P-ClearPath

Multi-robot collision avoidance on a GPU

Why do we care?

Why do we care?

Constraints?

- Must be:
- Fast!
- Reliable!
- Safe!
- Actually get to goals!

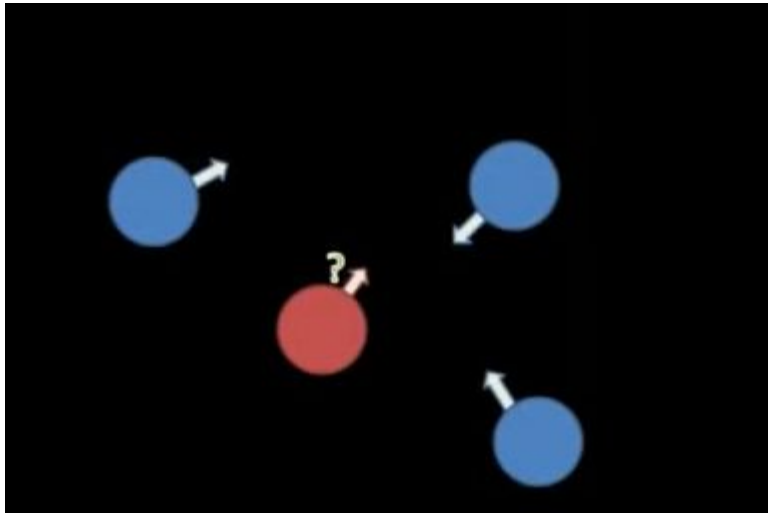
P-ClearPath

http://gamma.cs.unc.edu/CA/

https://www.youtube.com/watch?feature=player_detailpage&v=Hc6kng5A8lQ#t=145

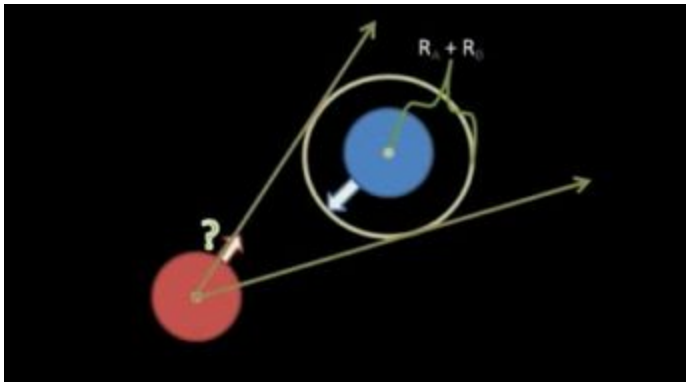
- Developed at UNC
- Mostly satisfies our constraints (sometimes not guaranteed to be safe, but it tries very very very hard to be!)
- Parallelizeable at different layers, GPU!

How it works: select 1 robot of interest

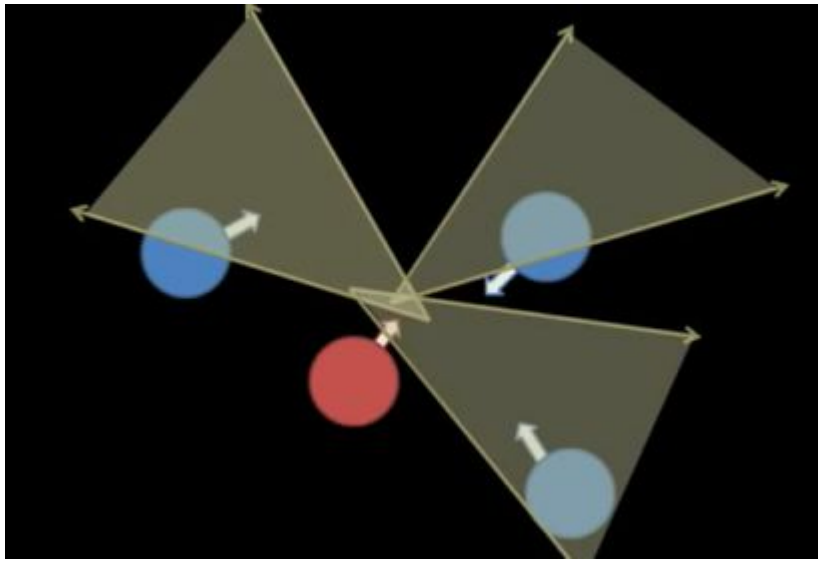


Credit of these images: http://gamma.cs.unc.edu/CA/

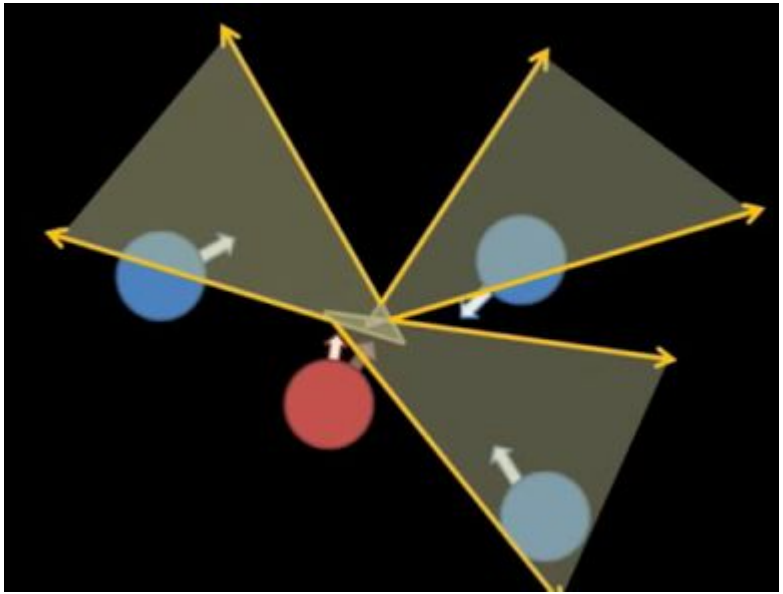
Find velocities that might cause collisions with others



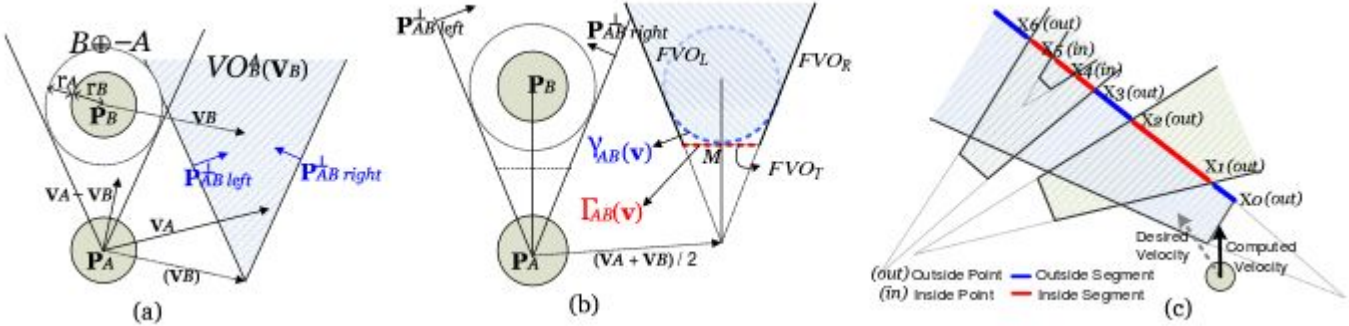
Repeat for all robots near the robot of interest



Compute bounding edges and use nearest intersection point as the velocity



A bit more complicated to avoid problem cases



Credit of these images: http://gamma.cs.unc.edu/CA/

Repeat for all the robots!

Parallelization possibilities:

- The repetition for all robots (we “freeze” at each time step, so no velocity updates affect each other)
- The computation of the boundary edges? (Do not think the paper does this, but uncertain, possibility for experimentation)

Not that simple though...

- Compute all edges
- Compute all intersections
- Compute edge segments that are outside of the “danger zone”
- Lots of ambiguities in the paper :(

What I’ve done

- Setup a test framework using the N-body simulation
- Solved for the FVO constraints
- Still need to actually compute resulting velocity (some ambiguities in equations)
- Reached out to author for clarification

Milestones

1. Complete basic implementation
    a. Only parallelize per robot FVO computation
    b. CPU implementation
    c. Some simple debug views
2. Implement uniform grid optimization
    a. Used to compute nearest neighbors (that might result in collisions, ignore far away bots)
3. Significant performance comparisons
    a. CPU, GPU, spatial data structures, parallelization of different portions