Motion Planning

Article · October 2007

CITATIONS
0

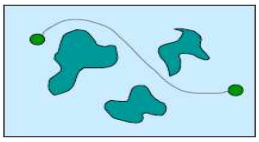
READS
122

1 author:

Mohamed Elhoseiny
Meta
97 PUBLICATIONS **2,422** CITATIONS

SEE PROFILE

Motion Planning



Mohamed Hamdy Mahoud,
Computer systems department, Faculty of computer and information sciences,
AinShams University

Abstract

Motion planning is one of the most challenging research topics .and so such point has big Interest of Academic audience. Thus a good point to be viewed here is a road map to Motion planning as a science, Giving details about Motion planning algorithms and the Math –Behind.

Introduction

There exist at this age of Information technology very greedy-humans to break a crack the great wall between human and computer, between things done by human and still not done by computer. One of these things are motion planning is the basic Idea that enable Machines to do things that It can not do before.

Some of the Ideas that give more enthusiasm for this field can be...

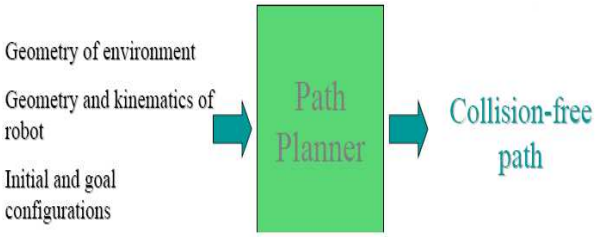
- Person 1 walks through some obstacles
- Person 1, looking at Person 2, directs Person 2 through obstacles.
- Person 1, looking at Person 2 with eyes closed, directs Person 2 through obstacles.
- Person 1, looking at a map and not Person 2,whose eyes are still closed, directs Person 2 through obstacles.
- Person 1, looking at an object and Person 2.whose eyes are closed, directs Person 2 to grab an obstacle.
- How do we do the last experiment with a map?

For solving these problems things need to be assumed to make us able to solve the problem programmatically.

- Perfect sensors-what information.
- Perfect control.
- Perfect Thinking. Knowledge of the world?complete?Processing the world?etc?

Here after a problem is illustrated. A formal definition of motion planning is needed.

Motion planning as a problem is the way to compute a continuous sequence of collision-free robot configurations connecting the initial and goal configurations.



This can be defined in mathematical Notations as follows.

- If **W** denotes the robot's workspace, and $\mathbf{W_{Oi}}$ denotes the i^{th} obstacle, Then the robot's free space, $\mathbf{W_{free}}$, is defined as: $\mathbf{W_{free} = W - (U\ W_{Oi})}$.
- The Collision-free path is defined as And a path **c** is $\mathbf{C^0}$ (i.e. continuous) | **c : [0,1]** → $\mathbf{W_{free}}$ where **c(0)** is $\mathbf{q_{start}}$ and **c(1)** is $\mathbf{q_{goal}}$.

Motion Planning is a big problem. So let its components be defined.

What are the problem aspects?
Aspects include Navigation, Converge, Localization and mapping.

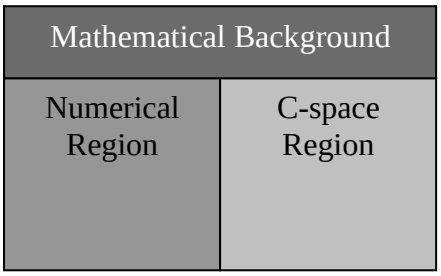
What aspects of Objects to deal with (e.g. Robots, obstacles)?
Aspects include degrees of freedom, Kinematics vs. Dynamic, and homonymic or not.

What ere the algorithmic aspects for Planning?
Aspects include optimality,computional complexity,completeness,Feedback or not,Sensor Based or-not and Online or offline

Mathematical Background

Motion planning as a science is inherently based on computational geometry algorithms and also numerical analysis. This section covers a good part of Motion planning mathematical picture.

Motion planning Algorithms have two mathematical regions a one depends on numerical approached and the other depends on Space definition and a term named C-Space (i.e. Configuration space).



Numerical Region

Numerical techniques have a good approached for digitizing many problems to be implemented on the computer. One of the problems is iterative solution of optimizing function value. That is based on *gradient* vector.

The basic idea that relates gradient and optimization is the directional derivative (i.e. Derivative of the function towards a specific vector **u**) that is defined as follows.
$F_u(x1,x2,\ldots,xn) = a1*F_{x1}(x1,x2,\ldots,xn)+ ...+\ldots ai*F_{xi}(x1,x2,\ldots,xn),...+ a3*F_{xn}(x1,x2,\ldots,xn)$.
Where F_{xi} is $\delta(F(x1, x2,\ldots,xn))/\delta xi$.
And **u** equals (a1,… ,an).

This can be abbreviated as
$F_u = \mathbf{u}.\nabla F$,where ∇F is the gradient vector of function F (i.e. $\nabla F = [F_{x1},\ldots F_{xn}]^T$).

But $F_u = \mathbf{u}.\nabla F = \|u\| \|\nabla F\| \cos(\Theta)$ .and **u** is almost defined as a normalized and so $\|u\|=1$. Then $F_u = \mathbf{u}.\nabla F = \|\nabla F\|*\cos(\Theta)$.the derivative means the rate of change of the function.

And so the rate of change that can lead to reduction of the value of the function is achieved when it reaches the largest negative value. This can be achieved when $\cos(\Theta)= -1$ (i.e. its minimum value)
$F_u = \mathbf{u}.\nabla F = -\|\nabla F\|$.This means that the greatest rate of change that minimizes the vector is always towards the opposite direction of the gradient vector, and the greatest rate change that minimizes the vector is always towards the direction of the gradient vector (i.e. $\cos(\Theta)= 1$) $F_u = \mathbf{u}.\nabla F = \|\nabla F\|$.).

The previously discussed idea can be used to find the path from the starting point to the goal minimizing the distance to the goal and maximizing the distance to obstacles.
Note: This is the basis of *potential field-algorithm* as will be discussed later.

Configuration-space(C-Space) Region

The source of configuration space as a term comes from mechanics. The **configuration space** is the space of possible positions that a physical system may attain, possibly subject to external constraints. In other words it is a complete specification of the position of every point in the system.

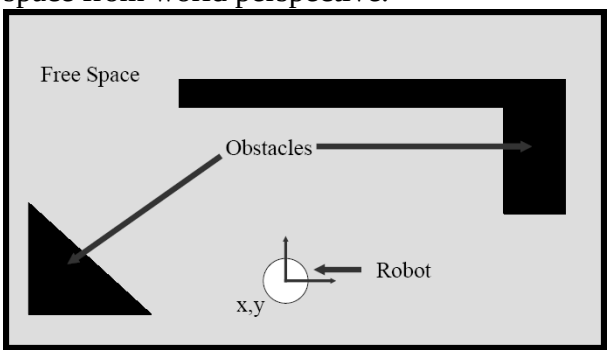
C-space can be simply exemplified as a sufficient representation of a robot that translates in the plane or a rotating bar fixed at a point or a rotating bar that translates along the rotation axis and etc.

C-space can be expressed mathematically as follows (for just a one robot).

- Let q denote a point in a configuration space Q.
- The path planning problem is to find a mapping $c:[0,1]\rightarrow Q$ such that no configuration along the path intersects an obstacle.
- A configuration space obstacle Q_{Oi} is the set of configurations q at which the robot intersects W_{Oi} (i.e. workspace of obstacle i) , that is $Q_{Oi} = \{q \in Q \mid R(q) \cap W_{Oi} \neq \emptyset\}$.
- The free configuration space (or just free space) $Q_{free} = Q \setminus (\cup Q_{Oi})$

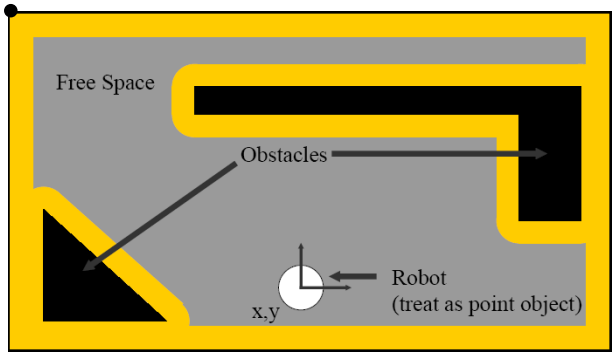
As previously illustrated, World has a free-space and obstacle space, also Configuration has a free-space and obstacle space, then what is the difference?

To illustrate the difference, look at the following figure, it shows free and obstacle space from world perspective.



World space just defines objects in such world and accordingly obstacle-space is the space that obstacle objects have (i.e. taking the world boundaries into account) and free-space is the rest of the world.

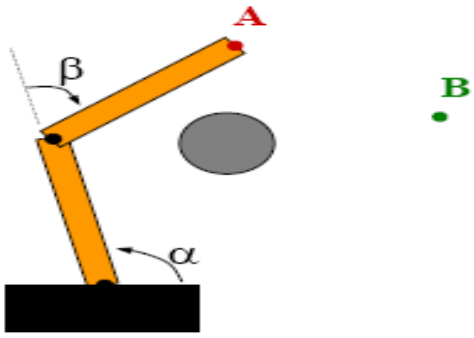
Then, Let the following figure be the illustration of Configuration space for the same case.



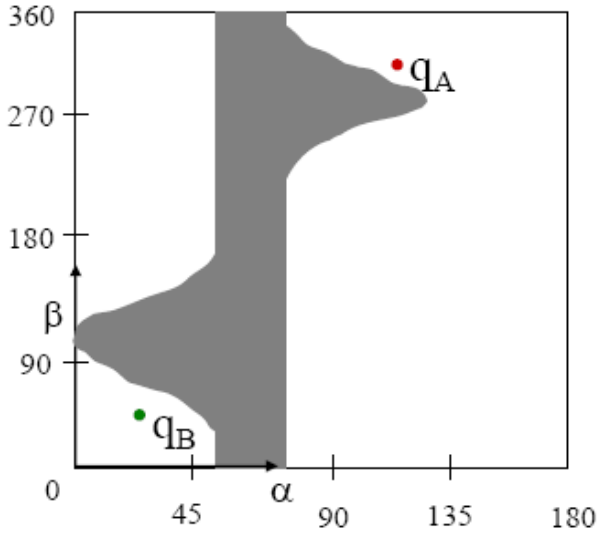
Configuration space can be simply defined by the area in which a robot can navigate through and because the moving objects has a space , let us choose centroid point to be point represents the robot. It can be noted that such point can not move to all points in the world free space. For example, it can not be moved to the upper-left point in the world, because robot has boundaries that disable its centroid to touch that point (i.e. Robot can not move with centroid in yellow colored area).

Here it can be noted that if we assume that the robot has zero-space (i.e. a point robot) then the configuration space is the same as world space.

C-space as defined depends or type of transformation that can be applied on a robot. The previous example has type of transformation to be translation only (i.e. rotation does not affect C-space because the robot is a circle).so let's have an example as follows



The robot consists two free rotating links (i.e. with α is an angle between 0 and 180 ,β is between 0 and 360).with the goal is to find the path from Point A to point B.
For the given situation, configuration space is defined on these angles as follows



C-space is defined in such way just because it is simpler to represent. given the value of angles α,β the transformed robot can be represented.

As noted in the previous example the robot consists of two parts. Two parts may collide or generally for multiple-part robot, any pair of the parts may collide. For such case taken into account mathematical representation of configuration space will differ to be as follows.

$Q_{free} = Q \setminus Q_{obs.}$

$Q_{obs} = (\bigcup Q_{Oi}) \cup (\bigcup_{\text{foreach } i,j,i\neq j} (Q_{Ai} \cap Q_{Aj}))$.

Transformations

As illustrated there is inherent relation between C-Space and Transformation, The question here how transformations are done mathematically?? The answer can be illustrated through the following lines.

Transformation is done using matrices. And this is also the base of Computer graphics as a science as well. But here are represented only rigid body transformations (i.e. this is because robot can not be changed, e.g. scaled).

Rigid body translation matrix

T=

$$\begin{bmatrix} 1 & 0 & 0 & \text{x-shift} \\ 0 & 1 & 0 & \text{y-shift} \\ 0 & 0 & 1 & \text{z-shift} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

x-shift=translation along the x-axis
y-shift=translation along the y-axis
z-shift=translation along the z-axis

Rigid body rotation matrix

R=

$$\begin{bmatrix} (\cos\psi*\cos\phi+\sin\psi*\sin\theta*\sin\phi) & (\sin\psi*\cos\phi-\cos\psi*\sin\theta*\sin\phi) & (\cos\theta*\sin\phi) & 0 \\ (-\sin\psi*\cos\theta) & (\cos\psi*\cos\theta) & (\sin\theta) & 0 \\ (\sin\psi*\sin\theta*\cos\phi-\cos\psi*\sin\phi) & (-\cos\psi*\sin\theta*\cos\phi-\sin\psi*\sin\phi) & (\cos\theta*\cos\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

θ=rotation around the x-axis (pitch)
ϕ=rotation around the y-axis (roll)
ψ=rotation around the z-axis (yaw)

Topology of Configuration Space

Topology is the "intrinsic character" of a space. Two spaces have a different topology if cutting and pasting is required to make them the same (e.g. a sheet of paper vs. a mobius strip).

Think of rubber figures, if we can stretch and reshape them continuously without tearing one into the other, they have the same topology. From here topological space can be defined as

Homeo- and Diffeomorphisms

- Recall mapping:
  - $\Phi: S \rightarrow T$
  - If each elements of φ goes to a unique T, φ is injective or (1-1).
  - If each element of T has a corresponding pre-image in S, then φ is surjective (or onto).
  - If Φ is surjective and injective, then it is bijective (in which case an inverse, Φ^{-1} exists).
  - Φ is smooth if derivatives of all orders exist (Φ is said to be C^{∞}).
- If $\varphi: S \rightarrow T$ is a bijection, and both φ and φ^{-1} are continuous, φ is a homeomorphism; if such a φ exists, S and T are homeomorphic.
- Homeomorphism where both φ and φ^{-1} are smooth defines diffeomorphism.

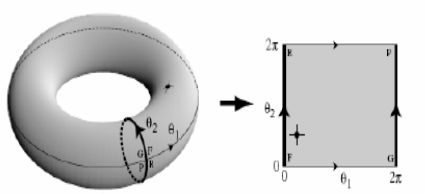
Manifolds

A manifold is a topological space that's locally Euclidean (i.e., around every point, there is a neighborhood that is topologically the same as the open unit ball in R^n). To illustrate this idea, consider the ancient belief that the Earth was flat as contrasted with the modern evidence that it is round. The discrepancy arises essentially from the fact that on the small scales that we see, the Earth does indeed look flat. In general, any object that is nearly "flat" on small scales is a manifold.

In general, any object that is nearly "flat" on small scales is a manifold, and in formal words any object that can be "charted" is a manifold.

A Chart is a pair (U,φ) such that U is an open set in a k-dimensional manifold and φ is a diffeomorphism from U to some open set in $\Re^k$.

- Think of this as a "coordinate system" for U (e.g. lines of latitude and longitude away form the poles).
- The inverse map is a parameterization of the manifold.

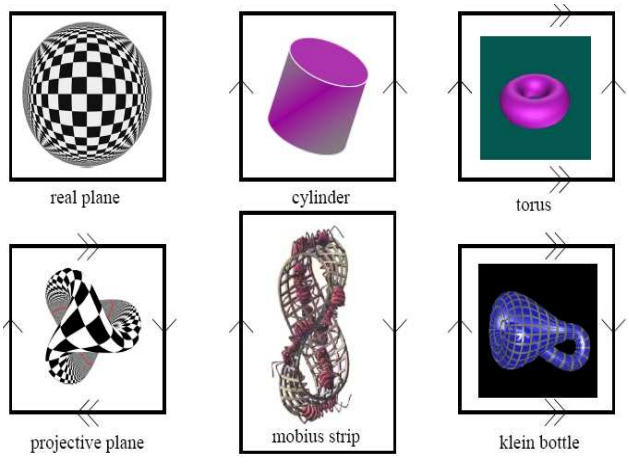


An atlas is a set of charts

- That cover a manifold.
- That are smooth where they overlap (the book defines the notion of C^∞ related for this).

A set S is a differentiable manifold of dimension n if there exists an atlas from S to $\Re^n$ (i.e. For example, this is what allows us locally to view the (spherical) earth as flat and talk about translational velocities upon it) .

The following figures are examples of two-dimensional manifolds.



In motion planning, manifold is used as a standard to represent C-space. The following table represents some types of configuration spaces for transformation that can be applied on a robot.

Type of robot	Representation of Q
Mobile robot translating in the plane	$\mathbb{R}^2$
Mobile robot translating and rotating in the plane	$SE(2)$ or $\mathbb{R}^2 \times S^1$
Rigid body translating in the three-space	$\mathbb{R}^3$
A spacecraft	$SE(3)$ or $\mathbb{R}^3 \times SO(3)$
An n-joint revolute arm	T^n
A planar mobile robot with an attached n-joint arm	$SE(2) \times T^n$

Holonomicity

A robot is **holonomic** if it *can* move to change its pose instantaneously in all available directions. Other wise it is non **holonomic**.

Path Planning Algorithms

What is needed (Background) is Over, what to do (trends) is coming .It's time to begin to. It's time for Algorithms.

Each of the following algorithms has assumptions so that it can work properly. Along the way of exploring Motion Planning algorithms solutions varies for covering simple, hard and harder problems.

Bug algorithms

Bug algorithms are our first step here towards completeness of Motion planning adventure. The 1st things to mention are the assumptions .

Bug algorithms have the following assumptions.

- Known direction to goal.
  - robot can measure distance d(x,y) between pts x and y
- otherwise local sensing
  - walls/obstacles & encoders
- *reasonable* world
  - Finitely many obstacles in any finite area.
  - A line will intersect an obstacle finitely many times**.**
  - Workspace is bounded $W \subset Br(x)$, $r < \infty$, $Br(x) = \{y \in \Re\ (2) \mid d(x,y) < r\ \}$.

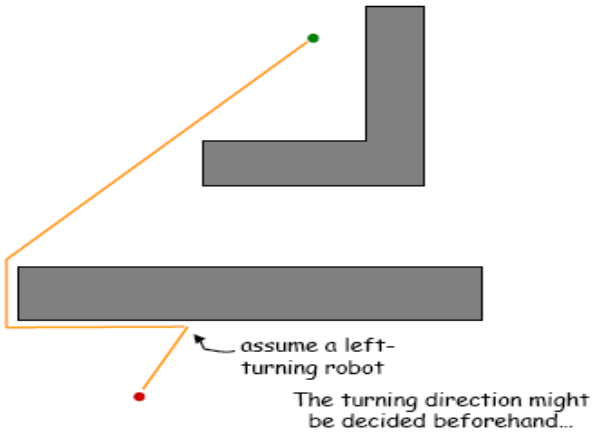
Bug "0" algorithm

Some notation:

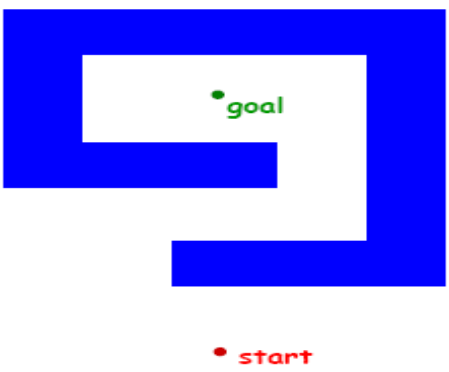
- qstart and qgoal
- "hit point" q^H_i.
- "leave point q^L_i.
- A *path* is a sequence of hit/leave pairs bounded by qstart and qgoal

Algorithm:

1) Head toward goal
2) Follow obstacles until you can head toward the goal again.
3) Continue.

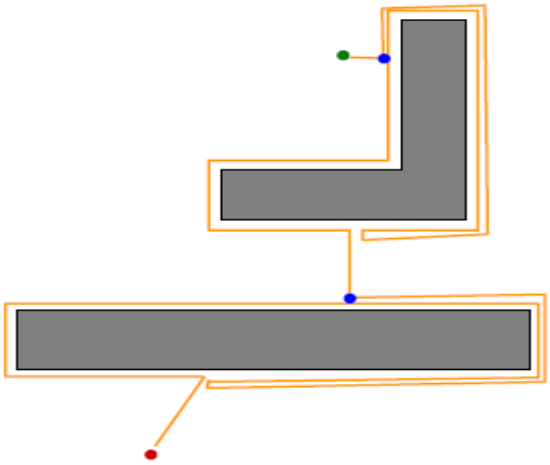


Stop! Bug "0" is bad-It can not solve this



Bug "1" algorithm

1) Head toward goal
2) If an obstacle is encountered circumnavigate it and remember how close you get to the goal
3) Return to that closest point (by wall-following) and continue.



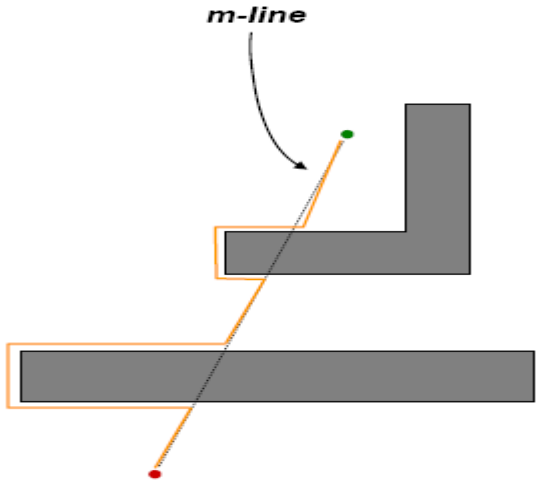
Bug1 be represented formally as follows

- Let q^L_0 = qstart; i = 1.
- Repeat.
  - Repeat.
    - From q^L_{i-1} move toward goal.
  - Until goal is reached or obstacle encountered at q^H_i.
  - If goal is reached, exit.
  - Repeat.
    - Follow boundary recording pt q^L_i with shortest distance to goal.
  - Until q_{goal} is reached or q^H_i is re-encountered.
  - If goal is reached, exit.
  - Go to q^H_i.
  - if move toward qgoal moves into obstacle
    - exit with failure
  - else
    - i = i+1.
    - Continue.

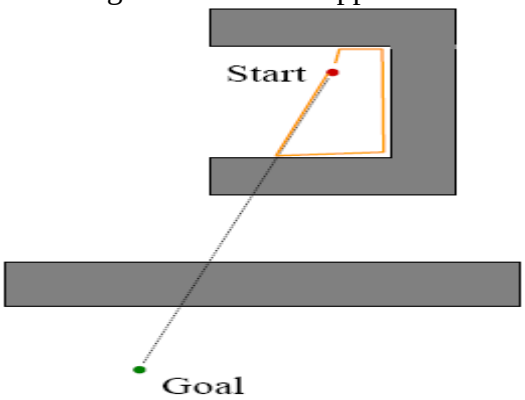
Lower bound is D,where D is straight-line distance from start to goal.
Upper bound is D + 1.5 Σ Pi where Pi is the perimeter of the i^{th} obstacle.

Bug "2" algorithm

1) Head toward goal on the m-line.
2) If an obstacle is in the way, follow it until you encounter the m-line again ***closer to the goal.***
3 Leave the obstacle and continue toward the goal.



If "***closer to the goal***" phrase does not exist the following situation will happen.

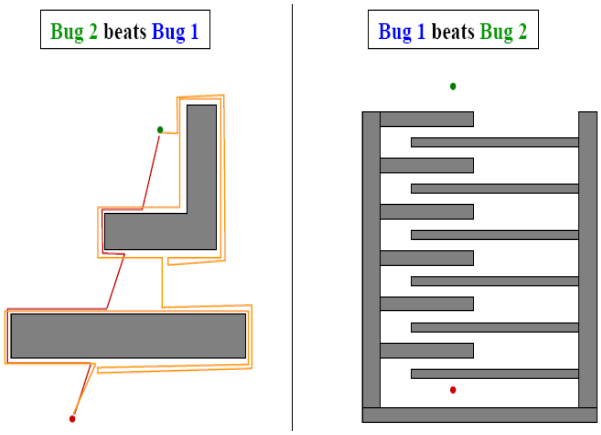


Bug2 be represented formally as follows

- Let q^L_0 = qstart; i = 1.
- Repeat.
  - Repeat.
    - From q^L_{i-1} move toward goal along m-line.
  - Until goal is reached or obstacle encountered at q^H_i.
  - If goal is reached, exit.
  - Repeat.
    - Follow boundary.
  - Until qgoal is reached or q^L_i is re-encountered .or m-line is re-encountered, x is not q^H_i, d(x, q_{goal}) < d(q^H_i,q_{goal}) and way to goal is unimpeded.
  - If goal is reached, exit.
  - If q^H_i is reached, return failure
  - else
    - q^L_i = m.
    - i = i+1.
    - Continue.

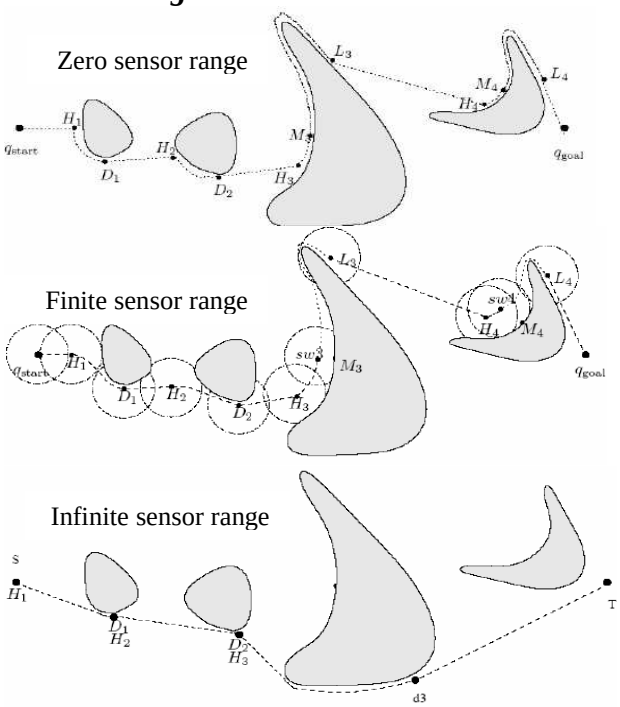
Lower bound is D,where D is straight-line distance from start to goal.
Upper bound is D + 1.5 Σ ni/2 *Pi where ni = # of m-line intersections of the i^{th} obstacle.

BUG1 VS BUG2



BUG 1 is an exhaustive search algorithm (i.e. it looks at all choices before committing)
BUG 2 is a greedy algorithm (i.e. it takes the first thing that looks better)
In many cases, BUG 2 will outperform BUG 1 , but. BUG 1 has a more predictable performance overall.

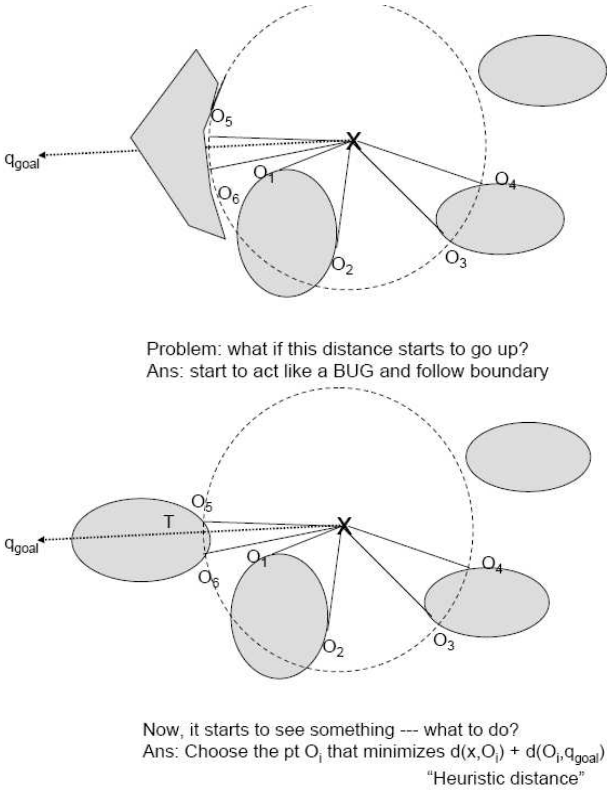
Sensors Ranges



Having non-zero range sensor is an advantage that must be taken into account for efficiency of algorithms .Tangent Bug algorithm is the one that take this point into account.

Tangent bug Algorithm.

Tangent Bug relies on finding endpoints of finite, continuous segments of ρ_R.

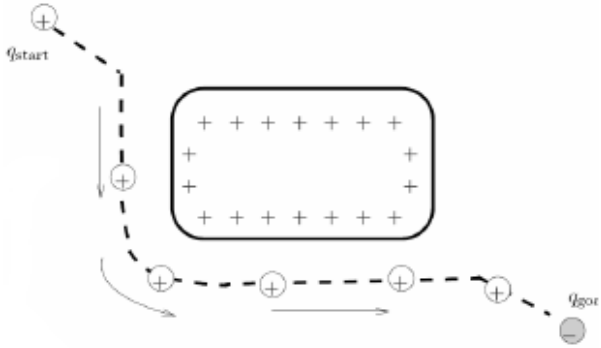


Tangent bug can be represented formally as follows

- Repeat.
  - Compute continuous range segments in view.
  - Move toward n ∈ {T,Oi} that minimizes h(x,n) = d(x,n) + d(n,qgoal) until
    - Goal is encountered, or.
    - The value of h(x,n) begins to increase.
- Follow boundary continuing in same direction as before repeating.
  - Update {Oi}, d-reach and d-followed until.
    - goal is reached
    - A complete cycle is performed (goal is unreachable).
    - d-reach < d-followed

Potential Functions

The idea behind potential functions can be mapped as a spring" drawing the robot toward the goal and away from obstacles or like and opposite charges as illustrated in the figure below. Robot and obstacles are with positive change but the goal is with negative charge. so the robot (i.e. moving object) attracted by the goal getting through a path repulsed from obstacle. The robot moves to a lower energy configuration.



A potential function is a function U: $\Re m \rightarrow \Re$ Energy (i.e. being far from the goal and closer to obstacle) is minimized by following the negative gradient of the potential energy function

$$\nabla U(q) = DU(q)^T = [\frac{\partial U}{\partial q_1}(q), \ldots, \frac{\partial U}{\partial q_m}(q)]^T$$

.

A vector field can be thought over the space of all q's- at every point in time, the robot looks at the vector at the point and goes in that direction.

$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

– Uatt is the "attractive" potential --- move to the goal.
– Urep is the "repulsive" potential --- avoid obstacle.
– U(q) must be differentiable for every q ∈ Qfree.

The Attractive Potential

Uatt(q) = 1/2(ξ*ρ^2_{goal}(q)), where ρ_{goal}(q)is ||q – q_{goal}||(i.e., Euclidean distance), ξ is some positive constant scaling factor.
$F_{att}(q) = -\nabla Uatt(q) = 1/2*(2\rho_{goal}(q)* \nabla\rho_{goal}(q))$
$\nabla\rho_{goal}(q) = \nabla (\sum (x_i - x_{gi})^2)^{1/2}$
= (q – qgoal)/ ρ_{goal} (q).

$$Fatt(q) = -\nabla Uatt(q) = -\xi^*(q - q_{goal}).$$

In some cases, it may be desirable to have distance functions that grow slower to avoid huge velocities far from the goal (one idea is to use the quadratic potential near the goal (< d*) and the conic farther away).

Combined Potential

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

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

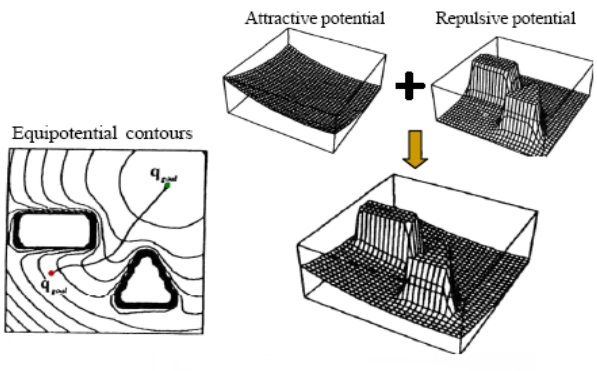
The Repulsive Potential

$$\begin{aligned} \vec{F}_{rep}(q) &= -\nabla U_{rep}(q) \\ &= -\nabla\left(\frac{1}{2}\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2\right) \\ &= -\frac{1}{2}\eta\nabla\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 \\ &= -\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\nabla\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right) \\ &= -\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)(-1)\left(\frac{1}{\rho^2(q)}\right)\nabla\rho(q) \\ &= \eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\left(\frac{1}{\rho^2(q)}\right)\nabla\rho(q) \end{aligned}$$

So

$$\vec{F}_{rep}(q) = \eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\left(\frac{1}{\rho^2(q)}\right)\frac{q - q_c}{\|q - q_c\|}$$

The following graph represents an illustration of using potential function to find path to the gal

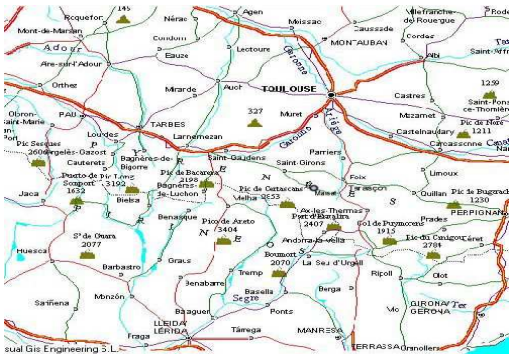


Gradient-descent (Implementation)

- q (0) =qstart.
- i=0.
- while || ∇ U(q(i)) || > ε do
  - q(i+1) = q(i) - α(i) ∇ U(q(i)).
  - i=i+1

Road-Map

The basic idea of road map is to reduce the N-dimensional configuration space to a set of one-D paths to search. The idea can be exemplified as a map of a country. the starting point is a car position the goal position is cinema. Can driver has the map, goal can be reached.



Roadmap

A roadmap is a union of curves such that for all start and goal points in Qfree that can be connected by a path. And thus

- There is a path from $q_{start} \in Q_{free}$ to some q' $\in$ RM.
- There is a path from some q" $\in$ RM to $q_{goal} \in Q_{free}$.
- There exists a path in RM between q' and q".

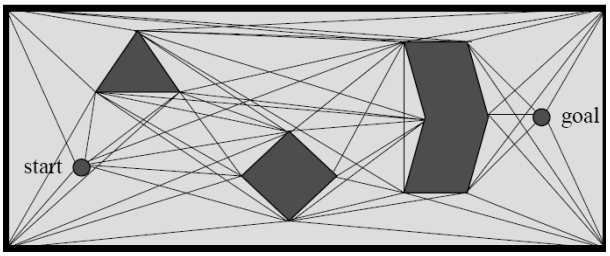
Roadmap has approaches that can be discussed as follows.

Visibility graph

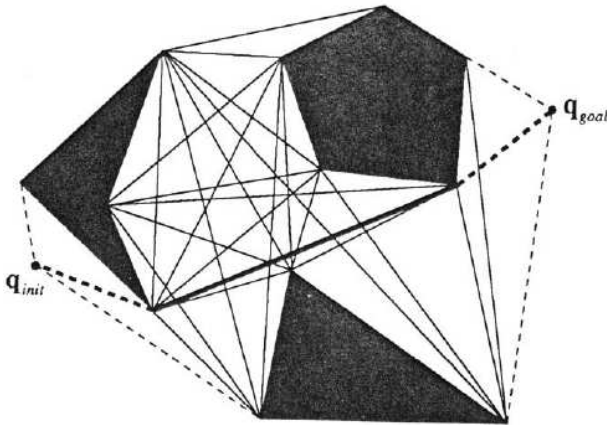
Visibility graph is defined for polygonal (or polyhedral) configuration space where a graph is constructed containing all lines that connect vertices to one another (and that do not intersect the obstacles themselves).some of this lines can exist on the perimeter of obstacles.

A systematic manner of constructing Visibility diagram can be listed as follows.

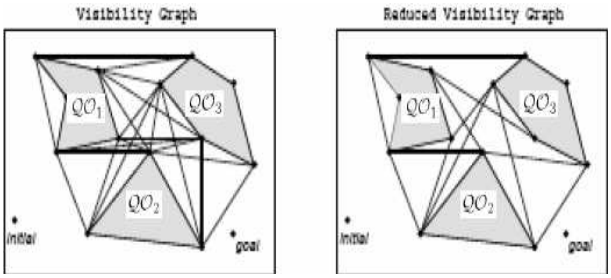
- Draw lines of sight from the start and goal to all "visible" vertices and corners of the world.
- Draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight
- Repeat until you're done.



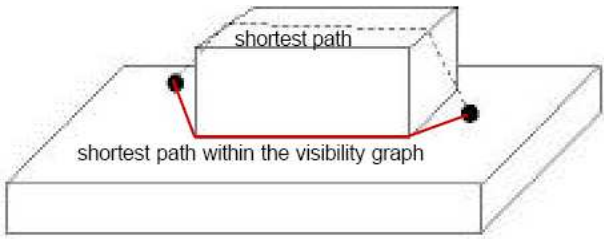
Once this graph is defined a graph search algorithm can be applied seeking minimum path to the goal (e.g. Dijkstra's algorithm $O(N^2)$ |N = the number of vertices in C-space).



Visibility graph has many edges; it can be reduced by making it consists of only nodes that are convex and edges that are tangent(i.e. do not head into the object at either endpoint).



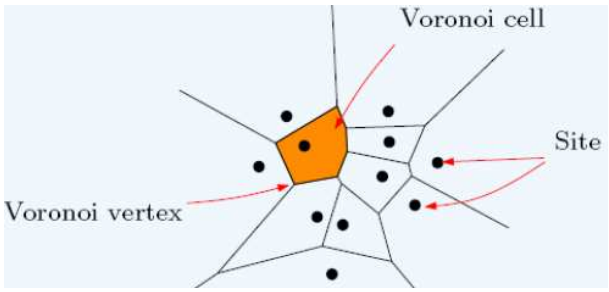
Drawback:
Visibility graphs do not preserve their optimality in higher dimensions In addition, the paths they find are "semi-free," i.e. in contact with obstacles (i.e. No clearance) this can be illustrated in the figure below.



This problem can be avoided in the voronoi diagram

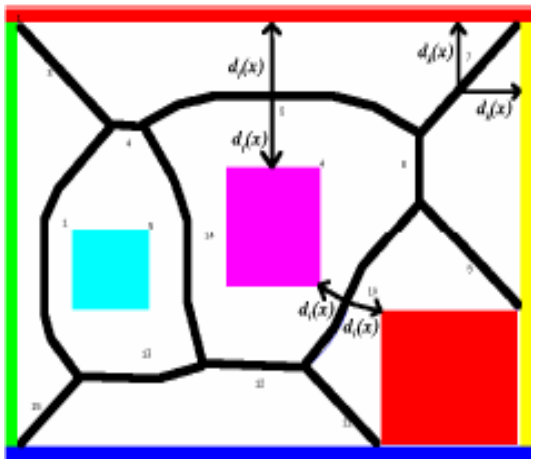
Voronoi diagram

Voronoi diagram maximizes the clearance between the points and obstacles.



Generalized Voronoi Diagram (GVD):
GVD is locus of points equidistant from the closest two or more obstacle boundaries, including the workspace boundary.

GVD is formed by paths equidistant from the two closest objects. GVD maximize the clearance between the obstacles.



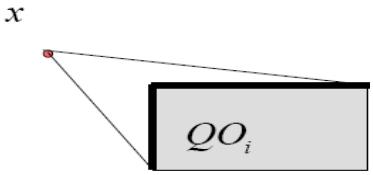
GVD can be mathematically represented by the following equation

$GVD = \cap_{i=1 \text{ to } n-1} \cap_{j=i+1 \text{ to } n} F_{ij}$.

Where $F_{ij} = \{x \in Q_{free}: d_i(x) = d_j(x) \leq dh(x), \forall h \neq i,j\}$.

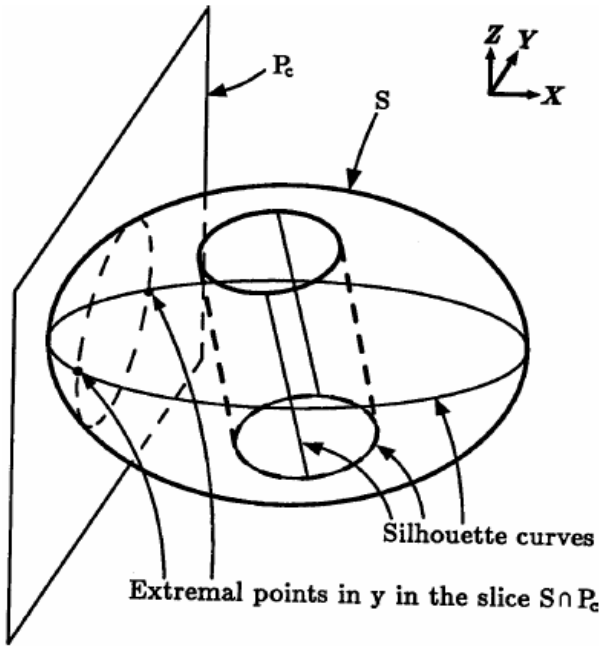
$d_i(x)$ is the distance to Q_{Oi} if $ci \in \tilde{C}(x)$, otherwise $d_i(x) = \infty$.

$\tilde{C}(x) = \{x \in Q_{Oi} \; \forall t \in [0, 1], x(1-t)+ct \in Q_{free}\}$



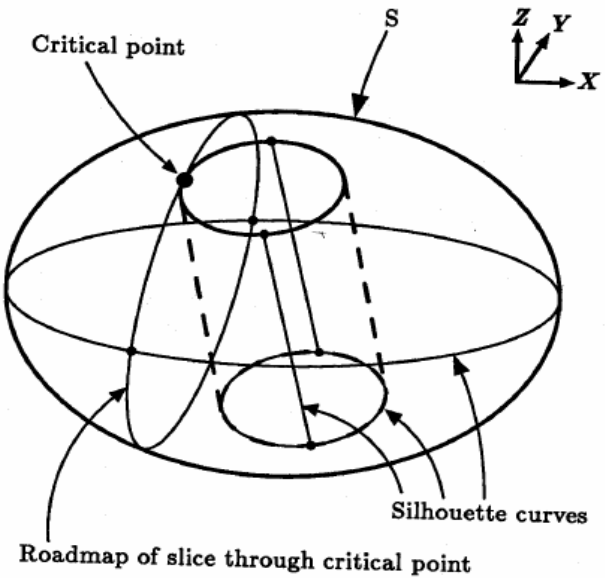
Canny's Roadmap Algorithm

- Determines cells that contain the start and goal. Let S be the ellipsoid with a through hole Pc is a hyper-plane of co-dimension 1 (x = c) which will be swept through S in the X direction.
- At each point the slice travels along X we'll find the extrema in $S \cap P_c$ in the Y direction. If we trace these out we get **silhouette curves**.



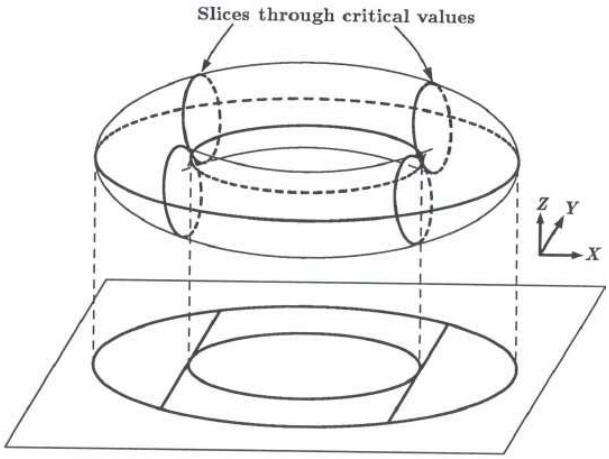
- Then critical point (i.e. A point on a silhouette curve with the tangent to the curve at the point lies in Pc) is connected to the rest of the silhouette curve with a path that lies within $S \cap P_c$. This can be done by running the algorithm recursively.

Each time, we increase the co-dimension of the hyper-plane by 1.



- The recursion is repeated until there are no more critical points or the critical slice has dimension 1 (it is its own roadmap).
- The roadmap is the union of all silhouette curves

Another example



Finding Extrema

This is done using *Lagrange Multiplier Theorem*:

Let S be an n-surface in $\Re n+1$, $S=f^{-1}(c)$ where $f: U \rightarrow \Re$ is such that $\nabla f(q) \neq 0 \; \forall q \in S$.

Suppose $h: U \rightarrow \Re$ is a smooth function and $p \in S$ is an extremum point of h on S.

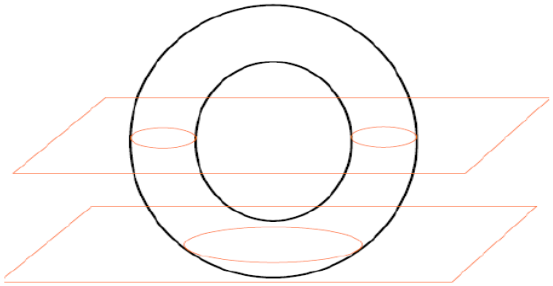
Then $\exists \lambda \in \Re$ such that $\nabla h(p) = \lambda \nabla f(p)$ (they are parallel)

Example:
Consider $S=f$ -1(0) where $f=x2+y2+z2-1$ (a solid unit sphere). Find extrema of $h=\pi1(x,y,z)=(x)$.

D (f,h) = $\begin{bmatrix} 2x & 2y & 2z \\ 1 & 0 & 0 \end{bmatrix}$

y = z = 0 (y-z plane) and only points on sphere is x = 1, x = -1, left most and right most.

After road map is built it will be like this.



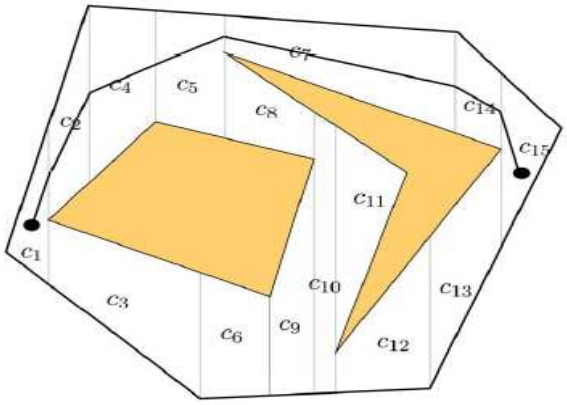
Cell-decomposition Methods

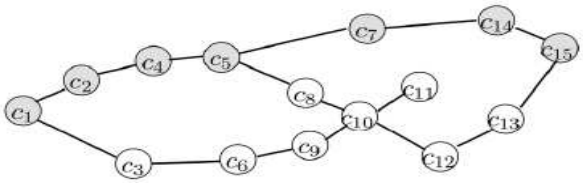
The idea behind is to decompose the world into cells. And the planner will do two steps.

- Determines cells that contain the start and goal.
- Planner searches for a path within adjacency graph.

Trapezoidal (2-D Vertical) decomposition

Trapezoidal decomposition is decomposition of the free space into trapezoidal & triangular cells



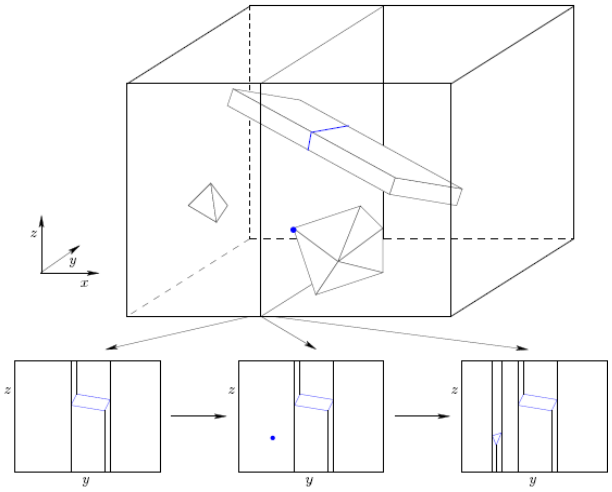


Implementation-O (n^2)

- Input is vertices and edges.
- Sort n vertices O(n log(n)).
- Determine vertical extensions.
- For each vertex
  - intersect vertical line with each edge { O(n)}.

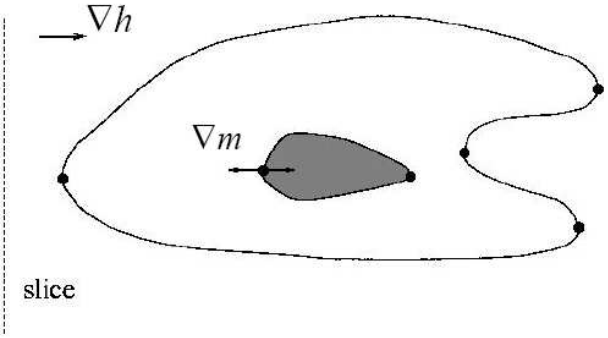
3-D Vertical decomposition.

Trapezoidal decomposition idea can be extended to any dimension by recursively applying the sweeping idea. The following figure illustrates the algorithm that constructs the 3D vertical decomposition.



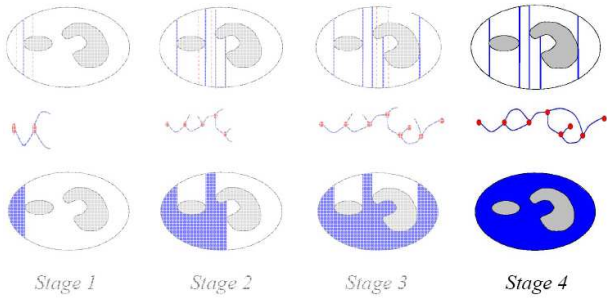
Morse Decomposition

It is better to define Morse in a diagram as follows

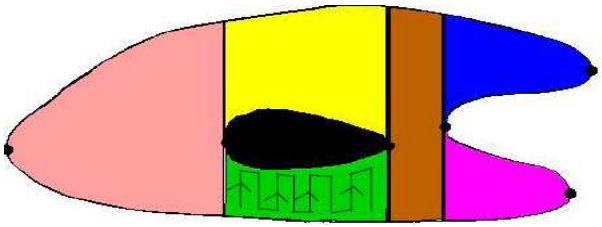


Where h(x,y) is a slice function with gradient ∇h.
At a critical point x of $h|_M$ ∇ h (x) = ∇m (x) where M = {x |m(x)=0}.

The following diagram is Morse decomposition in stages (i.e. incremental construction) with h(x,y)= x.

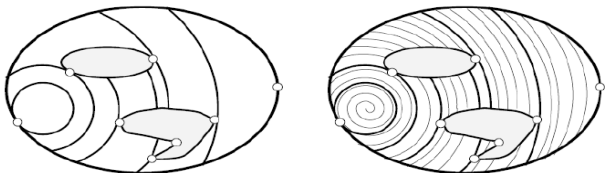


It must be noted that each cell can not be identified by only critical points (i.e. they are curved) so each cell can be covered here by back and forth motions.

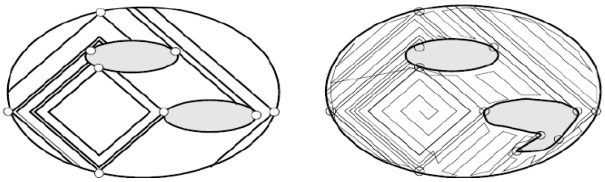


Morse decomposition can also have different slicing function giving different cellular topology as follows.

$$h(x,y) = x^2 + y^2$$



$$h(x,y) = |x| + |y|$$

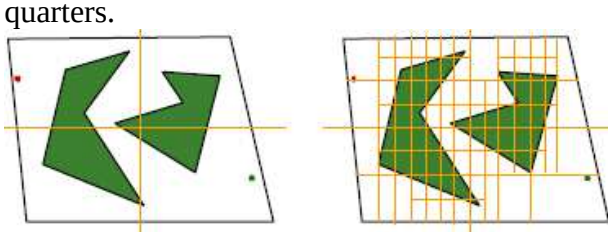


Notes

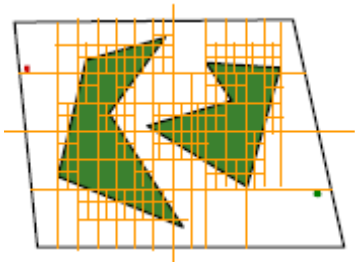
- Hybrid techniques can use Morse decomposition with voronoied world (i.e. world partitioned into voronoi regions).
- As an advanced extension for the topic is the decomposition for non-linear model .e.g. Line segment with SE (3)-configuration-For more information refer to references-section.

Quad-tree decomposition (as an example of approximate cell decomposition)

It recursively subdivides each *mixed* Quad-tree: obstacle/free (sub) region into four quarters.



At a certain level of resolution, only the cells whose interiors lie entirely in the free space are used.

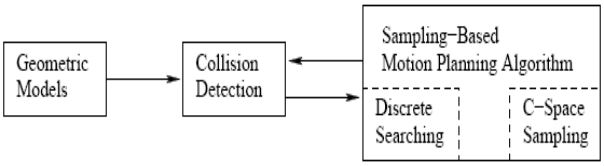


A search in this graph yields a collision free path, has disceretized-systems- drawbacks.

Probability (Sampling-Based) planning

The key idea is rather than exhaustively to explore *ALL* possibilities; randomly explore a smaller subset of possibilities while keeping track of progress

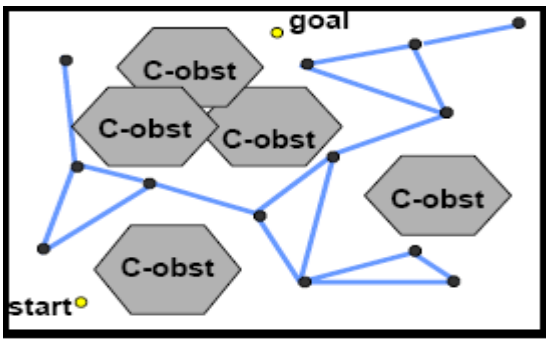
The sampling-based planning philosophy uses collision detection as a “black box” that separates the motion planning from the particular geometric and kinematic models. C-space sampling and discrete planning (i.e., searching) are performed. These words can be figured as follows.



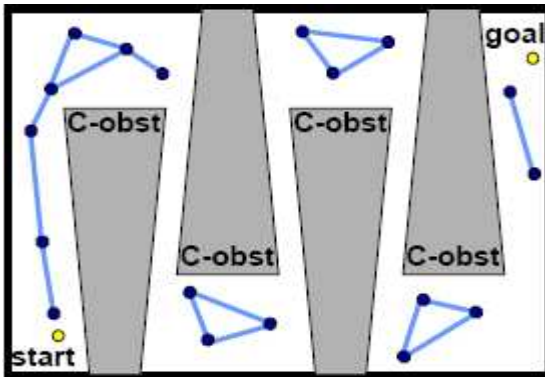
Examples includes Potential field with probability approach (i.e. depend on expectation rather than exact function)

Also there exists a road map approach for such kind of planning, named "Probabilistic RoadMap"

Sample-based: The Good News
1. Probabilistically *complete*
2. Do not construct the C-space
3. Applies easily to high-dimensional C-space
4. Support fast queries with enough preprocessing



Sample-Based: The Bad News
1. Don't work as well for some problems:
   – Unlikely to sample nodes in *narrow passages*
   – Hard to sample/connect nodes on constraint surfaces
2. No optimality or completeness



Sample-based planning can be procedured into two steps.

- Learning Phase.
  - Construction Step.
  - Expansion Step
- Query Phase.

Learning phase

this phase construct a probabilistic roadmap by generating random free configurations of the robot and connecting them using a simple, but very fast motion planner, know as a *local*

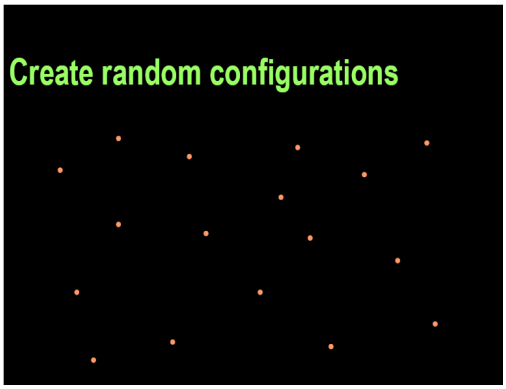
In addition .a graph is stored whose nodes are the configurations and whose edges are the paths computed by the *local planner*.

Construction Step

- Initially, the graph **G = (V, E)** is empty.
- Then, repeatedly, a *random free configuration* is generated and added to V.
- For every new node c, select a number of nodes from V and try to connect c to each of them using the *local planner*.
- If a path is found between c and the selected node v, the edge (c,v) is added to E. The path itself is not memorized (usually).

Random free configuration determination

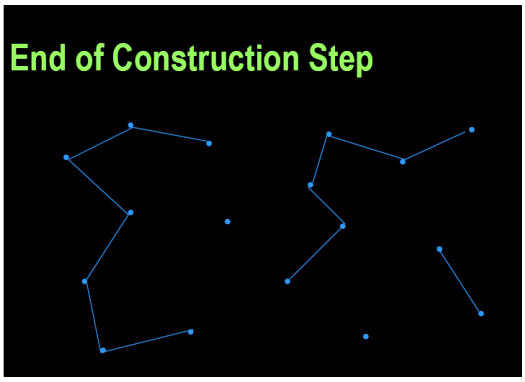
- Draw each of its coordinates from the interval of values of the corresponding degrees of freedom. (Use the uniform probability distribution over the interval).
- Check for collision both with robot itself and with obstacles.
- If collision free, add to V, otherwise discard.
- What about rotations? Sample Euler angles give samples near poles, what about quaternion?- This is HUGE TOPIC, for more info refer to references section.



local planner

Can pick different ones

- Nondeterministic – have to store local paths in roadmap
- Powerful - slower but could take fewer nodes but takes more time
- Fast - less powerful, needs more nodes.



Expansion Step

- Sometimes G consists of several large and small components which do not effectively capture the connectivity of Qfree
- The graph can be disconnected at some narrow region
- Assign a positive weight w(c) to each node c in V

W(c) is a heuristic measure of the "difficulty" of the region around c. So w(c) is large when c is considered to be in a difficult region. We normalize w so that all weights together add up to one. The higher the weight, the higher the chances the node will get selected for expansion

How to choose W(c)?

Can pick different heuristics

- Count number of nodes of V lying within some predefined distance of c.
- Check distance D from c to nearest connected component not containing c.
- Use information collected by the local planner.(If the planner often fails to connect a node to others, then this indicates the node is in a difficult area).

One example is

$$r_f(c) = \frac{f(c)}{n(c)+1}$$

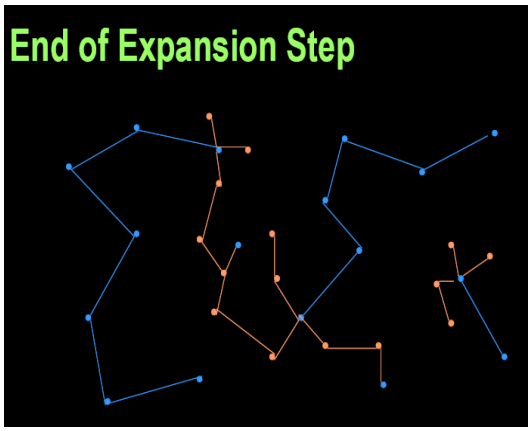
Where n(c) is the total number of times the local planner tried to connect c to another node and f(c) is the number of times it failed.

At the beginning of the expansion step, for every node c in V, compute w(c) proportional to the failure ratio.

$$w(c) = \frac{r_f(c)}{\sum_{a \in V} r_f(a)}$$

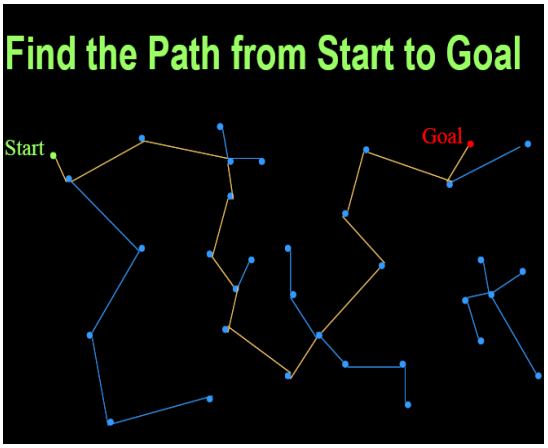
To expand a node c, we compute a short random-bounce walk starting from c.
This means

- Repeatedly pick at random a direction of motion in C-space and move in this direction until an obstacle is hit.
- When a collision occurs, choose a new random direction.
- The final configuration n and the edge (c,n) are inserted into R and the path is memorized.
- Try to connect n to the other connected components like in the construction step.
- Weights are only computed once at the beginning and not modified as nodes are added to G.



Query phase

- Let start configuration be s
- Let goal configuration be g
- Try to connect s and g to Roadmap R at two nodes ŝ and ĝ, with feasible paths Ps and Pg. If this fails, the query fails.
  - Consider nodes in G in order of increasing distance from s (according to D) and try to connect s to each of them with the local planner, until one succeeds.
  - Random-bounce walks.
- Compute a path P in R connecting ŝ to ĝ.
- Concatenate Ps, P and reversed Pg to get the final path.



Conclusion

Such report is not a complete coverage of motion planning as a great science .It is a big picture about it and exposing most of algorithms to be used. The report can be used as just the shortest gateway for Motion planning area.

As a summarization, The Report can cover Motion Planning to be as follows.

Approaches.

- Hybrid local/global (Bug Algorithms).
- Potential Fields.
- Roadmap approaches
  - Visibility Graph.
  - Voronoi Diagram,..etc
- Cell decomposition
  - Exact Cell Decomposition (All algorithms mentioned in the report except quad-tree, e.g. Trapezoidal).
  - Approximate Cell Decomposition (e.g. Quad-tree)

Sampling Approach(Probability planning)

- Probability Roadmap (PRM) is an example

References

- Planning Algorithms, Steven M. LaValle.
- http://voronoi.sbp.ri.cmu.edu/~motion/
- http://parasol.tamu.edu/~amato/Courses/padova04/lectures/.
- http://en.wikipedia.org/wiki/Configuratio_space.
- http://bishopw.loni.ucla.edu/AIR5/rigidbody.html#R.
- http://mathworld.wolfram.com.
- http://www.ece.mtu.edu/ee/faculty/jitan/course/MobileRobotics/notes.
- http://en.wikipedia.org

Stanford University has built a good kit for motion planning. It can be contacted and also software can be downloaded by the following links

- http://robotics.stanford.edu/~mitul/mpk/
- http://ai.stanford.edu/~mitul/mpk/pqp/