Robotics and Autonomous System

Chapter 6: Planning and Navigation

Görkem Şafak
Quentin Bernigaud
Sánchez Álvarez Juan Héctor

27th November 2008

Introduction

- Up to now focused on elements critical to robust mobility;
  - Kinematics
  - Perception
  - Localization
- Cognitive level – the decision making and execution that a system utilizes to achieve its end-goals.
- Navigation competence – given knowledge about its environment and goal position, the ability to act on knowledge and sensor values to reach a specific goal efficiently and reliably.
- Path Planning – Identifying a trajectory that will cause the robot reach the goal when executed.
- Obstacle Avoidance – Based on sensor readings, modulating the trajectory to avoid collisions.

Planning and Reacting

- For mobile robots, planning and reacting complementary.
- Navigation challenge involves executing a course of plan to reach its goal. During execution, robot must react to obstacles to reach the goal.
- Without any plan, reacting effort can not guide the robot.
- Information – Theory :

  Robot R with map M_i at time i and initial belief b_i.

  $loc_g(R) = p\ (g \leq n)$
- The robot can sense its belief state, not physical position so plan is the transition from b_i to $b_g(loc_g(R)=p)$.
- The problems;
  - b_i and M_i can be different from the reality.
  - M changes over time and the model for its change may be imperfect
- React! The planner will also receive unanticipated information and update its plans, i.e. reacting.
- Reacting – Planning merge = Integrated Planning and

Completeness

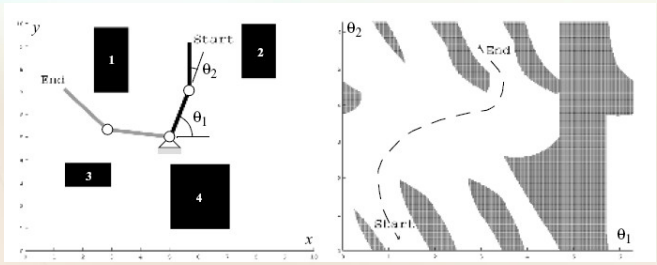
- An important concept in robot architecture involves whether particular design decisions sacrifice system's ability to achieved a desired goal whenever a solution exists.
- Formally, the robot system is complete if and only if, for all possible problems(initial state, map, goals), whenever there exists a trajectory to goal belief state, the system will achieve the goal belief state.
- So, an incomplete systems fails to generate a solution although there is one.
- Completeness is an ambitious goal but sometimes sacrificed to reduce computational complexity at representation level or reasoning.

Path Planning

- Previously was done for manipulators and industrial robots with up to 6 degrees of freedom. Path planning for such robots are much complex when compared to differential-drive mobile robots.
- High degrees of freedom and high speeds required not only kinematics but also dynamics to be involved.
- In mobile robots, a “fair” number of robots move at comperatively slow speed so dynamics rarely considered during path-planning.
- Configuration Space: A robot has k degrees of freedom. Every state or configuration of the robot can be described with k real values. The k-values can be regarded as points in k-dimensional space, called configuration space C of robot.
- Convenient, allowing complex 3D shapes with a single

Configuration Space

- Consider a robot-arm moving through workspace with obstacles. Path-planning aims to find a trajectory from initial state to final state, avoiding obstacles.
- Hard to visualize and solve, but configuration space definition makes it simpler.
- If O is defined as configuration space obstacle, a subset of C, then freespace F = C – O.



Configuration Space

- In mobile robotics, generally robots represented by (x, y, θ). For non-holonomic robots, robot’s velocity is limited by non-holonomic constraints in each configuration.
- The most common approach for path-planning is to assume the robot as holonomic, simplifying the process.
- Differential drive robots can rotate in place so a holonomic path can easily be mimicked if rotational position of robot is not important.
- Further, usually a mobile robot is represented as a single point in 2D space, as there’s no more rotational difference.
- Enlarge the objects by the radius of robot to compensate this assumption.

Path Planning Overview

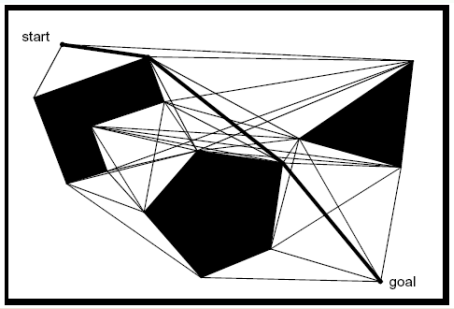
- Robot’s environment can be presented in many ways.
- First step for path planning – transform the environmental model into a discrete map suitable for chosen path-planning algorithm.
- Three common decomposition strategies;
  - Road map : Identify a set of routes within free space.
  - Cell decomposition: Discriminate between free and occupied cells.
  - Potential Field : a mathematical function over the space.

Road Map Planning

- Captures the connectivity of robot's free space in a 1D network of lines, roads. Decomposition is based on usually obstacle geometry.
- The challenge is to construct a set of roads that together enable the robot to go anywhere in its free space, while minimizing the number of total roads.
- Connecting the initial and goal position to road network and searching the path.
- Completeness preserved.
- Two extreme approaches;
  - Visibility graph : Stay close to object, minimize the distance.
  - Voronai graph : Keep distance to objects.

Visibility Graph

- Nodes of the graph = Initial and goal points + vertices of the configuration space obstacles (polygons).
- All nodes which are visible from each other are connected by straight-line segments, defining the road map. This means there are also edges along each polygon's.

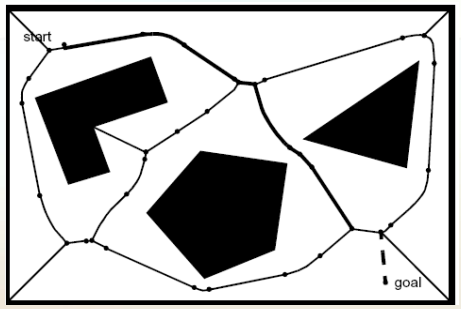


Visibility Graph

- Offers minimal-length solutions due to its representation.
- Moderately popular in robotics because of simple implemantation, especially environment representation describes objects as polygons.
- Two important caveats ;
  - The number of nodes and edges, the size of representation increases when there are many objects. Fast and efficient for sparse, slow and inefficient for dense environments.
  - Gives optimal solution in terms of minimal length, but sacrifying safety. Solution is to grow the objects by more than robot's radius.

Voronai Graphs

- Contrasting with the visibility graph approach, a Voronoi diagram is a complete road map method that tends to maximize the distance between the robot and obstacles in the map.



- The Voronoi diagram consists of the lines constructed from all points that are equidistant from two or more obstacles.
- The direction of movement on the Voronoi diagram is also selected so that the distance to the boundaries increases fastest.

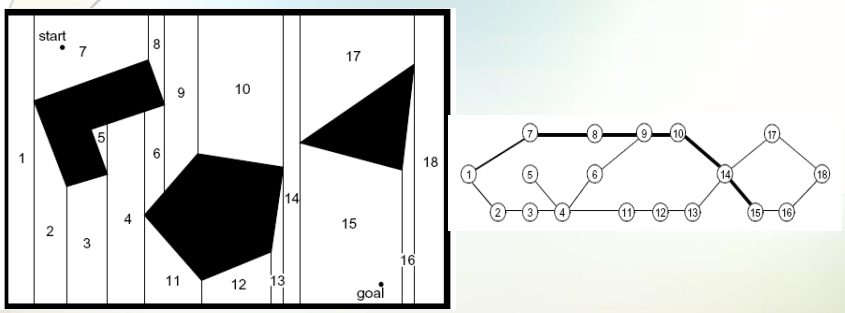
Voronoi Graphs

- Complete like Visibility graphs, but far from optimal in terms of length.
- An important important weakness in the case of limited range localization sensors.
- Any short-range sensor on the robot will be in danger of failing to sense its surroundings , chosen path will be poor.
- But an important advantage, executability. Given planned path via Voronoi planning, a robot can follow a Voronoi edge in the physical world by simple control rules: Maximize the minima read from local sensors. Migitation for encoder inaccuracy
- Can be used in automatic mapping, by finding and moving on unknown edges and then constructing a consistent Voronoi map.

Cell Decompositin Path Planning

- The idea is to discriminate between geometric areas or cells that are free or occupied.
- Generalized idea for cell decomposition path planning;
  - Divide F into simple connected regions called cells.
  - Determine adjacent open cell, build a connectivity graph.
  - Find the cells that contains initial and final configurations, search for a path in connectivity graph.
  - From the sequence of cells found with an appropriate searching algorithm, compute a path within each cell, for example, passing through the midpoints of the cell boundaries
- The placement of boundaries :
  - Exact Cell Decomposition : Boundaries are function of obstacles, decomposition is lossless.
  - Approximate Decomposition

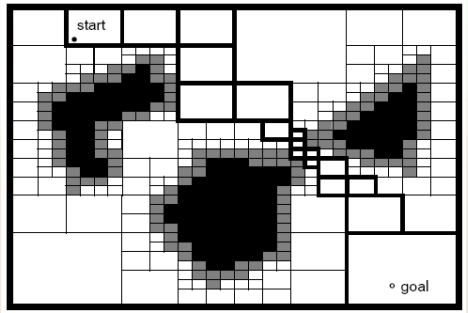
Exact Cell Decomposition



- The particular position of the robot within each cell of free space does not matter ; the robot's ability to traverse from each free cell to adjacent free cells is important.
- Disadvantage: The number of cells and, therefore, overall path planning computational efficiency depends upon the density and complexity of objects in the environment, just as with road map based systems

Approximate Cell Decomposition

- A popular technique for mobile robot path planning due to lower computational complexity.
- Cell size not dependent on objects in the environment. Narrow passages can be lost, but practically not a problem due to very small cell sizes used.

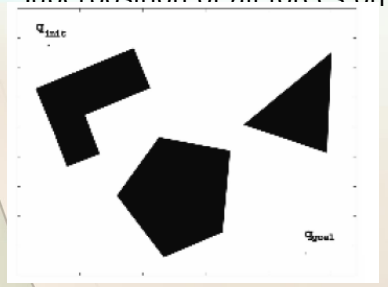


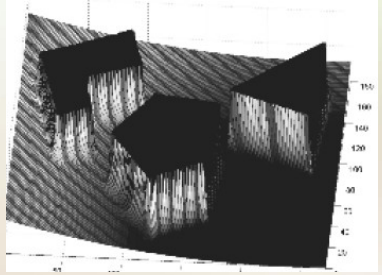
Approximate Cell Decomposition

- Grassfire(NF1), an efficient algorithm for finding routes.
- Wavefront expansion from the goal position outward, marking for each cell its distance to goal until the initial cell is reached.
- At this point, planner can estimate the distance to the goal and create a trajectory by linking adjacent cells that are always closer to goal.
- Each cell stored in memory so visited only once. Simply a breadth-first search in the space of adjacent cells. Complexity not dependent on density of the environment.
- Fundamental cost is the memory requirement, but migitated due to latest improvements in computers.

Potential Field Path Planning

- Creates a field, or gradient, across the robot's map that directs robot to the goal avoiding obstacles.
- Treats the robot as a point under the influence of potential field *U(q)*.
- Robot moves by following the field, like a ball rolling down hill. Obstacles act as peaks, repulsive forces whereas goal attracts the robot.
- Superposition of all forces on the point(robot) guides it to the





Potential Field Path Planning

- Assuming the robot is a point, we have a potential field in 2D(x,y). If we assume a differentiable potential field function U(q), resulting force F(q) acting on position q = (x,y) ;
  F(q) = - ∇*U(q),* where ∇*U(q)* defines the gradient vector of U at point q.

$$\nabla U = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix}$$

- The potential field acting on the robot is then;
  $U(q) = U_{att}(q) + U_{rep}(q)$
  $F(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q)$

Potential Field Path Planning

- **Attractive Potential:**

$$U_{att}(q) = \frac{1}{2} k_{att} \cdot \rho^2_{goal}(q)$$ where $p_{goal}(q) = |q - q_{goal}|$

- $F_{att}(q) = -\nabla U_{att}(q) = k_{att}.(q-q_{goal})$, F tends to zero when q->goal.
- **Repulsive Potential:** A strong repulsion when close to en far away. Example;

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) \ge \rho_0 \end{cases}$$

- *p(q)* is the minimal distance from q to the object and $p_0$the distance of influence of the object.
- Tends to infinity if q gets closer to the object.

Limitations & Extensions

- Some limitations;
  - Local minima that appear dependent on object shape and size.
  - For Concave objects, there may be several minimal distances p(q), resulting in oscillations between two closest points to the object, sacrifices completeness.

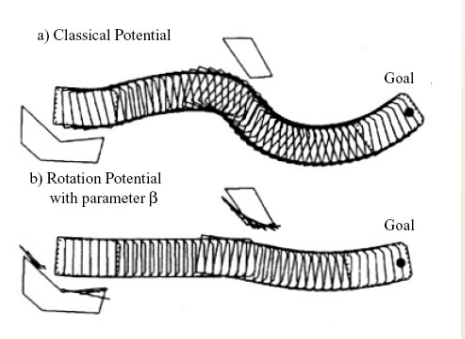
Extended Potential Field Approach

Proposed by Khatib and Chatila, makes use of two additional fields;

- Rotational potential field
- Task Potential Field

Extended Potential Field

- The rotation potential field assumes that the repulsive force is a function of the distance from the obstacle and the orientation of the robot relative to the obstacle.
- A gain factor which reduces the repulsive force when an obstacle is parallel to the robot's direction of travel, since such an object does not pose an immediate threat to the robot's trajectory.



Task Potential Field

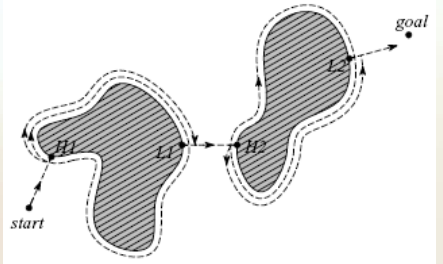
- The task potential field considers the present robot velocity and from that it filters out those obstacles that should not affect the near-term potential based on robot velocity. The result can be smoother trajectories through space.
- Many variations and improvements made on potential fields(lowering oscillations and instability). Easy to implement like grassfire and widely used in robotics.

Obstacle Avoidance

- Focuses on changing robot's trajectory as informed by its sensors during the motion.
- Resulting motion is a function of robot's current and recent sensor readings and its goal position and relative position to the goal.
- Obstacle avoidance algorithms, differences- similarities.

Bug Algorithm

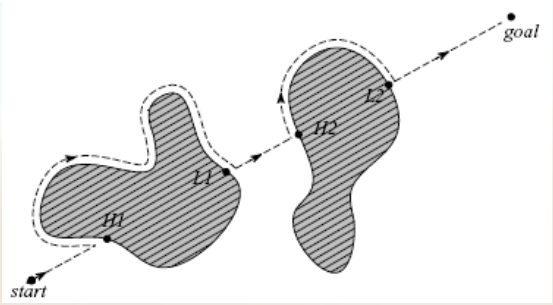
- Several bug algorithm
- Bug1:
  - Fully circle the object
  - Leave it at the nearest point from the goal



- Not really efficient

Bug Algorithm

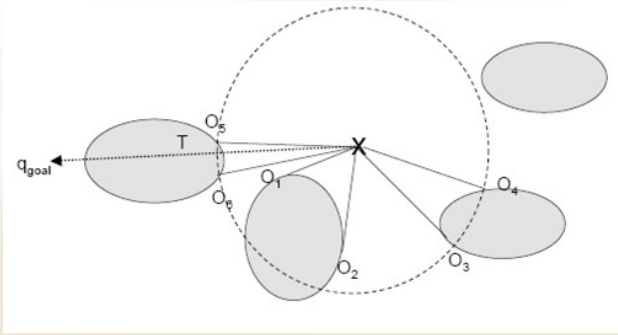
- Bug2:
  - Remember the direct line to the goal
  - Follow the object's contour until it find the direct line



  - Faster than bug1
  - Sometimes can be inefficiant

Bug Algorithm

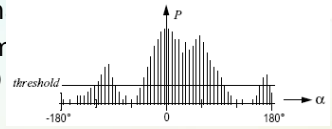
- Tangent Bug
  - Evaluates distance to object



  - Choose the point O_i that minimizes $d(x,O_i)+d(O_i, q_{goal})$

Vector field histogram

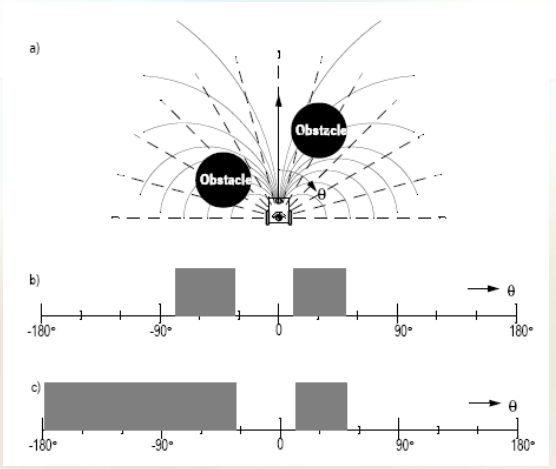
- Prevents problems due to a lack of information about objetcs
- Creating a local map of the environment
- Use occupancy grid as a m
- Generate a polar histogram
  - X-axis: angle of the object (α)
  - Y-axis: probabilities (P) of the object based on the map.



- All the direction are calculated
- The $G = a \cdot \text{target\_direction} + b \cdot \text{wheel\_orientation} + c \cdot \text{previous\_direction}$
  - target_direction = alignment of the robot path with the goal;
  - wheel_orientation = difference between the new direction and the current wheel orientation;
  - previous_direction = difference between the previously selected direction and the new
  - direction.
  - a, b,c depends of the behavior of the robot

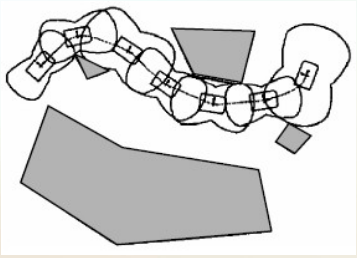
Vector field histogram

- VFH+
  - Takes into account the movements of the robot



The bubble band technique

- Bubble: the maximum local subset of the free space around the robot
- Require a global map or global path-planner
- Used along the trajectory of the robot

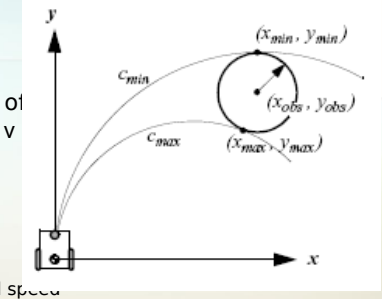


- When object, used to change the trajectory minimizing bubble band tension

Curvature velocity techniques

•The basic curvature velocity approach.

•Velocity space:
- ω : Rotational velocity
- v : Translation velocity



•Robot only move along arc of circle with curvature c= ω x v

•Best way is chosen by an objective function

•Two types of constraints :
- •Limitations in acceleration and speed
- •Object blocking certain v and ω values

•obstacles are approximated by circular objects

•The lane curvature method.

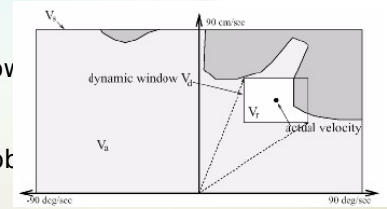
•Calculates a set of lanes (length and width)

Dynamic window approaches

- **The local dynamic window approach.**

•Velocity space:
- ω : Angular velocity
- v : Velocity

•First, selects all the tuple(v, ω) reacheable = dynamic windov



•Second, keeps only safe tuples (the robot wont hit a ob = admissible velocities.

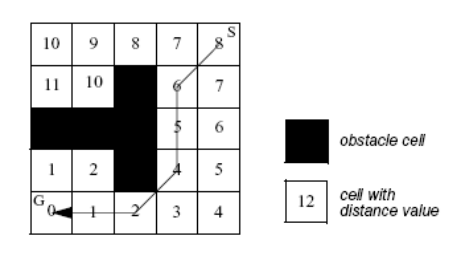
•Objective function will choose the direction :

$$O = a \cdot heading(v, \omega) + b \cdot velocity(v, \omega) + c \cdot dist(v, \omega)$$

•heading = Measure of progress toward the goal location;
•velocity = Forward velocity of the robot → encouraging fast movements;
•dist = Distance to the closest obstacle in the trajectory.

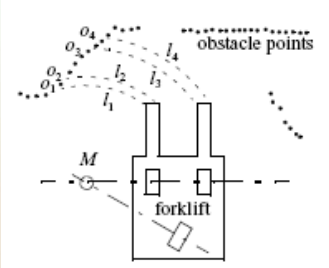
Dynamic window approaches

- **The global dynamic window approach.**
  - **Adds NF1 to the objective function**



The Schlegel approach

- Considers the shape of the robot
- Using a Cartesian grid
- Using precalculated lookup tables
- i_c = curvation
- l_i = distance between the robot and object for i_c



Others approachs

- ASL:
  - Path planning by NF1
  - Path is converted in elastic band
  - Move is chosen by dynamic window
- Nearness diagram
- Gradient method
- Adding dynamic constraints

….