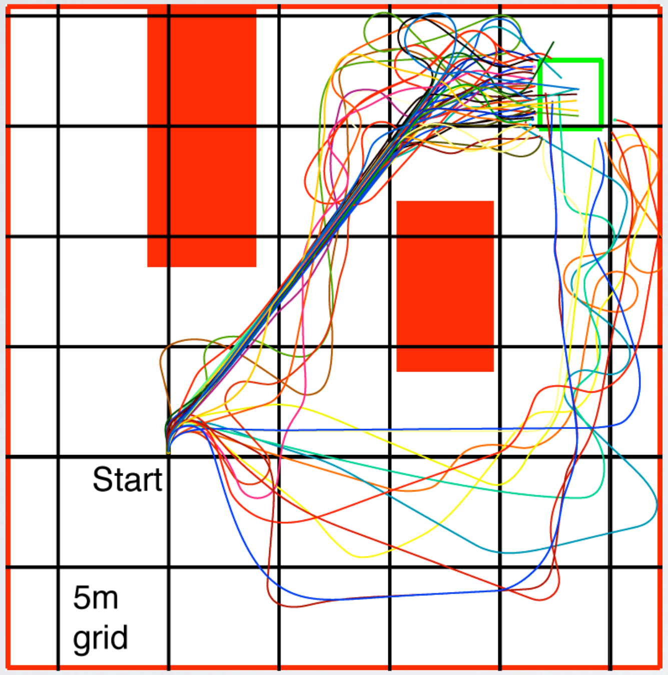
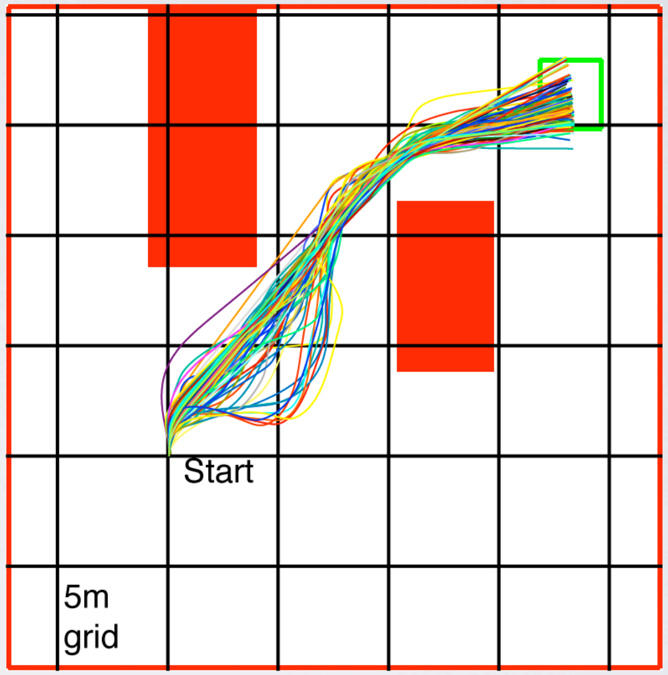
ANYTIME MOTION PLANNING USING THE RRT*



Anytime RRT



Anytime RRT*

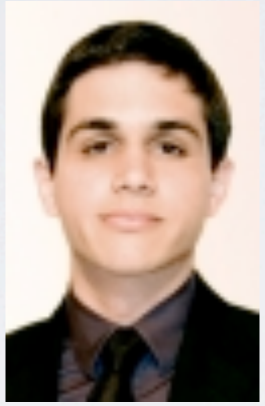
Sertac Karaman[1], **Matthew Walter**[2], Alejandro Perez[2],
Emilio Frazzoli[1], & Seth Teller[2]

[1]MIT / LIDS [2]MIT / CSAIL

10 May 2011

JOINT WORK WITH

Sertac Karaman
(MIT/LIDS)



Alejandro Perez
(MIT/CSAIL)

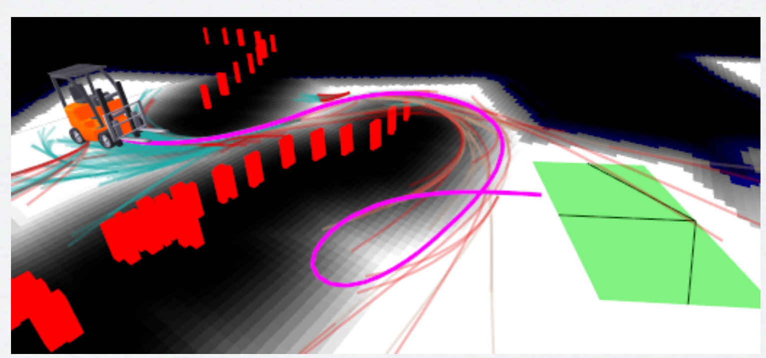
Emilio Frazzoli
(MIT/LIDS)

Seth Teller
(MIT/CSAIL)

PRACTICAL MOTION PLANNING

- (Probabilistic) completeness
- Quickly find a kinodynamically feasible solution
- Computationally efficiency (limited resources)
- Plan despite incomplete, imperfect knowledge
- Accommodate dynamic environments

[Kuwata et al., GNC 2008]

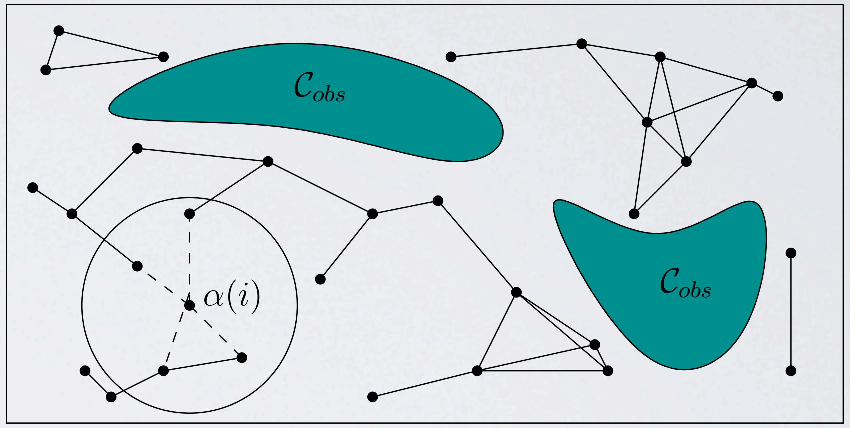


[Teller et al., ICRA 2010]

SAMPLE-BASED MOTION PLANNING

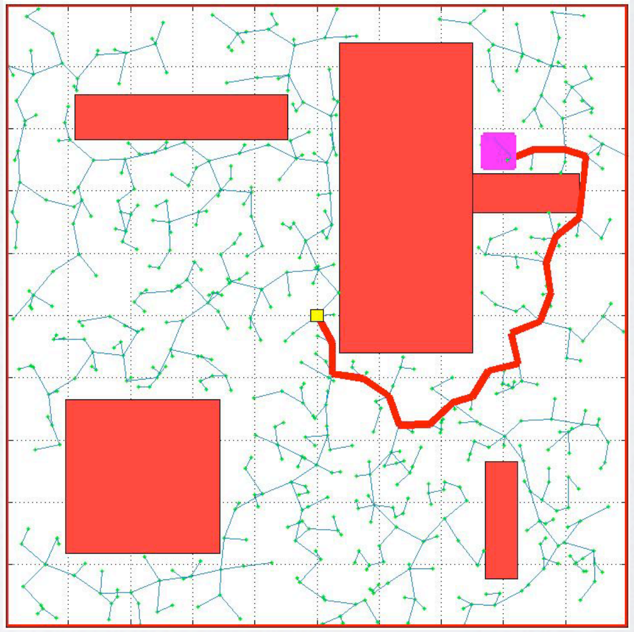
Sample-based motion planning provides an effective solution

- Probabilistic RoadMap (PRM)
  [Kavraki et al., T-RA 1996]
  - Multiple query



[Credit: LaValle, Planning Algorithms 2006]

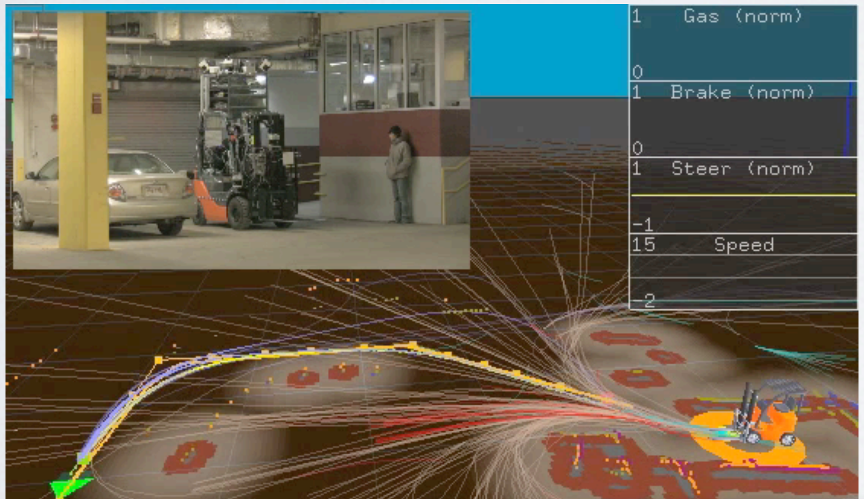
- Rapidly-exploring Random Tree (RRT)
  [LaValle & Kufner, IJRR 2001]
  - Single query
  - Incremental
  - Online



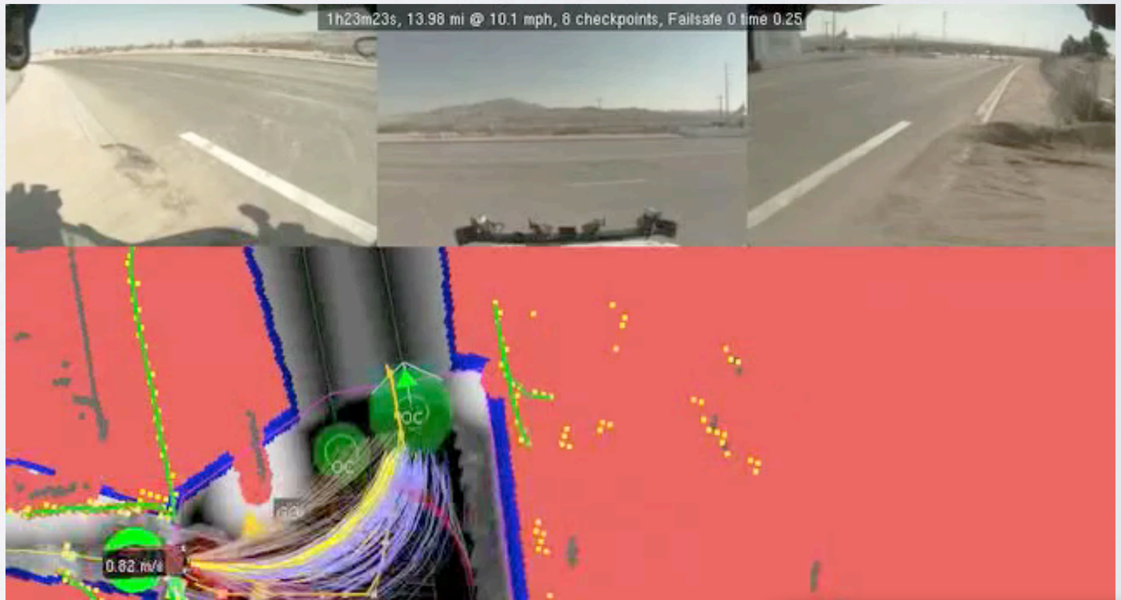
INCREMENTAL SAMPLE-BASED MOTION PLANNING

- Rapidly-exploring Random Tree (RRT) [LaValle & Kufner, IJRR 2001]
  - Probabilistically complete
  - Respects kinodynamic (non-holonomic) constraints
  - Computationally efficient, scales to high dimensions
  - Relatively simple to implement

Effectively demonstrated on state-of-the-art robotic platforms



video: 2009_02_agile_rrt.mp4



video: 2007_11_duc_25_mph.mp4

ANYTIME MOTION PLANNING

During execution, improve solution toward optimal

- Overall approach:

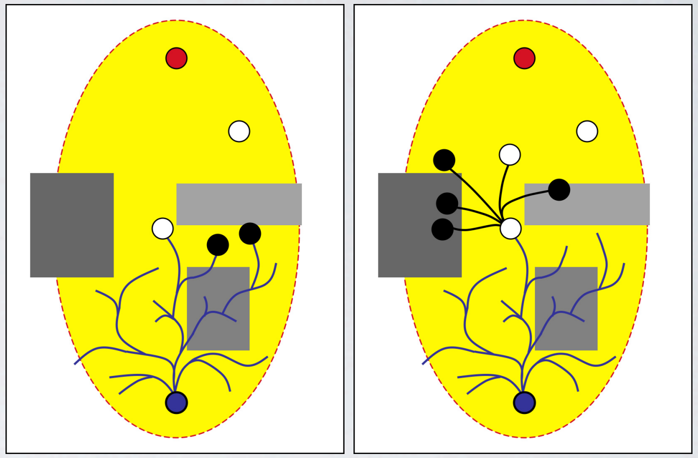
  1. Quickly find a solution that is feasible, but not necessarily optimal
  2. Exploit execution time to incrementally improve towards optimal solution

- Desired properties:

  1. Form of completeness guarantees
  2. Asymptotic optimality given more computation time

ANYTIME MOTION PLANNING

- Anytime RRTs [Ferguson & Stentz, IROS 2006]
  - Quickly finds an initial solution with a vanilla RRT
  - Successively generates new trees that improve solution costs via biased sampling
  - The cost of successive solutions is guaranteed to decrease, though they **do not converge to the optimum**
- CL-RRT [Kuwata et al., T-CST 2009]
  - Quickly finds an initial solution with a closed-loop RRT
  - Continues to search for other solutions (during execution)
  - Estimates an upper-bound on the cost of each solution via a cost-to-go heuristic
  - Chooses the solution with the lowest upper-bound cost
  - **No convergence guarantees**



[Credit: Ferguson & Stentz, IROS 2006]

ANYTIME MOTION PLANNING

The RRT is not asymptotically optimal

- Y_n^{RRT} denotes the cost of the best path in the RRT after n iterations
- c^* denotes the cost of an optimal path

Theorem [Karaman, Frazzoli, IJRR 2011]
The probability that the RRT converges to an optimum solution is zero

$$\mathbb{P}\Big(\big\{\lim_{n\to\infty} Y_n^{\mathrm{RRT}} = c^*\big\}\Big) = 0$$

ANYTIME RRT*

Our approach: Leverage the RRT* to converge to optimal

- RRT* [Karaman & Frazzoli, RSS 2010] is both **asymptotically optimal** and **computationally efficient**
  - $Y_n^{\mathrm{RRT}^*}$: cost of the best path in the RRT*
  - c^* : cost of an optimal solution
  - M_n^{RRT} : number of steps executed by the RRT at iteration n
  - $M_n^{\mathrm{RRT}^*}$: number of steps executed by the RRT* in iteration n

Theorem [Karaman & Frazzoli, IJRR 2011]

(i) The RRT* algorithm is asymptotically optimal

$$\mathbb{P}\Big(\big\{ \lim_{n\to\infty} Y_n^{\mathrm{RRT}^*} = c^* \big\}\Big) = 1$$

(ii) RRT* algorithm has no substantial computational overhead when compared to the RRT:

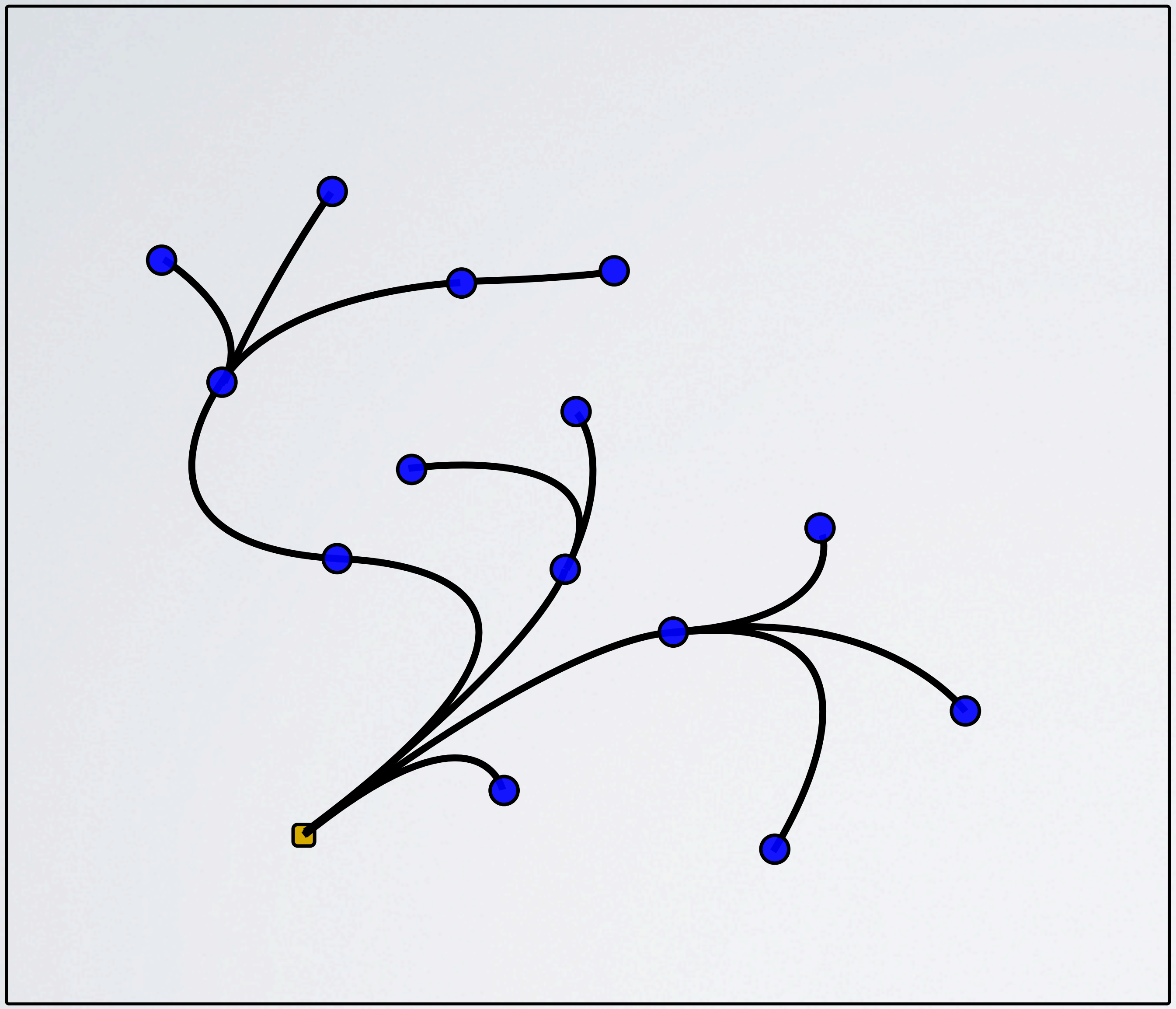
$$\lim_{n\to\infty} \mathbb{E}\left[\frac{M_n^{\mathrm{RRT}^*}}{M_n^{\mathrm{RRT}}}\right] = \mathrm{constant}$$

ANYTIME RRT*

Our approach: Leverage the RRT* to converge to optimal

- Closed-loop formulation of the RRT*
  - Samples from the space of control inputs utilizing a prediction model to grow the tree
- Quickly find a feasible, possibly sub-optimal solution
- Exploit available computation time during execution to rewire the tree
- Introduce heuristics for online implementation and efficiency
  - Committed trajectory
  - Branch-and-bound

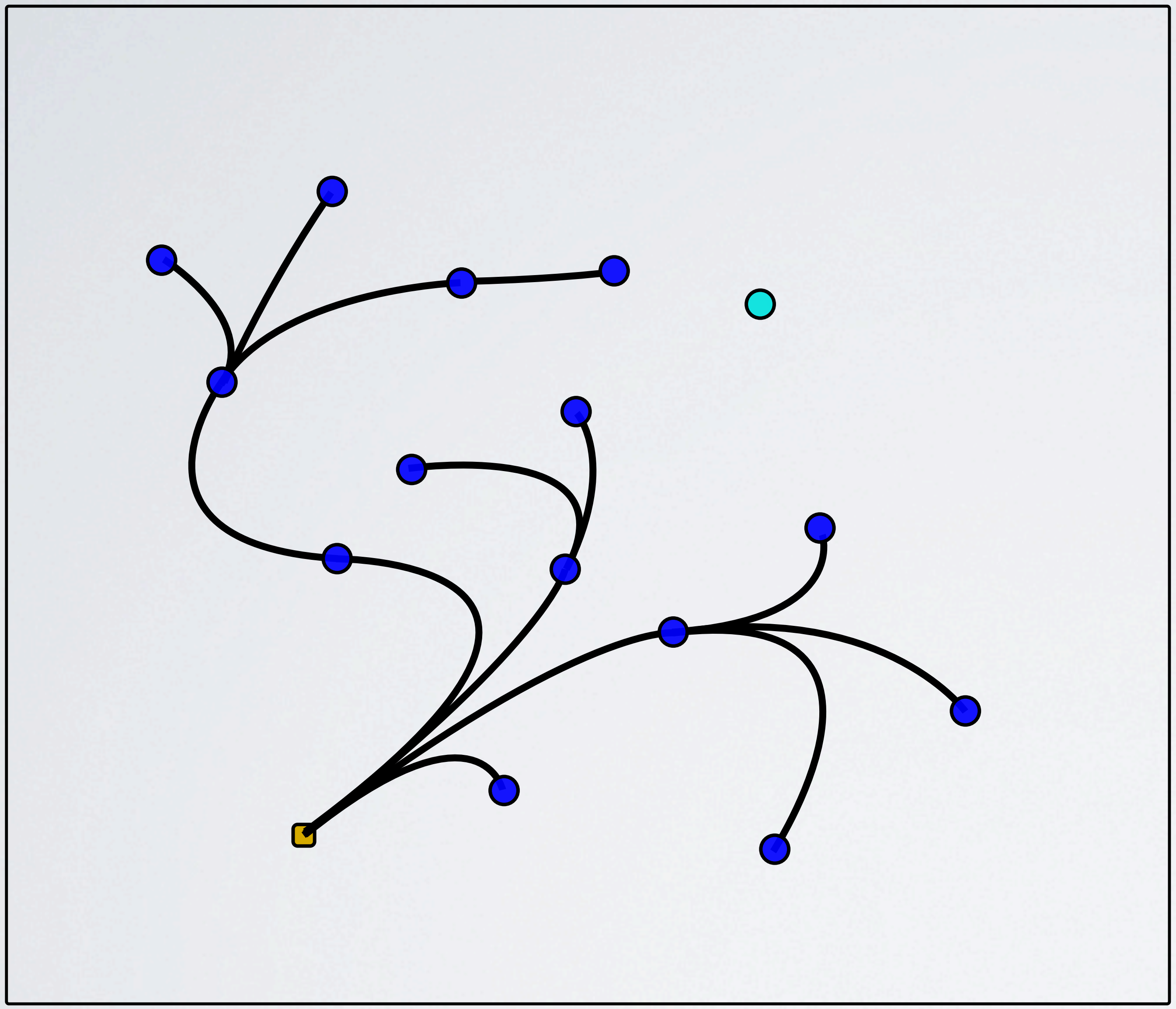
THE RRT*



$\mathcal{T} \leftarrow \texttt{InitializeTree}();$
$\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, z_{\text{init}}, \mathcal{T});$
for $i = 1$ *to* $i = N$ **do**
    $z_{\text{rand}} \leftarrow \texttt{Sample}(i);$
    $z_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, z_{\text{rand}});$
    $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}});$
    if $\texttt{ObstacleFree}(x_{new})$ **then**
        $Z_{\text{near}} \leftarrow \texttt{Near}(\mathcal{T}, z_{\text{new}}, |V|);$
        $z_{\text{min}} \leftarrow \texttt{ChooseParent}(Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}});$
        $\mathcal{T} \leftarrow \texttt{InsertNode}(z_{\text{min}}, z_{\text{new}}, \mathcal{T});$
        $\mathcal{T} \leftarrow \texttt{ReWire}(\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}});$
return $\mathcal{T}$

So far, a standard RRT

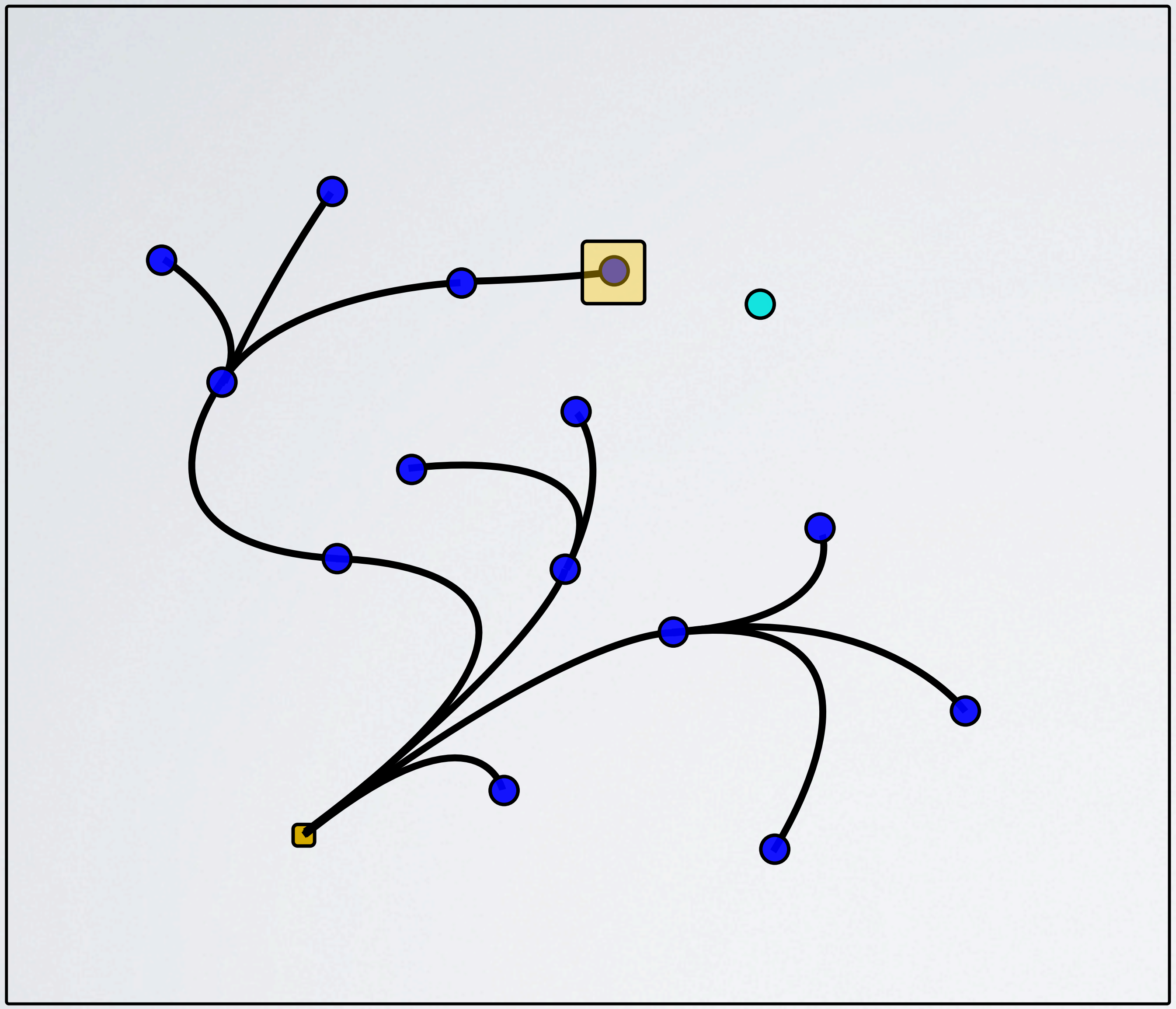
THE RRT*



```
T ← InitializeTree();
T ← InsertNode(∅, z_init, T);
for i = 1 to i = N do
    z_rand ← Sample(i);
    z_nearest ← Nearest(T, z_rand);
    (x_new, u_new, T_new) ← Steer(z_nearest, z_rand);
    if ObstacleFree(x_new) then
        Z_near ← Near(T, z_new, |V|);
        z_min ← ChooseParent(Z_near, z_nearest, z_new, x_new);
        T ← InsertNode(z_min, z_new, T);
        T ← ReWire(T, Z_near, z_min, z_new);
return T
```

So far, a standard RRT

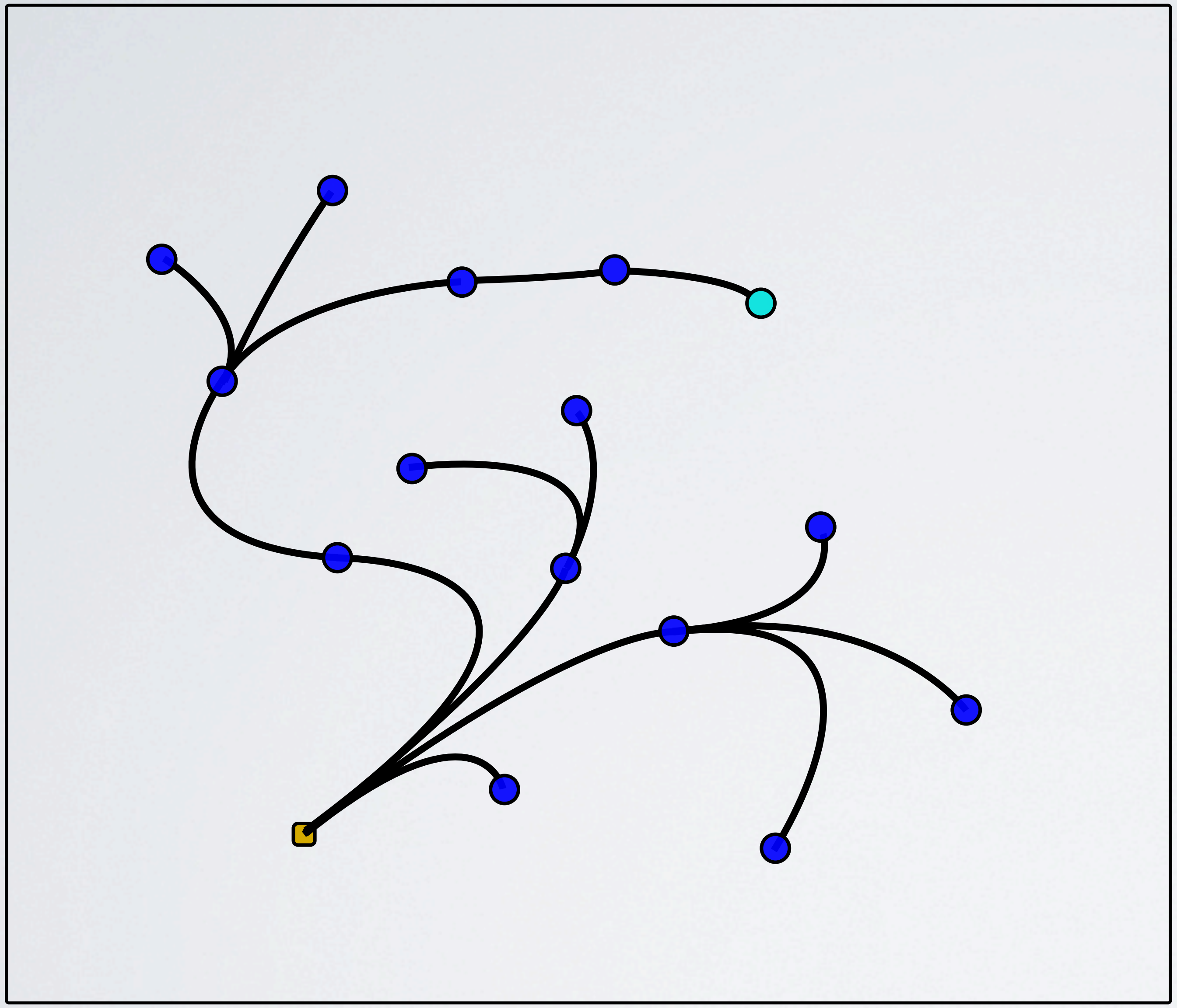
THE RRT*



$\mathcal{T} \leftarrow$ `InitializeTree`();
$\mathcal{T} \leftarrow$ `InsertNode`($\emptyset, z_{\text{init}}, \mathcal{T}$);
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow$ `Sample`(i);
  $z_{\text{nearest}} \leftarrow$ `Nearest`($\mathcal{T}, z_{\text{rand}}$);
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow$ `Steer`($z_{\text{nearest}}, z_{\text{rand}}$);
  if `ObstacleFree`(x_{new}) **then**
    $Z_{\text{near}} \leftarrow$ `Near`($\mathcal{T}, z_{\text{new}}, |V|$);
    $z_{\text{min}} \leftarrow$ `ChooseParent`($Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}}$);
    $\mathcal{T} \leftarrow$ `InsertNode`($z_{\text{min}}, z_{\text{new}}, \mathcal{T}$);
    $\mathcal{T} \leftarrow$ `ReWire`($\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}}$);
return $\mathcal{T}$

So far, a standard RRT

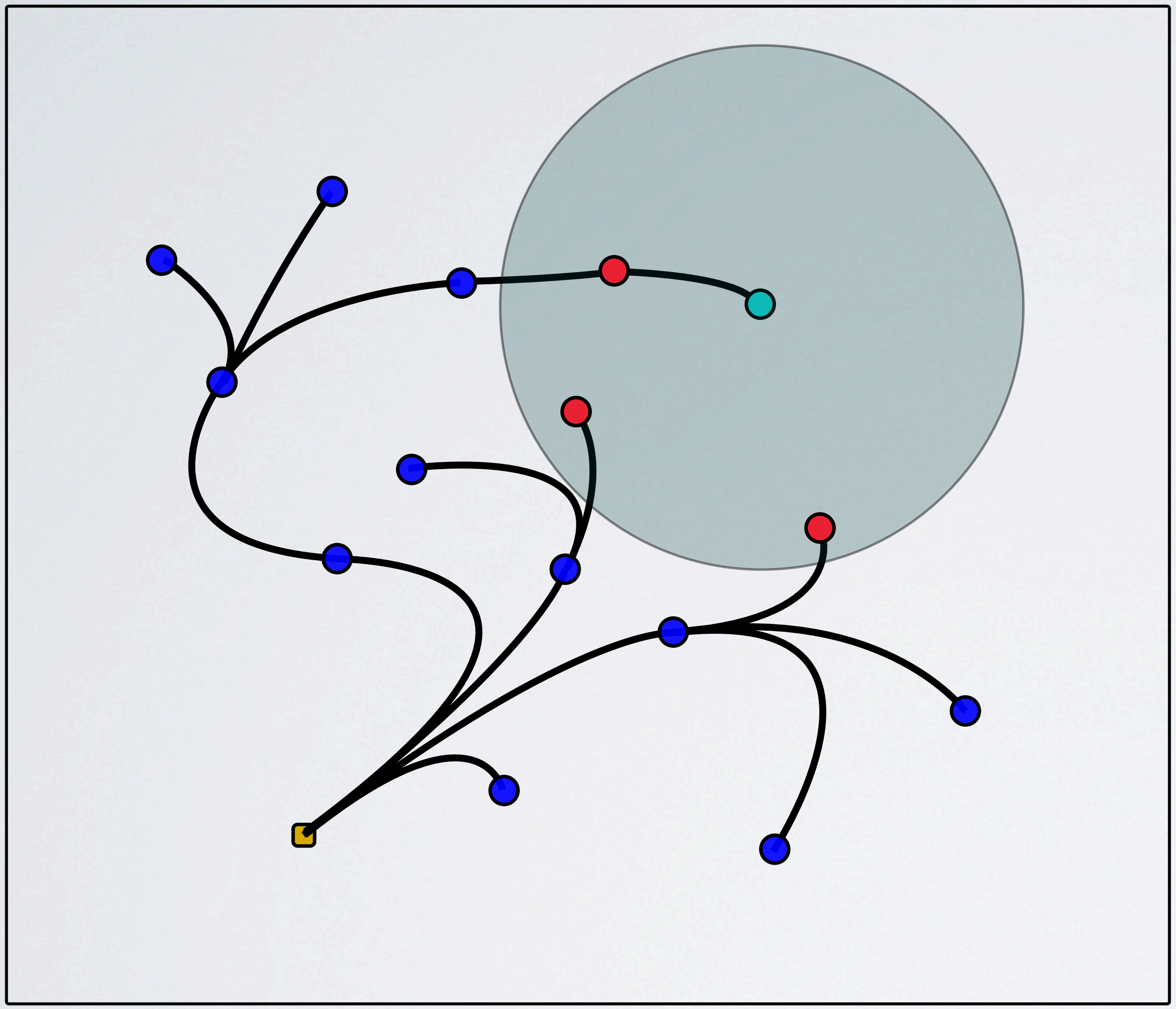
THE RRT*



```
T ← InitializeTree();
T ← InsertNode(∅, z_init, T);
for i = 1 to i = N do
    z_rand ← Sample(i);
    z_nearest ← Nearest(T, z_rand);
    (x_new, u_new, T_new) ← Steer(z_nearest, z_rand);
    if ObstacleFree(x_new) then
        Z_near ← Near(T, z_new, |V|);
        z_min ← ChooseParent(Z_near, z_nearest, z_new, x_new);
        T ← InsertNode(z_min, z_new, T);
        T ← ReWire(T, Z_near, z_min, z_new);
return T
```

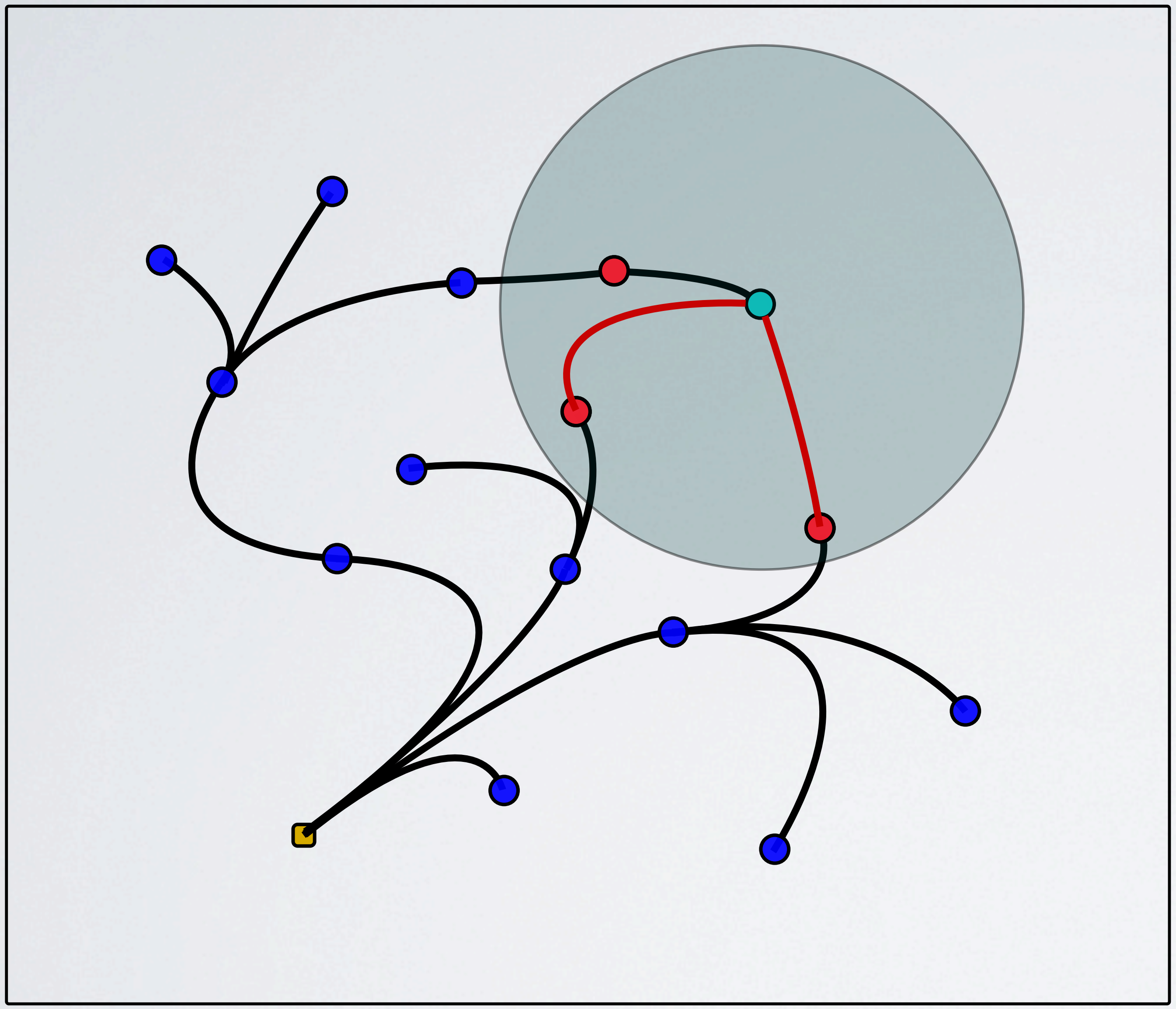
So far, a standard RRT

THE RRT*



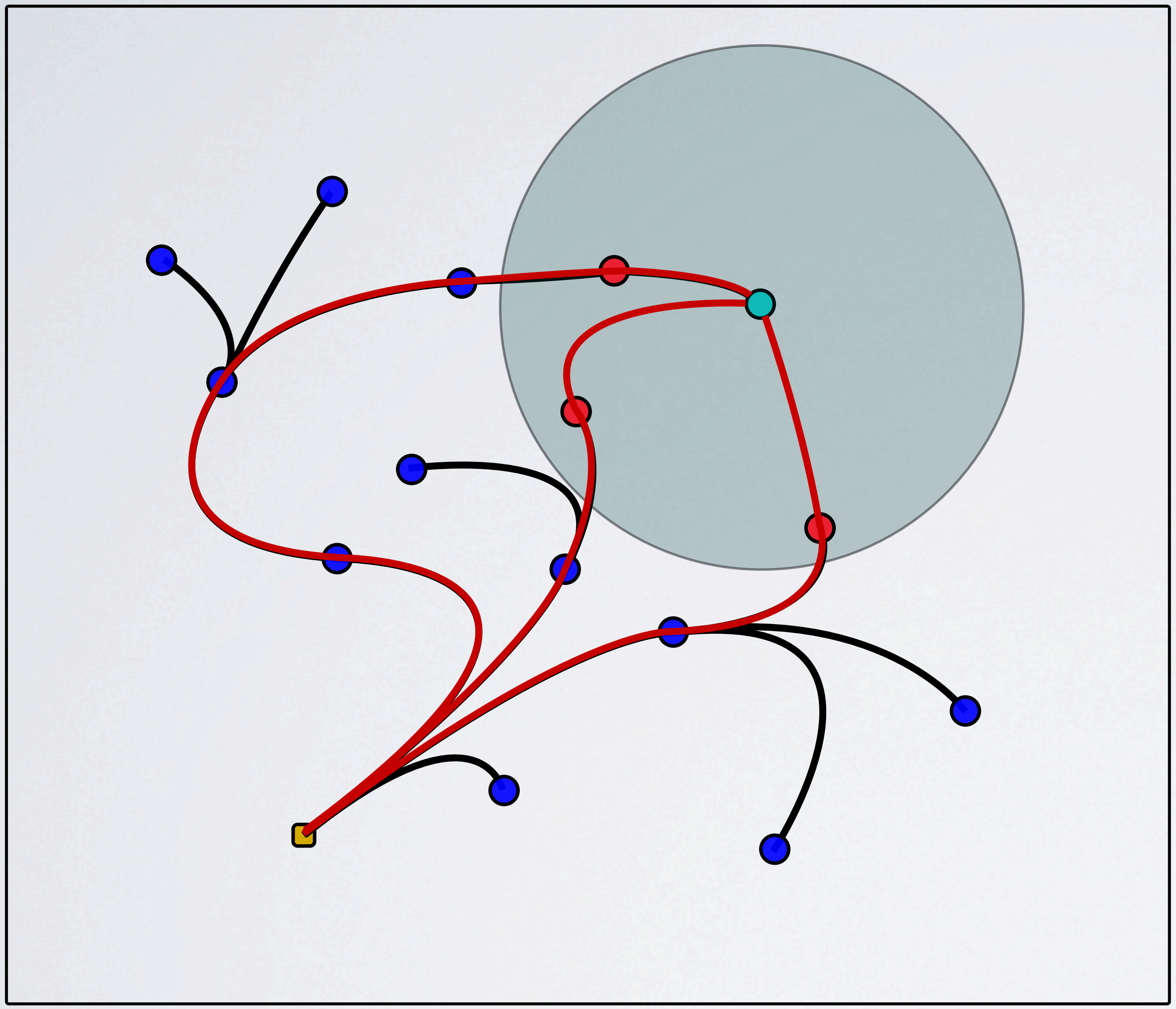
$\mathcal{T} \leftarrow \texttt{InitializeTree}()$;
$\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, z_{\text{init}}, \mathcal{T})$;
for $i = 1$ *to* $i = N$ **do**
    $z_{\text{rand}} \leftarrow \texttt{Sample}(i)$;
    $z_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, z_{\text{rand}})$;
    $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
    if $\texttt{ObstacleFree}(x_{new})$ **then**
        $Z_{\text{near}} \leftarrow \texttt{Near}(\mathcal{T}, z_{\text{new}}, |V|)$;
        $z_{\text{min}} \leftarrow \texttt{ChooseParent}(Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}})$;
        $\mathcal{T} \leftarrow \texttt{InsertNode}(z_{\text{min}}, z_{\text{new}}, \mathcal{T})$;
        $\mathcal{T} \leftarrow \texttt{ReWire}(\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}})$;
return $\mathcal{T}$

THE RRT*



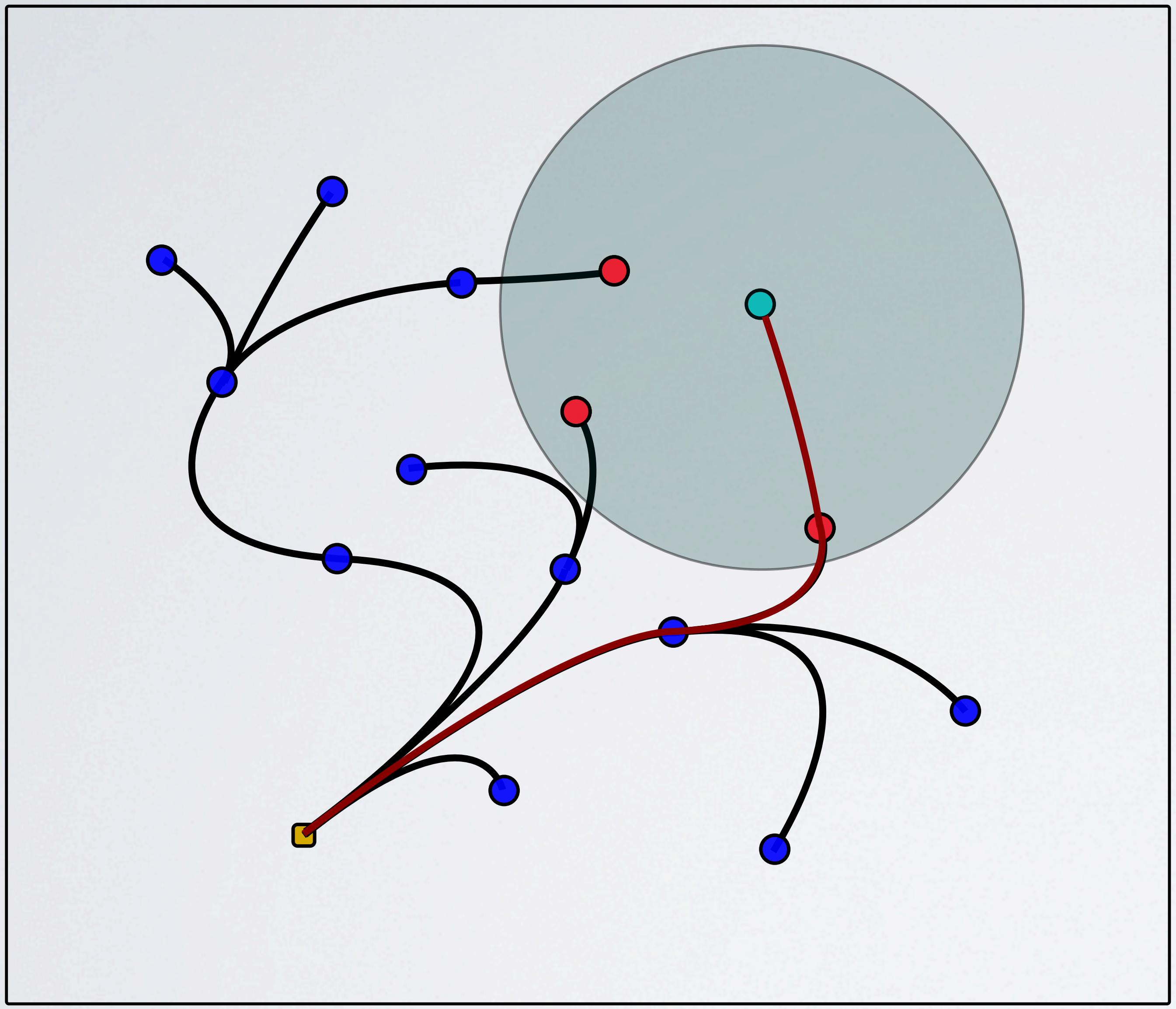
$\mathcal{T} \leftarrow$ InitializeTree();
$\mathcal{T} \leftarrow$ InsertNode($\emptyset, z_{\text{init}}, \mathcal{T}$);
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow$ Sample(i);
  $z_{\text{nearest}} \leftarrow$ Nearest($\mathcal{T}, z_{\text{rand}}$);
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow$ Steer($z_{\text{nearest}}, z_{\text{rand}}$);
  if ObstacleFree(x_{new}) **then**
    $Z_{\text{near}} \leftarrow$ Near($\mathcal{T}, z_{\text{new}}, |V|$);
    $z_{\text{min}} \leftarrow$ ChooseParent($Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}}$);
    $\mathcal{T} \leftarrow$ InsertNode($z_{\text{min}}, z_{\text{new}}, \mathcal{T}$);
    $\mathcal{T} \leftarrow$ ReWire($\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}}$);
return $\mathcal{T}$

THE RRT*



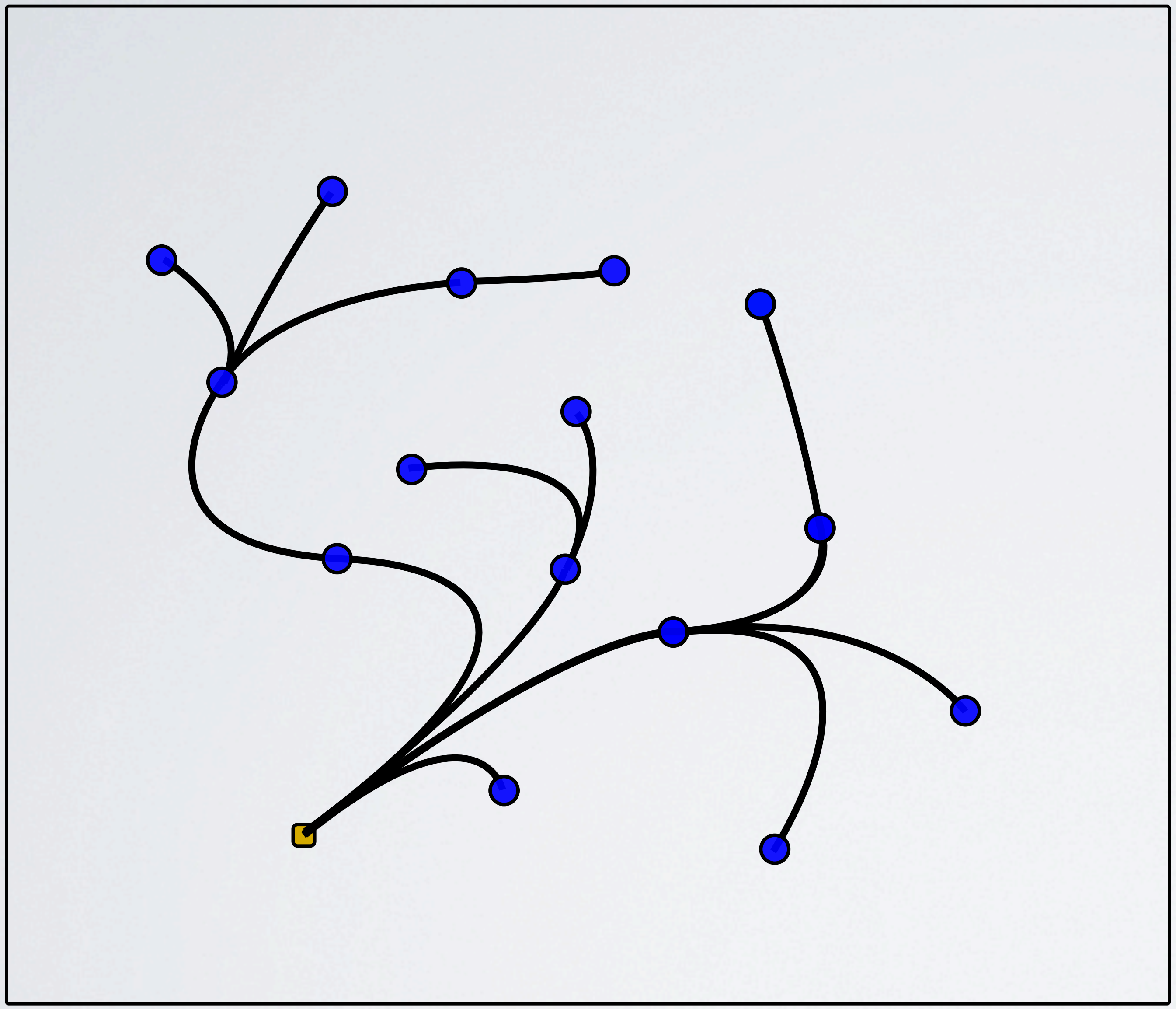
$\mathcal{T} \leftarrow \texttt{InitializeTree}()$;
$\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, z_{\text{init}}, \mathcal{T})$;
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow \texttt{Sample}(i)$;
  $z_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, z_{\text{rand}})$;
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
  if $\texttt{ObstacleFree}(x_{new})$ **then**
    $Z_{\text{near}} \leftarrow \texttt{Near}(\mathcal{T}, z_{\text{new}}, |V|)$;
    $z_{\text{min}} \leftarrow \texttt{ChooseParent}(Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}})$;
    $\mathcal{T} \leftarrow \texttt{InsertNode}(z_{\text{min}}, z_{\text{new}}, \mathcal{T})$;
    $\mathcal{T} \leftarrow \texttt{ReWire}(\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}})$;
return $\mathcal{T}$

THE RRT*



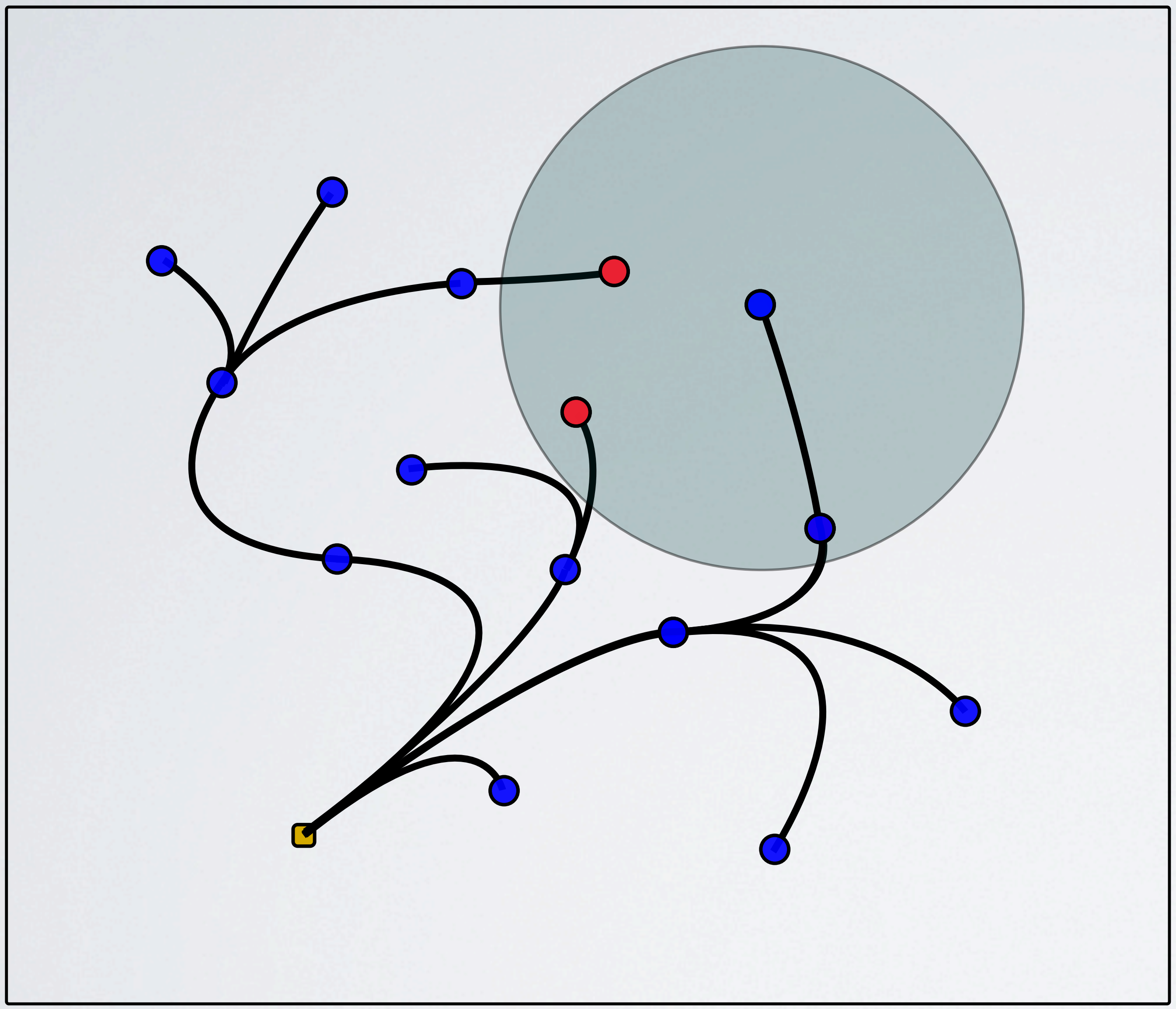
```
T ← InitializeTree();
T ← InsertNode(∅, z_init, T);
for i = 1 to i = N do
    z_rand ← Sample(i);
    z_nearest ← Nearest(T, z_rand);
    (x_new, u_new, T_new) ← Steer(z_nearest, z_rand);
    if ObstacleFree(x_new) then
        Z_near ← Near(T, z_new, |V|);
        z_min ← ChooseParent(Z_near, z_nearest, z_new, x_new);
        T ← InsertNode(z_min, z_new, T);
        T ← ReWire(T, Z_near, z_min, z_new);
return T
```

THE RRT*



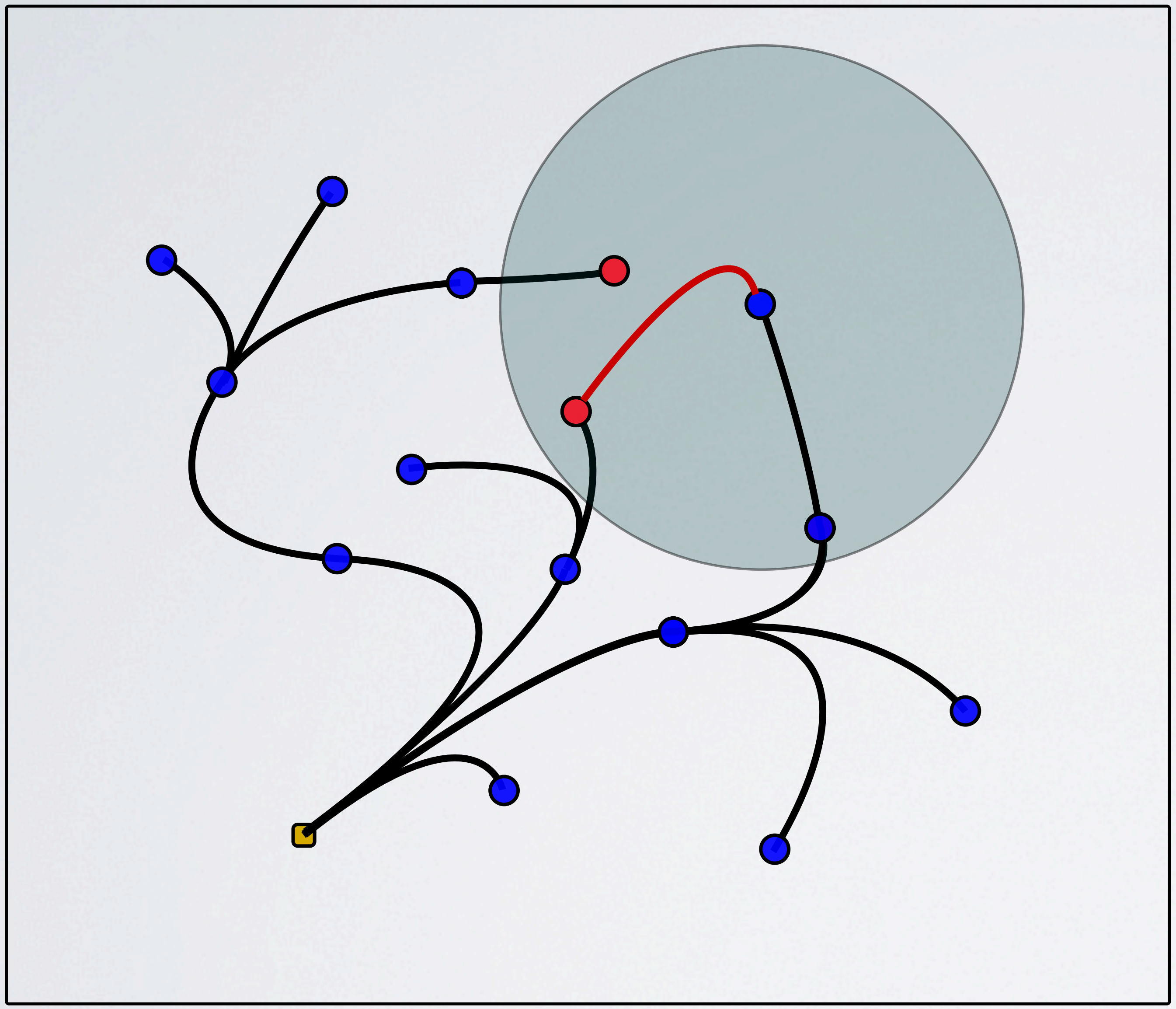
$\mathcal{T} \leftarrow \texttt{InitializeTree}()$;
$\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, z_{\text{init}}, \mathcal{T})$;
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow \texttt{Sample}(i)$;
  $z_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, z_{\text{rand}})$;
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
  if $\texttt{ObstacleFree}(x_{new})$ **then**
    $Z_{\text{near}} \leftarrow \texttt{Near}(\mathcal{T}, z_{\text{new}}, |V|)$;
    $z_{\text{min}} \leftarrow \texttt{ChooseParent}(Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}})$;
    $\mathcal{T} \leftarrow \texttt{InsertNode}(z_{\text{min}}, z_{\text{new}}, \mathcal{T})$;
    $\mathcal{T} \leftarrow \texttt{ReWire}(\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}})$;
return $\mathcal{T}$

THE RRT*



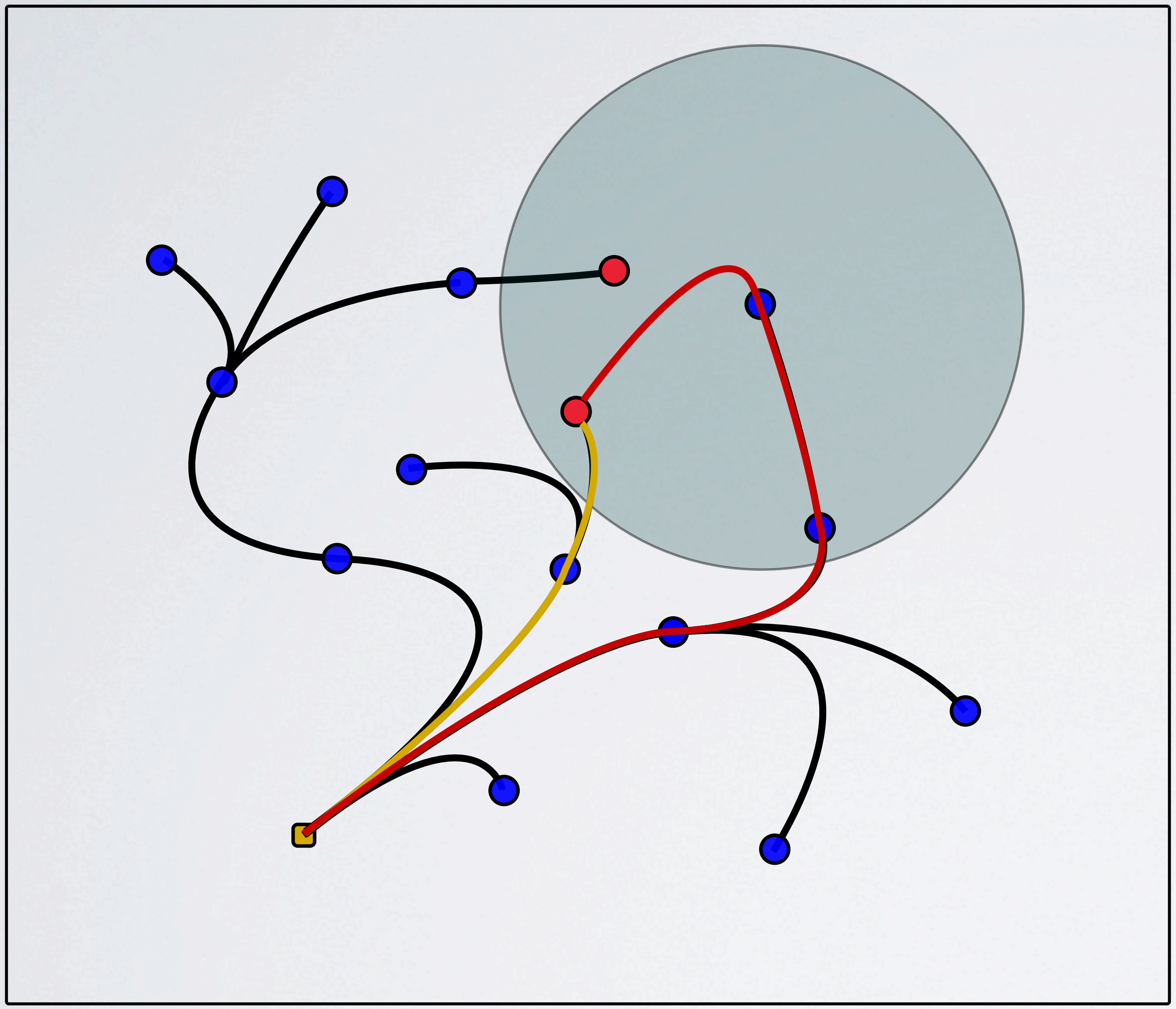
$\mathcal{T} \leftarrow$ `InitializeTree()`;
$\mathcal{T} \leftarrow$ `InsertNode`$(\emptyset, z_{\text{init}}, \mathcal{T})$;
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow$ `Sample`(i);
  $z_{\text{nearest}} \leftarrow$ `Nearest`$(\mathcal{T}, z_{\text{rand}})$;
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow$ `Steer`$(z_{\text{nearest}}, z_{\text{rand}})$;
  if `ObstacleFree`(x_{new}) **then**
    $Z_{\text{near}} \leftarrow$ `Near`$(\mathcal{T}, z_{\text{new}}, |V|)$;
    $z_{\text{min}} \leftarrow$ `ChooseParent`$(Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}})$;
    $\mathcal{T} \leftarrow$ `InsertNode`$(z_{\text{min}}, z_{\text{new}}, \mathcal{T})$;
    $\mathcal{T} \leftarrow$ `ReWire`$(\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}})$;
return $\mathcal{T}$

THE RRT*



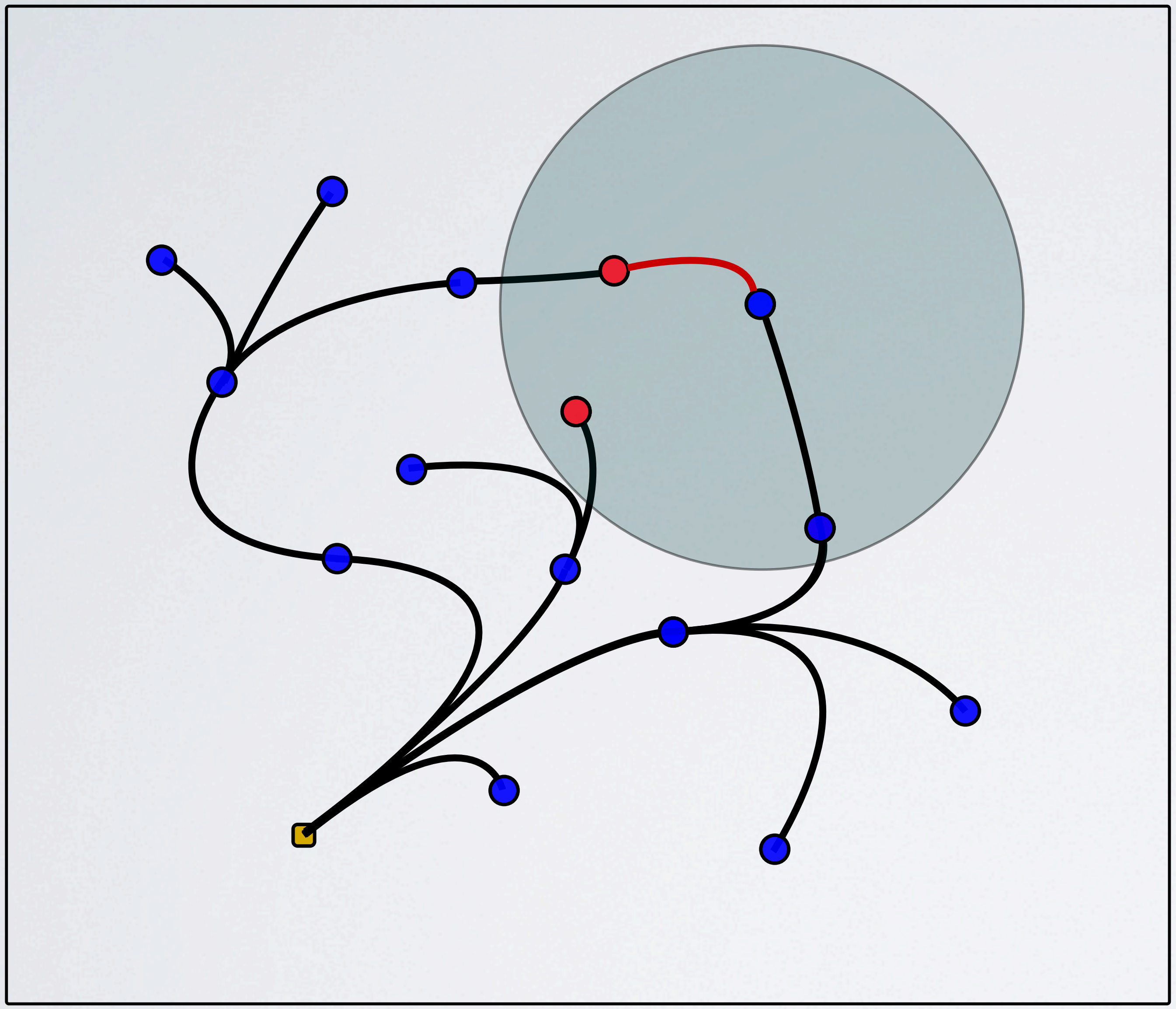
$\mathcal{T} \leftarrow$ `InitializeTree`();
$\mathcal{T} \leftarrow$ `InsertNode`($\emptyset, z_{\text{init}}, \mathcal{T}$);
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow$ `Sample`(i);
  $z_{\text{nearest}} \leftarrow$ `Nearest`($\mathcal{T}, z_{\text{rand}}$);
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow$ `Steer`($z_{\text{nearest}}, z_{\text{rand}}$);
  if `ObstacleFree`(x_{new}) **then**
    $Z_{\text{near}} \leftarrow$ `Near`($\mathcal{T}, z_{\text{new}}, |V|$);
    $z_{\text{min}} \leftarrow$ `ChooseParent`($Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}}$);
    $\mathcal{T} \leftarrow$ `InsertNode`($z_{\text{min}}, z_{\text{new}}, \mathcal{T}$);
    $\mathcal{T} \leftarrow$ `ReWire`($\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}}$);
return $\mathcal{T}$

THE RRT*



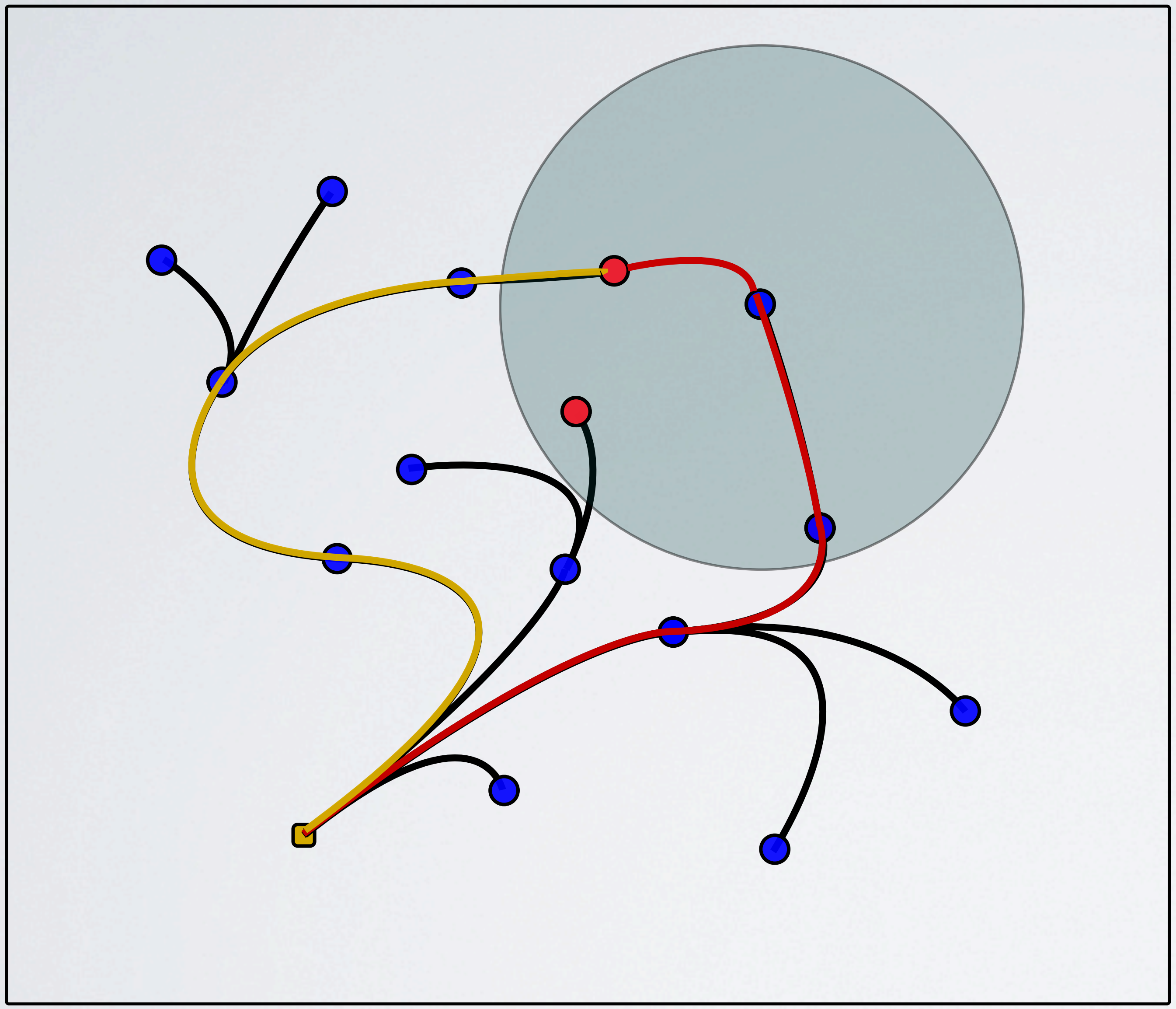
$\mathcal{T} \leftarrow \texttt{InitializeTree}()$;
$\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, z_{\text{init}}, \mathcal{T})$;
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow \texttt{Sample}(i)$;
  $z_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, z_{\text{rand}})$;
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
  if $\texttt{ObstacleFree}(x_{new})$ **then**
    $Z_{\text{near}} \leftarrow \texttt{Near}(\mathcal{T}, z_{\text{new}}, |V|)$;
    $z_{\text{min}} \leftarrow \texttt{ChooseParent}(Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}})$;
    $\mathcal{T} \leftarrow \texttt{InsertNode}(z_{\text{min}}, z_{\text{new}}, \mathcal{T})$;
    $\mathcal{T} \leftarrow \texttt{ReWire}(\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}})$;
return $\mathcal{T}$

THE RRT*



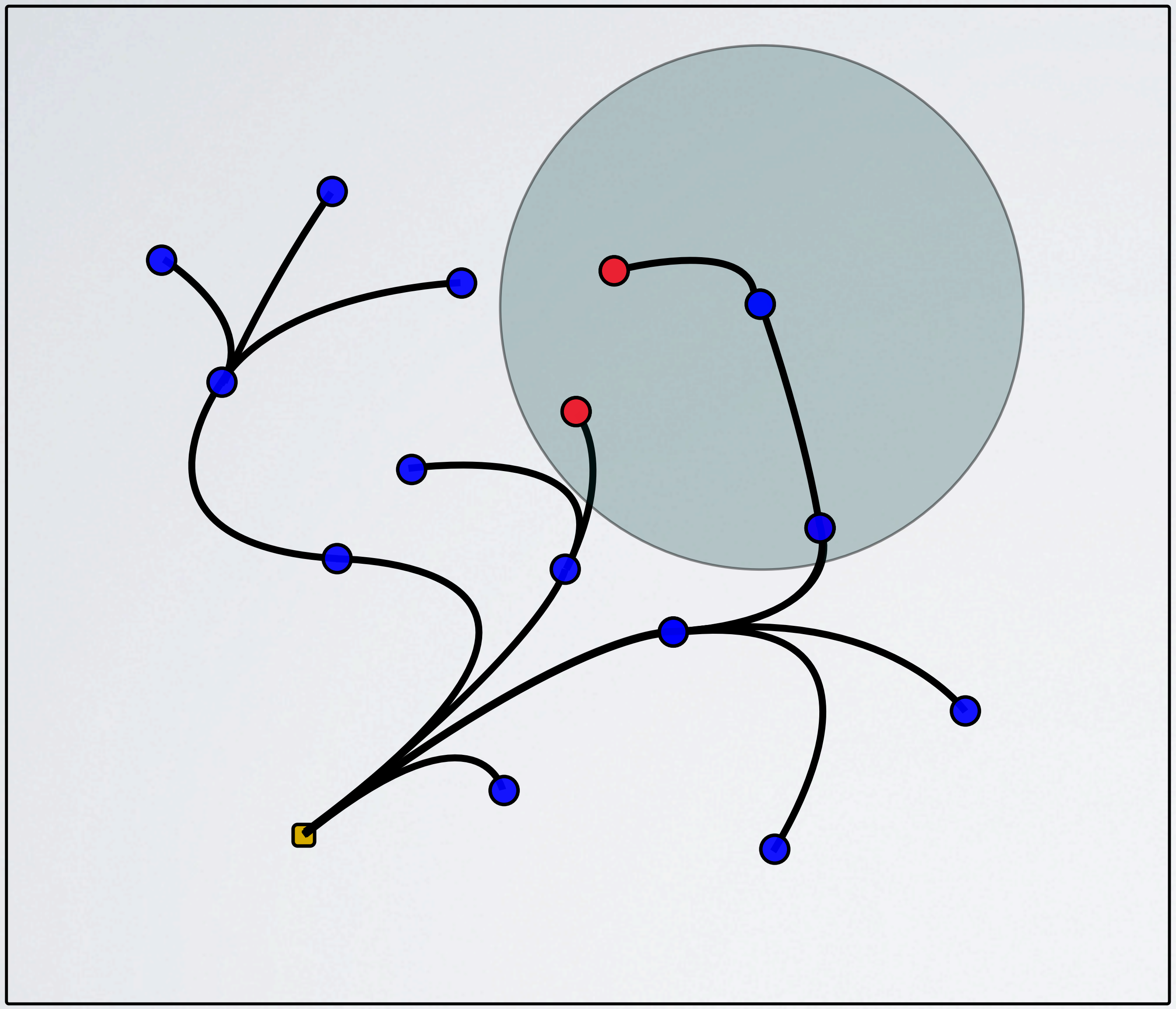
$\mathcal{T} \leftarrow \texttt{InitializeTree}()$;
$\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, z_{\text{init}}, \mathcal{T})$;
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow \texttt{Sample}(i)$;
  $z_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, z_{\text{rand}})$;
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
  if $\texttt{ObstacleFree}(x_{new})$ **then**
    $Z_{\text{near}} \leftarrow \texttt{Near}(\mathcal{T}, z_{\text{new}}, |V|)$;
    $z_{\text{min}} \leftarrow \texttt{ChooseParent}(Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}})$;
    $\mathcal{T} \leftarrow \texttt{InsertNode}(z_{\text{min}}, z_{\text{new}}, \mathcal{T})$;
    $\mathcal{T} \leftarrow \texttt{ReWire}(\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}})$;
return $\mathcal{T}$

THE RRT*



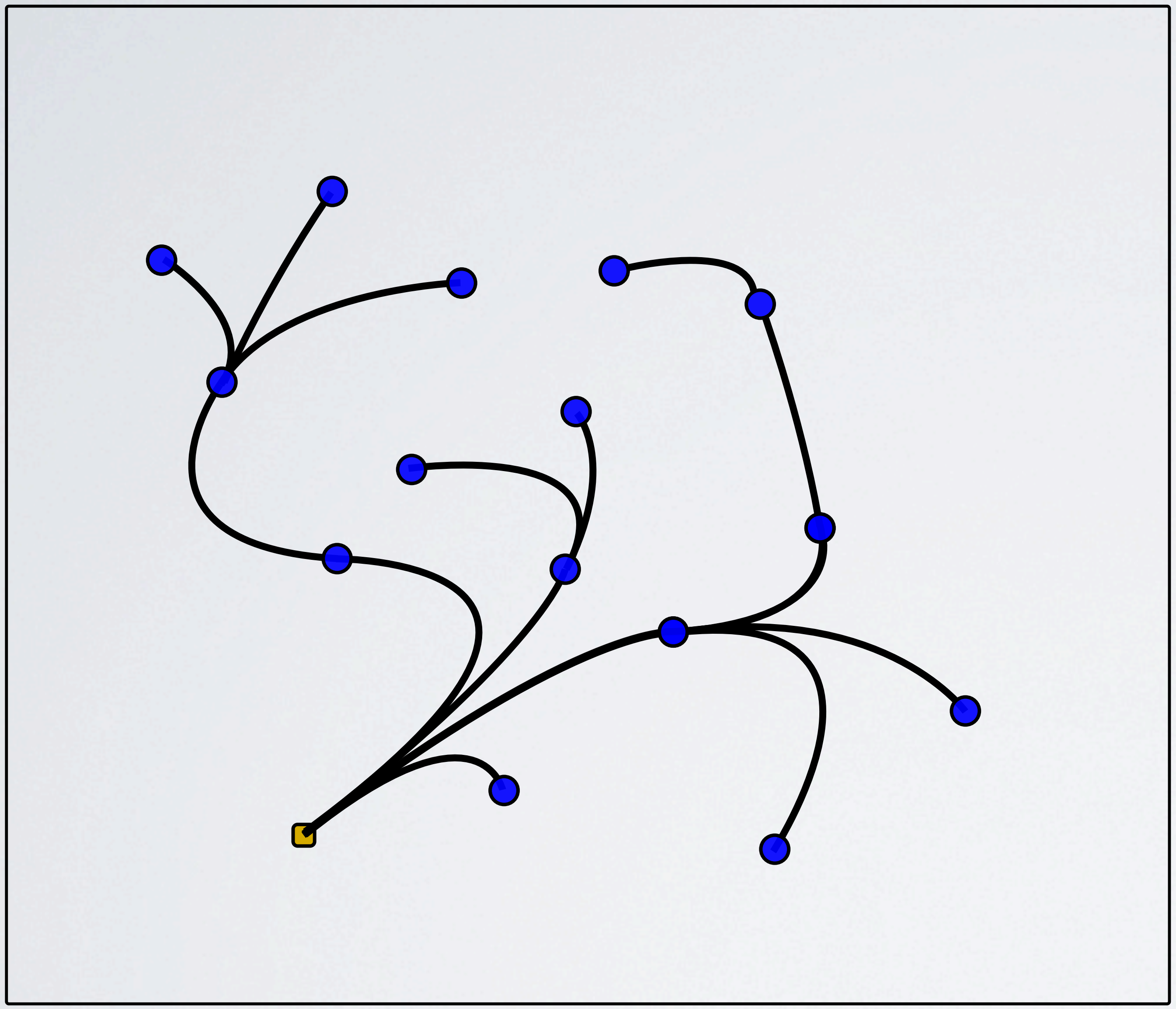
$\mathcal{T} \leftarrow$ `InitializeTree`();
$\mathcal{T} \leftarrow$ `InsertNode`($\emptyset, z_{\text{init}}, \mathcal{T}$);
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow$ `Sample`(i);
  $z_{\text{nearest}} \leftarrow$ `Nearest`($\mathcal{T}, z_{\text{rand}}$);
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow$ `Steer`($z_{\text{nearest}}, z_{\text{rand}}$);
  if `ObstacleFree`(x_{new}) **then**
    $Z_{\text{near}} \leftarrow$ `Near`($\mathcal{T}, z_{\text{new}}, |V|$);
    $z_{\text{min}} \leftarrow$ `ChooseParent`($Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}}$);
    $\mathcal{T} \leftarrow$ `InsertNode`($z_{\text{min}}, z_{\text{new}}, \mathcal{T}$);
    $\mathcal{T} \leftarrow$ `ReWire`($\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}}$);
return $\mathcal{T}$

THE RRT*



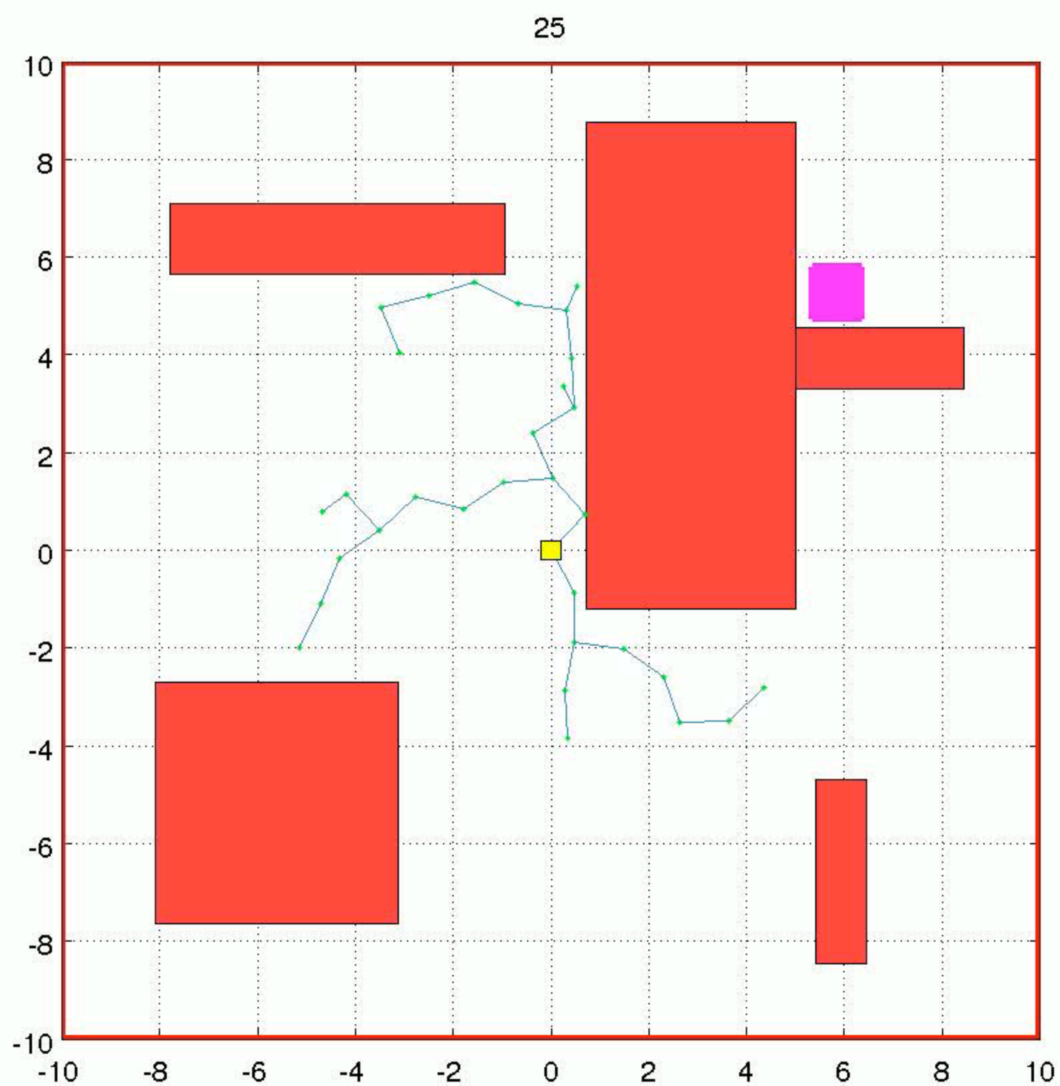
$\mathcal{T} \leftarrow \texttt{InitializeTree}()$;
$\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, z_{\text{init}}, \mathcal{T})$;
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow \texttt{Sample}(i)$;
  $z_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, z_{\text{rand}})$;
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
  if $\texttt{ObstacleFree}(x_{new})$ **then**
    $Z_{\text{near}} \leftarrow \texttt{Near}(\mathcal{T}, z_{\text{new}}, |V|)$;
    $z_{\text{min}} \leftarrow \texttt{ChooseParent}(Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}})$;
    $\mathcal{T} \leftarrow \texttt{InsertNode}(z_{\text{min}}, z_{\text{new}}, \mathcal{T})$;
    $\mathcal{T} \leftarrow \texttt{ReWire}(\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}})$;
return $\mathcal{T}$

THE RRT*

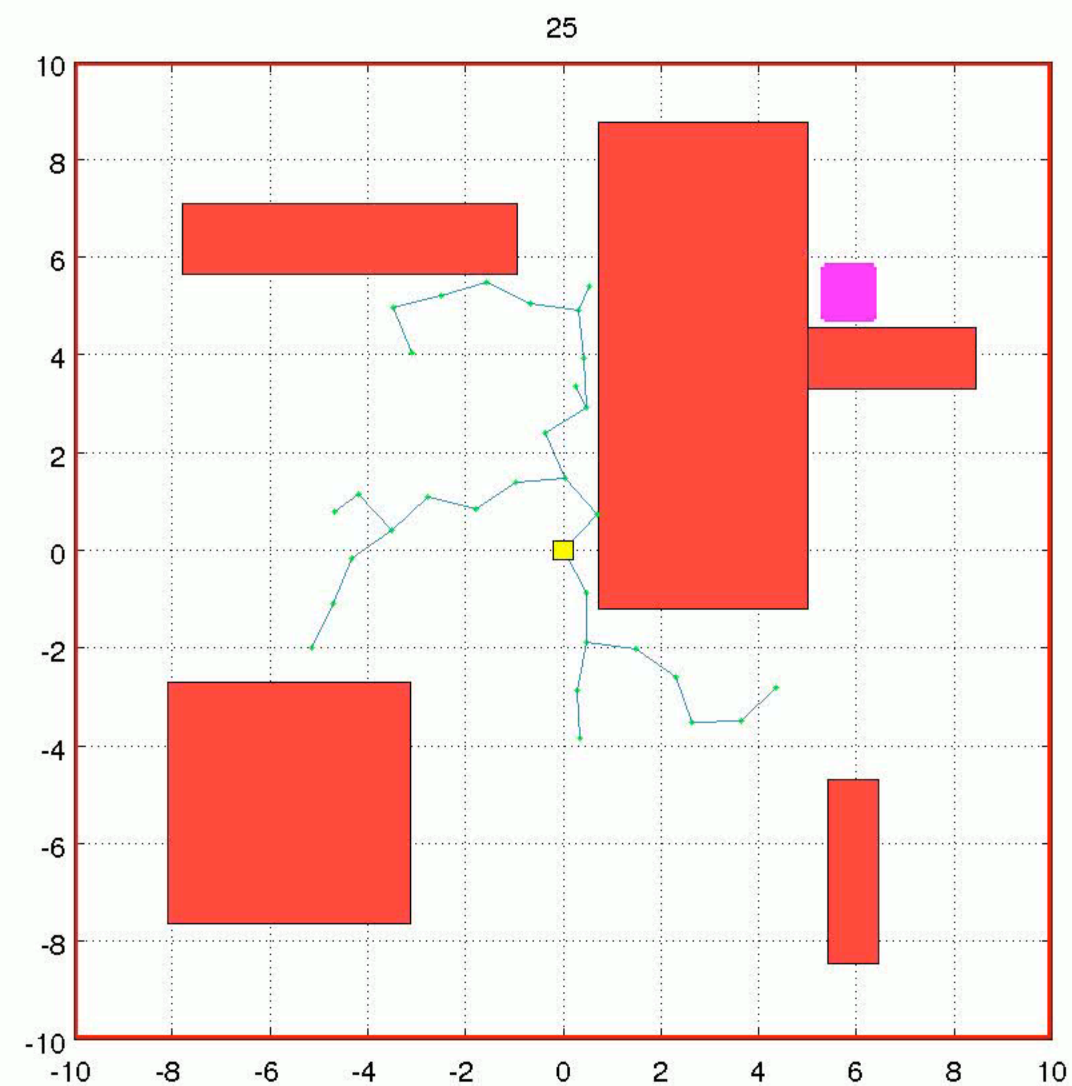


$\mathcal{T} \leftarrow$ `InitializeTree`();
$\mathcal{T} \leftarrow$ `InsertNode`($\emptyset, z_{\text{init}}, \mathcal{T}$);
for $i = 1$ *to* $i = N$ **do**
  $z_{\text{rand}} \leftarrow$ `Sample`(i);
  $z_{\text{nearest}} \leftarrow$ `Nearest`($\mathcal{T}, z_{\text{rand}}$);
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow$ `Steer`($z_{\text{nearest}}, z_{\text{rand}}$);
  if `ObstacleFree`(x_{new}) **then**
    $Z_{\text{near}} \leftarrow$ `Near`($\mathcal{T}, z_{\text{new}}, |V|$);
    $z_{\text{min}} \leftarrow$ `ChooseParent`($Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}}$);
    $\mathcal{T} \leftarrow$ `InsertNode`($z_{\text{min}}, z_{\text{new}}, \mathcal{T}$);
    $\mathcal{T} \leftarrow$ `ReWire`($\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}}$);
return $\mathcal{T}$

THE RRT*



RRT



RRT*

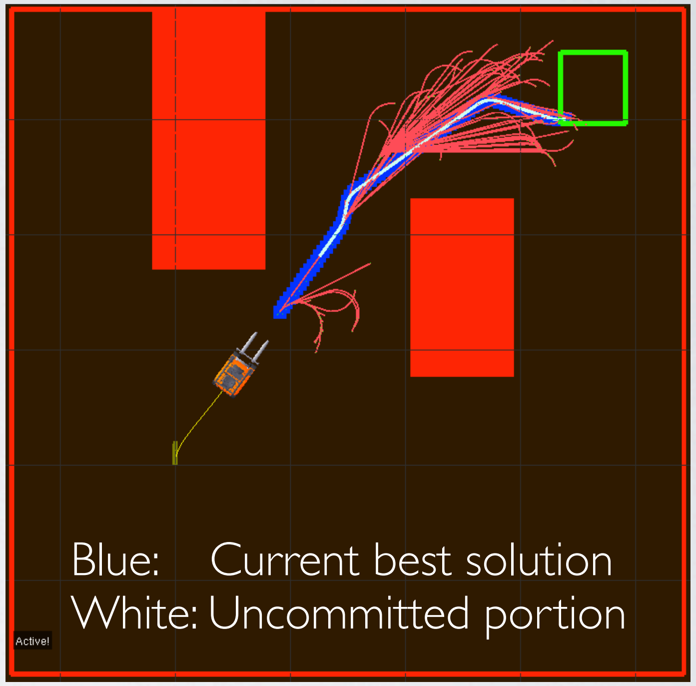
video: 2011_05_10_icra_rrt.mp4

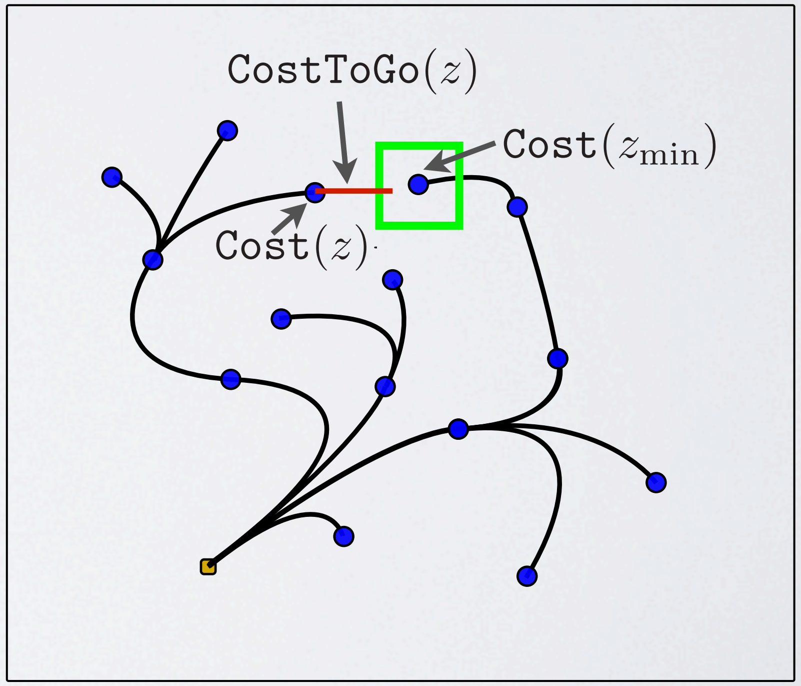
video: 2011_05_10_icra_rrtstar.mp4

ANYTIME EXTENSIONS

- Committed trajectory
  - Robot commits to execute immediate portion of current solution
  - Delete branches off committed trajectory, making the endpoint the new tree root
  - The planner improves paths (rewires) beyond committed trajectory
- Branch-and-bound
  - Maintain a lower-bound on the cost to get to the goal from each node in the tree (e.g., Euclidean distance)
  - Delete nodes for which

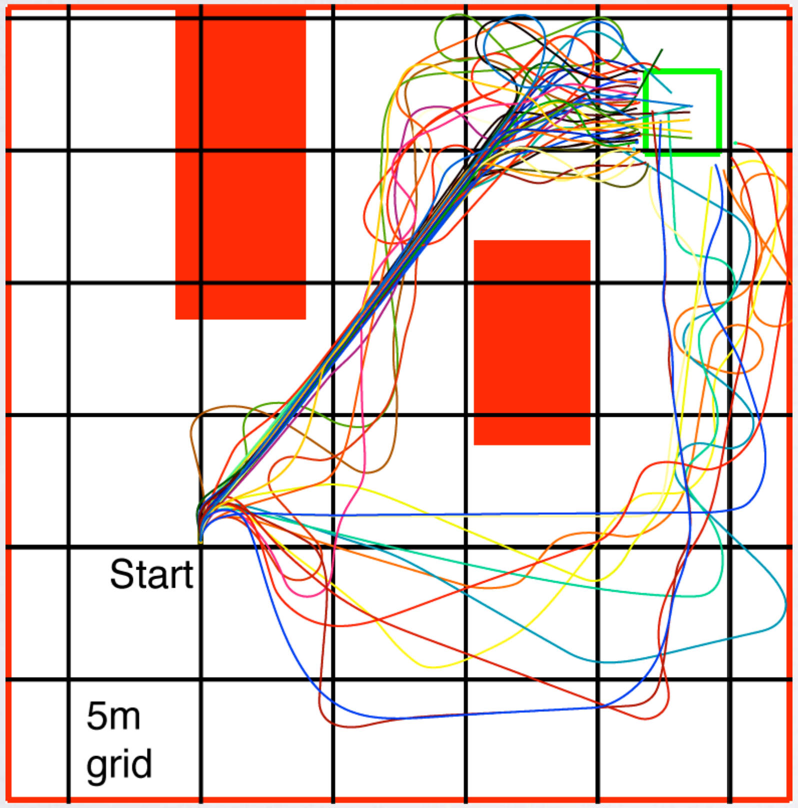
$$\texttt{Cost}(z) + \texttt{CostToGo}(z) \geq \texttt{Cost}(z_{\min})$$



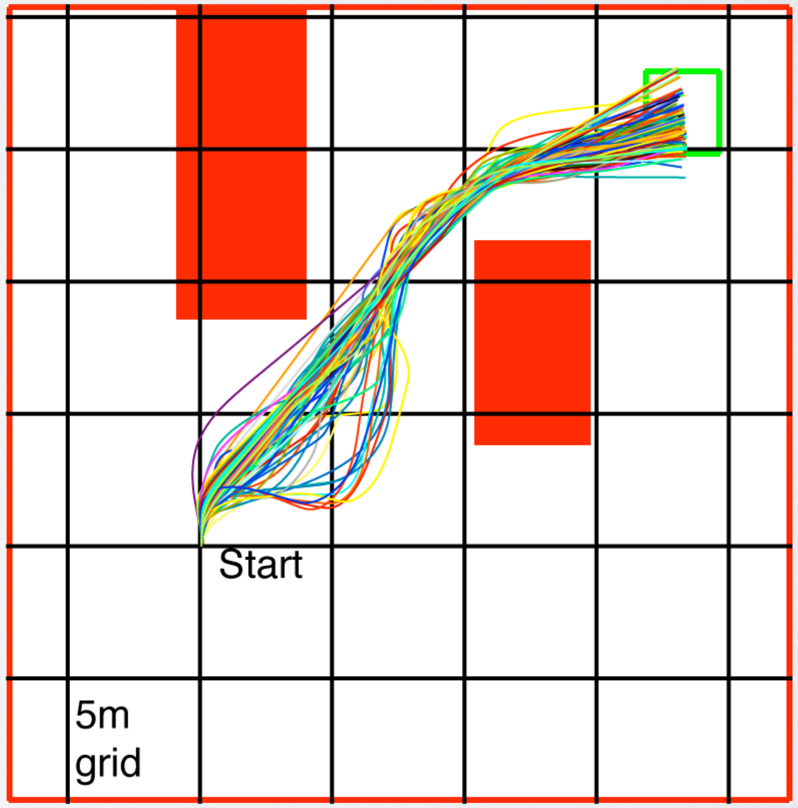


PERFORMANCE ANALYSIS

- Series of Monte Carlo simulations of a non-holonomic vehicle
  - Compare our anytime RRT* with anytime RRT
  - Both planners utilized committed trajectory and branch-and-bound heuristics
  - Both planners were allowed to maintain the tree until robot reached the goal
  - High-fidelity forklift dynamics model



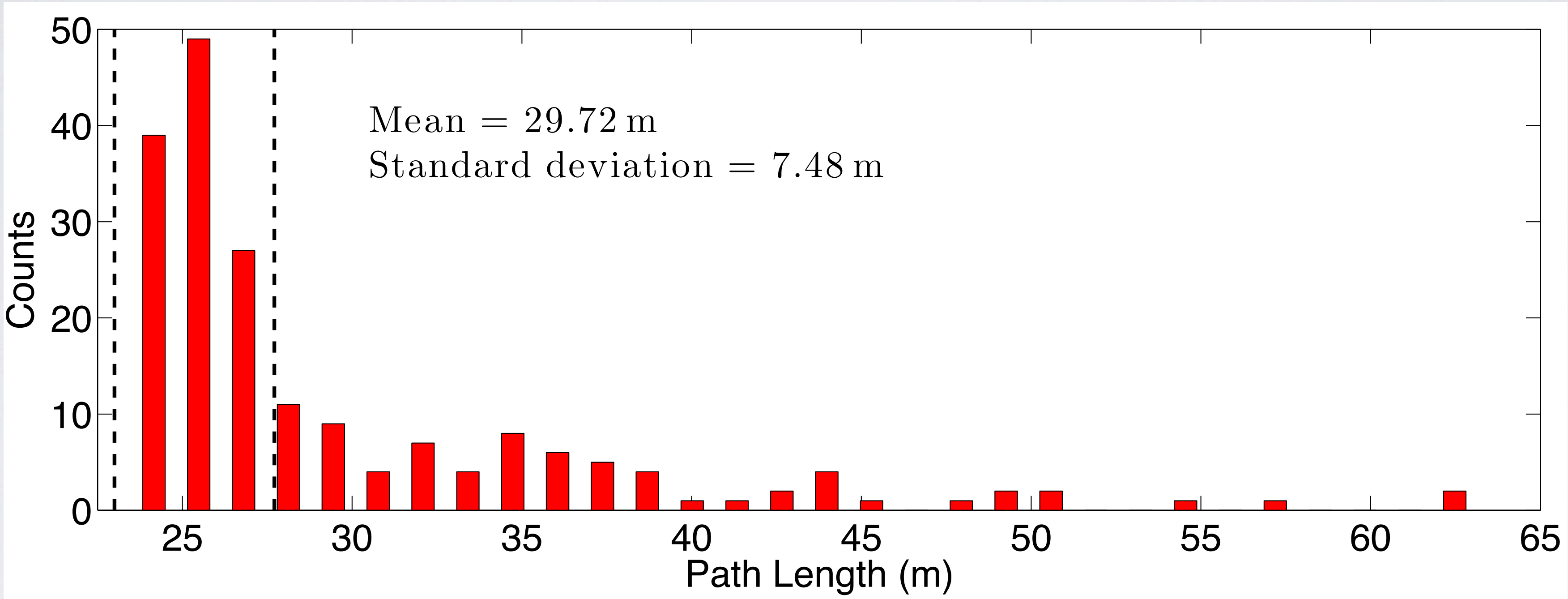
Anytime RRT



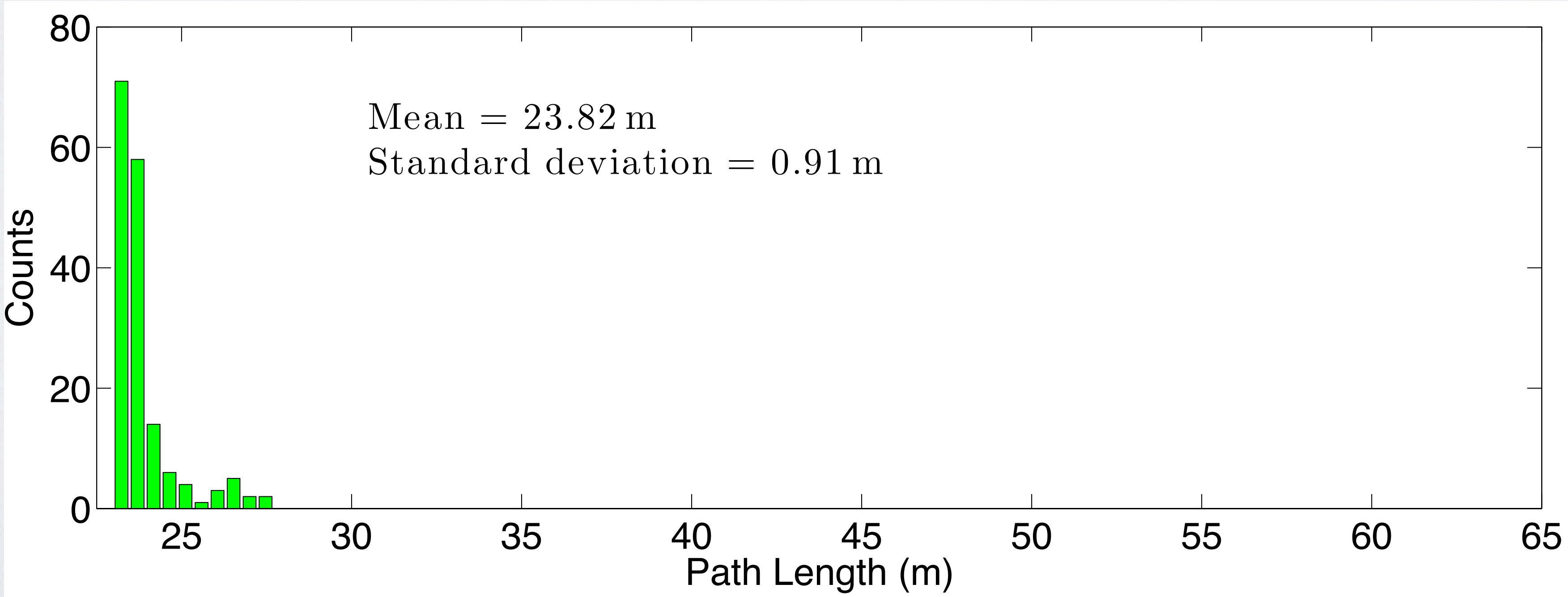
Anytime RRT*

PERFORMANCE ANALYSIS

Histogram of final path lengths



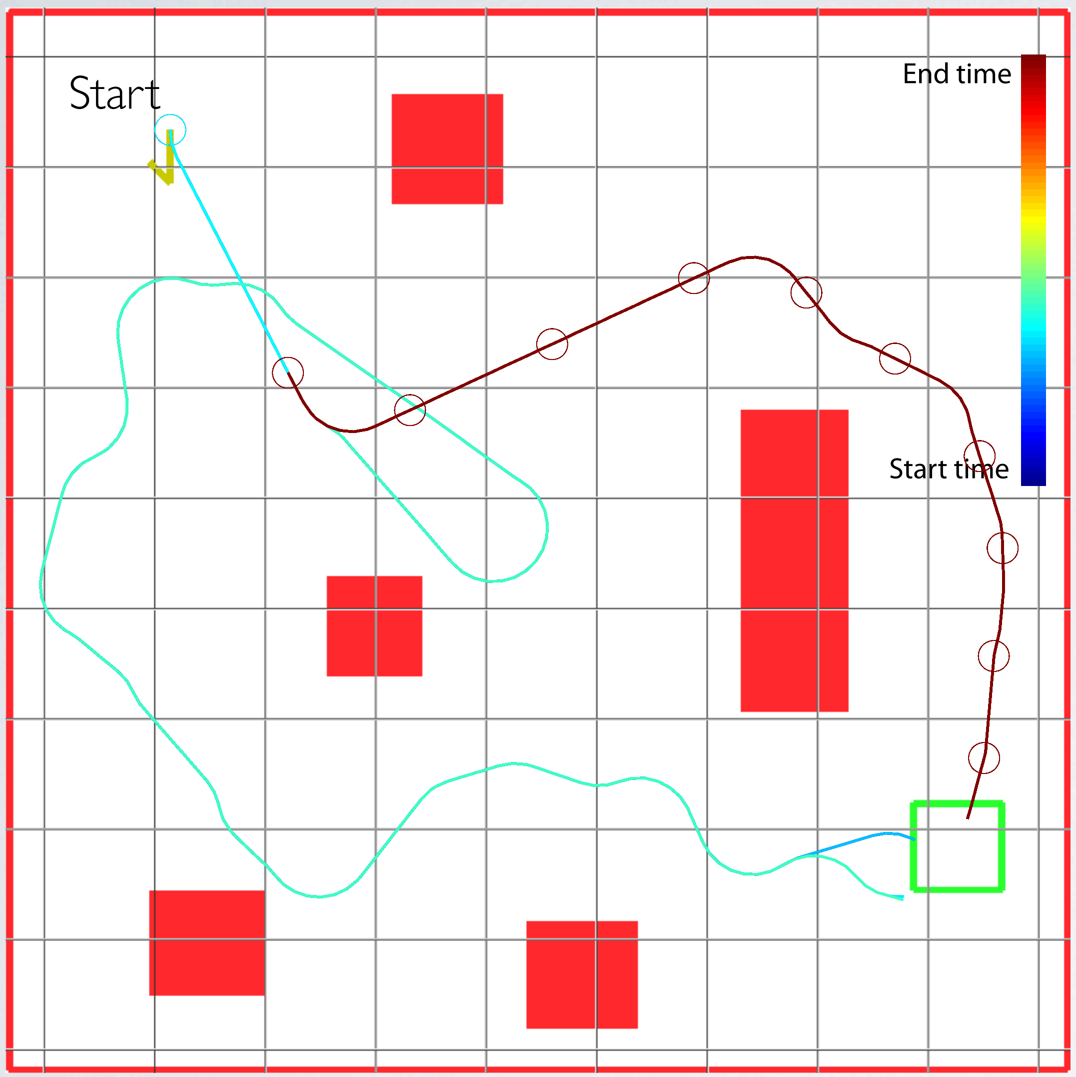
Anytime RRT



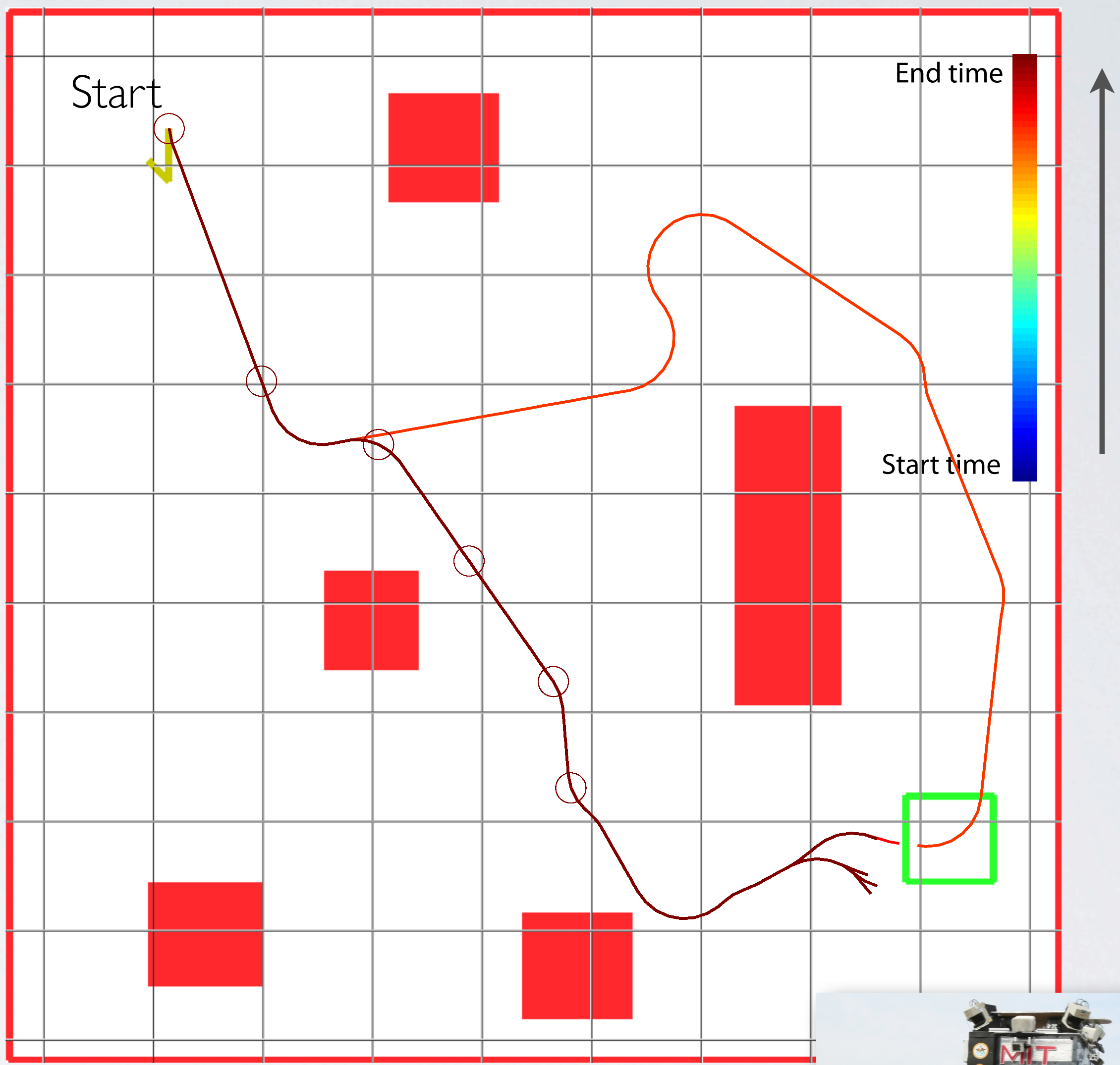
Anytime RRT*

FORKLIFT EXPERIMENTS: ANYTIME RRT

Run 1



Run 2

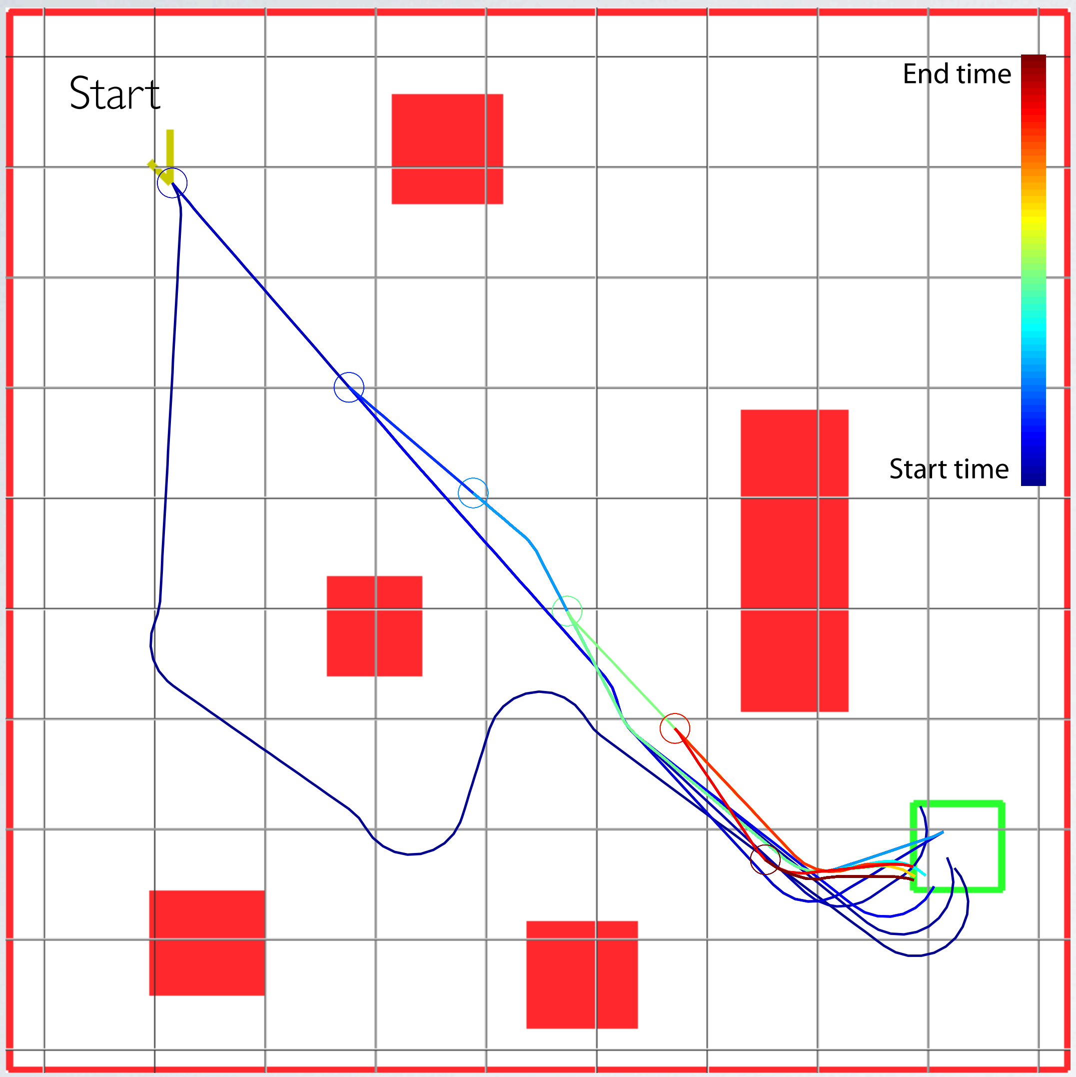


Increasing plan/execution time

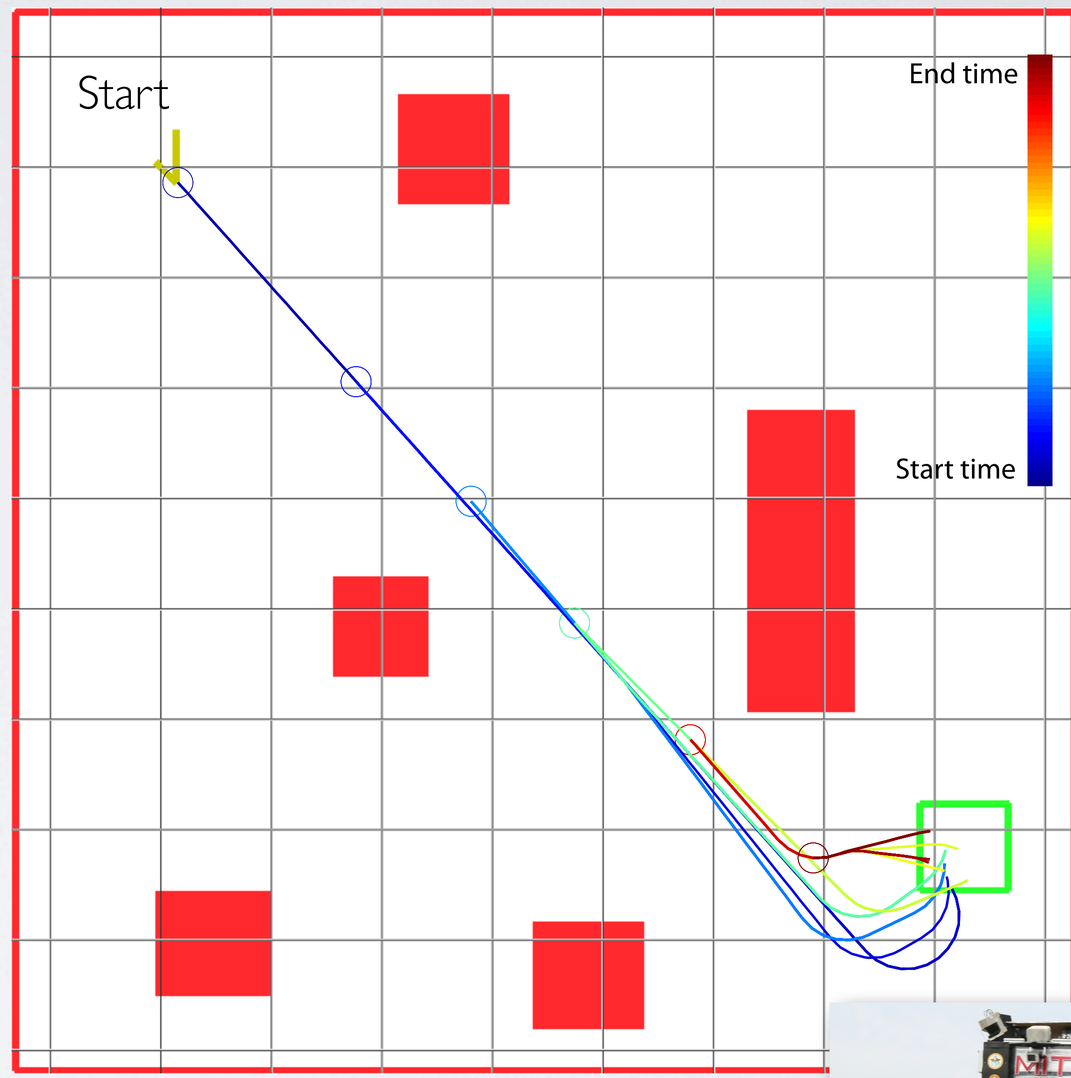
Circles denote initial positions of the planned (uncommitted) paths

FORKLIFT EXPERIMENTS: ANYTIME RRT*

Run 1



Run 2



Increasing plan/execution time

Circles denote initial positions of the planned (uncommitted) paths

CONCLUSION

The algorithm demonstrates the desired anytime properties:

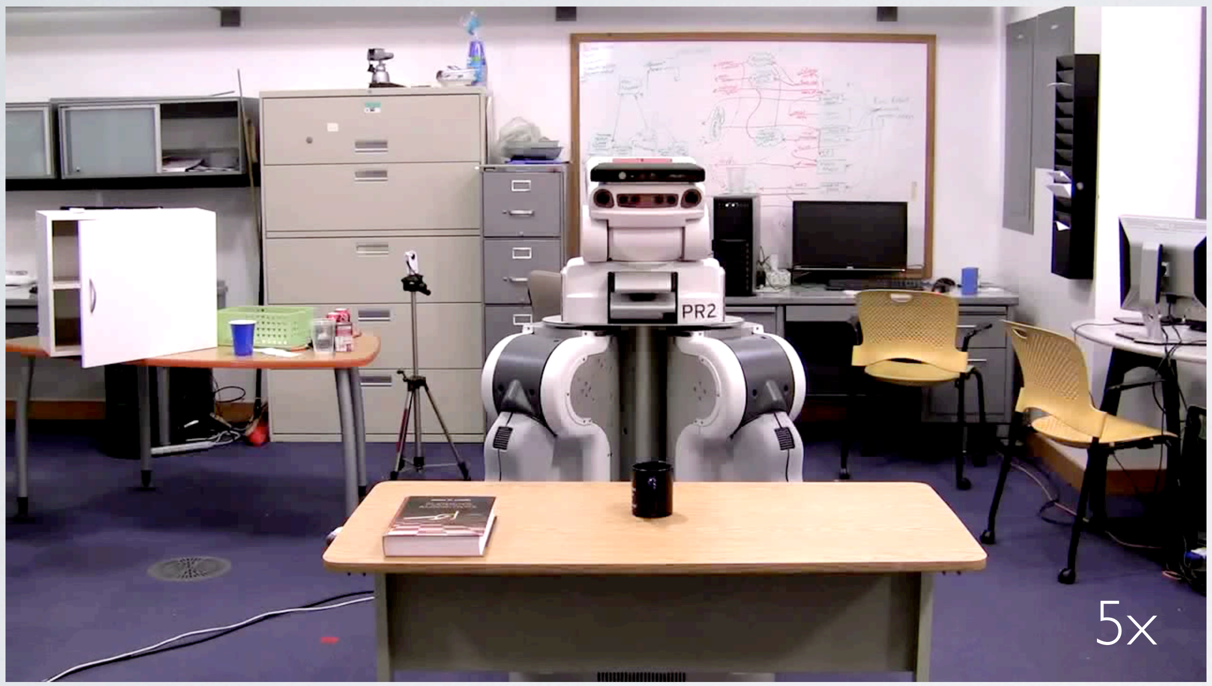
- Quickly find a feasible solution that the agent can begin to execute
- Take advantage of valuable execution time to asymptotically improve to optimal

CURRENT WORK

- Quickly find a solution that is feasible, but not necessarily optimal
- Asymptotic optimality: Exploit execution time to incrementally converge towards optimal solution
- (Probabilistic) completeness
- **Computationally efficiency (limited resources)**
- **Plan despite incomplete, imperfect knowledge**
- **Accommodate dynamic environments**

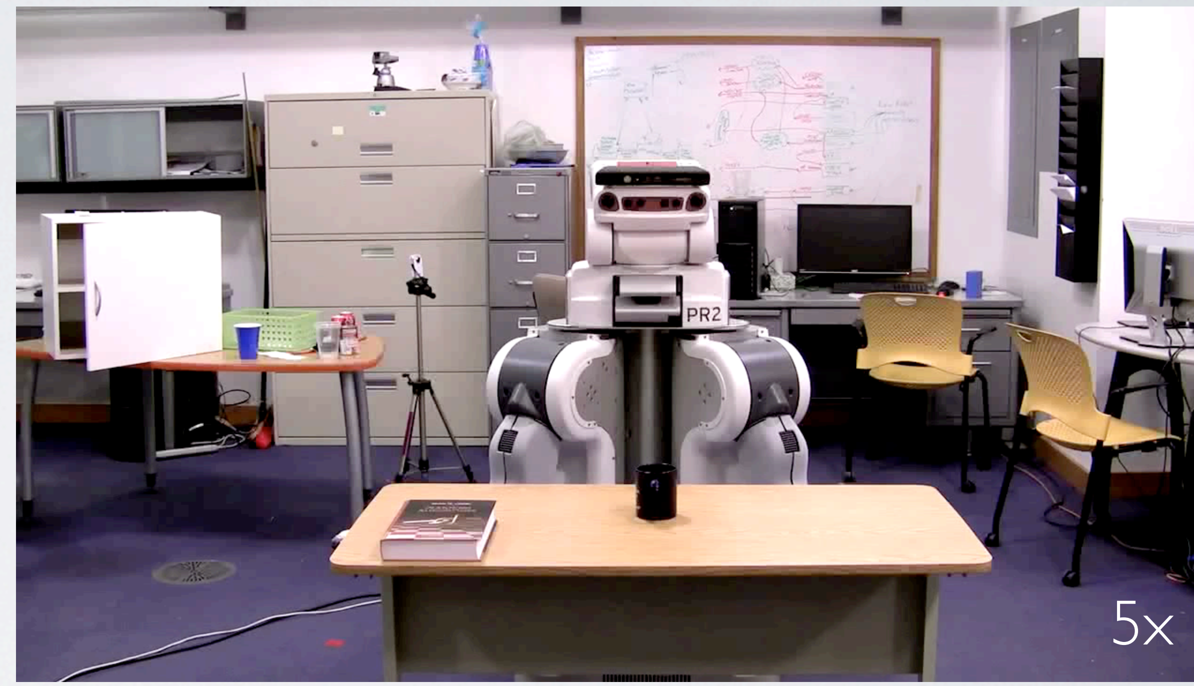
OPTIMAL MANIPULATION PLANNING

12-DOF pre-grasp planning on the PR2



RRT

video: 2011_02_pr2_12dof_rrt.mp4



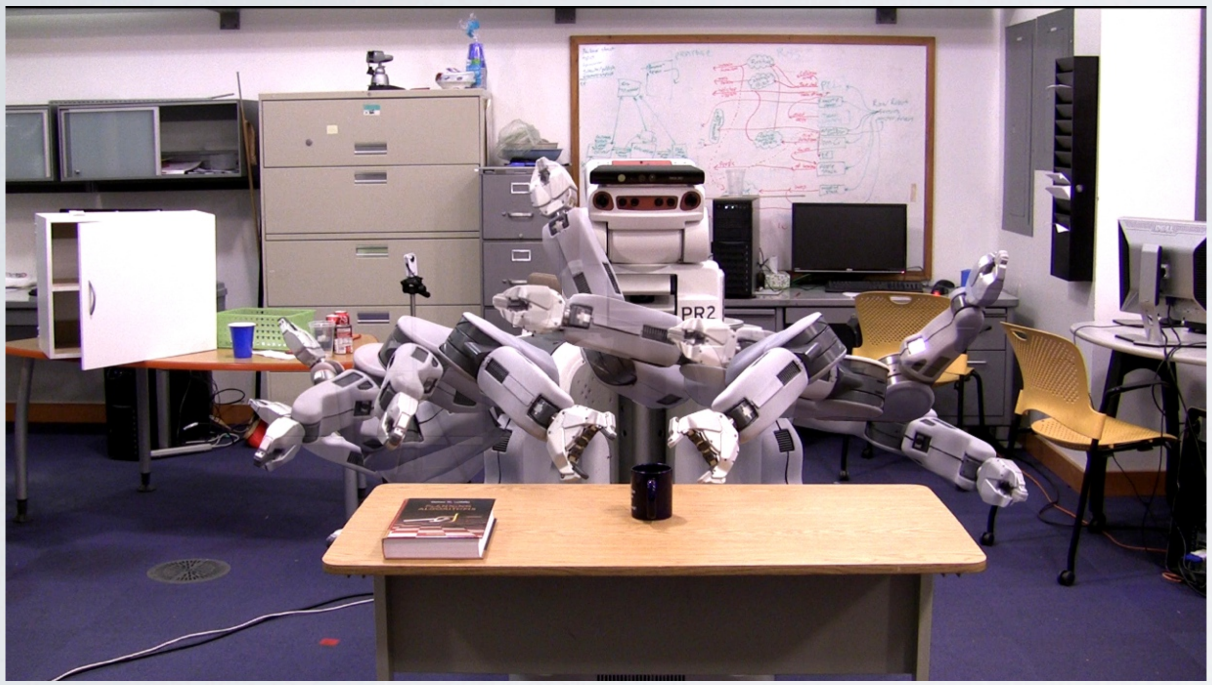
RRBT*

video: 2011_02_pr2_12dof_rrbtstar.mp4

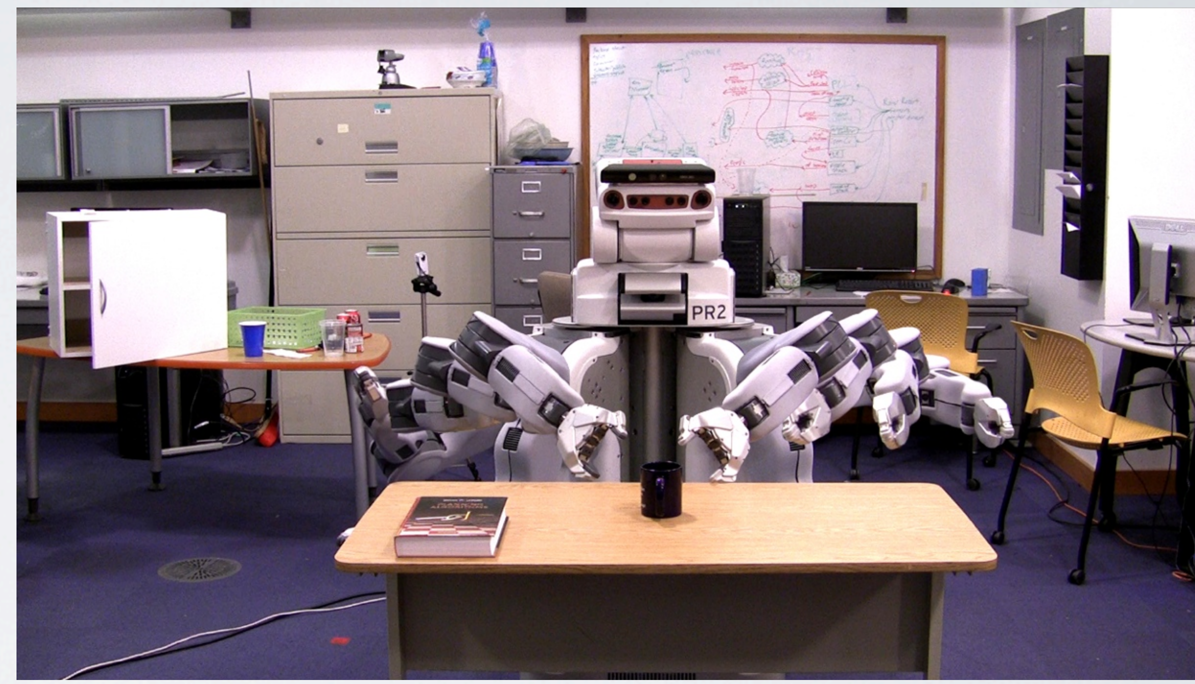
[Perez, Karaman, Walter, Shkolnik, Frazzoli, & Teller, IROS 2011 (submitted, under review)]

OPTIMAL MANIPULATION PLANNING

12-DOF pre-grasp planning on the PR2



RRT



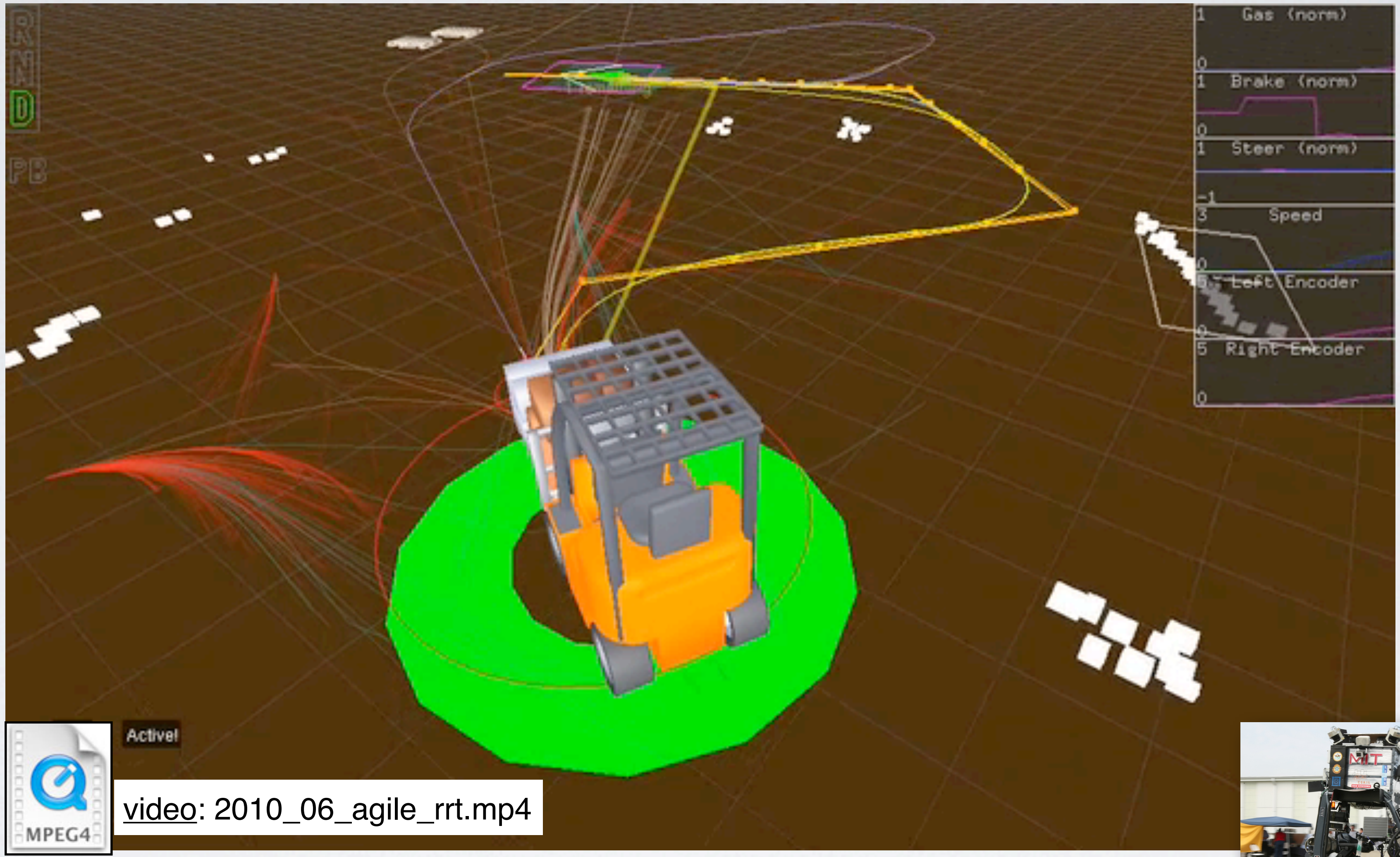
RRBT*

	RRT	RRBT*
First solution time (sec)	**29.9**	**9.7**
Cost of first solution (rad)	**19.8**	**8.6**
Cost of final solution (rad)	19.8	7.5

Averaged over several runs

[Perez, Karaman, Walter, Shkolnik, Frazzoli, & Teller, IROS 2011 (submitted, under review)]

PLANNING IN UNCERTAIN ENVIRONMENTS



Closed-loop RRT with false obstacle detections

[Joint work with Adam Bry and Nicholas Roy]

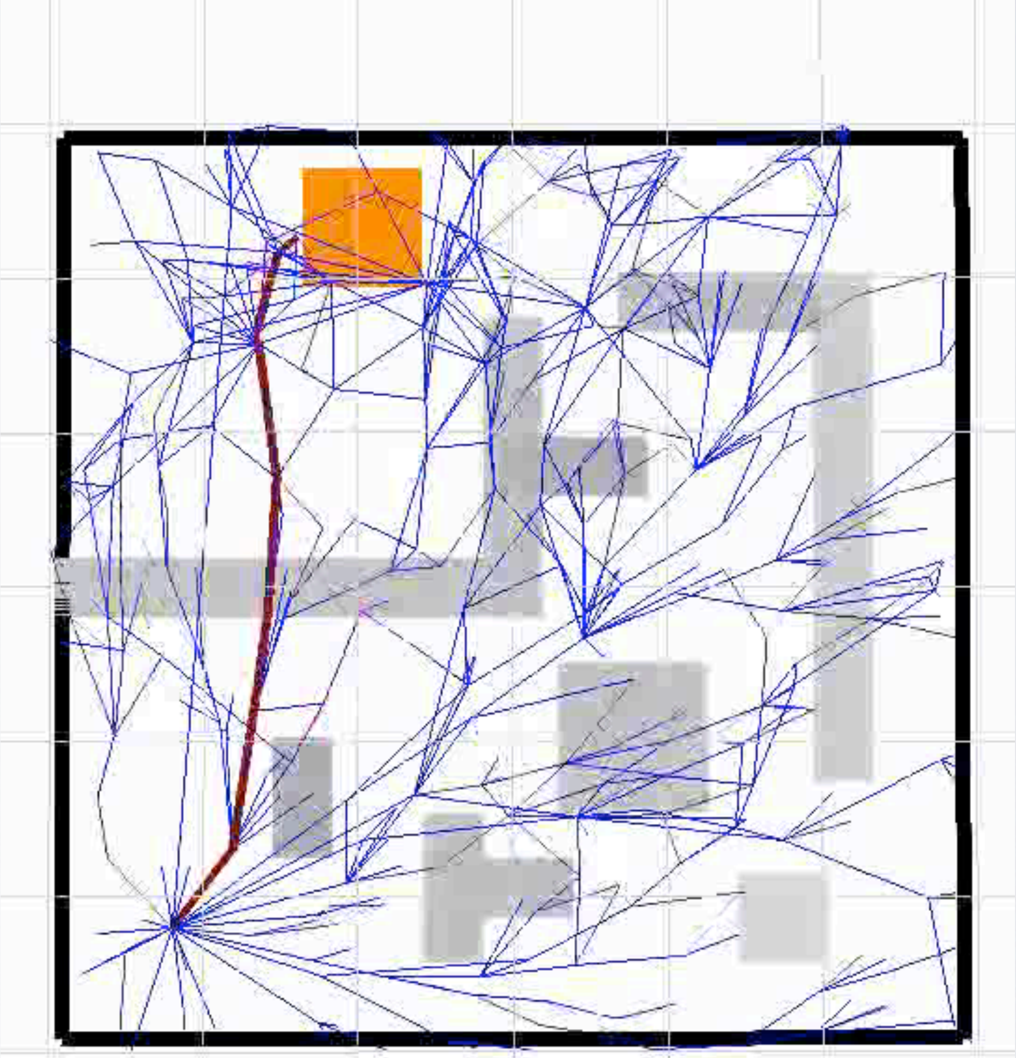
PLANNING IN UNCERTAIN ENVIRONMENTS

Chance-constrained optimization using incremental, sampling-based techniques

$$\min\left[c(\sigma)\right]$$

subject to:

$$P_{\mathrm{col}} < \delta$$
$$\sigma(0) = x_{\mathrm{init}}$$
$$\sigma(s) \in \mathcal{X}_{\mathrm{goal}}$$



Darker shades imply higher collision likelihood

video: 2011_04_constraint_optimization.mp4

[Joint work with Adam Bry and Nicholas Roy]

QUESTIONS?

mwalter@csail.mit.edu

http://people.csail.mit.edu/mwalter