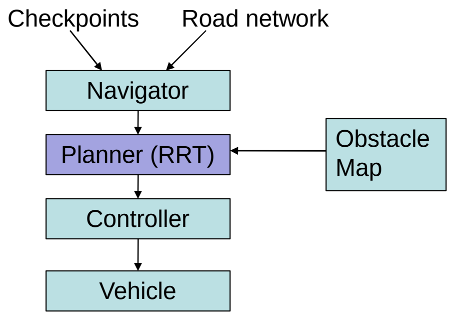


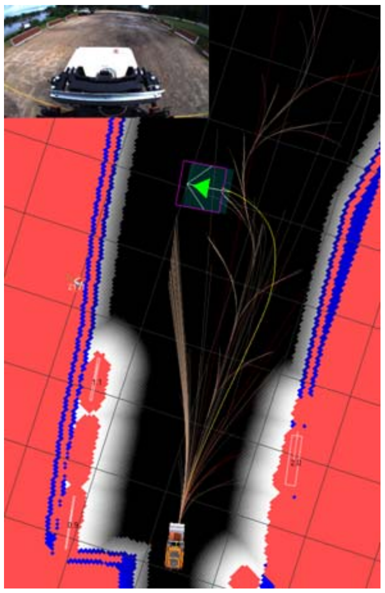
Legend for images, videos you'll see next:

▮	Instantaneous vehicle pose	🟥	Obstacle	—	Reaching, low cost
▲	Goal pose	🟦 ▨	High-cost regions	—	Reaching, high cost
				—	Non-reaching

System Architecture

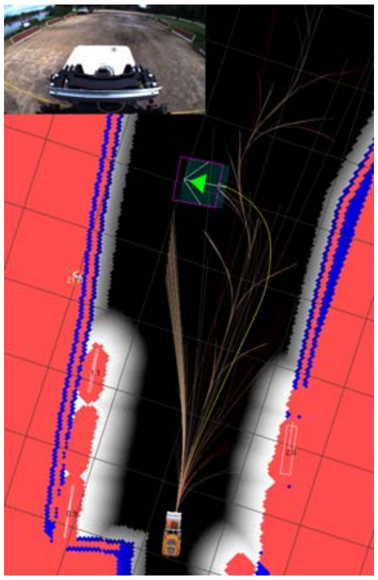
- System has 40 CPUs, 70+ processes
- Processes communicate with each other *only* via message passing
- The core planning and control processes and some components that they are directly connected to





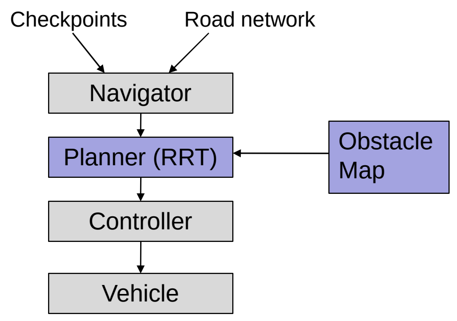
Dynamical Feasibility

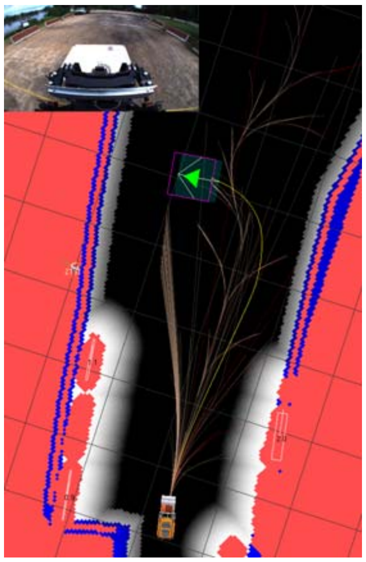
- Vehicle (car, forklift) is modeled as a dynamical system with 5 states:
  - X
  - Y
  - Theta (heading)
  - Speed
  - Sideslip – not used for planning
- A closed-loop controller is designed to stabilize the system in
  - Position/orientation (X, Y, Theta)
  - Speed control
- The RRT samples **controller setpoints** (which a lower-level controller then tracks)
- This process ensures dynamical feasibility (i.e. generates only trajectories that can indeed be executed by the vehicle)



Grid Map and Moving Obstacles

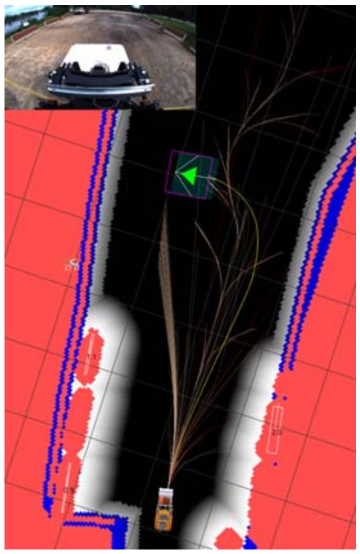
- The RRT uses the **grid map** data structure, which provides one efficient query:
  - Check whether a specified hypothetical trajectory collides with any (dilated) obstacle
- The perception subsystem also detects moving obstacles and predicts their future trajectories





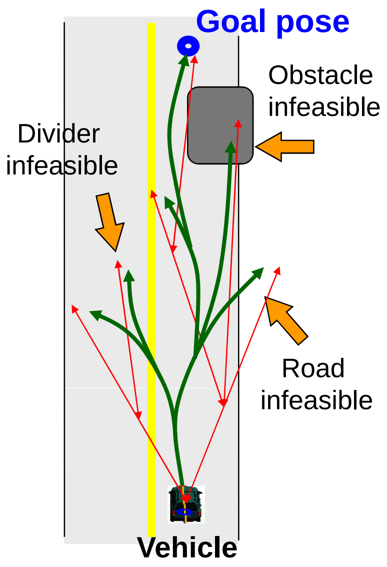
Stop Nodes

- Remember the **safety** requirement.
- All leaf nodes constrained to have speed=0
- The RRT attempts to maintain a path from every internal node to some leaf node
- If, due to some newly discovered obstacle, no such trajectory exists in the tree, the robot “E-stops” (i.e., slams on the brakes)
- **Safety** is perhaps the most important requirement in real-time motion planning for robotic vehicles.

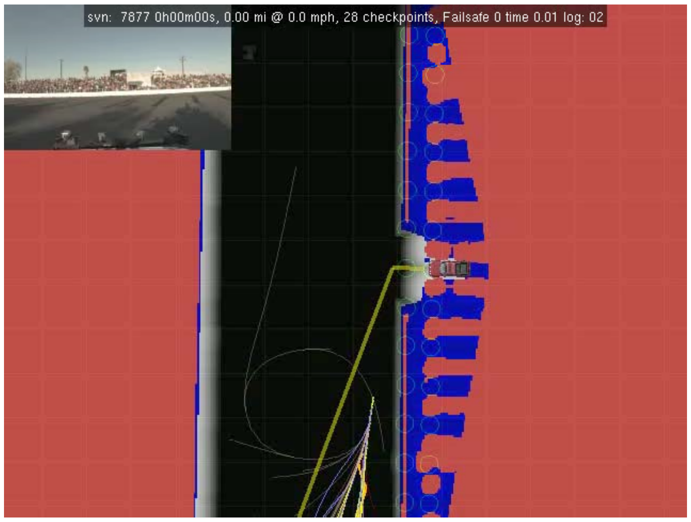


RRT – Real-World Example

- **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



RRT at work: Urban Challenge


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

Summary

- Studied the Rapidly-exploring Random Tree (RRT) algorithm
- Discussed challenges for motion planning methods in real-world applications
- Discussed *magic* behind sampling-based methods
- Looked at two applications:
  – Urban Challenge vehicle, Agile Robotics forklift