# SPITE: Simple Polyhedral Intersection Techniques for modified Environments

**Marta Markowicz*, Maria Lusardi*, Stav Ashur, James Motes, Marco Morales, Sariel Har-Peled, Nancy M. Amato**
**Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign**
***Equal Contribution**

## Motivation

- Multi-agent task and motion planning requires reutilization of previous efforts for efficiency
- Fast algorithms can't afford to plan from scratch

## Problem Definition

- Planning in <u>modified environments:</u> obstacles can make discrete position changes before queries.
- Inputs:
  - robot
  - environment with obstacles (workspace)
  - discrete obstacle position changes
  - roadmap
- Outputs:
  - Updated validities of nodes/edges
  - Valid path for a given query

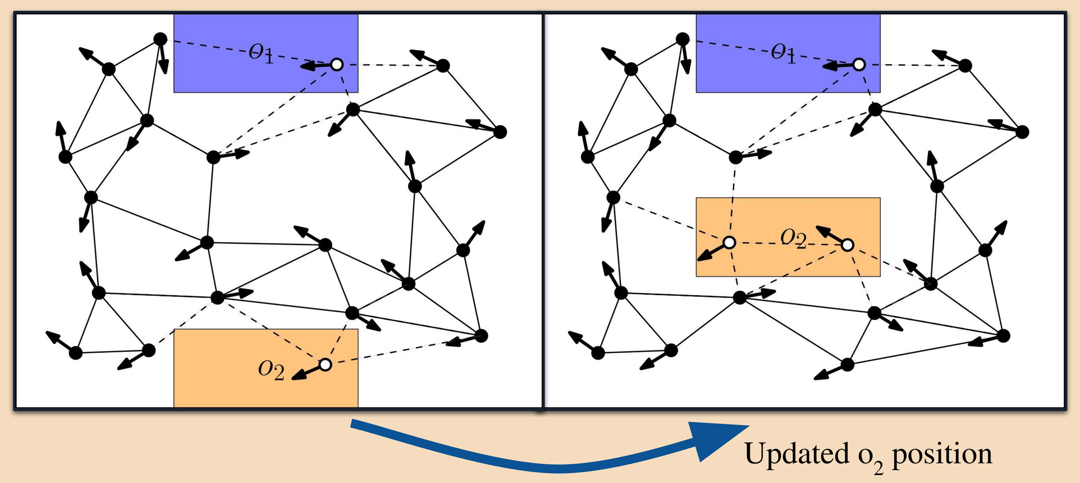


*Figure 1: Updating dynamic roadmap in modified environment.*

## Contribution

- Successfully created data structure to <u>efficiently maintain dynamic roadmap</u>
- Achieved faster update and query times than previous SOTA (Grid method [1])
- Provided improved hierarchical collision checking with constant-size 'cigar' approximationsn

## Link to full paper

## Method

- Construct point cloud from C-space edge intermediates
- Construct arbitrarily oriented "cigars" (capped cylinders) to over-approximate the swept volume of an edge (Fig. 2)

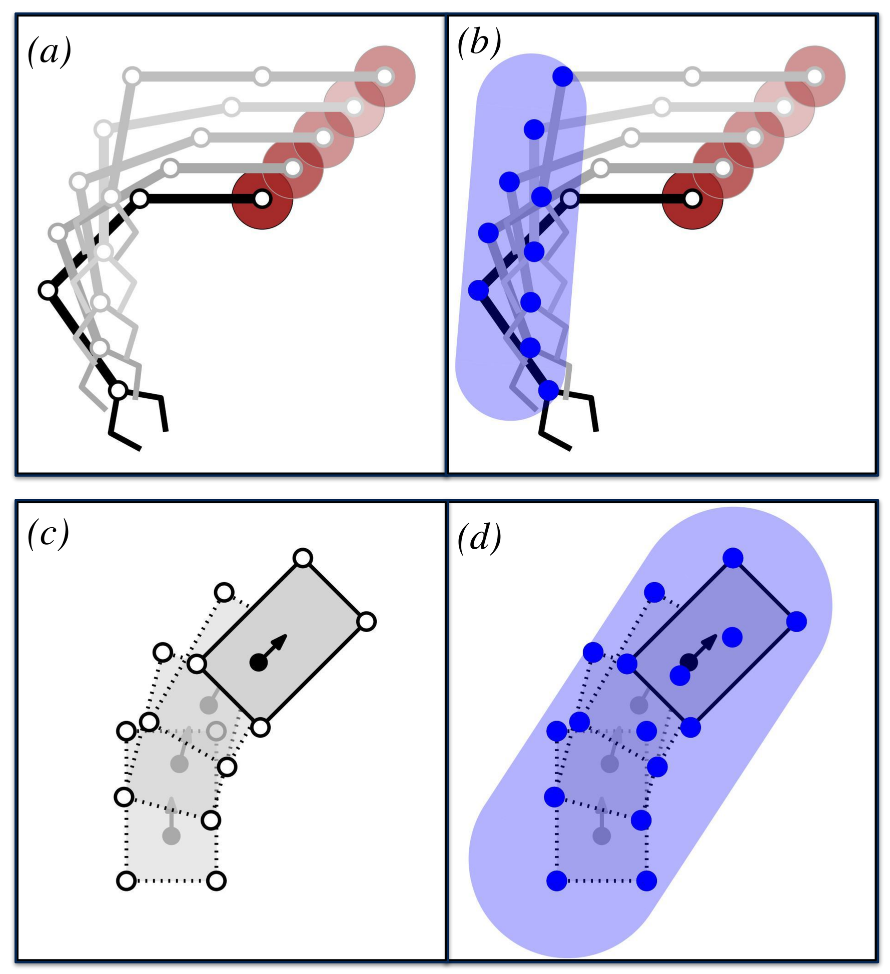


*Figure 2: Intermediates of roadmap edge for manipulator (a) and mobile robot (c). Swept volume approximation of edge for manipulator (b) and mobile robot (d) using cigars.*

- Cigars stored in AABB tree
- When an obstacle moves, run intersection query with new obstacle position against tree
- Reexamine validity of resulting nodes and edges potentially in collision and previously invalid ones
- Store mapping of obstacle to invalidated nodes and edges

```
Algorithm 1: Roadmap Update
input: Obstacle o, Transformation t
o.UpdateLocation(t)
invC ← Tree.GetIntersectingObjects(o)
for c ∈ invC do
    if CD(c,o) == blocked then
        c.invalidate()                              # Invalidating nodes and edges
        c.intersectionList.add(o)
    end
end
revC ← Tree.GetIntersectedVolumes(o)
for c ∈ revC do
    if CD(c,o) == free then
        c.intersectionList.remove(o)
        if c.intersectionList.IsEmpty() then
            c.validate()                            # Re-validating nodes and edges
        end
    end
end
```

## Results

- UR5e (5 DOF manipulator) experiment with two shelves, avoiding two obstacles placed in center of either shelf (Fig. 3)

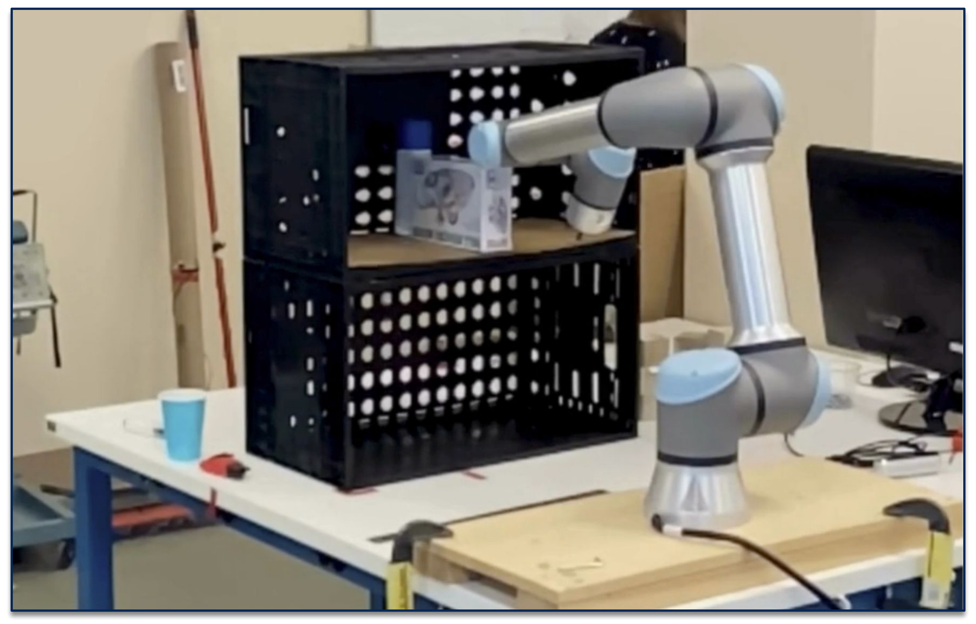

*Figure 3: Setup for UR5e experiment with shelf*

- Tested query times of dynamic roadmaps and single-query methods
  - SPITE outperforms Grid, ~75% less update time and ~66% less pre-processing (Table 1)
  - SPITE outperforms LazyPRM (91% less query time) and RRT (low success rate) (Table 1)

| Query Method | Update(s) | Query (s) | Total | Pre-processing (min) |
|---|---|---|---|---|
| PRM + SPITE | 0.199 | 0.028 | 0.227 | 27.6 |
| PRM + Grid_0.5 | 0.773 | 0.029 | 0.802 | 89.9 |
| PRM + Grid_1 | 9.310 | 0.165 | 9.475 | 12.3 |
| LazyPRM | - | 0.349 | 0.349 | - |
| RRT | - | *0.016 | *0.016 | - |

*Table 1: Results of UR5e experiment with different planners. Methods with low success rate (53%) highlighted red*

- Roadmap update times tested, SPITE outperforms Grid at every cell size (Table 2)

| obstacle | SPITE(ms) | Grid_1(ms) | Grid_2(ms) | Grid_4(ms) | Grid_8(ms) | BF |
|---|---|---|---|---|---|---|
| $1 \times 1 \times 1$ | 5.3 | 8.9 | 15.8 | 36.0 | 116.3 | $\sim$ 4 sec |
| $1 \times 1 \times 1$ | 50.3 | 67.0 | 93.9 | 149.8 | 296.9 | |
| $20 \times 2 \times 2$ | 39.7 | 57.2 | 85.2 | 152.6 | 343.1 | |
| $5 \times 5 \times 5$ | 58.0 | 70.7 | 95.7 | 153.4 | 301.4 | |
| $20 \times 20 \times 1$ | 212.2 | 246.9 | 314.3 | 453.0 | 718.8 | |
| $20 \times 5 \times 5$ | 136.7 | 166.1 | 212.7 | 300.9 | 478.3 | |
| $20 \times 20 \times 2$ | 292.1 | 337.9 | 417.5 | 577.8 | 872.9 | |
| $20 \times 20 \times 5$ | 422.4 | 465.6 | 538.7 | 708.6 | 974.9 | |
| $20 \times 20 \times 20$ | 610.7 | 778.9 | 773.8 | 924.4 | 1126.8 | |
| Pre-processing time | 2 min | 29 hrs | 4 hrs | 34 min | 4 min | - |

*Table 2: Results of testing roadmap update times with SPITE and grid.*

- Tested query times of mobile robot moving through changing passages in walls (Fig. 4)
- SPITE outperforms dynamic roadmap with Grid and single query methods (Table 3)

| Query Method | Update(s) | Query (s) | Total | Pre-processing (min) |
|---|---|---|---|---|
| PRM + SPITE | 0.13 | 0.12 | 0.25 | 0.95 |
| PRM + Grid_1 | 0.14 | 0.15 | 0.28 | 40.3 |
| PRM + Grid_2 | 0.39 | 0.14 | 0.52 | 7.6 |
| LazyPRM | - | 1.84 | 1.84 | - |
| RRT | - | 10.68 | 10.68 | - |

*Table 3: Results of mobile robot experiment with different planners.*

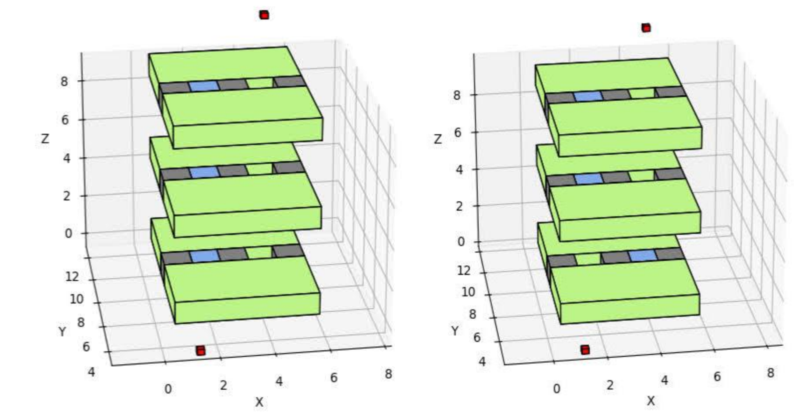

*Figure 4: Mobile robot environment with 3 walls and randomly changing passages.*

## Conclusions

- Introduced SPITE, a fast method for updating dynamic roadmaps for multi-query planning using 3D volume approximation and hierarchical collision checking
- Demonstrated up to 67% decrease in update times compared to Grid [1], using >100x less preprocessing time
- Achieved >40x decrease in query times compared to single query methods
- Future Work:
  - Quick check of nodes/edges at start of Lazy SPITE with over and under approximations
  - Parallelize bounding box collision checks

## References

[1] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), vol. 5, Apr. 2004, pp. 4399–4404.

## Acknowledgements

This work was supported in part by the U.S. National Science Foundation's ``Expeditions: Mind in Vitro: Computing with Living Neurons'' under award No. IIS-2123781, by the IBM-Illinois Discovery Accelerator Institute and the Center for Networked Intelligent Components and Environments (C-NICE) at the University of Illinois, and by Asociación Mexicana de Cultura AC.

**A simple update technique for roadmaps in changing environments to significantly reduce planning time!**