RMP-Bug Algorithms & Path Planning

Key words	Bug algorithms, Bug0 -Bug2, Cell decomposition, Potential field, Roadmap, Tangential Bug, Visibility graph, Voronoi diagramm
Status	note complete

Base Navigation: How to get from A to B?

- **random walk**: no path planning.
  - move in **random direction**
  - respond to **direct collision**: change **new random direction**
  - reactive
- **bug algorithms:** simliest and earliest **sensor-based planners** with provable guarantees
  - robot assumption: **point robot !!!!!**
  - known: global **goal**
  - unknown: obstacles
  - local sensing possible:
    - respond to **direct collision**
    - odometry (wheel): **distance traveled and orientation**
  - navigation path: a sequence of hit/leave point pairs of obstalces
    - start: q_s
    - end: q_d
    - hit point and leave point of obstacle i: q_i^H , q_i^L
    - eg: path = $\left(q_s\,, q_1^H\,, q_1^L\,, q_2^H\,, q_2^L\,, q_d\right)$

        - different algorithms defines different hit and leave points.
    - characteristics: **motion-to-goal, boundary following**
    - **entire planning is done online!!**

Assumption: point robot !!!!! no consideration of robot configurations.

Algorithms: reacts only to direct collision

Robot with only **contact sensor/ 0-range sensor** to detect obstacles.

- **Bug 0 algorithm**
    - Process:
        1. head towards goal
        2. when hits obstacle (hit point), follow the wall **until it can move towards goal —> leave point**
        3. repeat, head towards goal again
    - **each time the direction towards goal can be different!!!!**
    - Problem: the robot can get stuck and surround obstacle endlessly.

- **Bug 1 algorithm**
    - **m-line:** line segment **from** q_i^L **to goal**
    - Process:
        1. start: head towards goal **along m-line (start to goal)**
        2. when hits obstacle (hit point), **circumnavigate the whole obstacle.** —> complete exploration of obstacle
        3. find out **closest point towards goal** and travel back **—> leave point**

           (keep/update a **minimum** in memory, calculate distance to goal of all points)
        4. repeat, head towards goal again **along m-line (leave point to goal)**

- the **orientation** of robot end effector **at goal is different from start**. —> grasp impossible
- Problem:
  - if start and goal are fully isolated, every leave point to goal **intersects with current obstacle** —> no path, failure
  - grasping operation impossible: different orientation at goal.
- path length: $D_{start,goal} \leq L \leq D_{start,goal} + 1.5 \cdot \Sigma_i^n p_i$,
  - p: perimeter/ circumference of an obstacle

Bug 2 algorithm

- **m-line: fixed** line **from start to goal**
- Process:
  1. head towards goal **along m-line**
  2. when hits obstacle (hit point), move along the obstacle **until m-line is met —> leave point**
  3. repeat, head towards goal again along **m-line**
- the **orientation** of robot end effector **at goal is the same as start** —> grasping possible
- path length: $L \leq D_{start,goal} + 0.5 \cdot \Sigma_i^n n_i \cdot p_i$
  - n: #obstacles
  - n_i: hit the i-th obstacle n_i times
- Problem:
  - could run into **cycles**
  - for **complex obstacles**: path gets even **longer**, or it **can't reach the goal** becasue of **cycles**.

Bug 1 VS. Bug 2

- Bug 1 beats Bug 2: convoluted obstacle, Bug 2 might run into cycles and never reaches the goal.
- Bug 2 beats Bug 1: shorter path, when each obstacle is rather simple

Algorithms: limited local sensing to avoid direct collision

Robot with **limited range sensor** to detect/avoid obstacles.

▼ **Tangential Bug: improvement of Bug 2**

- 360°- local sensing with **limited range**

  —> **deviate from detected obstacle** before hitting.

  —> local knowledge: **hitting unseen obstacle** still possible
- strategy: avoid obstacle by **distance gradient** before hitting.
- Process:
  1. **motion-to-goal**: head towards the **straight line** connecting robot to goal. No sensing of obstacle between robot and goal
  2. **sensing of obstacle between robot and goal**: tries to pass the obstacle **tangentially.** It picks the endpoint that minimizes the heuristic distance to goal.
  3. **following boundary**: when the **heuristic distance** reaches its **minimum**, it starts to follow the boundary (**not necessarily hitting obstacle!**)
- possible heuristic: $d(\cdot, O_i) + d(O_i, goal)$
- Tangential Bug with senser range = 0: Bug 0 algorithm (**no specific leaving point strategy**)

Classic Path Planning Approaches

- Divide path planning into 2 processes:

    - **offline**: collision tests, mapping
    - **online:** connect start/goal with the space, path planning
- Motivation:
    - **multiple** path planning **tasks** implemented in **same environment —> mapping**

Complete algorithms: a solution exists in finite time

Combinatorial algorithms in a plane

▼ **Roadmap**

- represents **connectivity of** __free space__ by a network of **1-dimensional curves**

▼ **visibility graph**

- assmuption: all obstacles and walls are **polygonal**
    - if not polygonal (c-obstacles): enclose with **polygonal hull**
- **node**:
    - obstacle vertices, wall vertices
    - start node, goal node
- **edge**:
    - edges of obstacles/walls
    - for each node, check all other nodes: if **no collision with obstacle** when connectin 2 nodes —> edge
- Problem: **large number of edges**, computation complexity $O(n^2)$, space $O(n^2)$
- **Path** from start to goal: a sequence of edges —> **shortest path**
    - **semi-free path (sliding along obstacles)**
    - could be online: find the closest node of visibility graph and connect.

- **reduced visibility graph:** reduce the number of edges by
    - **supporting line**: 2 lines that fully enclose 2 obstacles
    - **separating line**: 2 lines that fully separate 2 obstacles
    - more than 2 obstacles: supporting and separating lines for **each 2 obstacle pairs**
- **Voronoi diagram**
    - Path: **maximizes** distance to obstacles.
    - Process: **Brushfire algorithm**
        1. **discretization** of the config-space into **grids**.
        2. each grid pixel measures **distance to closest point of closest obtacle.**
        3. **starts at obstacle**: **value = 1**
        4. wave-front: propagates from obstacle to next wave, **value = 2**
        5. wave fronts collide when **distance to 2 closest obstacles are the same** —> **point on Vonoroi diagram.**
    - Complexity: $O(n \cdot logn)$, space: $O(n)$
    - Wave-Front planner:
        - Process:
            1. similar to Brushfire algorithm.
            2. **starts at goal: value = 0, obstacle has no value**
            3. **wave-front: propagates** from goal to next wave, value = 1
        - Path:
            - ensures a **shortest path from start to goal,** avoids local minima.
            - a path can come very close to obstacle, only for one predefined goal.

- **Exact Cell decomposition**

- decompose the **free space into cells** and represent the **connectivity of free space** by **adjacency graph**.
- cells are **non-overlapping,** the **union** of all cells is **exactly F** (freespace).

▼ **Trapezoidal decomposition**

- assumption: polygonal environment and obstacles
- Process:
  1. **line-sweep:** sweeps vertical lines in the space. Each line intersects with wall corner / obstacle vertices.
  2. **adjacency graph:** each cell represents a subspace (**node: center of cell**). Connect adjacency subspaces (**edge**) and build an adjacency graph.
- Path: a **sequence of nodes** (subspaces) that starts from node of start point, and ends at node of goal point.
- Complexity: $O(n \cdot logn)$, space: $O(n)$
- Problem: **useless nodes** where a robot doesn't make decision: go up/down

  —> adjacent cell **gives no new information** —> can be eliminated.

▼ **Boustrophedon decomposition**

- Motivation: reduce the #nodes, reduce graph complexity
- **critical points:** vertices, where a vertical line can be **extended both up and down in free space** —> decision making
- Process:
  1. same process as Trapezoidal decomposition, **only keep lines at critical points**.
  2. adjacency graph

▼ Canny's roadmap algorithm

- for **arbitrary shapes** of obstacles —> exact knowledge of obstacles needed.
- line-sweep at **tangential line**

- complete configuration space necessary before motion planning

Planning in higher dimensions:

- motion: use combinatorial approaches —> **complete**

▼ **vertical cell decomposition** to higher dimensions (n)

1. **Plane sweep** the config-space using (n-1)-dimensional plane orthogonal to n-th dimension.
2. repeat

eg: 3D — first do 2D plane sweep, then 1D line sweep.

▼ **k-d Tree:** requires complete knowledge of the environment. inflexible if one upper level is updated/deleted, every level below will be changed. —> rebalancing necessary

▼ **Approximate Cell decomposition:** facilitates **hierarchical** decomposition

- Motivation: **avoid rebalancing** in tree structure
- cells are **non-overlapping,** the **union** of all cells **contains F** (freespace)
- cells: regular shapes, eg: rectangle, square.

▼ **Quadtree decomposition**

- only mixed nodes will be expanded
- Advantages:
  - quadtree boundaries don't depend on obstacle boundary.
  - if one element is removed, only local trees update, no rebalancing of whole tree
  - if one element is added, only update the node its subtrees

▼ **Octree decomposition: in higher dimension**

▼ Potential fields

- Bug algorithms: no construction of config-space, limited to 2-D configuration space

  —> algorithms for a **more general class of configuration spaces** (**higher dimensions**, non-Euclidean) needed

- **potential function over the free space** that has a **global minimum.**

  —> **gradient descent**

- **navigation function** (towards the goal):
  - goal: global minimum
  - obstacles: infinity
  - no local minima
  - smooth —> differentiable

▼ Attractive Field

- Characteristics:
  - the **goal attracts the robot.** —> **minimum** of the field
  - the potential **monotonously increases** as the distance to goal $d(q, q_{goal})$ increases.
  - its **negative gradient** points to the **steepest descent towards goal**
- quadratic potential: if robot **nears the goal,** below threshold d^*_{goal}
- conic potential: if robot is **far from goal,** above threshold d^*_{goal}

$$U_{attr}(q) = \begin{cases} \frac{1}{2} \cdot \zeta \cdot d^2(q, q_{goal}) & d(q, q_{goal}) \leq d^*_{goal} \\ d^*_{goal} \cdot \zeta \cdot d(q, q_{goal}) - \frac{1}{2} \cdot \zeta \cdot (d^*_{goal})^2 & d(q, q_{goal}) > d^*_{goal} \end{cases}$$

$$\nabla U_{attr}(q) = \begin{cases} \zeta \cdot (q - q_{goal}) & d(q, q_{goal}) \leq d^*_{goal} \\ \zeta \cdot d^*_{goal} & d(q, q_{goal}) > d^*_{goal} \end{cases}$$

▼ Repulsive Field

- Characteristics:
    - generated by obstacles, the **obstacles repel the robot.**
    - each obstacle has its own **max. impact distance** Q_i^*, generates its own repulsive field $U_{rep,i}(q)$
    - the repulsive field is the **sum of all repulsive fields of obstales.**

$$U_{rep,i}(q) = \begin{cases} \frac{1}{2} \cdot \eta \cdot (\frac{1}{D_i(q)} - \frac{1}{Q_i^*})^2 & D_i(q) \leq Q_i^* \\ 0 & D_i > Q_i^* \end{cases}$$

$$\nabla U_{rep,i}(q) = \begin{cases} \eta \cdot (\frac{1}{Q_i^*} - \frac{1}{D_i(q)}) \cdot \frac{1}{D_i(q)^2} \cdot \nabla D_i(q) & D_i(q) \leq Q_i^* \\ 0 & D_i(q) > Q_i^* \end{cases}$$

—> The potential function: **sum of attractive and repulsive potentials !!**

- Advantages:
    - **no need of complete knowledge of environment**. The goal attracts the robot, when the robot gets near to obstacle, it detects the repulsive field.
- ▼ Problem: **Local Minima** —> **not complete**
    - no guarantee of reaching the goal
    - examples: $F_{rep} = F_{attr}$ —> local minimum, the robot stops.
    - possible solutions:
        - U-form obstacles(concave): increase Q^* or η —> larger repulsive force / influence area. —> if goal inside U-obstacle, never reaches the goal
        - start at **multiple positions**
        - 2 convex obstacles: decrease Q^* or η —> smaller repulsive force / incluence area
        - if saddle point: tiny perturbation will lead to robot motion.