10. Cspace transform and path planning

Mechanics of Manipulation

Matt Mason
matt.mason@cs.cmu.edu

http://www.cs.cmu.edu/~mason

Carnegie Mellon

plannin

Outline.

- Background
  - Pick and place
  - Problems
  - Practice
- Cspace transform
  - An example
  - Simple cases
- Path planning
  - Visibility graph
  - Best first
  - Nonholo

Kinematic manipulation

Don't all manipulation problems involve force?

We can treat lots of manipulation problems as pure kinematics problems.

We can address kinematic issues faced by all manipulation tasks.

Pick and place

Let `nextblock()` describe next block to be moved;

Let `start(block)` give the initial location of `block`;

Let `goal(block)` give the goal location of `block`.

Pick and place follows the pattern:

```
FOR block = nextblock()
    MOVETO start(block)      ; pick block
    CLOSE
    MOVETO goal(block)       ; place block
    OPEN
```

Pick and place assumptions

Each object is attached either to the fixed frame or the moving (effector) frame.

Pick and place manipulation assumes:

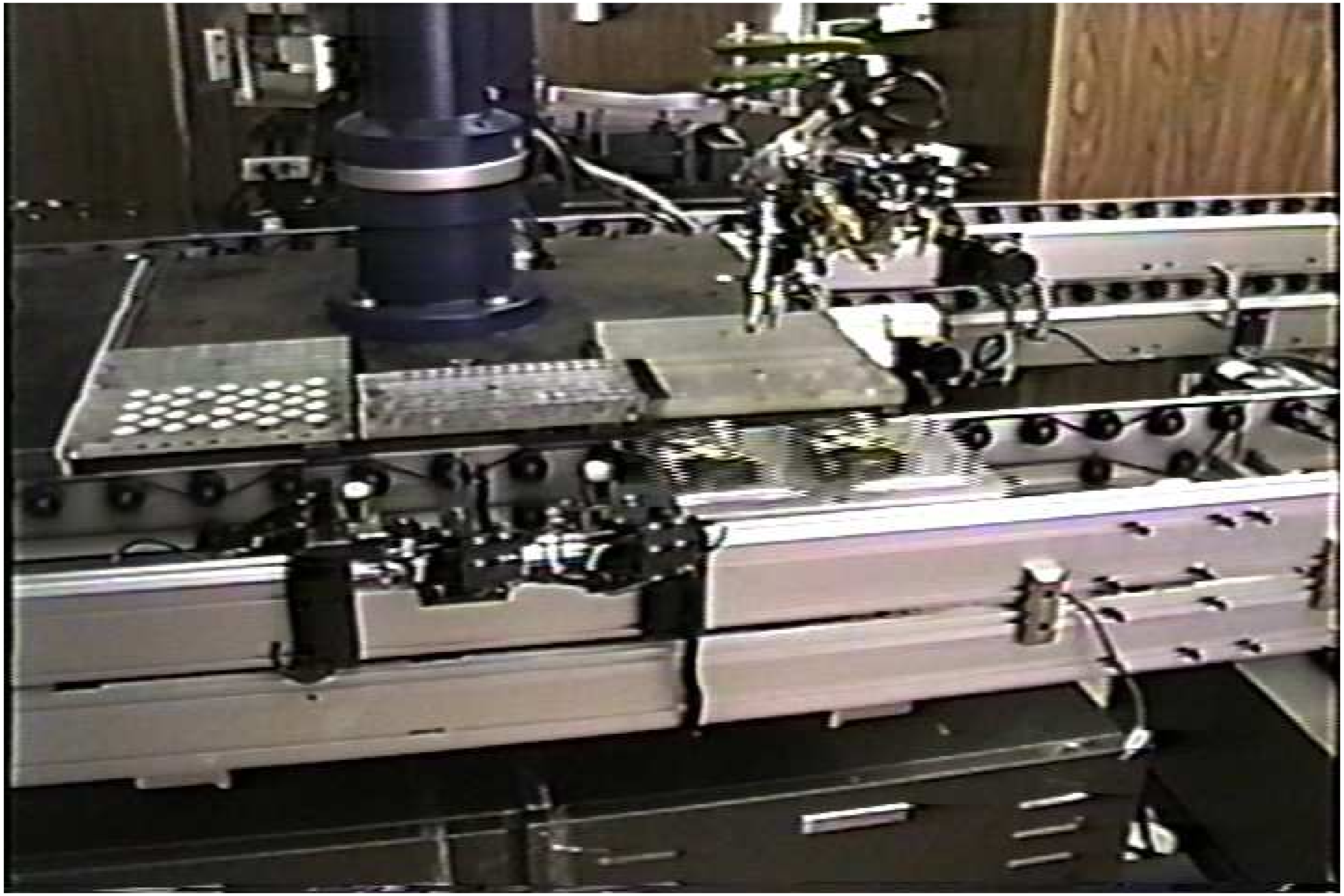
- `CLOSE` attaches the object rigidly to the effector;
- `OPEN` attaches the object to the fixed frame;
- The robot follows the path exactly.

Each assumption is a postponed problem:

- Grasp planning: choosing effector and finger motions that will produce a stable grasp for each object.
- Stable placement.
- Compliant motion: errors in sensing and control lead to collisions interleaved with compliant motions.

Pick and place in practice

Pick and place is common in factory automation. (SONY video)



(There's a lot more than just pick and place going on!!)

Moving problems offline

Grasp planning: design a different gripper for each part;

Known initial pose: parts orienting machine;

Stable placement: design for assembly.

Collision avoidance:

> Every assembly is vertical;
>
> Space above is guaranteed clear—a motion bus;
>
> `DEPROACH`; `MOVETO`; `APPROACH`.

Industrial manipulation has all the problems of general purpose robotics, but they deal with many of them offline.

The Cspace transform

Let $\mathbf{q}$ be the configuration of some moving object A;

let $\mathbf{Q}$ be the set of all $\mathbf{q}$, the *configuration space*;

let B be some fixed object;

let $CO_A(B)$ be the set of all $\mathbf{q}$ such that A at $\mathbf{q}$ intersects B.

$CO_A(B)$ is the **Cspace obstacle**

Path planning (collision avoidance) often means finding a path $\mathbf{q}(t)$ from given $\mathbf{q}_{\text{init}}$ to given $\mathbf{q}_{\text{goal}}$ without passing through the interior of $CO_A(B)$.

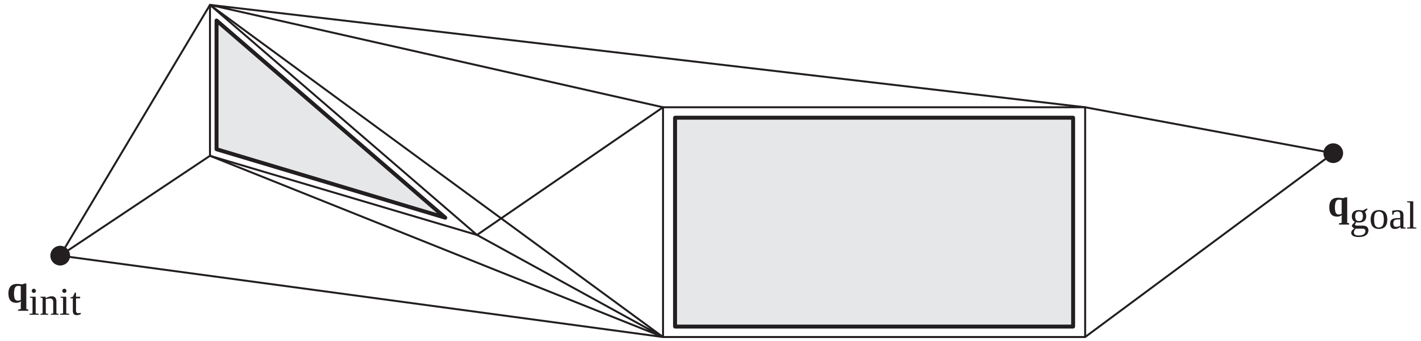
Example 1: point in plane

The moving object A is a point in the plane;

the fixed objects B_i are polygons;

define **visibility graph**: for every vertex pair, include the line segment if it is in free space;

search the visibility graph.



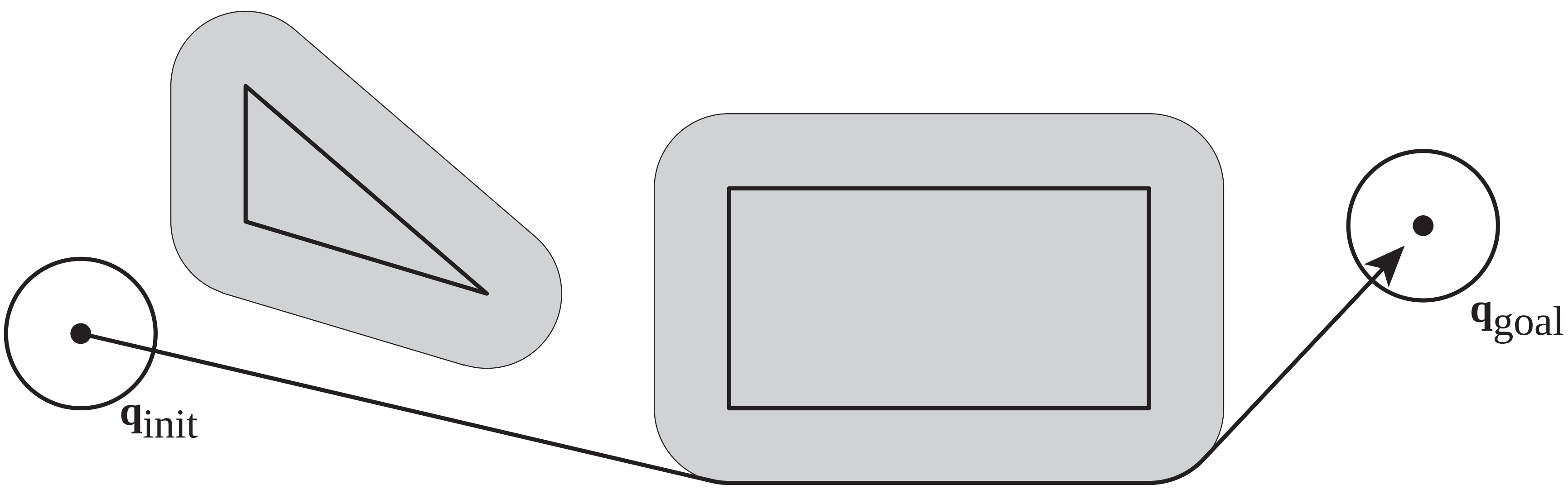
Example 2: disk in plane

Especially good for cylindrical indoor mobile robots!

The moving object A is a disk in the plane;

Fixed objects are polygons (circular arcs easily incorporated);

Visibility graph includes all bi-tangents in free space.



Example 3: Translating polygon in plane

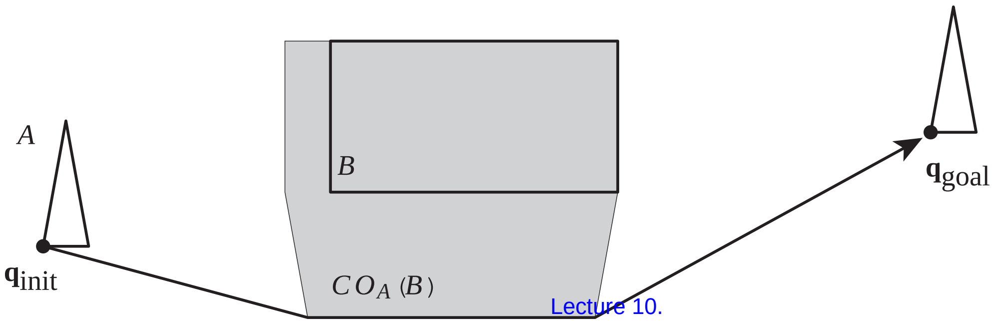
$$\text{collision} \leftrightarrow \exists_{\mathbf{a}\in A,\mathbf{b}\in B}\ \mathbf{a}+\mathbf{q}=\mathbf{b}$$

The Cspace obstacle is

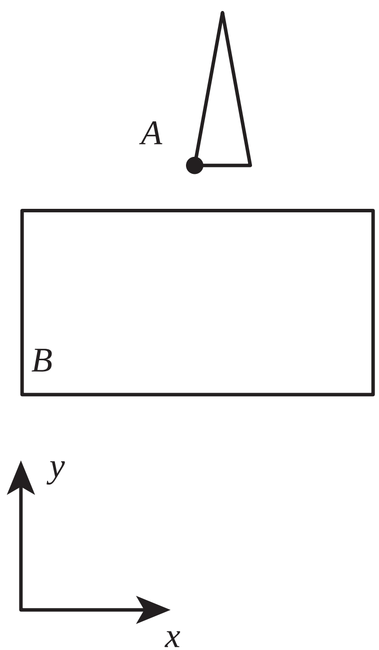
$$\begin{aligned} CO_A(B) &= \{\mathbf{q} \mid \exists_{\mathbf{a}\in A,\mathbf{b}\in B}\ \mathbf{a}+\mathbf{q}=\mathbf{b}\} \\ &= \{\mathbf{b}-\mathbf{a} \mid \mathbf{a}\in A, \mathbf{b}\in B\} \\ &= B \ominus A \end{aligned}$$

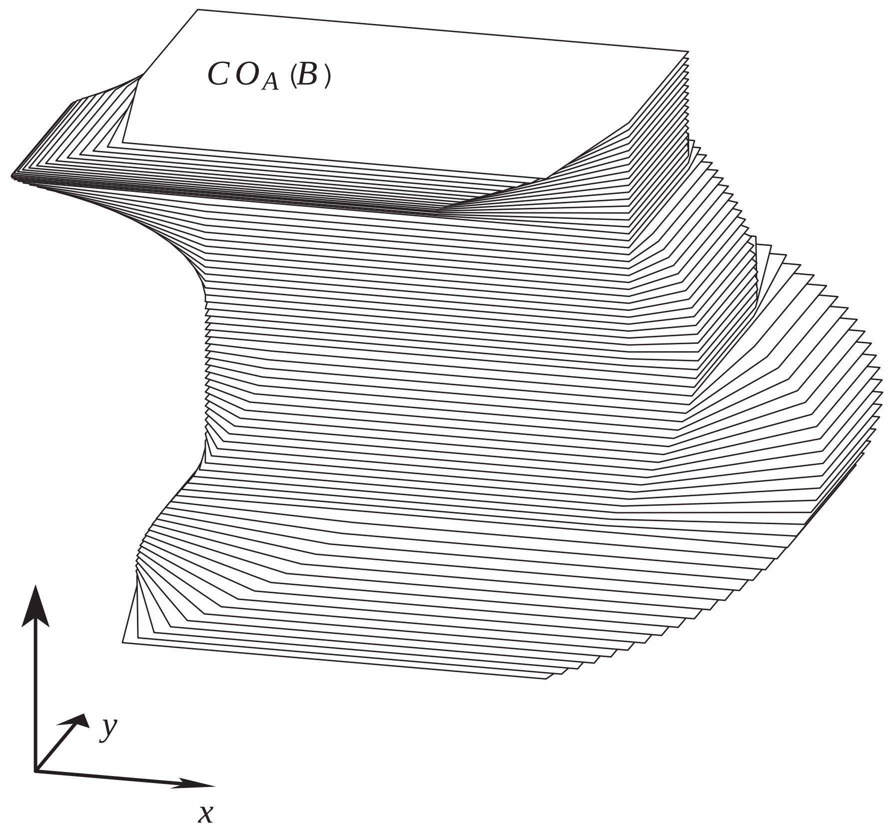
("$\ominus$" is *Minkowski difference*.) For A and B convex:

$$CO_A(B) = \mathrm{conv}(\mathrm{vert}(B) \ominus \mathrm{vert}(A))$$

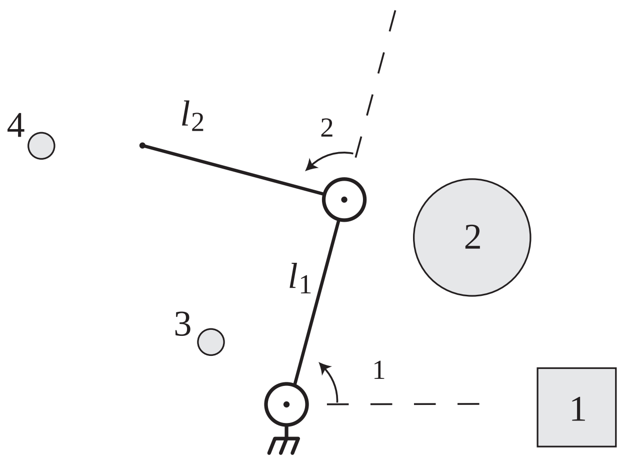


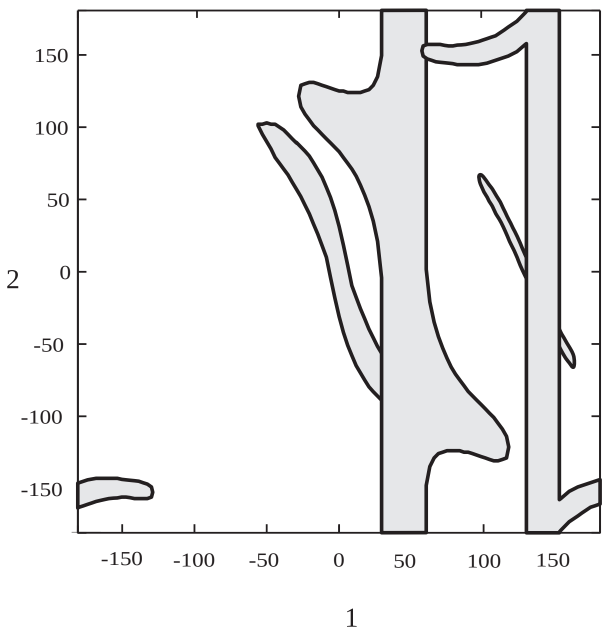
Planar polygon with rotation





Two link planar arm





Path planning

Visibility graph works well in two dimensions, but not three or more.

Voronoi graphs have been generalized to higher dimensions (see Choset).

Free space graphs.

Maze searching algorithms.

Random trees.

Best First Planner

Best First Planner

Used by Barraquand and Latombe to search high dimensional space.

Discretized search in Cspace with potential fields.

- Obstacles have high potential, goal is low potential.
- Potential is used as search heuristic, not as control.

Searches in Cspace, but doesn't construct Cspace obstacles.

Best First Planner

```
procedure BFP
    open ← {q_init}
    mark q_init visited
    while open ≠ {}
        q ← best( open )
        if q = q_goal return( success )
        for n ∈ neighbors( q )
            if n unvisited and not a collision
                insert( n, open )
                mark n visited
    return( failure )
```

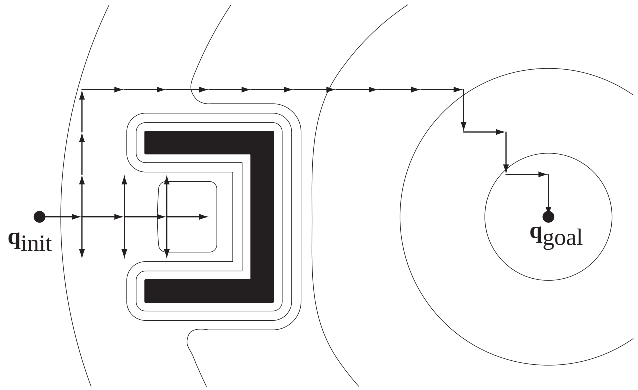
Example of Best First Planner

Searches from $\mathbf{q}_{\text{init}}$, taking best nodes according to potential field.

Discards collisions.

If lucky, goes straight to goal.

More generally, has to fill in local minima in potential field.



Nonholonomic path planning

Suppose we have a unicycle and we want to plan collision free paths.

What's wrong with Best First Planner?

```
procedure BFP
    open ← {q_init}
    mark q_init visited
    while open ≠ {}
        q ← best( open )
        if q = q_goal return( success )
        for n ∈ neighbors( q )
            if n unvisited and not a collision
                insert( n, open )
                mark n visited
    return( failure )
```

Approach

Use actions that satisfy velocity constraints.

Search sequences of actions.

Prune collisions.

Prune sequences that go where we’ve been before.

NonHolo Planner

Define a grid in configuration space.

Let δt be small time increment.

Let `grid( q )` returns the grid node closest q.

Let `actions` be a finite set of actions.

Let `int(` q`, a,` δt `)` simulate the system from q using action a for time δt.

NonHolo Planner

```
procedure NHP
    open ← {q_init}
    mark grid( q_init ) visited
    while open ≠ {}
        q ← best( open )
        if q ≈ q_goal return( success )
        for a ∈ actions
            n ← int( q, a, δt )
            if n not a collision and grid( n ) not vi
                insert( n, open )
                mark grid( n ) visited
    return( failure )
```

Example: NHP for unicycle

Let the grid be 10cm by 10cm by 5 degrees.

Let δt be 0.1 seconds.

Let `actions` be `fwd`, `rev`, `ccw`, `cw`. Speeds are 2m / sec and 90 degrees / sec.

Let `best` return node with shortest path so far, with big penalty for switching actions.

Draw here!