Trajectory Design for Space Systems

EN.530.626 (Fall 2025)

Lecture 14

Instructor: Prof. Abhishek Cauligi
Course Assistant 1: Arnab Chatterjee
Course Assistant 2: Mark Gonzales

Nomenclature

- Optimal control
- Trajectory optimization
- Motion planning

Nomenclature

Optimal control vs. trajectory optimization

- Optimal control: solving for the control inputs for a dynamical system that minimizes some cost function while satisfying system constraints

Nomenclature

Optimal control vs. trajectory optimization

- Optimal control: solving for the control inputs for a dynamical system that minimizes some cost function while satisfying system constraints
- Trajectory optimization: solving for the *trajectory of* control inputs for a dynamical system that minimizes some cost function while satisfying system constraints

Nomenclature

Motion planning

- Computing a sequence of actions that allows for a robot to navigate from an initial condition x_{init} to some terminal region $\mathcal{X}_{\text{goal}}$ while avoiding collisions with obstacles and violating system constraints

Suggested readings:

1. *Planning Algorithms*, Steven M. LaValle (2006), Cambridge University Press.

Nomenclature

Motion planning

Kinodynamic Planning

Edward Schmerling and Marco Pavone

1 Synonyms

Trajectory planning, planning under differential constraints.

2 Definition

Kinodynamic planning concerns the task of driving a robot from an initial state to a goal region while avoiding obstacles and obeying kinematic and dynamic—in short, kinodynamic—constraints dictating the relationship between a robot's controls and its motion.

"While it is true that motion planning may be regarded as a special case of optimal control where the main distinguishing element is a (typically highly non-convex) collision-avoidance constraint on the robot's state trajectory, the global combinatorial search necessitated by this non-convex constraint is truly the hallmark of a motion planning problem."

Motion planning
Configuration space

- C-space: captures all rigid-body transformations that can be applied

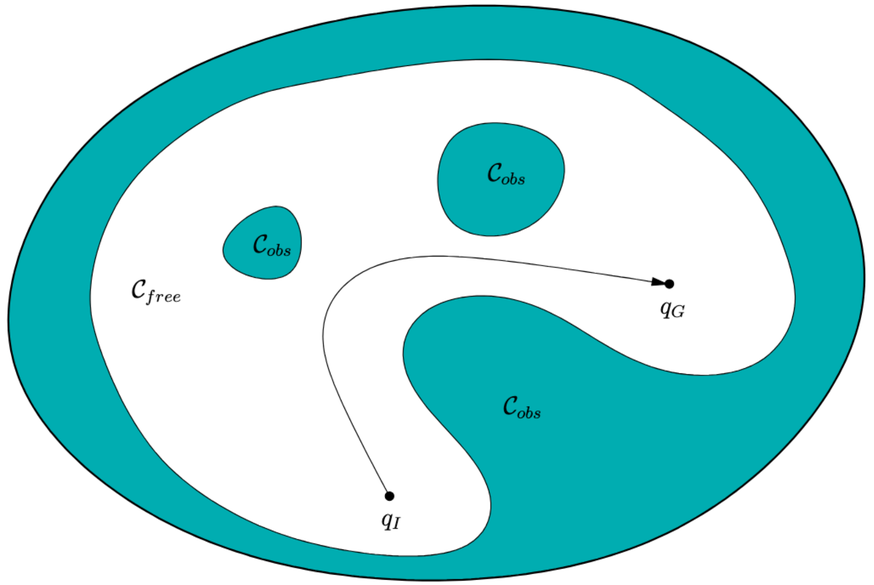


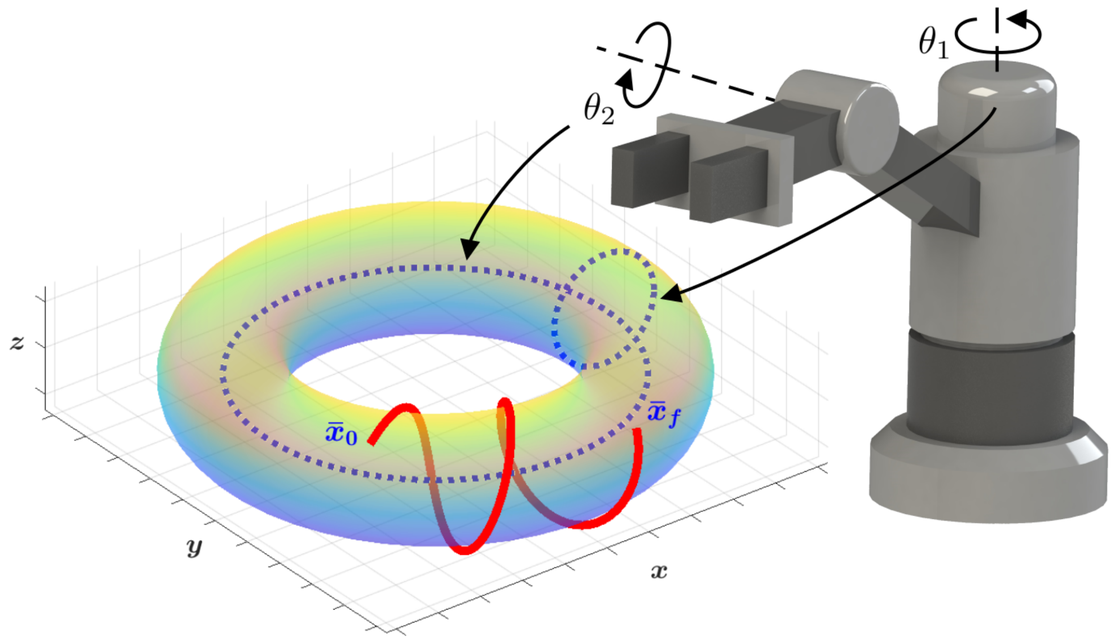
Figure 4.11: The basic motion planning problem is conceptually very simple using C-space ideas. The task is to find a path from q_I to q_G in $\mathcal{C}_{free}$. The entire blob represents $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$.

Planning Algorithms, Steven M. LaValle (2006)

Motion planning

Configuration space

- C-space: captures all rigid-body transformations that can be applied



Motion planning
Ingredients

- Determining the vertices
  - Sampling configurations
  - Sampling controls
- Connecting the edges
  - Two-point boundary value problem (2PBVP)
  - Collision checker

Motion planning

Optimal motion planning

$$\min_{\mathbf{x}(t),\mathbf{u}(t),t_f} \int_{t=t_0}^{t=t_f} J(\mathbf{x}(t),\mathbf{u}(t))\mathrm{d}t$$

subject to:

$$x(t_0) = x_{\text{init}}$$

$$x(t_f) \in \mathcal{X}_{\text{goal}}$$

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t),\mathbf{u}(t),t) \quad \forall t \in [t_0, t_f]$$

$$\mathbf{x}(t) \in \mathcal{X}_{\text{free}} \quad \forall t \in [t_0, t_f]$$

Motion planning

Optimal motion planning

$$\min_{\mathbf{x}(t),\mathbf{u}(t),t_f} \int_{t=t_0}^{t=t_f} J(\mathbf{x}(t),\mathbf{u}(t))\mathrm{d}t$$

Cost functional $J : \mathcal{X} \times \mathcal{U} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$

subject to:

$$x(t_0) = x_{\text{init}}$$

$$x(t_f) \in \mathcal{X}_{\text{goal}}$$

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t) \quad \forall t \in [t_0, t_f]$$

$$\mathbf{x}(t) \in \mathcal{X}_{\text{free}} \quad \forall t \in [t_0, t_f]$$

Motion planning

Optimal motion planning

$$\min_{\mathbf{x}(t),\mathbf{u}(t),t_f} \int_{t=t_0}^{t=t_f} J(\mathbf{x}(t),\mathbf{u}(t))\mathrm{d}t$$

subject to:

$$x(t_0) = x_{\text{init}}$$

$$x(t_f) \in \mathcal{X}_{\text{goal}}$$

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t),\mathbf{u}(t),t) \quad \forall t \in [t_0, t_f]$$

$$\mathbf{x}(t) \in \mathcal{X}_{\text{free}} \quad \forall t \in [t_0, t_f]$$

Boundary conditions

Motion planning

Optimal motion planning

$$\min_{\mathbf{x}(t),\mathbf{u}(t),t_f} \int_{t=t_0}^{t=t_f} J(\mathbf{x}(t),\mathbf{u}(t))\mathrm{d}t$$

subject to:

$$x(t_0) = x_{\text{init}}$$

$$x(t_f) \in \mathcal{X}_{\text{goal}}$$

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t),\mathbf{u}(t),t) \quad \forall t \in [t_0, t_f]$$

$$\mathbf{x}(t) \in \mathcal{X}_{\text{free}} \quad \forall t \in [t_0, t_f]$$

Dynamics constraints

Motion planning

Optimal motion planning

$$\min_{\mathbf{x}(t),\mathbf{u}(t),t_f} \int_{t=t_0}^{t=t_f} J(\mathbf{x}(t),\mathbf{u}(t))\mathrm{d}t$$

subject to:

$$x(t_0) = x_{\text{init}}$$

$$x(t_f) \in \mathcal{X}_{\text{goal}}$$

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t),\mathbf{u}(t),t) \quad \forall t \in [t_0, t_f]$$

$$\mathbf{x}(t) \in \mathcal{X}_{\text{free}} \quad \forall t \in [t_0, t_f]$$

Collision avoidance

MIPs for motion planning

- Last time, we focused on mixed-integer programs (MIPs) to solve combinatorial problems in robot planning and control
- While MIPs provide the “optimal” solution, in many practical settings, a *feasible* solution suffices

MIPs for motion planning

JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS
Vol. 25, No. 4, July–August 2002

Spacecraft Trajectory Planning with Avoidance Constraints Using Mixed-Integer Linear Programming

Arthur Richards,* Tom Schouwenaars,† Jonathan P. How,‡ and Eric Feron§
Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

A method for finding fuel-optimal trajectories for spacecraft subjected to avoidance requirements is introduced. These include avoidance of collisions with obstacles or other vehicles and prevention of thruster plumes from one spacecraft impinging on another spacecraft. The necessary logical constraints for avoidance are appended to a fuel-optimizing linear program by including binary variables in the optimization. The resulting problem is a mixed-integer linear program (MILP) that can be solved using available software. The logical constraints can also be used to express the configuration requirements for maneuvers where only the final relative alignment of the vehicles is important and the assignment of spacecraft within the fleet is not specified. The collision avoidance, trajectory optimization, and fleet assignment problems can be combined into a single MILP to obtain the optimal solution for these maneuvers. The MILP problem formulation, including these various avoidance constraints, is presented, and then several examples of their application to spacecraft maneuvers, including reconfiguration of a satellite formation and close inspection of the International Space Station by a microsatellite, are shown. These examples clearly show that the trajectory design methods presented are particularly well suited to proposed formation flying missions that involve multiple vehicles operating in close proximity.

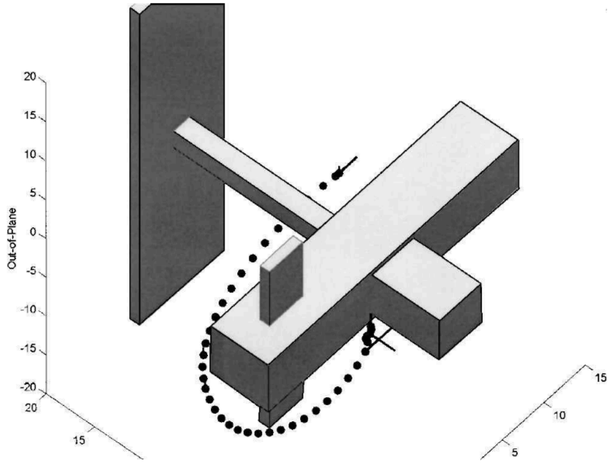


Fig. 5 ISS remote camera maneuver with plume impingement constraints added; start and end positions are the same as in Fig. 4.

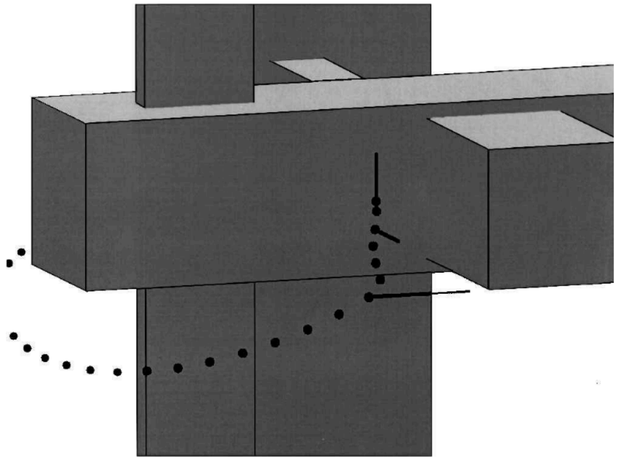


Fig. 6 Final stages of maneuver from Fig. 5, shown in close-up from below.

MIPs for motion planning

Pros

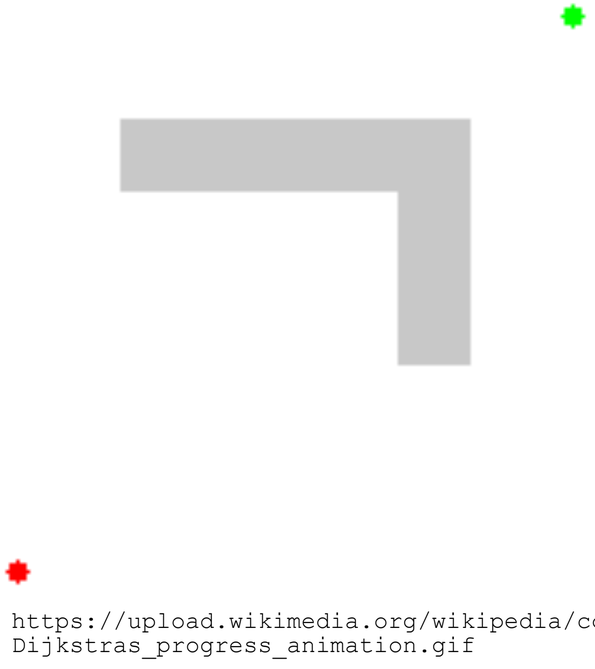
- For MICPs, finds globally optimal solution
- Optimization framework captures more challenging dynamical constraints

Cons

- Rewriting collision avoidance using integer constraints scales poorly
- Exponential complexity practically too slow for most systems

Grid-based planning

- Discretize the configuration space into a uniform grid
- Each grid cell is free or occupied
- Solve subsequent path planning problem using graph-search



https://upload.wikimedia.org/wikipedia/commons/2/23/Dijkstras_progress_animation.gif

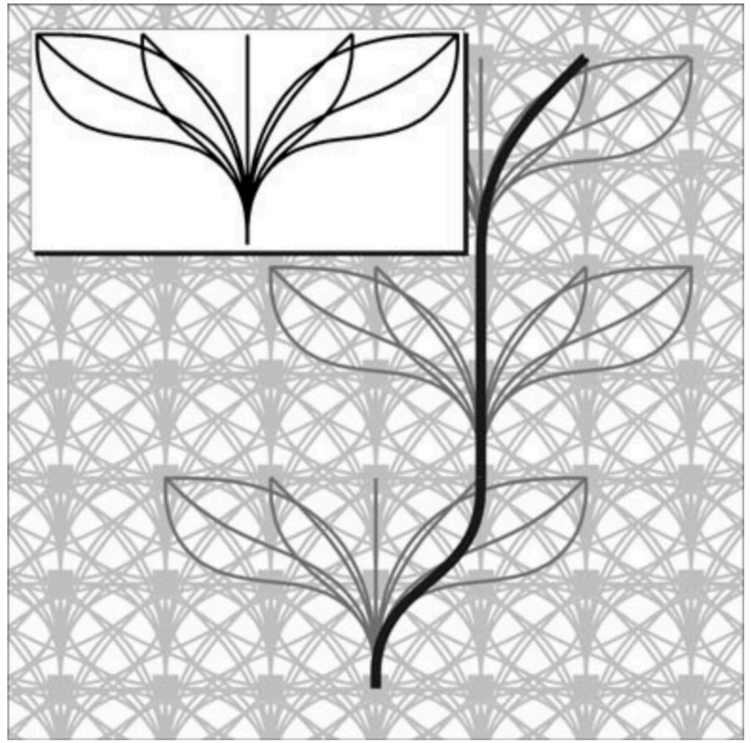
Grid-based planning

State lattices

Differentially Constrained Mobile Robot Motion Planning in State Lattices

Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
e-mail: mihail@cs.cmu.edu, rak@ri.cmu.edu, alonzo@ri.cmu.edu

Received 6 August 2008; accepted 4 January 2009



Grid-based planning

State lattices

Pros

- Simple and easy to use
- Many efficient and optimized implementations

Cons

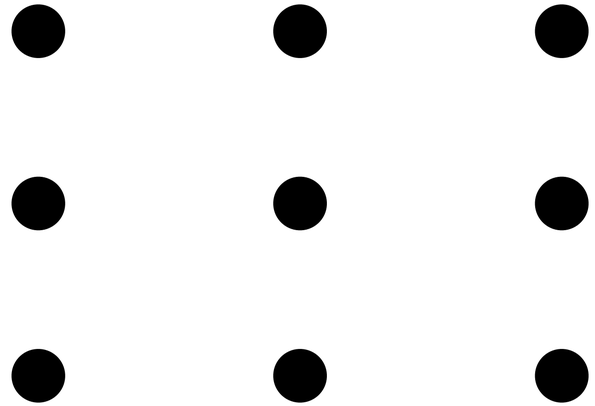
- Depends on the resolution (i.e., coarse grid —> no solution)
- Scales poorly with configuration space dimensionality

Grid-based planning

State lattices

Pros

- Simple and easy to use
- Many efficient and optimized implementations



Cons

- Depends on the resolution (i.e., coarse grid —> no solution)
- Scales poorly with configuration space dimensionality

Grid-based planning

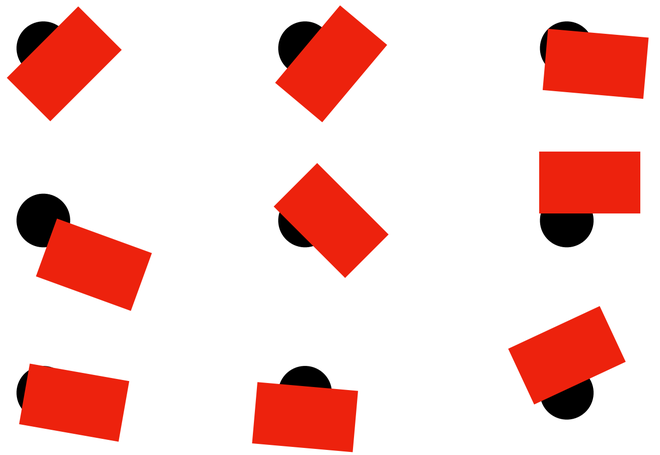
State lattices

Pros

- Simple and easy to use
- Many efficient and optimized implementations

Cons

- Depends on the resolution (i.e., coarse grid —> no solution)
- Scales poorly with configuration space dimensionality



Combinatorial planning

- Construct an exact geometric representation of the free configuration space

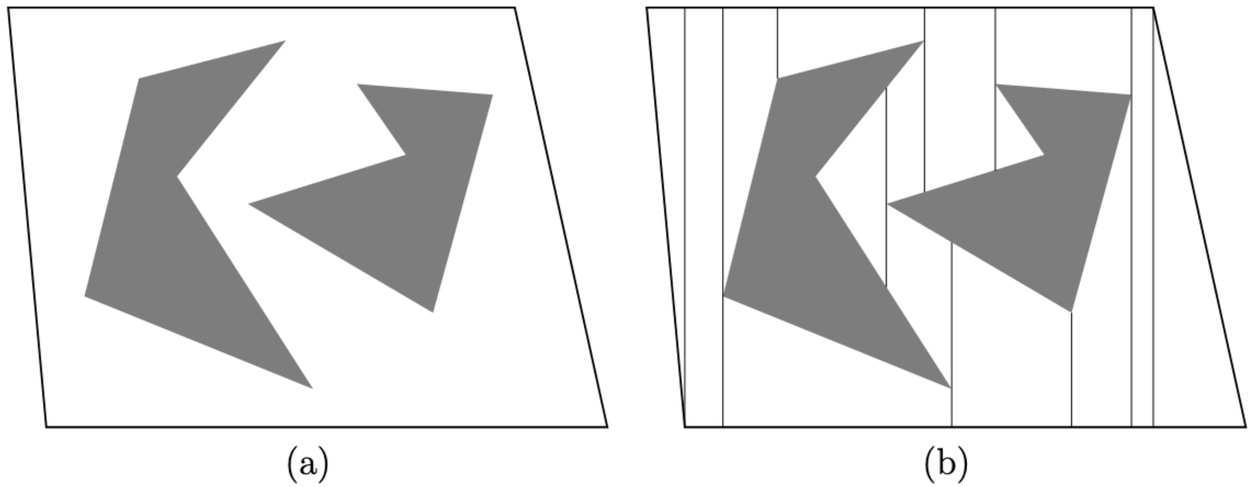


Figure 6.3: The vertical cell decomposition method uses the cells to construct a roadmap, which is searched to yield a solution to a query.

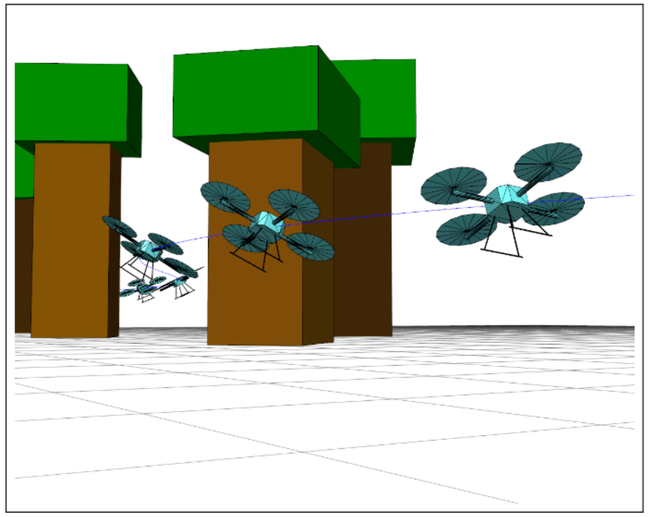
Planning Algorithms, Steven M. LaValle (2006)

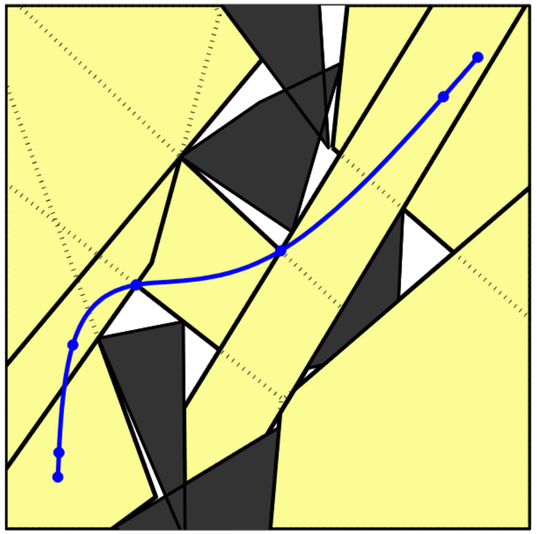
Combinatorial planning

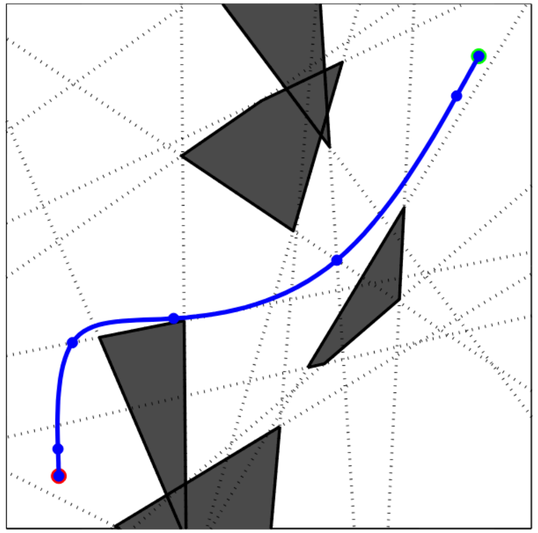
Efficient Mixed-Integer Planning for UAVs in Cluttered Environments

Robin Deits[1] and Russ Tedrake[2]

***Abstract*— We present a new approach to the design of smooth trajectories for quadrotor unmanned aerial vehicles (UAVs), which are free of collisions with obstacles along their entire length. To avoid the non-convex constraints normally required for obstacle-avoidance, we perform a mixed-integer optimization in which polynomial trajectories are assigned to convex regions which are known to be obstacle-free. Prior approaches have used the faces of the obstacles themselves to define these convex regions. We instead use IRIS, a recently developed technique for greedy convex segmentation [1], to pre-compute convex regions of safe space. This results in a substantially reduced number of integer variables, which improves the speed with which the optimization can be solved to its global optimum, even for tens or hundreds of obstacle faces. In addition, prior approaches have typically enforced obstacle avoidance at a finite set of sample or knot points. We introduce a technique based on sums-of-squares (SOS) programming that allows us to ensure that the entire piecewise polynomial trajectory is free of collisions using convex constraints. We demonstrate this technique in 2D and in 3D using a dynamical model in the Drake toolbox for MATLAB [2].**







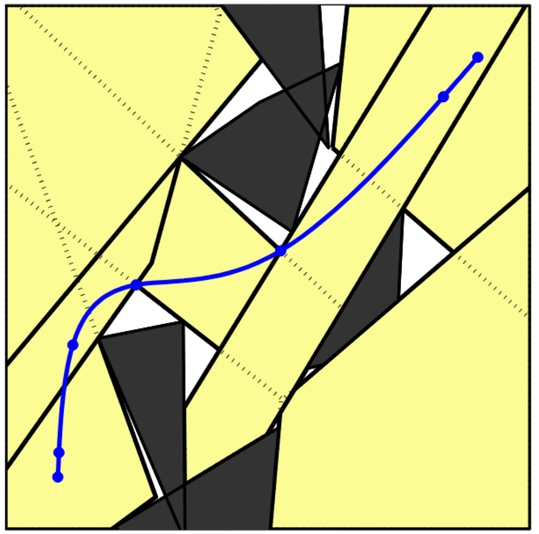
Combinatorial planning

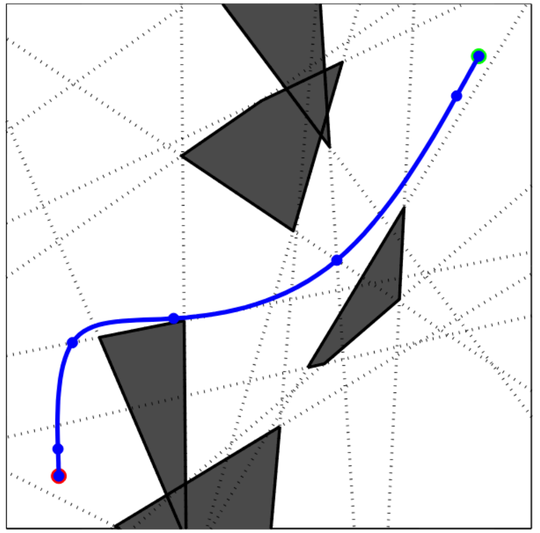
Pros

- Can formulate trajectory parameterizations that are guaranteed to lie within free space (e.g., Bezier curves)

Cons

- Explicit deconstruction of configuration space is computationally expensive
- Scales poorly with configuration space dimension





Sampling-based planning

- Key idea is to explore configuration space by (randomly) sampling configurations

Sampling-based planning

Probabilistic roadmap

Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces

Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars

***Abstract*— A new motion planning method for robots in static workspaces is presented. This method proceeds in two phases: a learning phase and a query phase. In the learning phase, a probabilistic roadmap is constructed and stored as a graph whose nodes correspond to collision-free configurations and whose edges correspond to feasible paths between these configurations. These paths are computed using a simple and fast local planner. In the query phase, any given start and goal configurations of the robot are connected to two nodes of the roadmap; the roadmap is then searched for a path joining these two nodes. The method is general and easy to implement. It can be applied to virtually any type of holonomic robot. It requires selecting certain parameters (e.g., the duration of the learning phase) whose values depend on the scene, that is the robot and its workspace. But these values turn out to be relatively easy to choose. Increased efficiency can also be achieved by tailoring some components of the method (e.g., the local planner) to the considered robots. In this paper the method is applied to planar articulated robots with many degrees of freedom. Experimental results show that path planning can be done in a fraction of a second on a contemporary workstation ($\approx$ 150 MIPS), after learning for relatively short periods of time (a few dozen seconds).**

space (C-space [37]) of the robot is stored as an undirected graph R. The configurations are the nodes of R and the paths computed by the local planner are the edges of R. The learning phase is concluded by some postprocessing of R to improve its connectivity.

Following the learning phase, multiple queries can be answered. A *query* asks for a path between two free configurations of the robot. To process a query the method first attempts to find a path from the start and goal configurations to two nodes of the roadmap. Next, a graph search is done to find a sequence of edges connecting these nodes in the roadmap. Concatenation of the successive path segments transforms the sequence found into a feasible path for the robot.

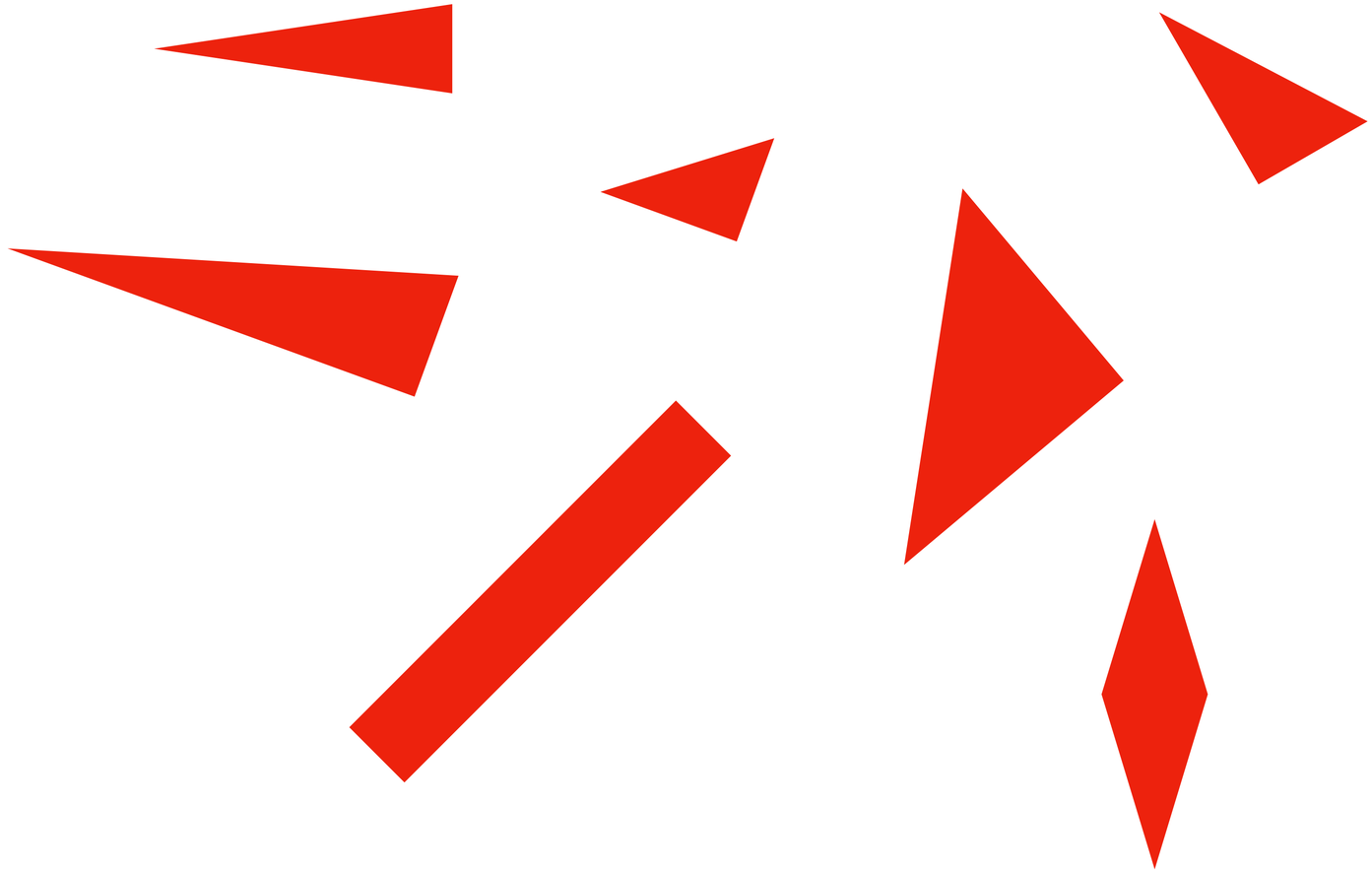
Notice that the learning and the query phases do not have to be executed sequentially. Instead, they can be interwoven to adapt the size of the roadmap to difficulties encountered during the query phase, thus increasing the learning flavor of our method. For instance, a small roadmap could be first constructed; this roadmap could then be augmented (or reduced)

Sampling-based planning

Probabilistic roadmap

Algorithm 1: PRM (preprocessing phase).

```
1 V ← ∅; E ← ∅;
2 for i = 0, ..., n do
3     x_rand ← SampleFree_i;
4     U ← Near(G = (V, E), x_rand, r);
5     V ← V ∪ {x_rand};
6     foreach u ∈ U, in order of increasing ‖u − x_rand‖, do
7         if x_rand and u are not in the same connected component of G = (V, E) then
8             if CollisionFree(x_rand, u) then
                  E ← E ∪ {(x_rand, u), (u, x_rand)};
9 return G = (V, E);
```

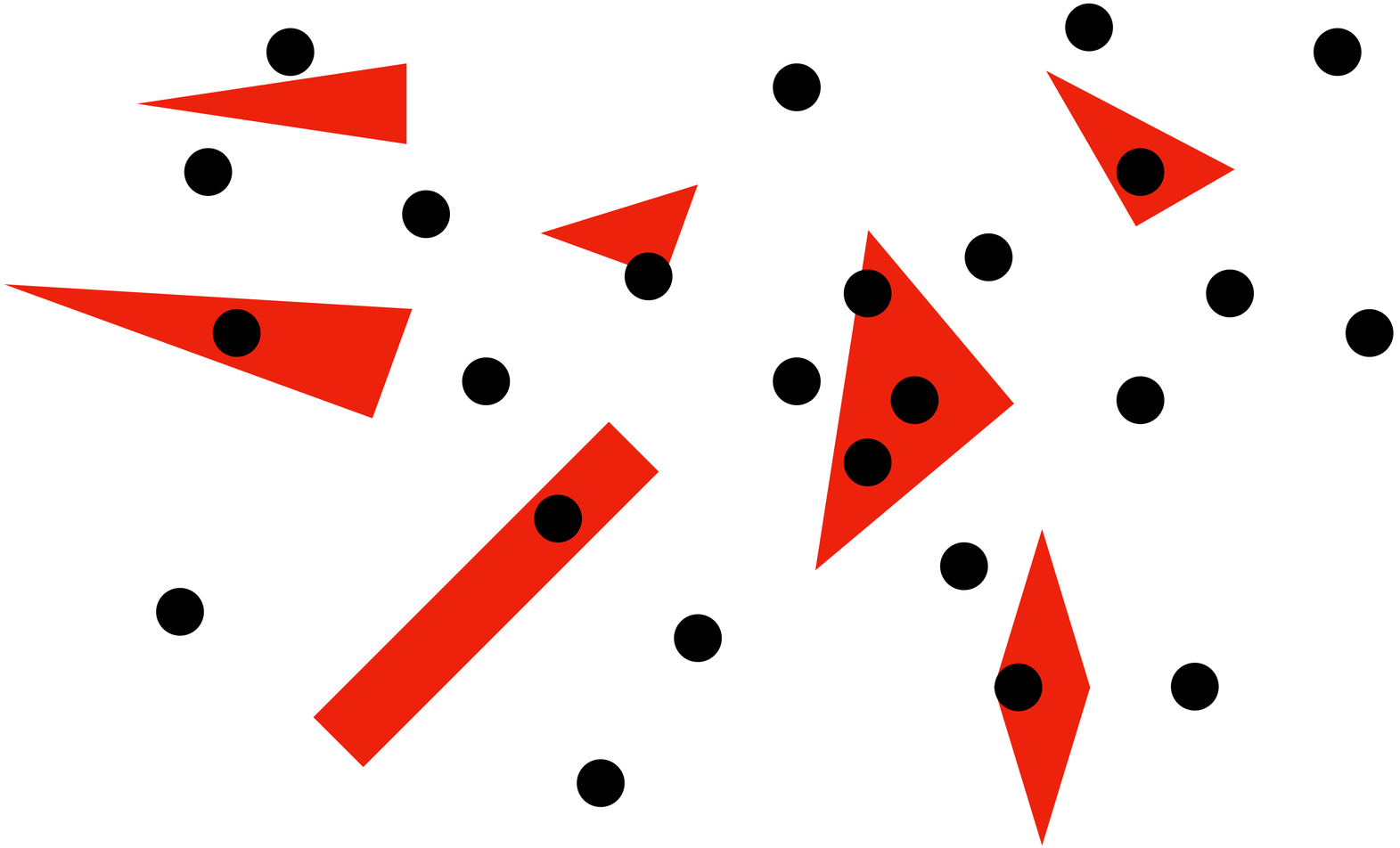


Sampling-based planning

Probabilistic roadmap

Algorithm 1: PRM (preprocessing phase).

```
1 V ← ∅; E ← ∅;
2 for i = 0, ..., n do
3     x_rand ← SampleFree_i;
4     U ← Near(G = (V, E), x_rand, r);
5     V ← V ∪ {x_rand};
6     foreach u ∈ U, in order of increasing ‖u − x_rand‖,
      do
7         if x_rand and u are not in the same connected
          component of G = (V, E) then
8             if CollisionFree(x_rand, u) then
                  E ← E ∪ {(x_rand, u), (u, x_rand)};
9 return G = (V, E);
```

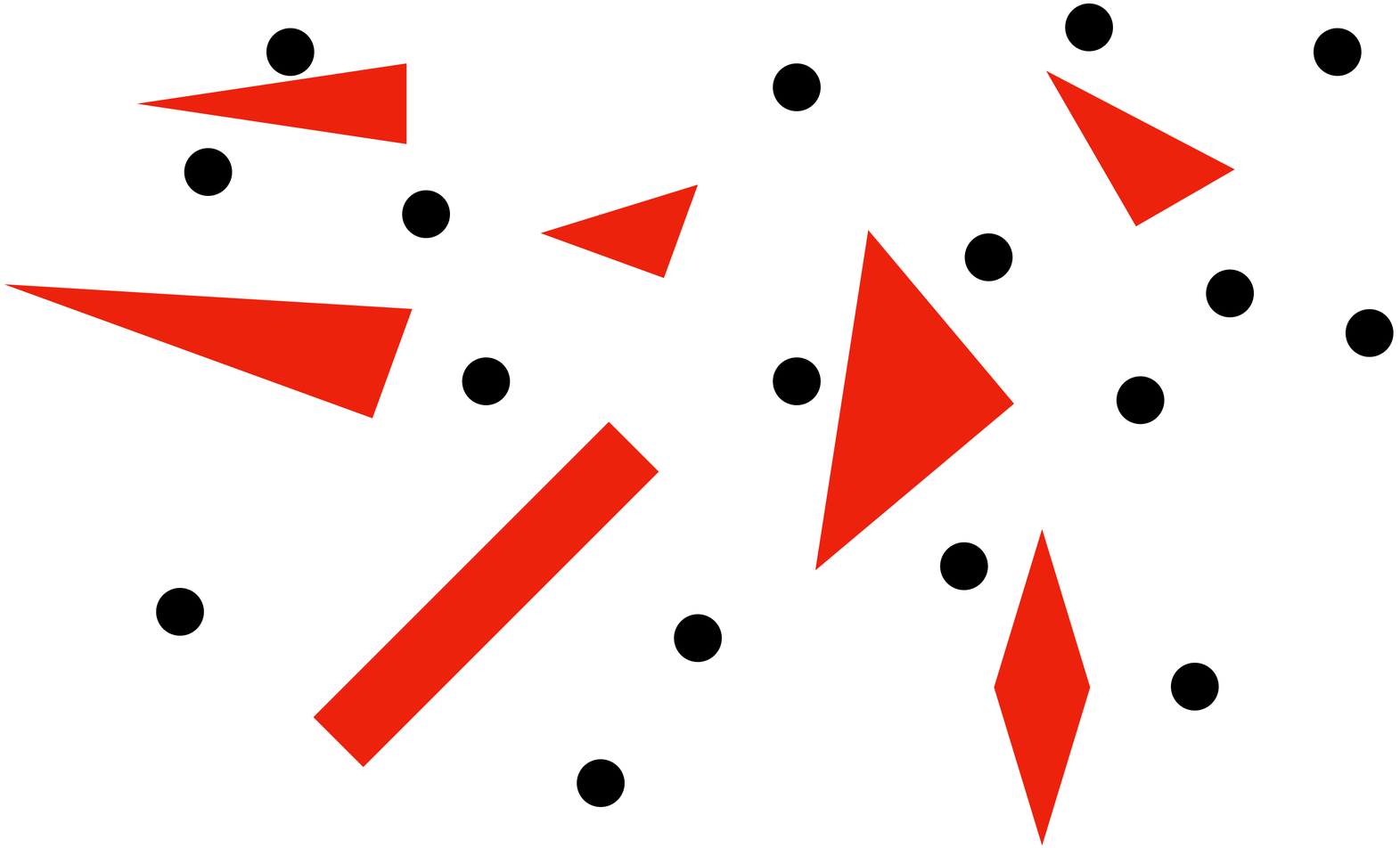


Sampling-based planning

Probabilistic roadmap

Algorithm 1: PRM (preprocessing phase).

```
1 V ← ∅; E ← ∅;
2 for i = 0, ..., n do
3     x_rand ← SampleFree_i;
4     U ← Near(G = (V, E), x_rand, r);
5     V ← V ∪ {x_rand};
6     foreach u ∈ U, in order of increasing ‖u − x_rand‖,
      do
7         if x_rand and u are not in the same connected
          component of G = (V, E) then
8             if CollisionFree(x_rand, u) then
                  E ← E ∪ {(x_rand, u), (u, x_rand)};
9 return G = (V, E);
```

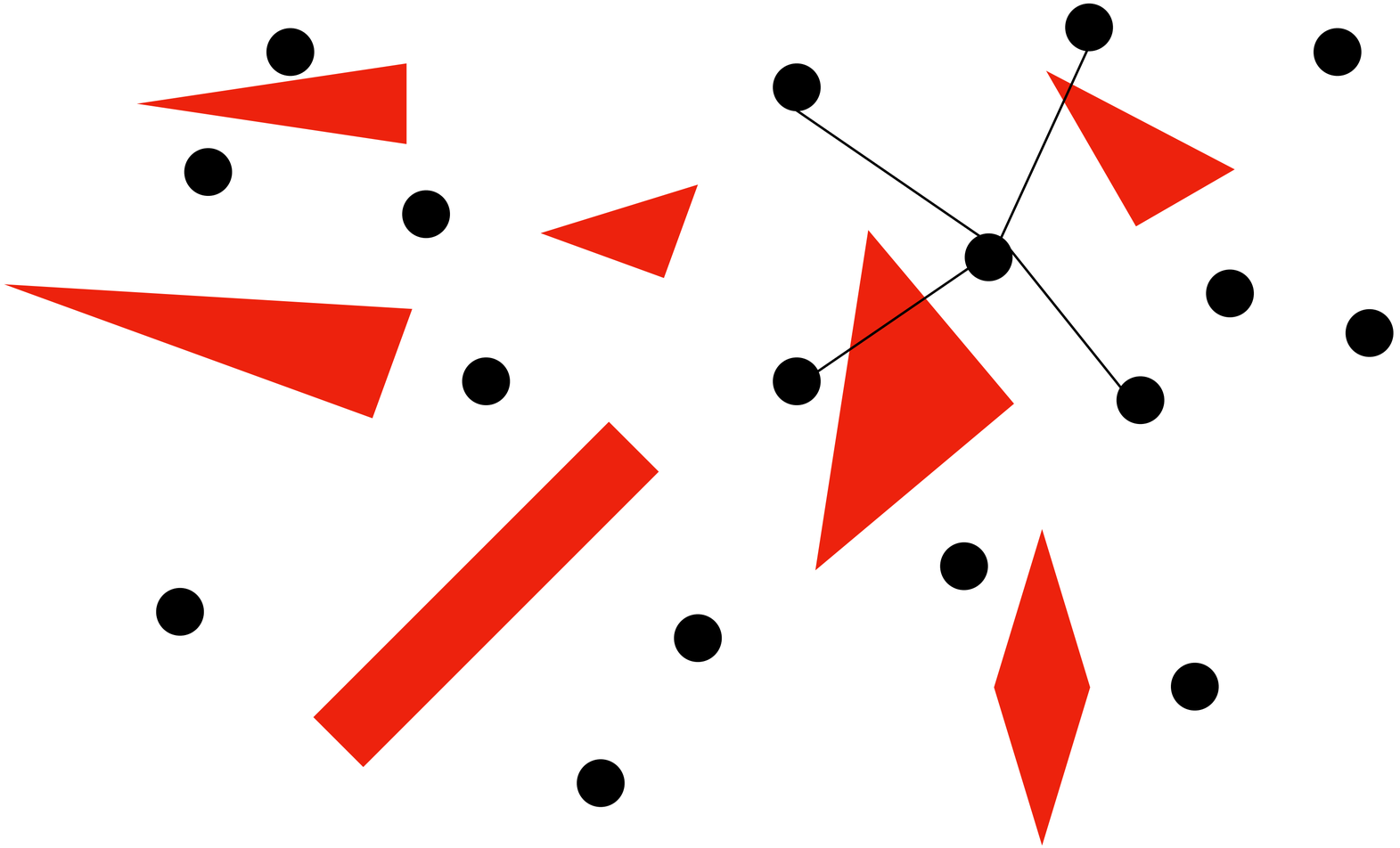


Sampling-based planning

Probabilistic roadmap

Algorithm 1: PRM (preprocessing phase).

```
1 V ← ∅; E ← ∅;
2 for i = 0, ..., n do
3     x_rand ← SampleFree_i;
4     U ← Near(G = (V, E), x_rand, r);
5     V ← V ∪ {x_rand};
6     foreach u ∈ U, in order of increasing ‖u − x_rand‖,
      do
7         if x_rand and u are not in the same connected
          component of G = (V, E) then
8             if CollisionFree(x_rand, u) then
                  E ← E ∪ {(x_rand, u), (u, x_rand)};
9 return G = (V, E);
```

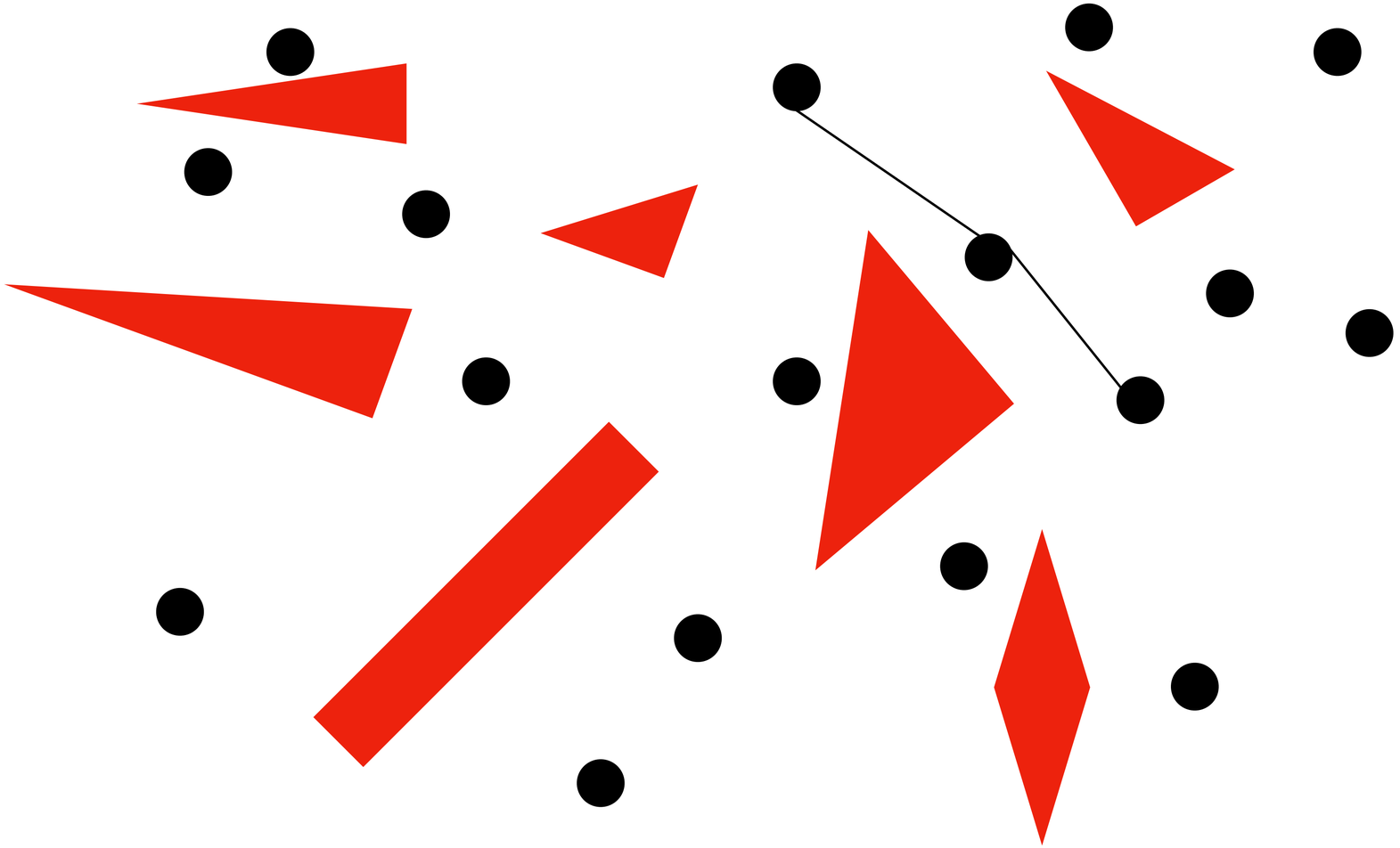


Sampling-based planning

Probabilistic roadmap

Algorithm 1: PRM (preprocessing phase).

```
1 V ← ∅; E ← ∅;
2 for i = 0, ..., n do
3     x_rand ← SampleFree_i;
4     U ← Near(G = (V, E), x_rand, r);
5     V ← V ∪ {x_rand};
6     foreach u ∈ U, in order of increasing ‖u − x_rand‖,
      do
7         if x_rand and u are not in the same connected
          component of G = (V, E) then
8             if CollisionFree(x_rand, u) then
                  E ← E ∪ {(x_rand, u), (u, x_rand)};
9 return G = (V, E);
```

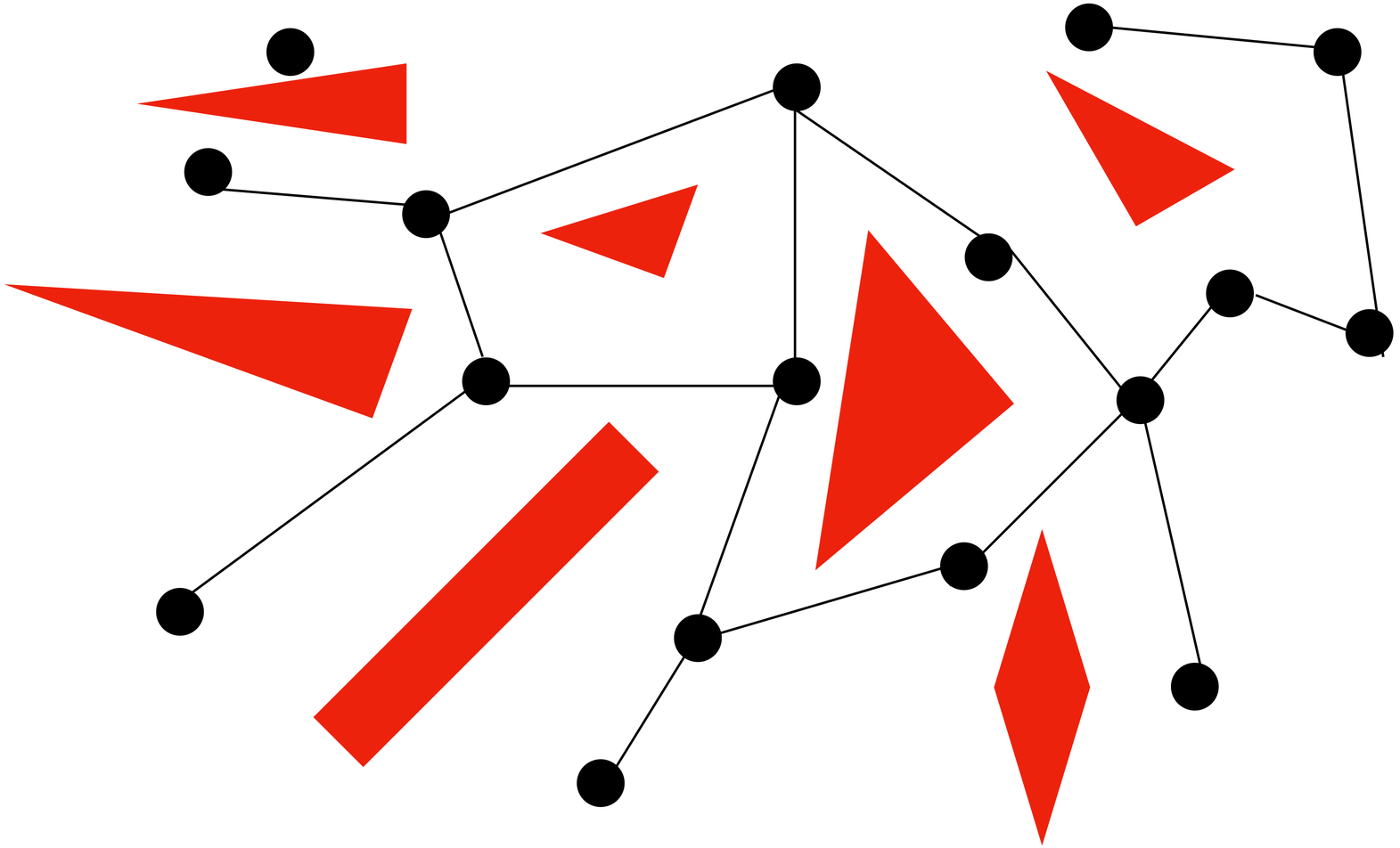


Sampling-based planning

Probabilistic roadmap

Algorithm 1: PRM (preprocessing phase).

```
1 V ← ∅; E ← ∅;
2 for i = 0, ..., n do
3     x_rand ← SampleFree_i;
4     U ← Near(G = (V, E), x_rand, r);
5     V ← V ∪ {x_rand};
6     foreach u ∈ U, in order of increasing ‖u − x_rand‖, do
7         if x_rand and u are not in the same connected component of G = (V, E) then
8             if CollisionFree(x_rand, u) then
                  E ← E ∪ {(x_rand, u), (u, x_rand)};
9 return G = (V, E);
```

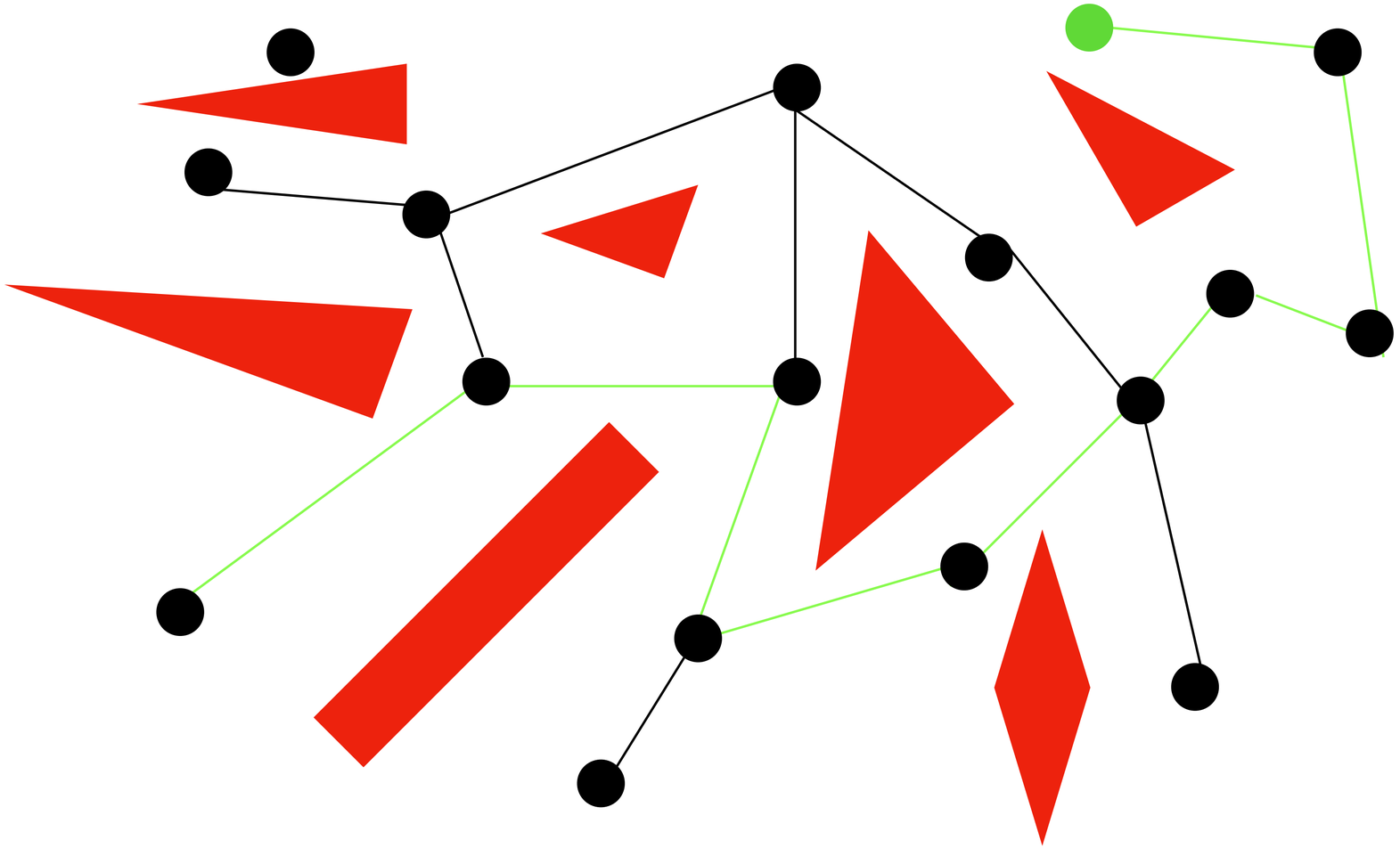


Sampling-based planning

Probabilistic roadmap

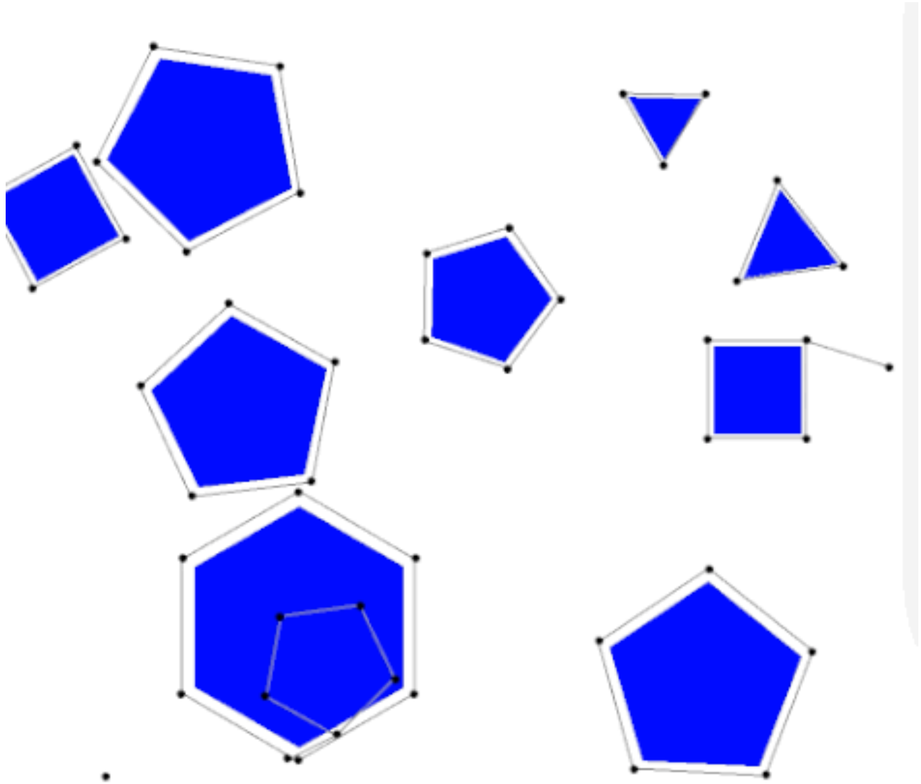
Algorithm 1: PRM (preprocessing phase).

```
1 V ← ∅; E ← ∅;
2 for i = 0, ..., n do
3     x_rand ← SampleFree_i;
4     U ← Near(G = (V, E), x_rand, r);
5     V ← V ∪ {x_rand};
6     foreach u ∈ U, in order of increasing ‖u − x_rand‖, do
7         if x_rand and u are not in the same connected component of G = (V, E) then
8             if CollisionFree(x_rand, u) then
                  E ← E ∪ {(x_rand, u), (u, x_rand)};
9 return G = (V, E);
```



Sampling-based planning

Probabilistic roadmap



https://en.wikipedia.org/wiki/Probabilistic_roadmap#/media/File:PRM_with_Ob-maps.gif

Sampling-based planning

Probabilistic roadmap

Pros

- Multi-query planning - can plan from multiple initial and goal configurations within the roadmap

Cons

- “Wasteful” computation of edges for parts of graph not relevant to planning problem
- Must reconstruct the whole graph each time obstacles change

Sampling-based planning

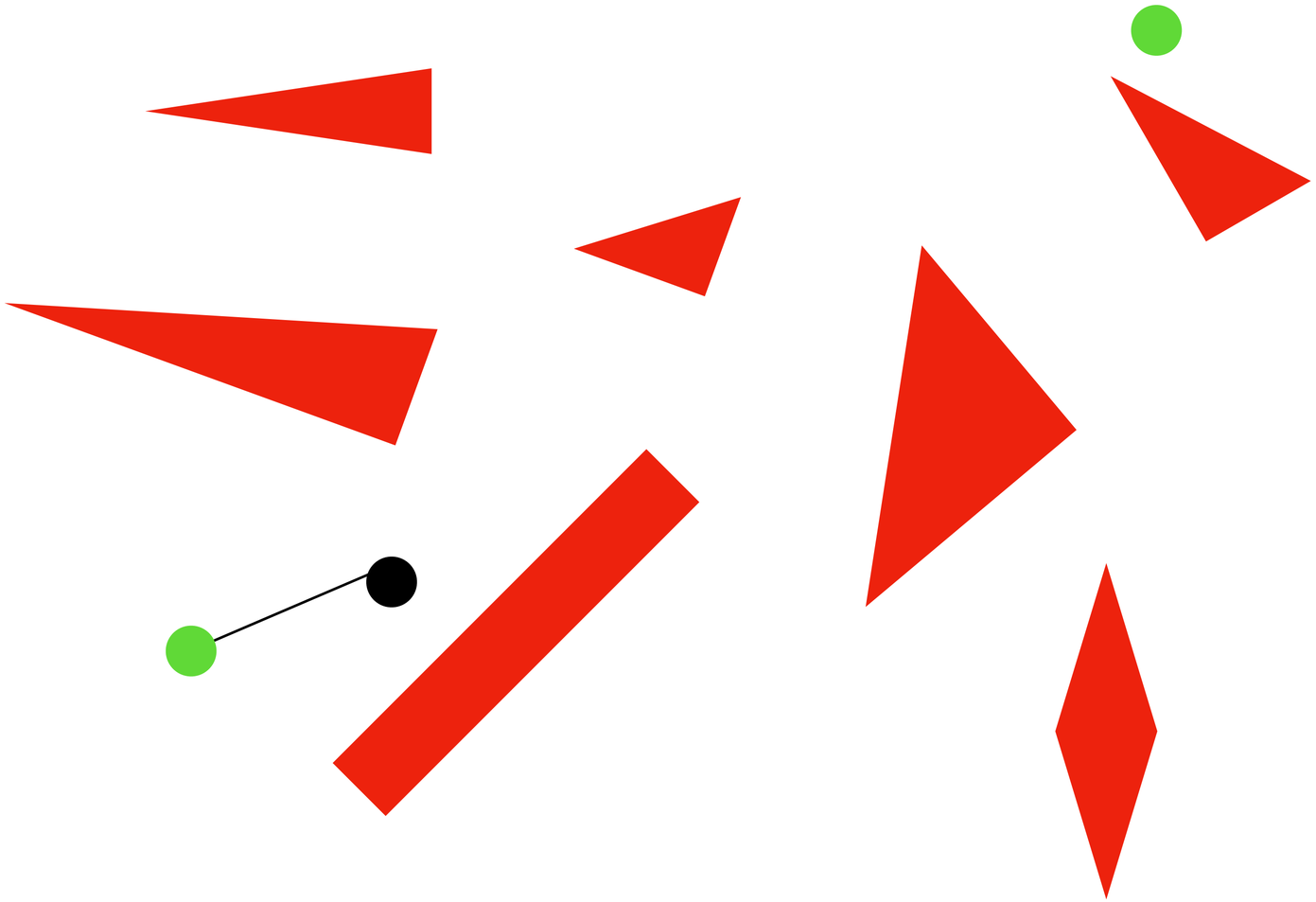
Rapidly exploring random tree

Sampling-based planning

Rapidly exploring random tree

Algorithm 3: RRT.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;

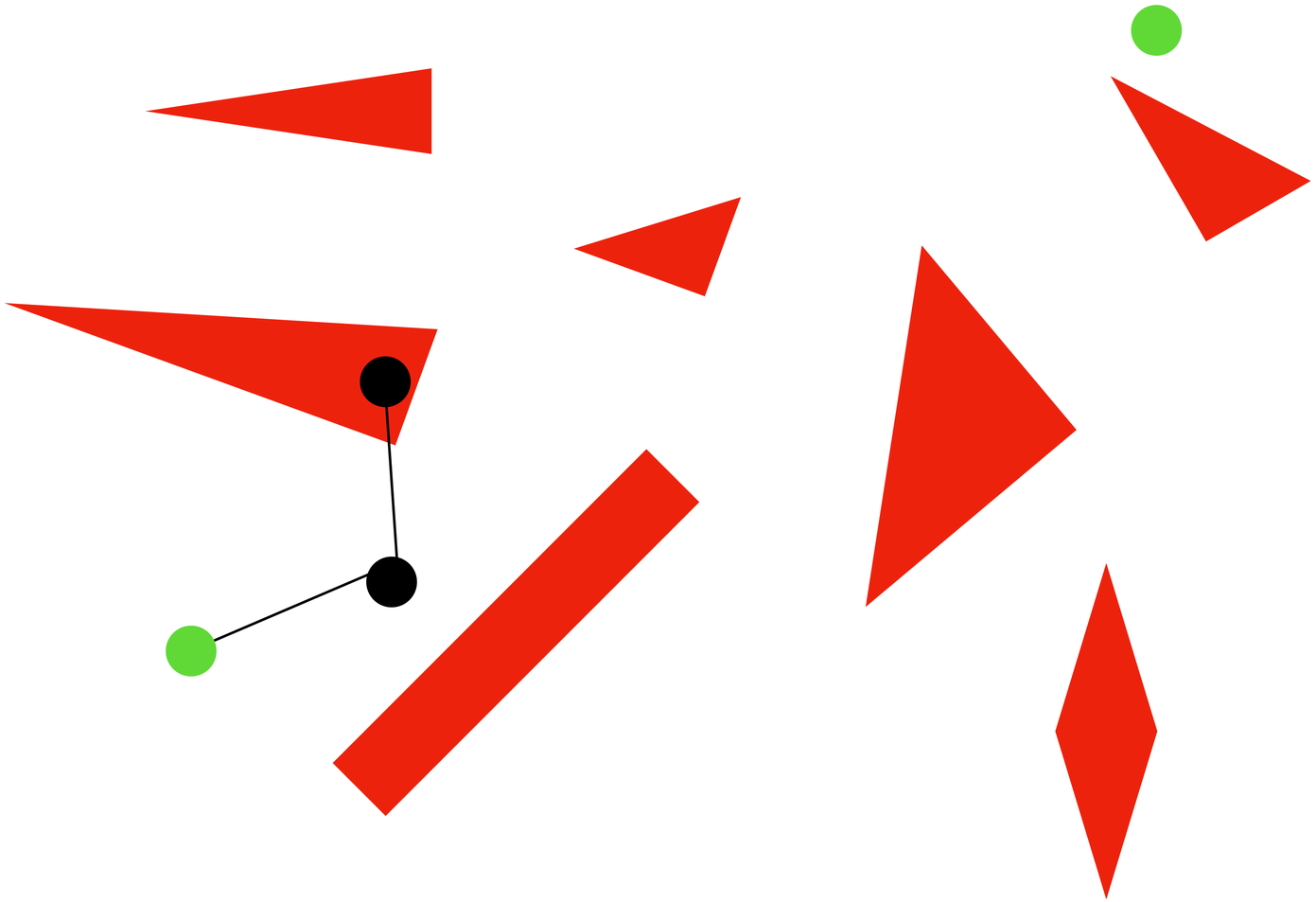


Sampling-based planning

Rapidly exploring random tree

Algorithm 3: RRT.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;

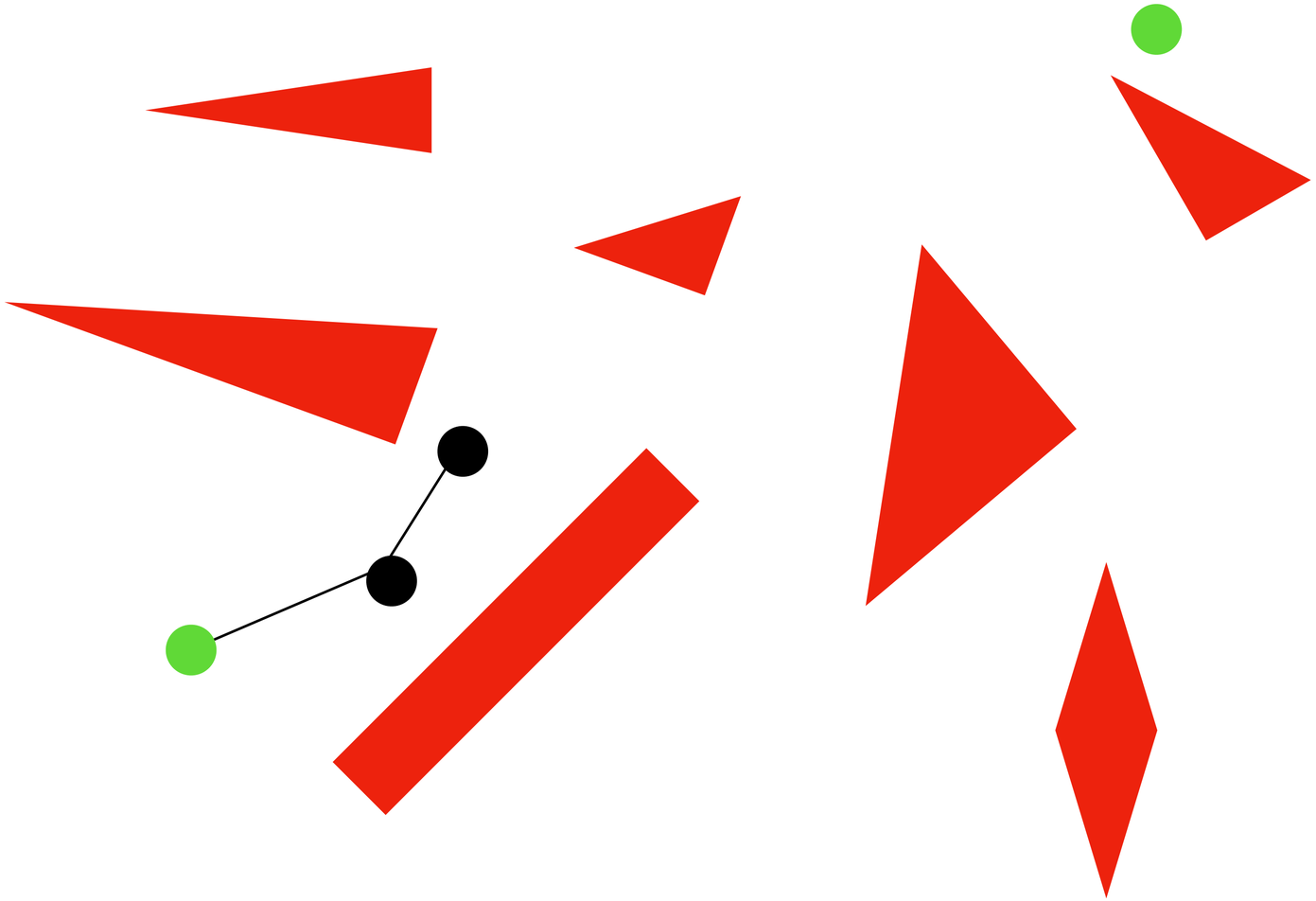


Sampling-based planning

Rapidly exploring random tree

Algorithm 3: RRT.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;

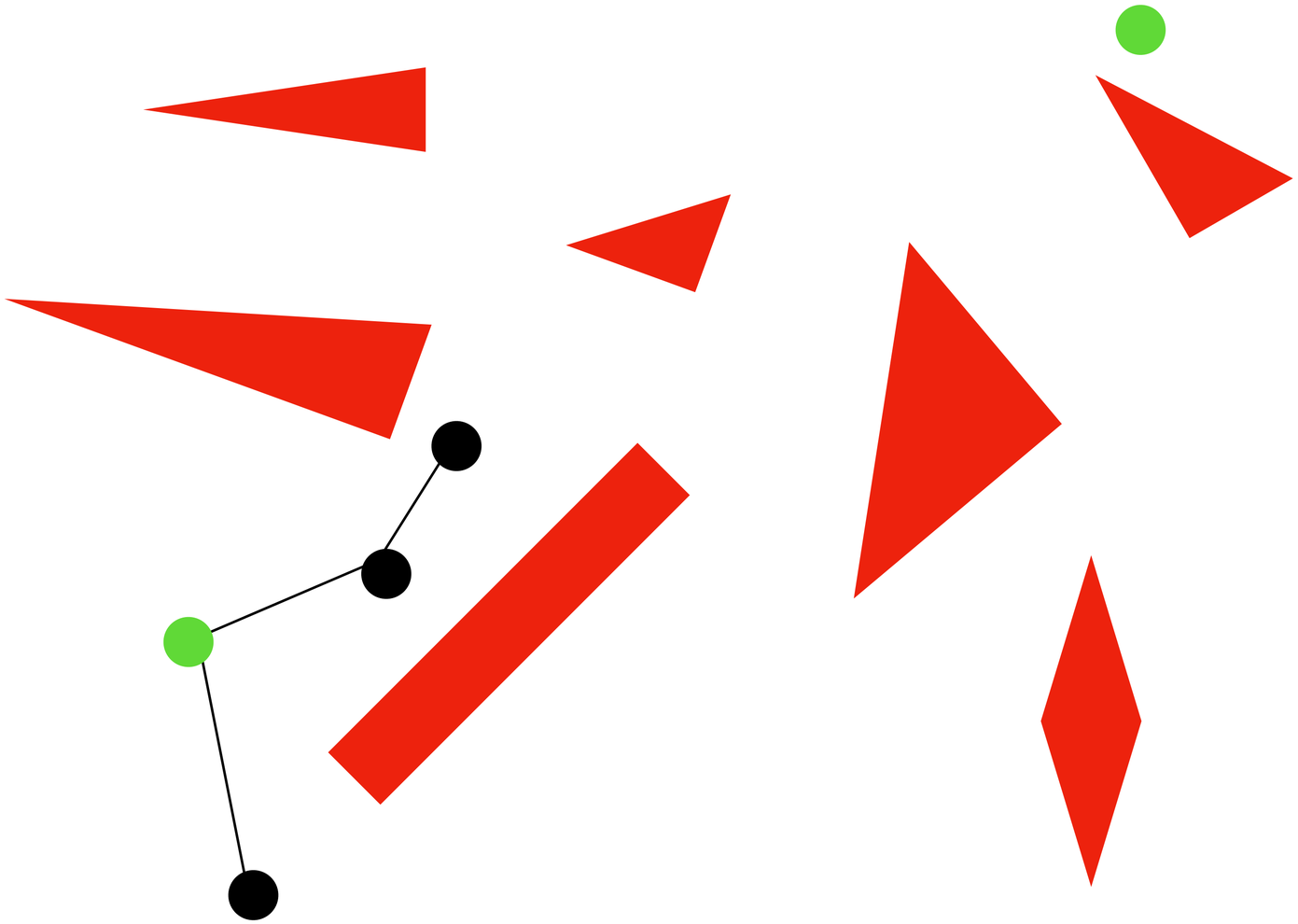


Sampling-based planning

Rapidly exploring random tree

Algorithm 3: RRT.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;

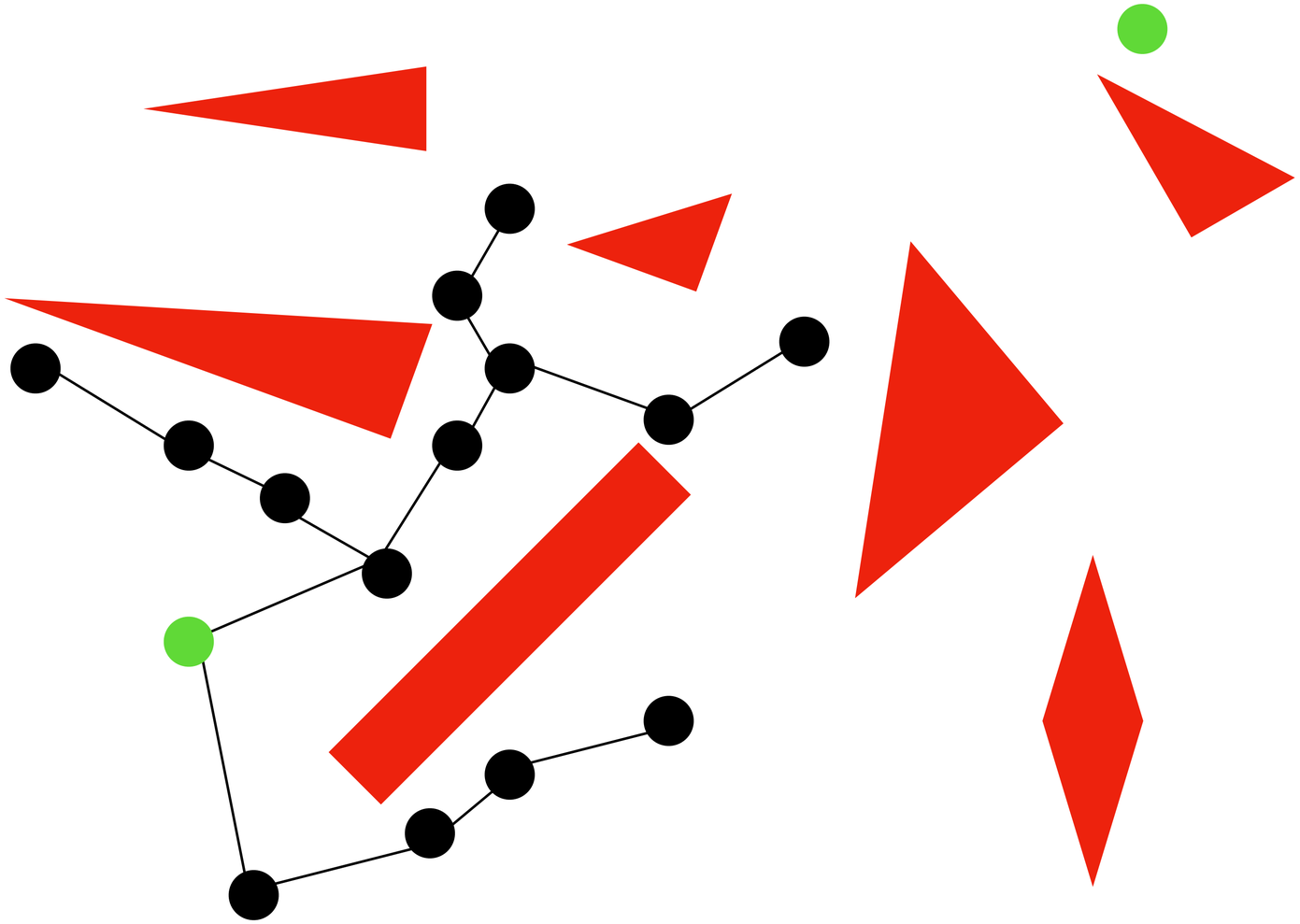


Sampling-based planning

Rapidly exploring random tree

Algorithm 3: RRT.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;

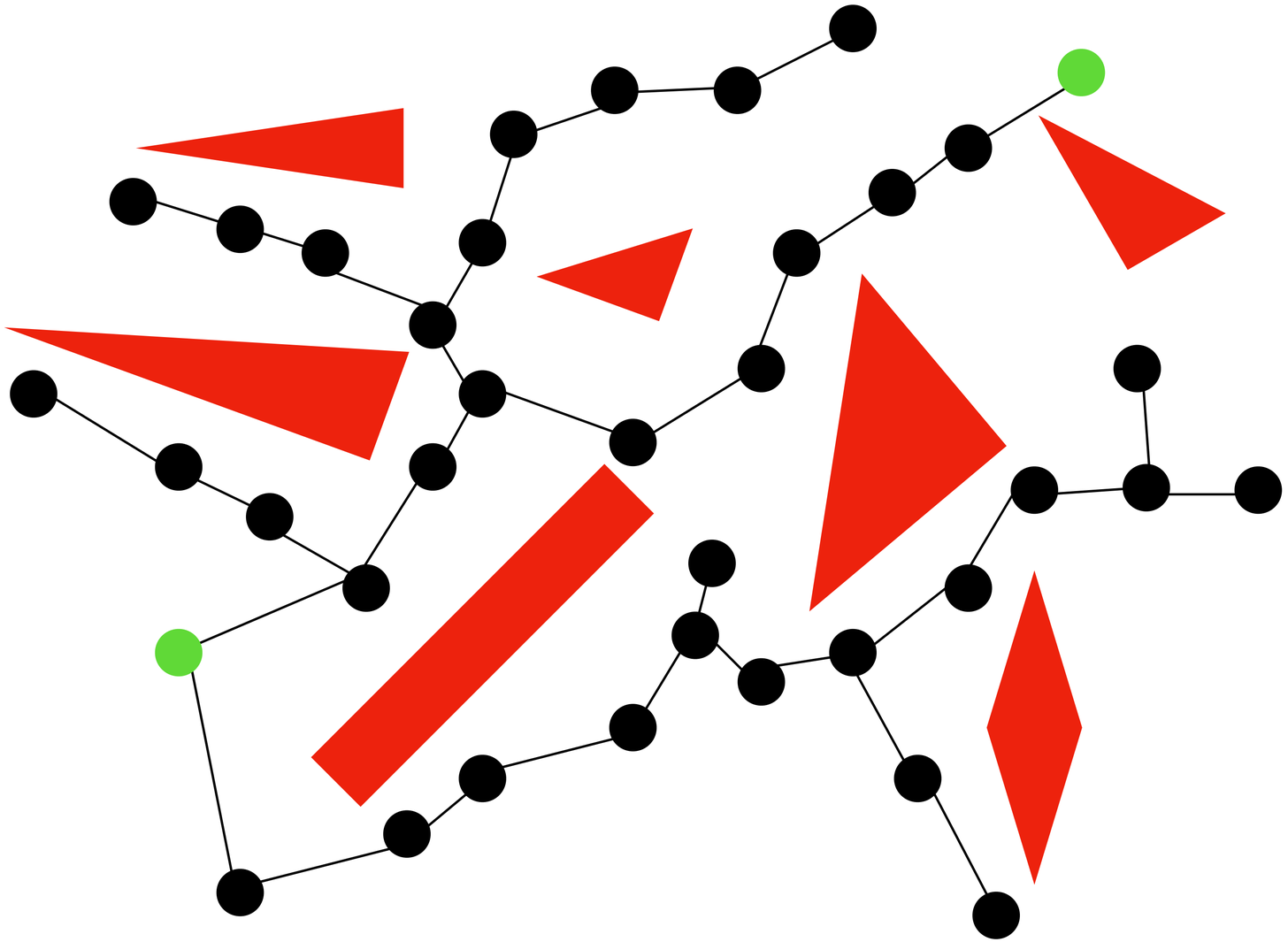


Sampling-based planning

Rapidly exploring random tree

Algorithm 3: RRT.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;

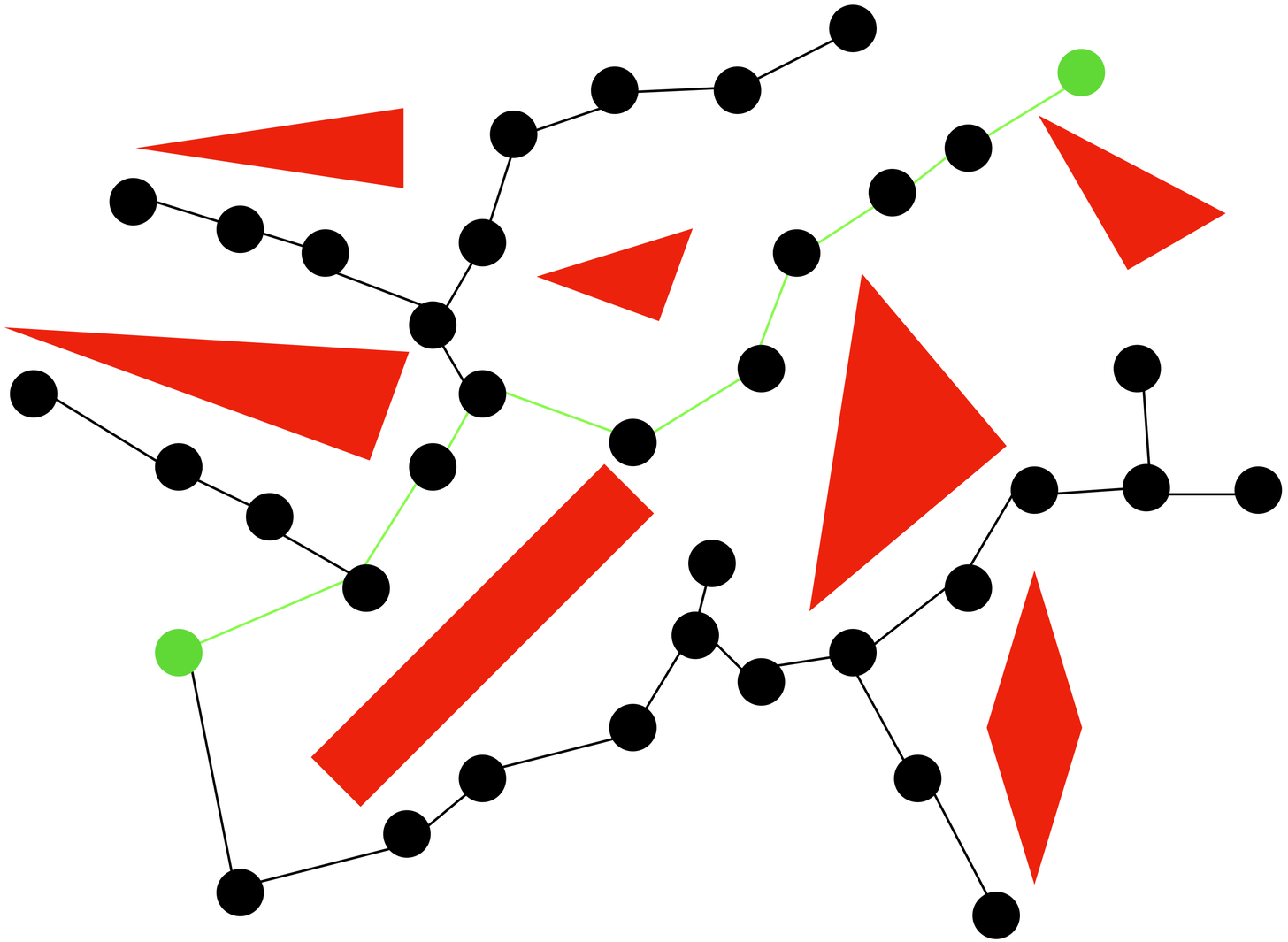


Sampling-based planning

Rapidly exploring random tree

Algorithm 3: RRT.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;



Sampling-based planning

Rapidly exploring random tree

Pros

- Single-query planning generally more effective for solving a single planning problem

Cons

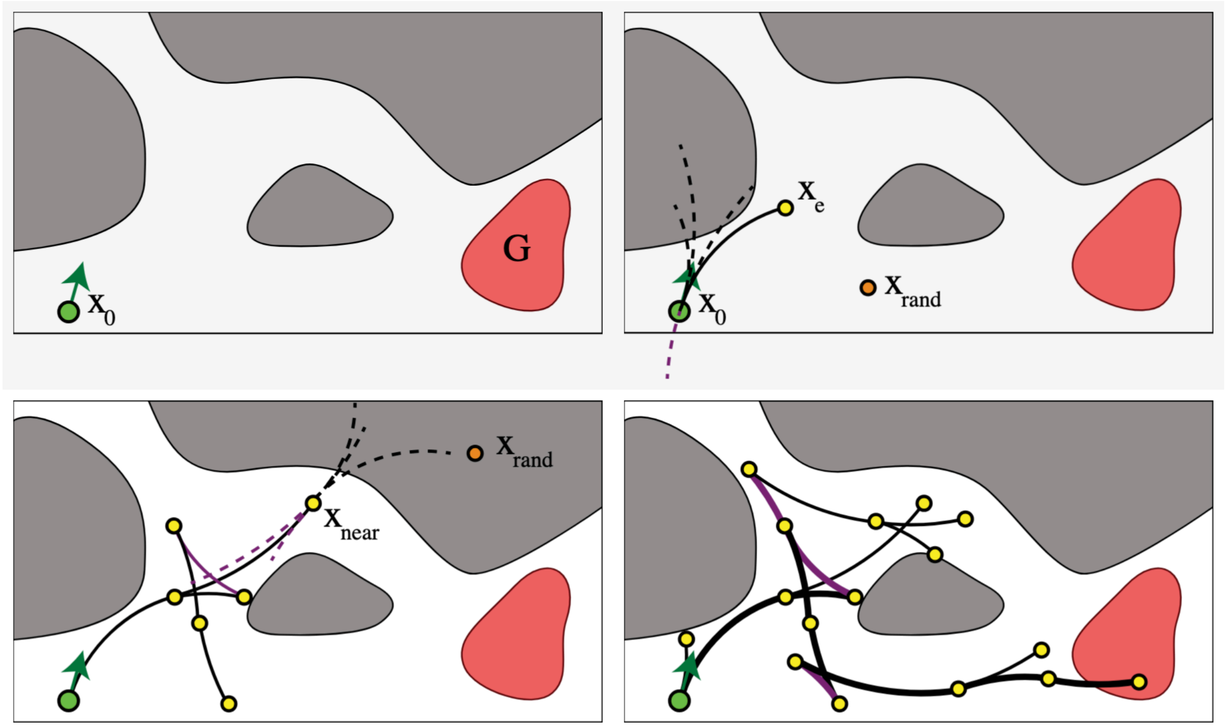
- Must reconstruct the tree given a change in planning query or obstacle configuration

Questions

- Can motion planners satisfy dynamical constraints?

Questions

- Can motion planners satisfy dynamical constraints?



Planning Algorithms, Steven M. LaValle (2006)

Questions

- Can motion planners satisfy dynamical constraints?
- Can constraints from the original optimal control problem be enforced here?

Questions

- Can motion planners satisfy dynamical constraints?
- Can constraints from the original optimal control problem be enforced here?
- Can sampling-based planners find the globally optimal solution?

Questions

- Can motion planners satisfy dynamical constraints?
- Can constraints from the original optimal control problem be enforced here?
- Can sampling-based planners find the globally optimal solution?

Probabilistic completeness: an algorithm ALG is probabilistically complete if, for any robustly feasible path planning problem $\mathscr{P} = (\mathscr{X}_{\text{free}}, x_{\text{init}}, \mathscr{X}_{\text{goal}})$,

$$\lim_{n \to \infty} \mathbb{P}(\{ \exists x_{\text{goal}} \in V_n^{\text{ALG}} \cap \mathscr{X}_{\text{goal}} \text{ such that } x_{\text{init}} \text{ is connected to } x_{\text{goal}} \text{ in } G_n^{\text{ALG}} \}) = 1$$

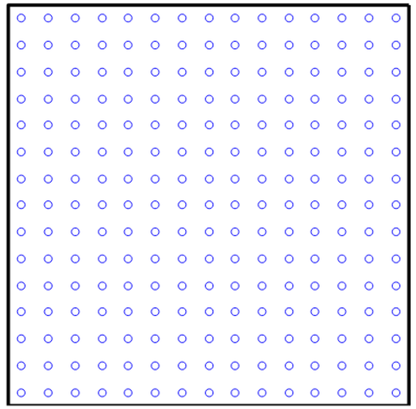
Asymptotic optimality: an algorithm ALG is asymptotically optimal if, for any path planning problem $\mathscr{P} = (\mathscr{X}_{\text{free}}, x_{\text{init}}, \mathscr{X}_{\text{goal}})$ and cost function $c : \Sigma \to \mathbb{R}_{\geq 0}$ that admit a robustly optimal solution with finite cost c^*

$$\mathbb{P}\left(\left\{ \limsup_{n \to \infty} \quad Y_n^{\text{ALG}} = c^* \right\} \right) = 1$$

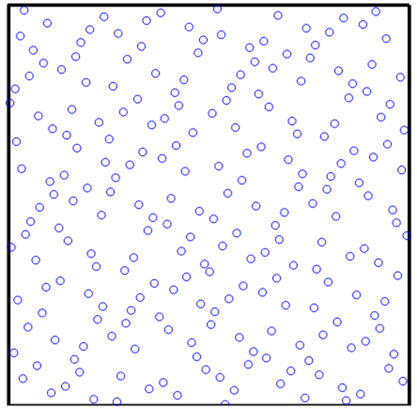
Questions

- Can motion planners satisfy dynamical constraints?
- Can constraints from the original optimal control problem be enforced here?
- Can sampling-based planners find the globally optimal solution?
- Can sampling-based planners produce reproducible results?

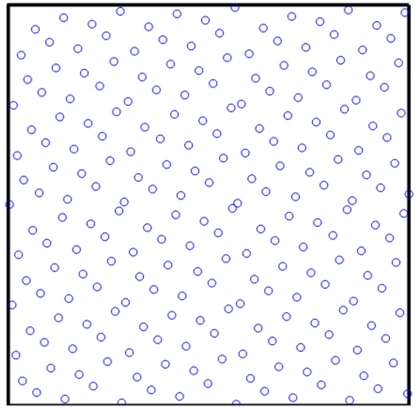
Questions



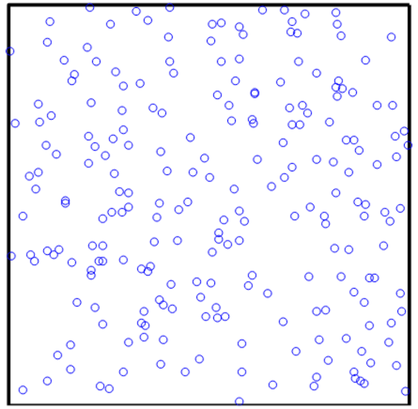
(a) Deterministic (Sukharev grid [98])



(b) Deterministic (Halton [99])



(c) Deterministic (Hammersley [100])



(d) Pseudo-random (Mersenne Twister [101])

J. A. Starek, "Sampling-Based Motion Planning for Safe and Efficient Spacecraft Proximity Operations," PhD thesis, 2016.

Takeaways

Sampling-based motion planning

- Difficult to enforce challenging dynamical constraints
- In practice do not find the globally optimal solution, but some algorithms have more favorable properties
- Plans continuous trajectories up to discretization of certain components (e.g., collision checker)

Software components

- Nearest neighbor search

Software components

- Nearest neighbor search

Fast Nearest Neighbor Search in SE(3) for Sampling-Based Motion Planning

Jeffrey Ichnowski and Ron Alterovitz

University of North Carolina at Chapel Hill, USA
{jeffi,ron}@cs.unc.edu

Abstract. Nearest neighbor searching is a fundamental building block of most sampling-based motion planners. We present a novel method for fast exact nearest neighbor searching in $SE(3)$—the 6 dimensional space that represents rotations and translations in 3 dimensions. $SE(3)$ is commonly used when planning the motions of rigid body robots. Our approach starts by projecting a 4-dimensional cube onto the 3-sphere that is created by the unit quaternion representation of rotations in the rotational group $SO(3)$. We then use 4 kd-trees to efficiently partition the projected faces (and their negatives). We propose efficient methods to handle the recursion pruning checks that arise with this kd-tree splitting approach, discuss splitting strategies that support dynamic data sets, and extend this approach to $SE(3)$ by incorporating translations. We integrate our approach into RRT and RRT* and demonstrate the fast performance and efficient scaling of our nearest neighbor search as the tree size increases.

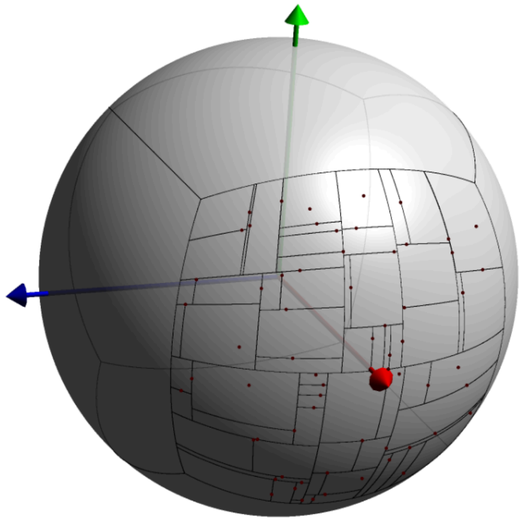


Fig. 1. A kd-tree projected onto the surface of a 2-sphere. An axis-orthogonal cube is projected into a sphere. Each face of the cube is a separately computed kd-tree; however, for illustrative purposes, we show the kd-tree of only one of the faces. In our method we extend the analogy to 4-dimensional space for use with quaternions.

Software components

- Nearest neighbor search
- Collision checker

Software components

- Nearest neighbor search
- Collision checker

The International Journal of Robotics Research
Volume 33, Issue 9, August 2014, Pages 1251-1270

https://doi.org/10.1177/0278364914528132

SAGE journals

Article

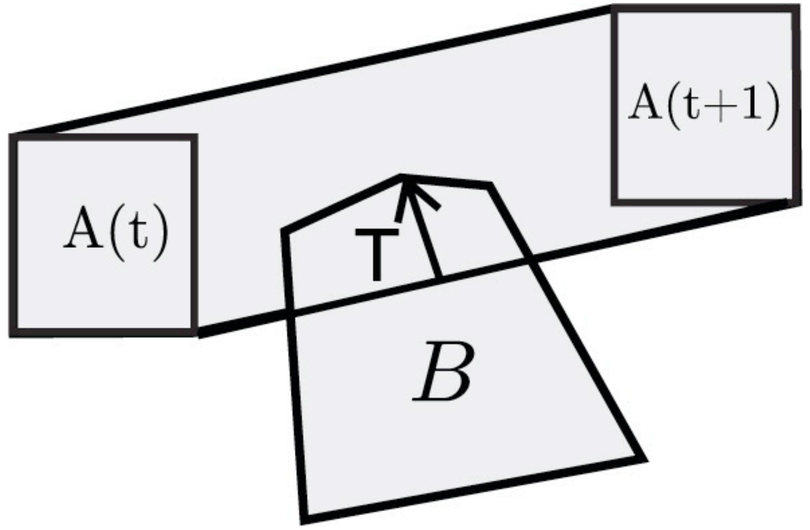
Motion planning with sequential convex optimization and convex collision checking

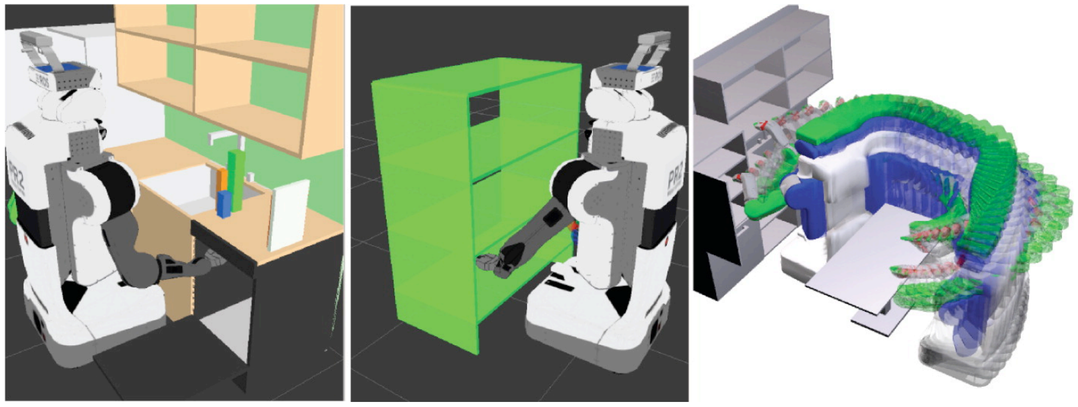
John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel

Abstract

We present a new optimization-based approach for robotic motion planning among obstacles. Like CHOMP (Covariant Hamiltonian Optimization for Motion Planning), our algorithm can be used to find collision-free trajectories from naïve, straight-line initializations that might be in collision. At the core of our approach are (a) a sequential convex optimization procedure, which penalizes collisions with a hinge loss and increases the penalty coefficients in an outer loop as necessary, and (b) an efficient formulation of the no-collisions constraint that directly considers continuous-time safety Our algorithm is implemented in a software package called TrajOpt.

We report results from a series of experiments comparing TrajOpt with CHOMP and randomized planners from OMPL, with regard to planning time and path quality. We consider motion planning for 7 DOF robot arms, 18 DOF full-body robots, statically stable walking motion for the 34 DOF Atlas humanoid robot, and physical experiments with the 18 DOF PR2. We also apply TrajOpt to plan curvature-constrained steerable needle trajectories in the SE(3) configuration space and multiple non-intersecting curved channels within 3D-printed implants for intracavitary brachytherapy. Details, videos, and source code are freely available at: http://rll.berkeley.edu/trajopt/ijrr.





Software components

- Nearest neighbor search
- Collision checker
- Two-point boundary value problem

Broad applications

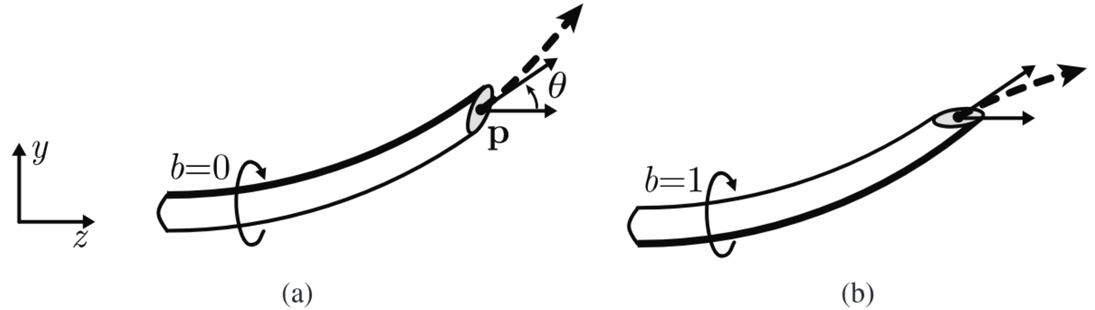
Surgical needle steering

Motion Planning Under Uncertainty for Image-guided Medical Needle Steering

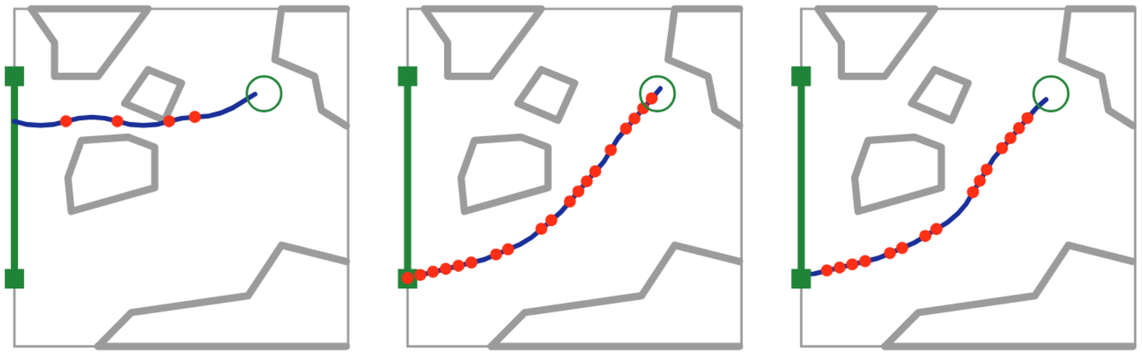
Ron Alterovitz
Department of Computer Science,
University of North Carolina at Chapel Hill,
Chapel Hill, NC 27599-3175, USA
ron@cs.unc.edu

Michael Branicky
Department of Electrical Engineering
and Computer Science,
Case Western Reserve University,
Cleveland, OH 44106, USA
mb@case.edu

Ken Goldberg
Department of Industrial Engineering
and Operations Research,
Department of Electrical Engineering
and Computer Sciences,
University of California,
Berkeley, CA 94720, USA
goldberg@berkeley.edu



(a) (b)



(a) (b) (c)

Broad applications

Protein folding

INSTITUTE OF PHYSICS PUBLISHING PHYSICAL BIOLOGY

Phys. Biol. **2** (2005) S148–S155 doi:10.1088/1478-3975/2/4/S09

Protein folding by motion planning*

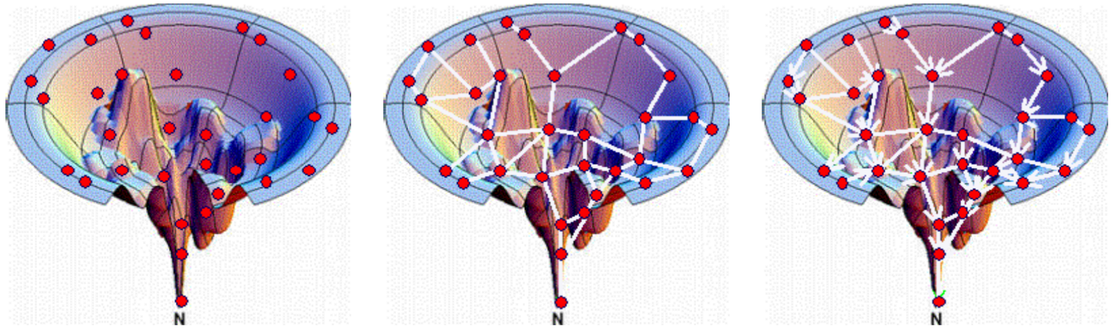
Shawna Thomas[1], **Guang Song**[2] **and Nancy M Amato**[1,3]

[1] Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA
[2] Baker Center for Bioinformatics and Biological Statistics, Iowa State University, Ames, IA 50011, USA

Received 24 June 2005
Accepted for publication 2 September 2005
Published 9 November 2005
Online at stacks.iop.org/PhysBio/2/S148

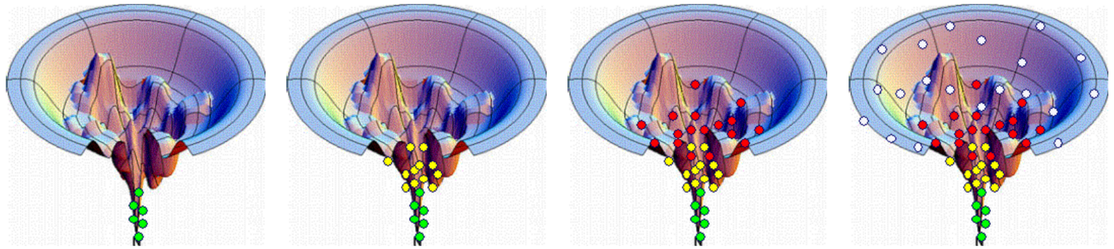
Abstract

We investigate a novel approach for studying protein folding that has evolved from robotics motion planning techniques called *probabilistic roadmap* methods (PRMs). Our focus is to study issues related to the folding process, such as the formation of secondary and tertiary structures, *assuming* we know the native fold. A feature of our PRM-based framework is that the large sets of folding pathways in the roadmaps it produces, in just a few hours on a desktop PC, provide global information about the protein's energy landscape. This is an advantage over other simulation methods such as molecular dynamics or Monte Carlo methods which require more computation and produce only a single trajectory in each run. In our initial studies, we obtained encouraging results for several small proteins. In this paper, we investigate more sophisticated techniques for analyzing the folding pathways in our roadmaps. In addition to more formally revalidating our previous results, we present a case study showing that our technique captures known folding differences between the structurally similar proteins G and L.



(*a*) (*b*) (*c*)

Figure 1. A PRM roadmap for protein folding shown imposed on a visualization of the potential energy landscape: (*a*) after node generation (note sampling is denser around **N**, the known native structure), (*b*) after the connection phase and (*c*) using it to extract folding paths to the known native structure.



(*a*) (*b*) (*c*) (*d*)

Figure 3. An illustration of our iterative perturbation sampling strategy shown imposed on a visualization of the potential energy landscape.

References

- S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *Int. Journal of Robotics Research*, vol. 30, no. 7, 2011.