Online Path Finding

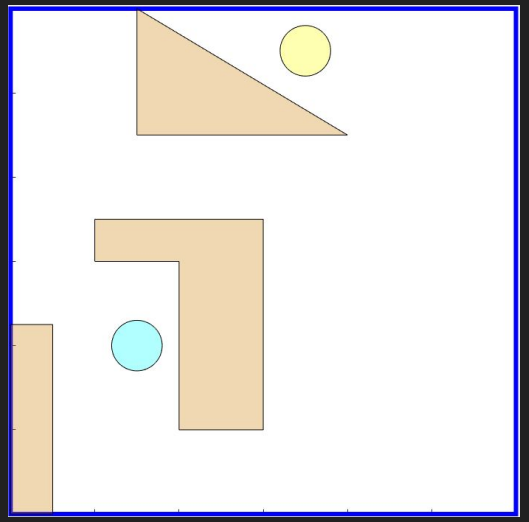
Kamal S. Fuseini & Shivam Swarnkar

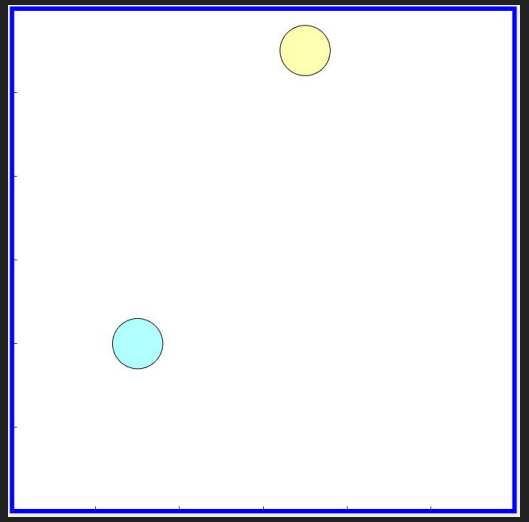
Outline

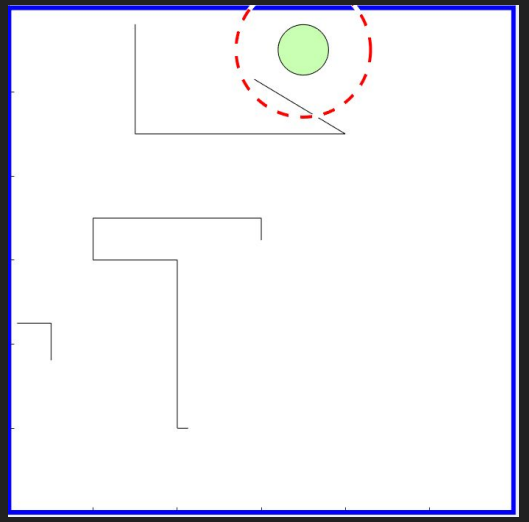
- Goal
- Background
- Issues
- Implementation
- Demo
- Next Step
- Bounded Modified Subdivision Search (BMSS)
- Conclusions

Goal

- Explore an unknown environment by scanning for obstacles and finding a path to the goal.

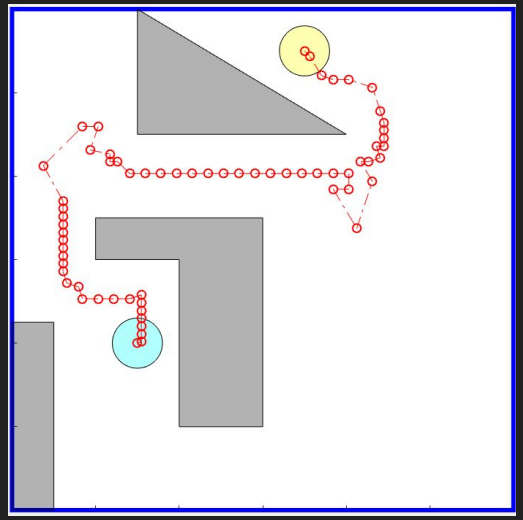






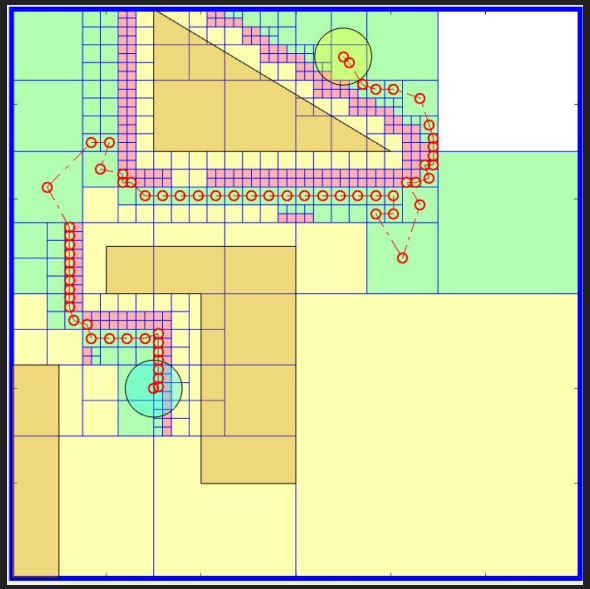
Background : Online Path Finding

- Agent (robot) explores an unknown map to find a goal position
- Continuously constructs a map based on its observations



Background : Subdivision Search

- Splitting an Environment into classified boxes
- Classification of boxes
  - Free
  - Stuck
  - Mixed
- Connection of free boxes to find a path to the goal



Issues

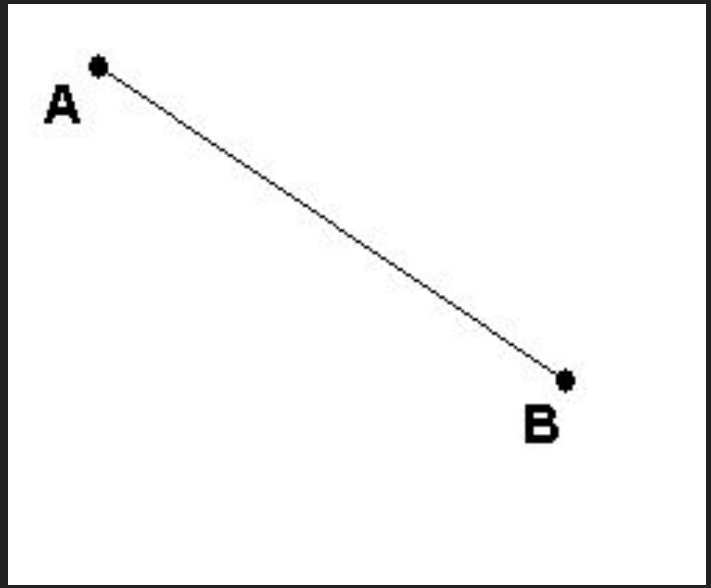
- Implementation of Laser
- Merger of Obstacles
- Keeping track of visited points
- Halting conditions

Implementation

- Line Class
- Laser Class
- Robot Class
- Soft Subdivision Search Class
- OPF Class

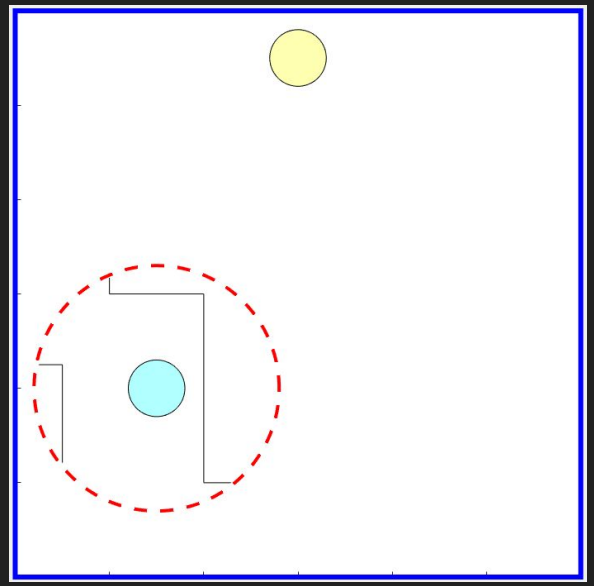
Implementation : Line Class

- Properties
  - 2 points
  - Slope
  - Intercept
  - Type
- Methods
  - Intersecs(position, radius)



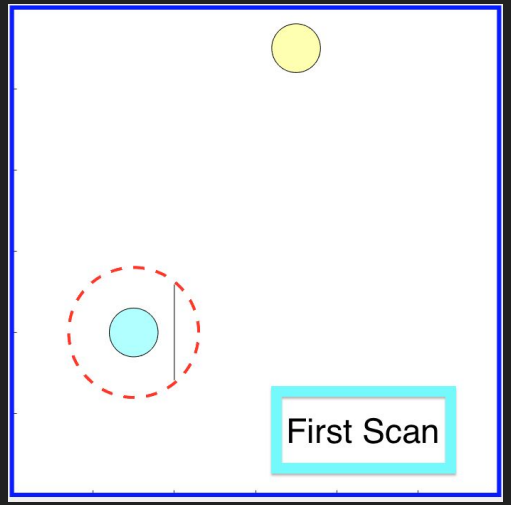
Implementation : Laser Class

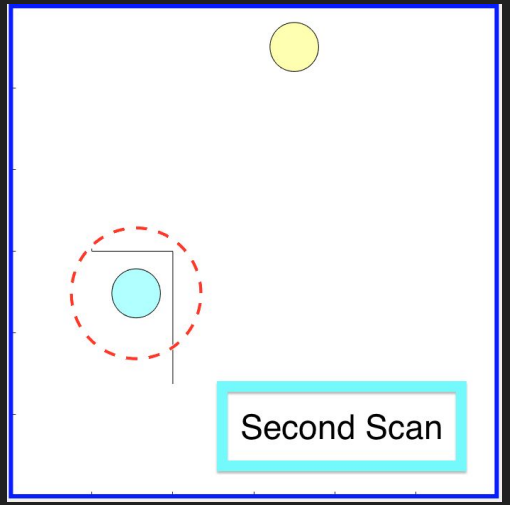
- Laser class returns the seen obstacles set in scan range as lines
- Properties
  - Range
  - Environment
  - Map
  - Polygons
  - SSS
- Methods
  - GetPolygons
  - Scan(position)



Implementation : Robot Class

- Robot class scans the environment and updates its Map
- Properties
  - Laser
  - Map
  - Current Position
- Methods
  - Scan





Implementation : SSS Class

- Uses Subdivision approach, Union Find data structure and Soft predicates to find path from start to goal
- Modified Version of HW4 SSS class implementation
- SSS constructor takes an environment type object
- Main method getPath() runs SSS algorithm, finds a path using Greedy BFS algorithm and returns the path as an array of points

Implementation : Online Pathfinding (OPF) Class

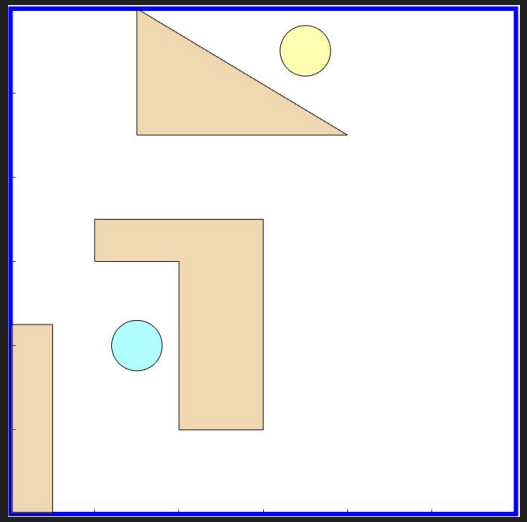
- OPF implements the main pathfinding algorithm
- Properties
  - Robot
- Methods
  - Run
  - Closest

Implementation : Pathfinding Algorithm

- Scan the environment, update the current map
- Send the updated map to SSS algorithm which returns a path (an array of points) to the goal
- Select the farthest point (bestPos) whose distance is less than the difference of scan range and robot’s radius (r - r0)
- Move to the bestPos, and repeats the above steps until reaches goal

Demo

- Path Finding in Following Map



Next Step

- Our current algorithm works, but it is inefficient
- It repeats and recomputes same values
- It wastes space and computation power
- Solution : Implementation of a more efficient algorithm

BMSS : Predicates and box classifiers

- A box is relevant if one of its vertex is visible in the scanned area and it’s not classified as “stuck”
- A box is classified “potentially free” if it’s classified as free based on current knowledge of obstacles
- A relevant box B is classified “free” if and only if it is “potentially free” and and all the vertices of B are visible and in the range $(r - r_0)$
- A box is classified “stuck” if it’s classified as stuck based on current knowledge of obstacles

Bounded Modified Subdivision Search (BMSS)

- Scan the environment, add new obstacles to the map
- Subdivide the current map and find a free box containing robot start position
- Add neighbours and adjacent relevant boxes (free and mixed) into the the queue
- Select the best box from the queue using heuristic
- If selected box free
  - Move to the box and scan
  - Add new obstacles to the map, merge overlapping obstacles and remove copies
  - Classify relevant boxes and add (free/mixed) nbrs to queue
- Else split the box, add the relevant (free/mixed) children to the queue and select a new box
- Keep selecting the boxes until selected free box contains goal or queue is empty

Conclusions

- Brute Force approach works but wastes computation power
- BMSS seems faster and more efficient approach
- BMSS requires new box classifications and predicates
- Current goal to implement BMSS algorithm

Thank you for your time