Homework 3: Sample-based Path Finding

Huan Chen

1. Matlab Part

The path search result of RRT is showed in Figure 1. The expanding length increases, path cost increase.

The path search result of RRT* with 800 iterations is showed in Figure 2. The final path has a cost of 950.09, which is a shorter path compared with RRT.

2. ROS Part

Use ompl (The Open Motion Planning Library) to achive RRT* path search function. Follows the tutorial of ompl under this link: https://ompl.kavrakilab.org/optimalPlanningTutorial.html. Include these two head files:
#include <ompl/geometric/planners/rrt/RRTstar.h>
#include <ompl/geometric/planners/rrt/InformedRRTstar.h>

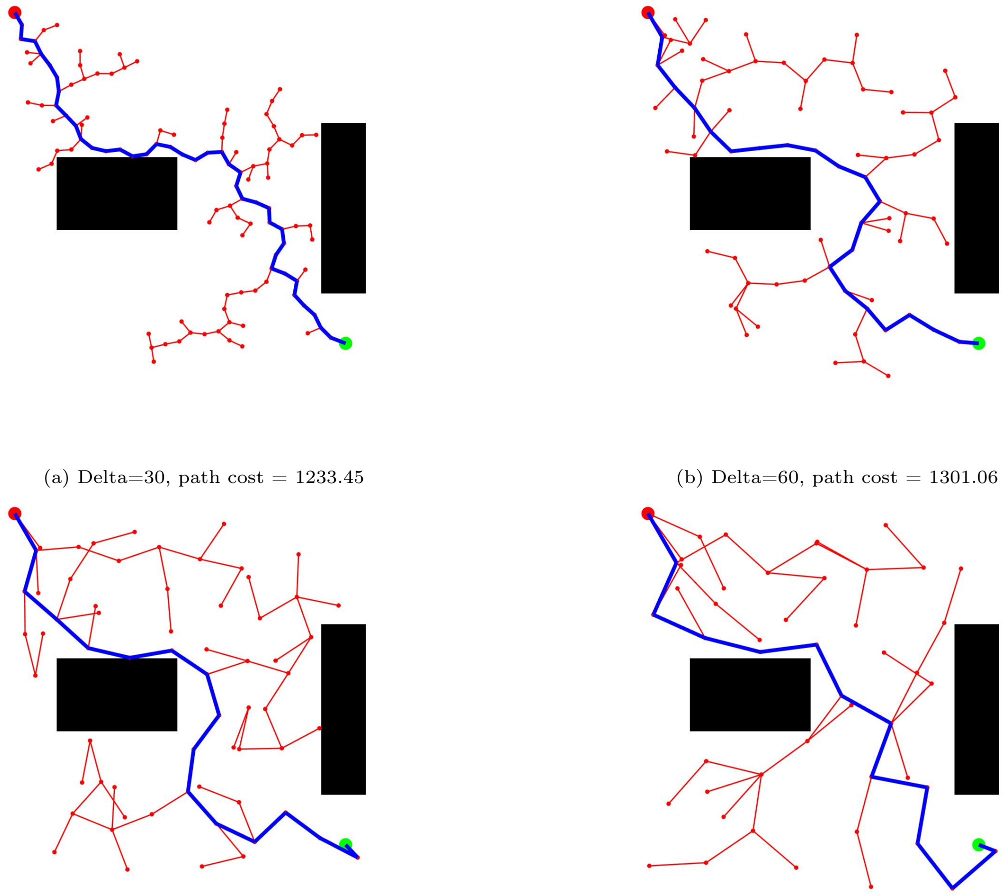
Use these two lines code to invoke RRT* or informed RRT* package:
ob::PlannerPtr optimizingPlanner(new og::RRTstar(si))
ob::PlannerPtr optimizingPlanner(new og::InformedRRTstar(si))

The RRT* path finding result and informed RRT* path finding result are showed in Figure 3 and Figure 4.

3. Main Takeaways

Sample-based planner do not attempt to explicitly construct the C-space and its boundaries.

1. Definition: complete planner, probabilistic complete planner, resolution complete planner.
2. PRM (probabilistic road map)
   - Learning phase build map
   - Query phase find path (e.g. with A*)
   - Cons: 2 steps, not efficient
   - Improvement: lazy collision check
3. RRT (rapidly-exploring random tree)



(a) Delta=30, path cost = 1233.45

(b) Delta=60, path cost = 1301.06

(c) Delta=90, path cost = 1387.07

(d) Delta=120, path cost = 1477.52

Figure 1: RRT path search result with different expanding length (Delta)

- Pros: find a path, target oriented
- Cons: not optimal
- Improvement: Kd-tree, bidirectional RRT

4. Optimal sampling-based planner with rewire function: RRT*, kinodynamic-RRT*

5. Advaced methods: only sample within certain region, e.g.informed RRT*, cross-entropy motion planning.

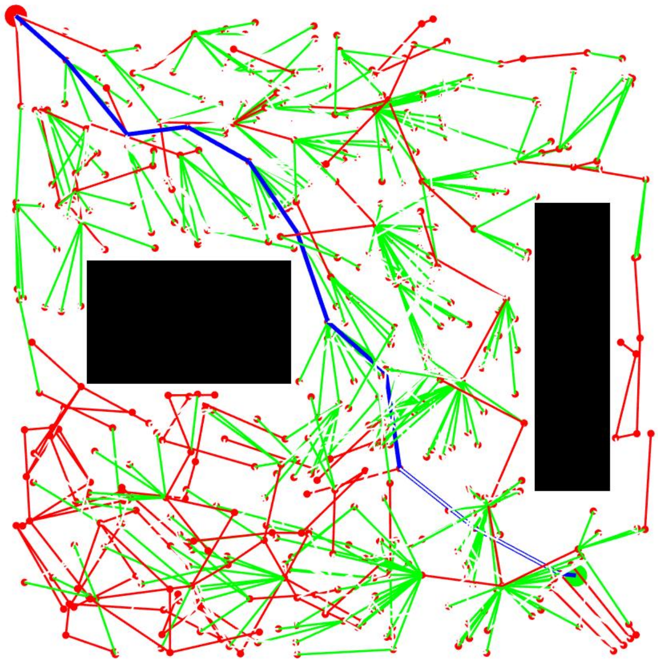


Figure 2: RRT* path search result, path cost is 950.09.

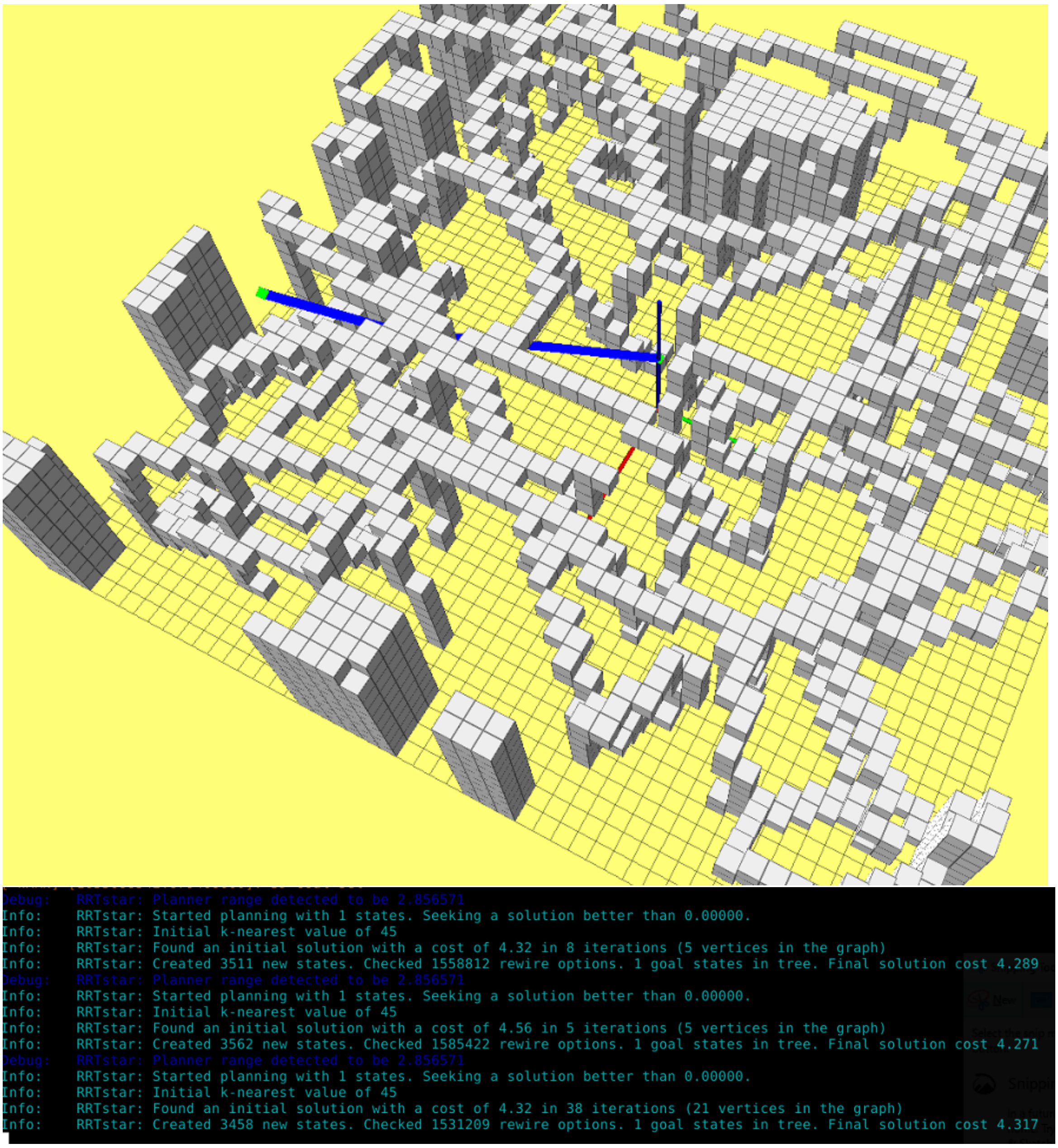


Figure 3: RRT* path finding result with ompl.

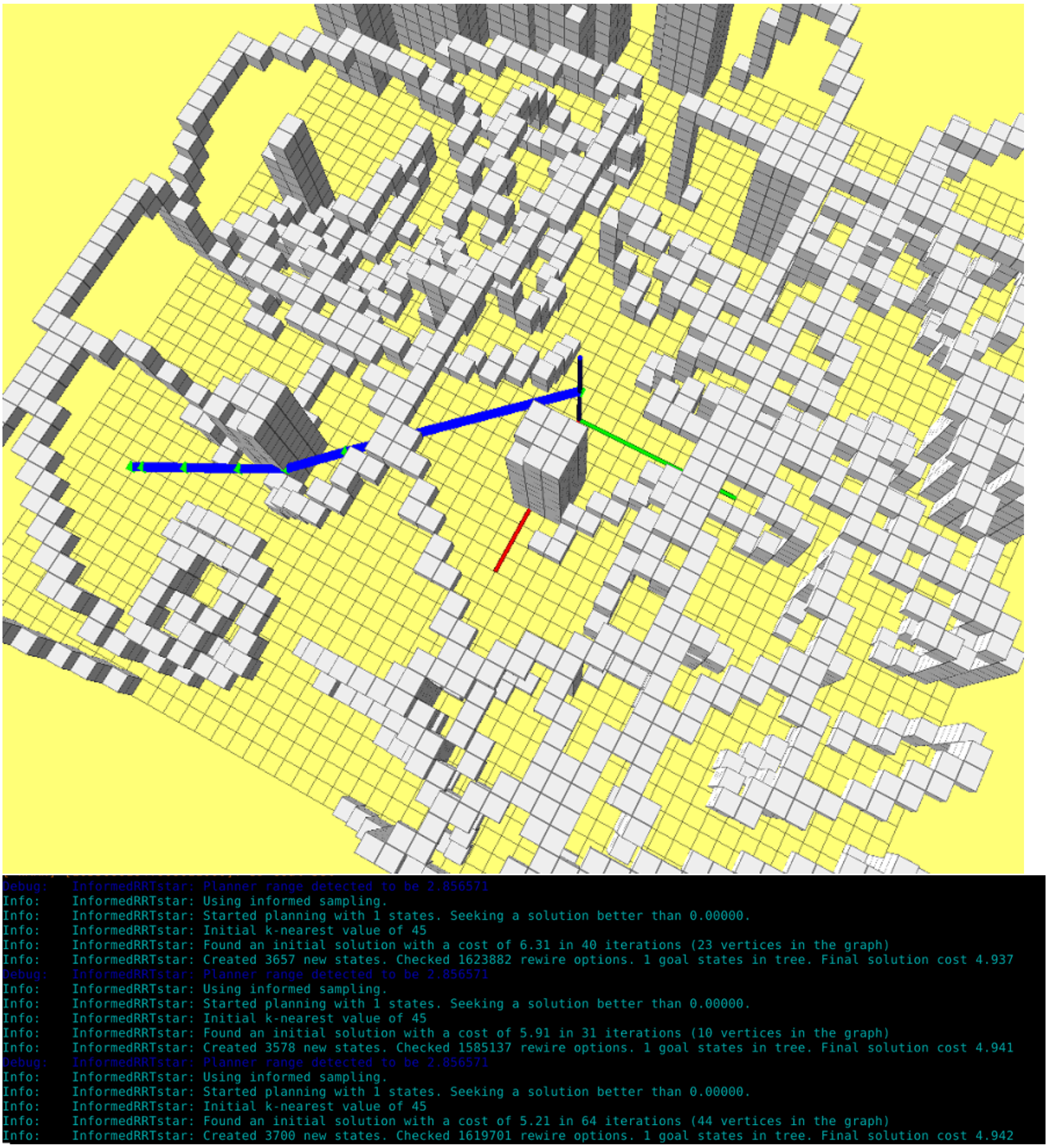


Figure 4: RRT* path finding result with ompl.

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
    $x_{rand} \leftarrow Sample(\mathcal{M})$;
    $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
    $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
    $E_i \leftarrow Edge(x_{new}, x_{near})$;
    if *CollisionFree*($\mathcal{M}$, E_i) **then**
        $\mathcal{T}.addNode(x_{new})$;
        $\mathcal{T}.addEdge(E_i)$;
    if $x_{new} = x_{goal}$ **then**
        Success();

Algorithm 2: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
    $x_{rand} \leftarrow Sample(\mathcal{M})$;
    $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
    $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
    if *CollisionFree*(x_{new}) **then**
        $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
        $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
        $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
        $\mathcal{T}.rewire()$;

Figure 5: RRT and RRT*.