3. Approximate path planning

csci3250: Computational Geometry
Laura Toma
Bowdoin College

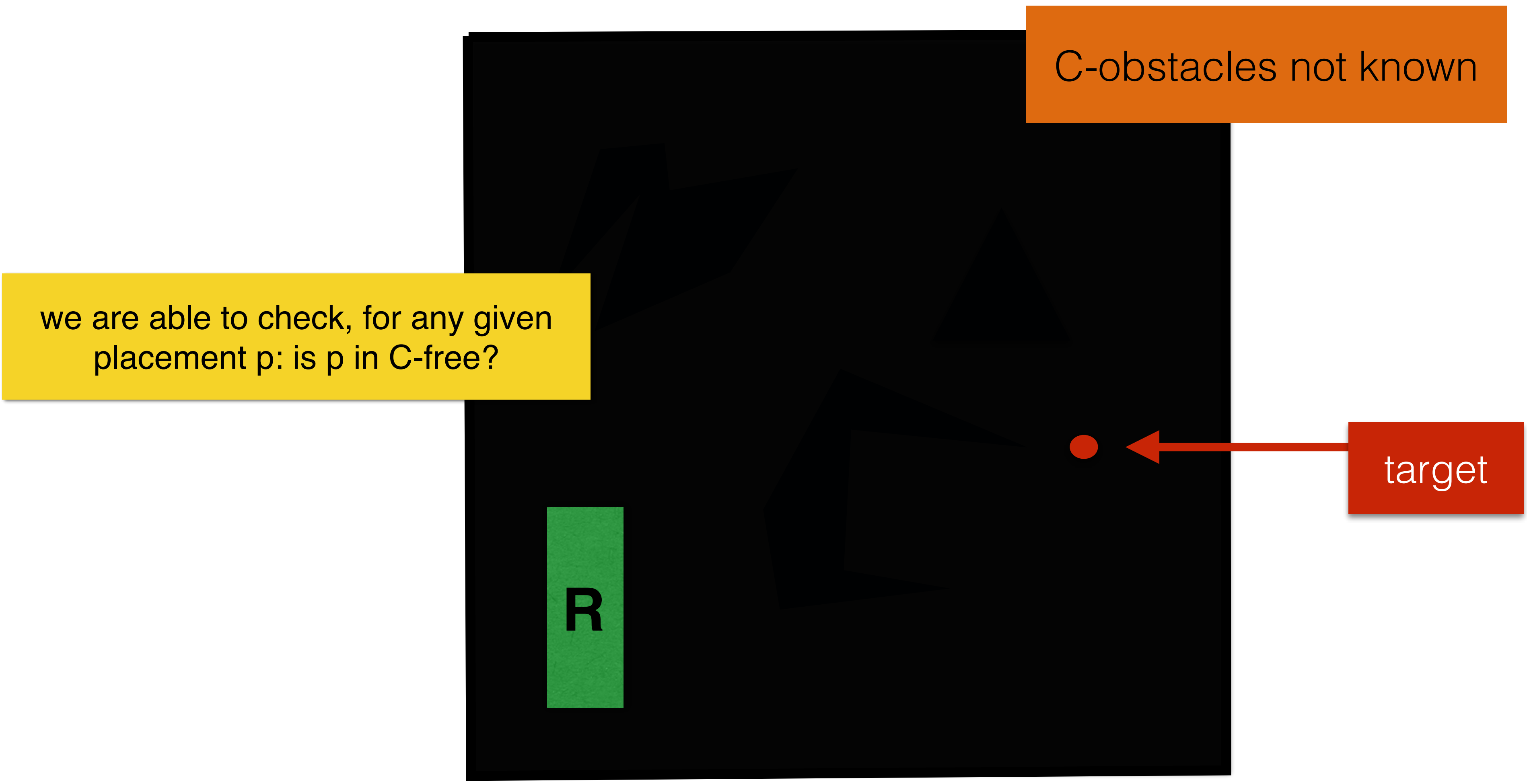
Path Planning Approaches

- Combinatorial / geometric planers
    - **Exact: Compute C-free geometrically**
    - Comments
        - This gives complete planners
        - Works beautifully in 2D and for some simple cases in 3D
        - Worst-case bound for combinatorial complexity of C-objects in 3D is high
        - A complete planner in 3D runs in $O(2^{n^{\#dof}})$
        - Impractical for high #dof
- Approximate planners
    - Approximate C-free

Approximate path planning

Knowing C-obstacles is like having a map: You know the roads and you can make a plan on how to get to the goal.

Without knowing the C-obstacles you are in the dark.

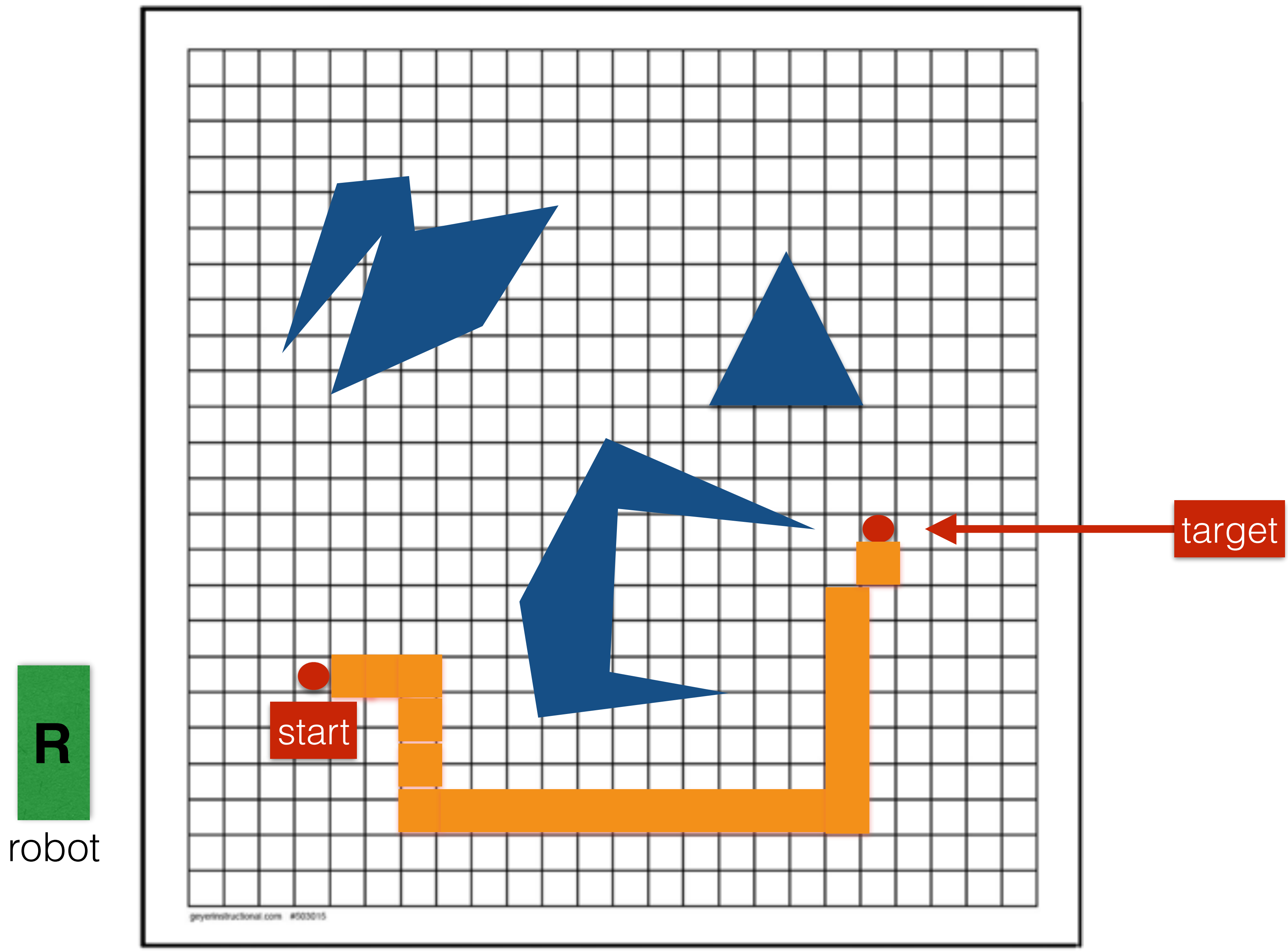


Approximate path planning

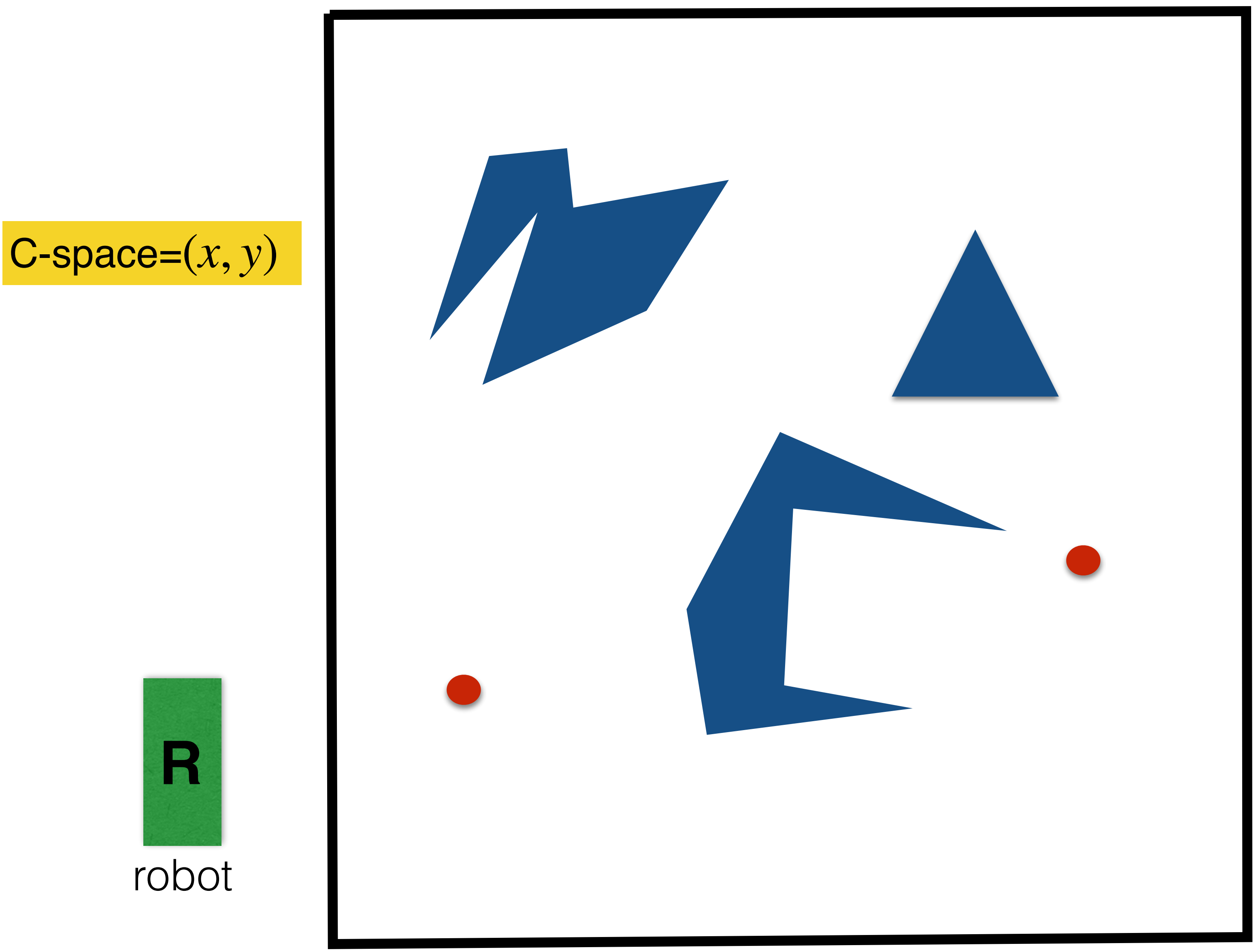
- Idea: Approximate C-free
- Approaches
    - Space partitioning/grid-based planners with A*
        - and variants (weighted A*, D*, ARA*,…)
    - Sampling-based
        - Rapidly-Exploring Random Tree (RRT)
        - Probabilistic RoadMap (PRM)
    - Potential field planners
    - Hybrid methods combing ideas from all of the above

Grid-based planners with A*

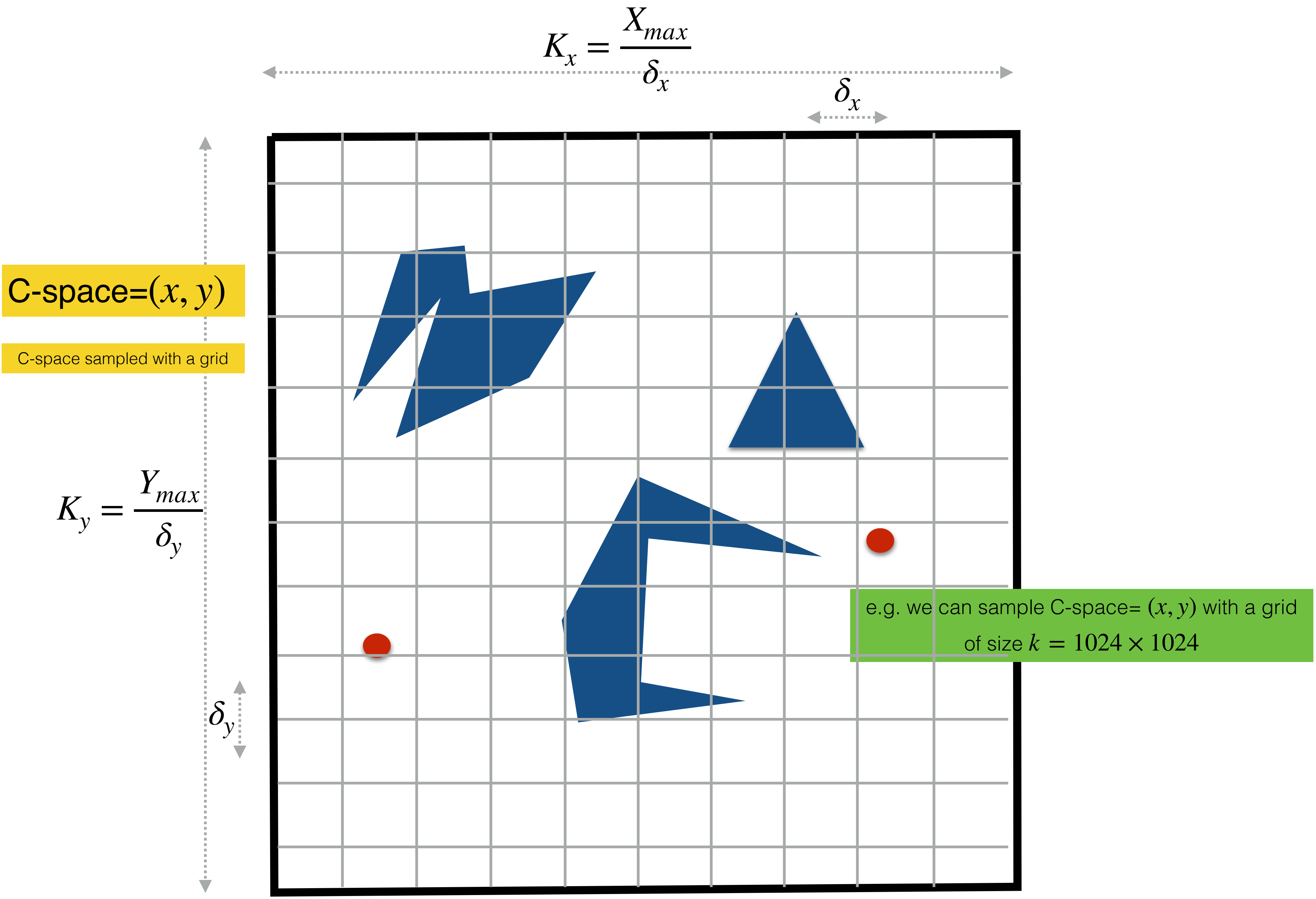
Grid based planners "pixelize" the space



Let’s say we have a robot moving in 2d without rotation and we want to implement a grid-based planner.



We decide what resolution we want on each axis, and from here we get the size of the grid in that dimension.



Grid-based planners with A*

- Sample C-space with a uniform grid/lattice
  - This “pixelizes" the C-space (pixels/voxels)
- Search for a path from start to end through “free” space
  - Dijkstra/A* and variants
  - Graph is implicit, given by lattice topology: move +/-1 in each direction, possibly diagonals

Dijkstra’s algorithm

- It’s basically a **best**-first search
- Initialize: $dist[v] = \infty, dist[s] = 0$,
- Repeat: select the **best** vertex (closest to start), and relax its edges

- Data structures
  - PQ of $(u, dist[u])$
  - priority(v): $dist[v]$

- Keeps track of :
  - $dist[v]$ = cost of getting from $start$ to v
  - $done[u]$: true if u has been explored
  - $pred[v]$: predecessor of v on the (optimal) path from $start$ to v

Dijkstra(vertex s)

- initialize
  - for all v: dist[v] = ∞, done[v]=false, pred[v]=null
  - dist[s] = 0, PQ.insert(<s, dist[s]>) ← insert the start
- while PQ not empty
  - $(u, dist[u])$ = PQ.deleteMin()
  - mark u as done ← //claim: dist[u] is the shortest path from s to u
  - for each edge (u,v), if v not done:
    - alt = dist[u] + edge(u,v)
    - if alt < dist[v]
      - dist[v] = alt, PQ.decreaseKey(v, dist[v])

↑ requires a structure that can search, or a PQueue with additional book-keeping

On a grid-graph

Dijkstra(vertex s)

- initialize
  - for all v: dist[v] = ∞, done[v]=false, pred[v]=null
  - dist[s] = 0 , PQ.insert(<s, dist[s]>)
- while PQ not empty
  - (u, dist[u]) = PQ.deleteMin()
  - if u is done, continue
  - mark u as done
  - for each neighbor v of u if v not done and isFree(v):
    - alt = dist[u] + edge(u,v)
    - if alt < dist[v]
      - dist[v] = alt, PQ.insert(v, dist[v])

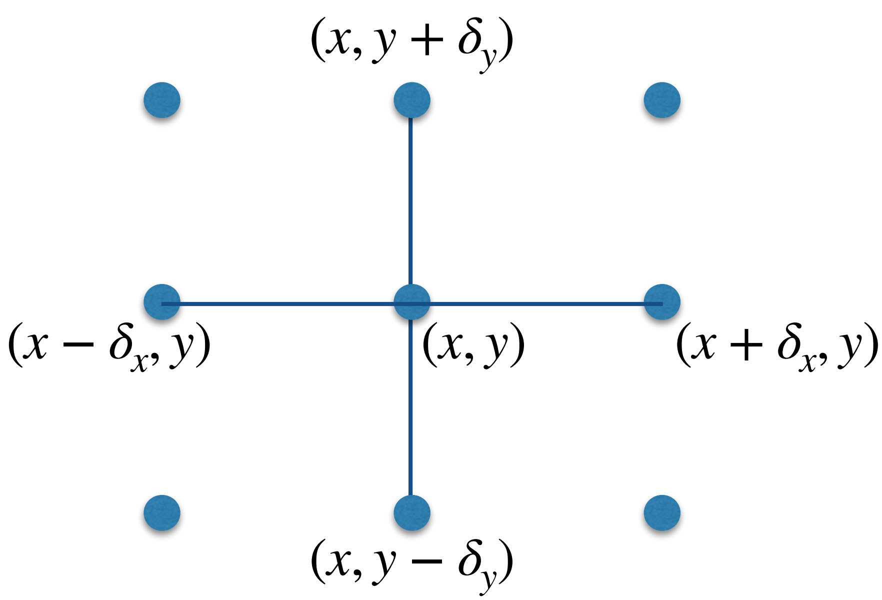
u is a placement, and also a pixel on the grid

isFree(v): is v in C-free

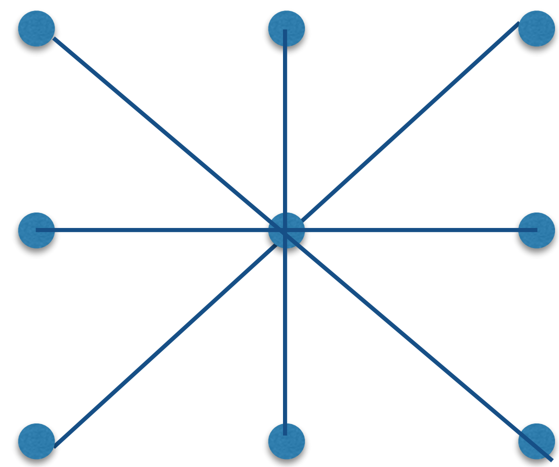
Grid graphs in 2d

u = (x, y) is a placement, and also a pixel on the grid

Neighbors of $u = (x, y)$



4-connected



8-connected

Grid graphs in 3d

neighbors of $v = (x, y, \theta)$

6-connected: $(x + \delta_x, y, \theta)$ $(x - \delta_x, y, \theta)$

$(x, y + \delta_y, \theta)$ $(x, y - \delta_y, \theta)$

$(x, y, \theta + \delta_\theta)$ $(x, y, \theta - \delta_\theta)$

or, connected diagonally as well

Motion planners assume the existence of a collision-detection routine that can check whether a given configuration, or path segment, is in free space.

Would my robot, if placed at this point p, intersect any obstacle?

```
//return true if placement p is in C-free
//put differently, return true if placing the robot R at p does not intersect any obstacle
bool isFree(placement p, the robot, the obstacles)
```

- translate and rotate the robot to p
- check whether any edge of the robot intersects any of the obstacles

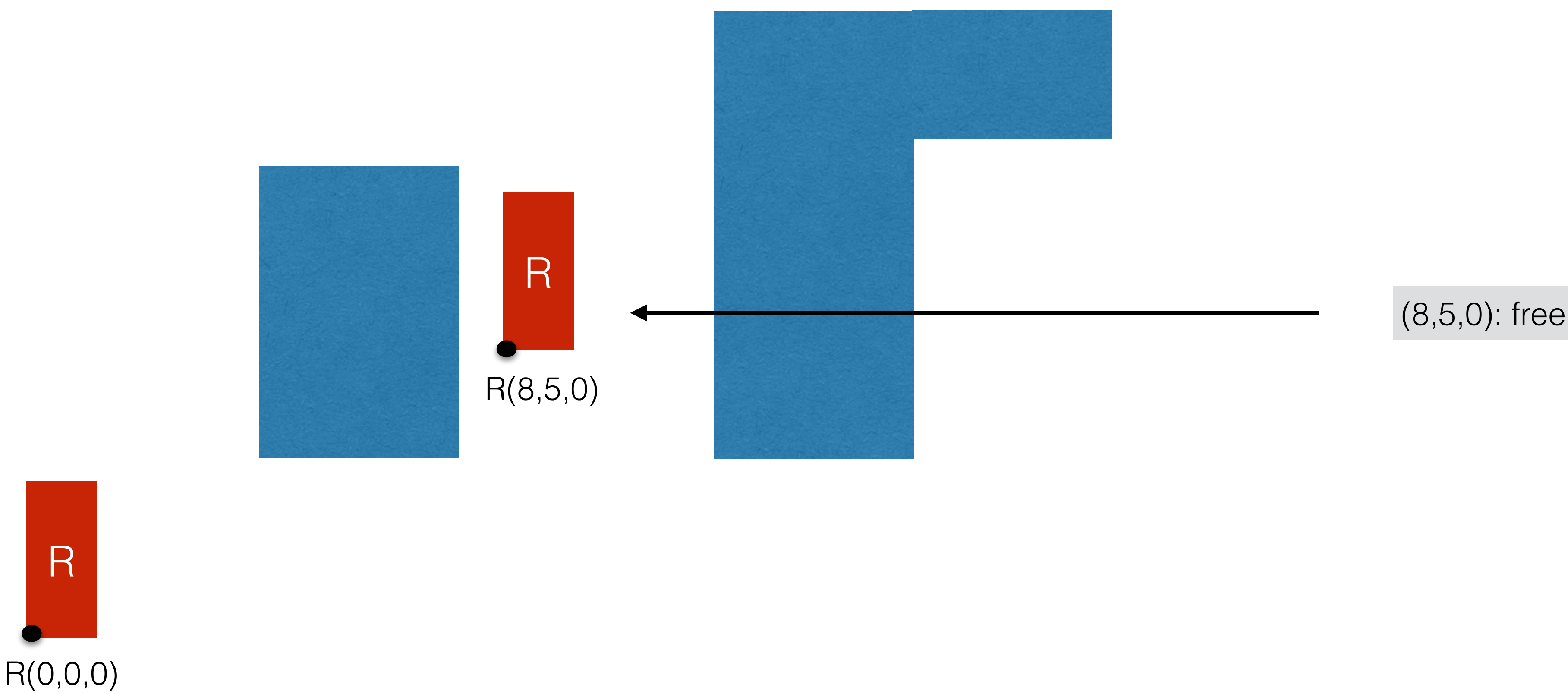
p is a point in C-space: (x, y) or (x, y, θ) or $(x, y, z, \theta_x, \theta_y, \theta_z)$ or …

EXAMPLE

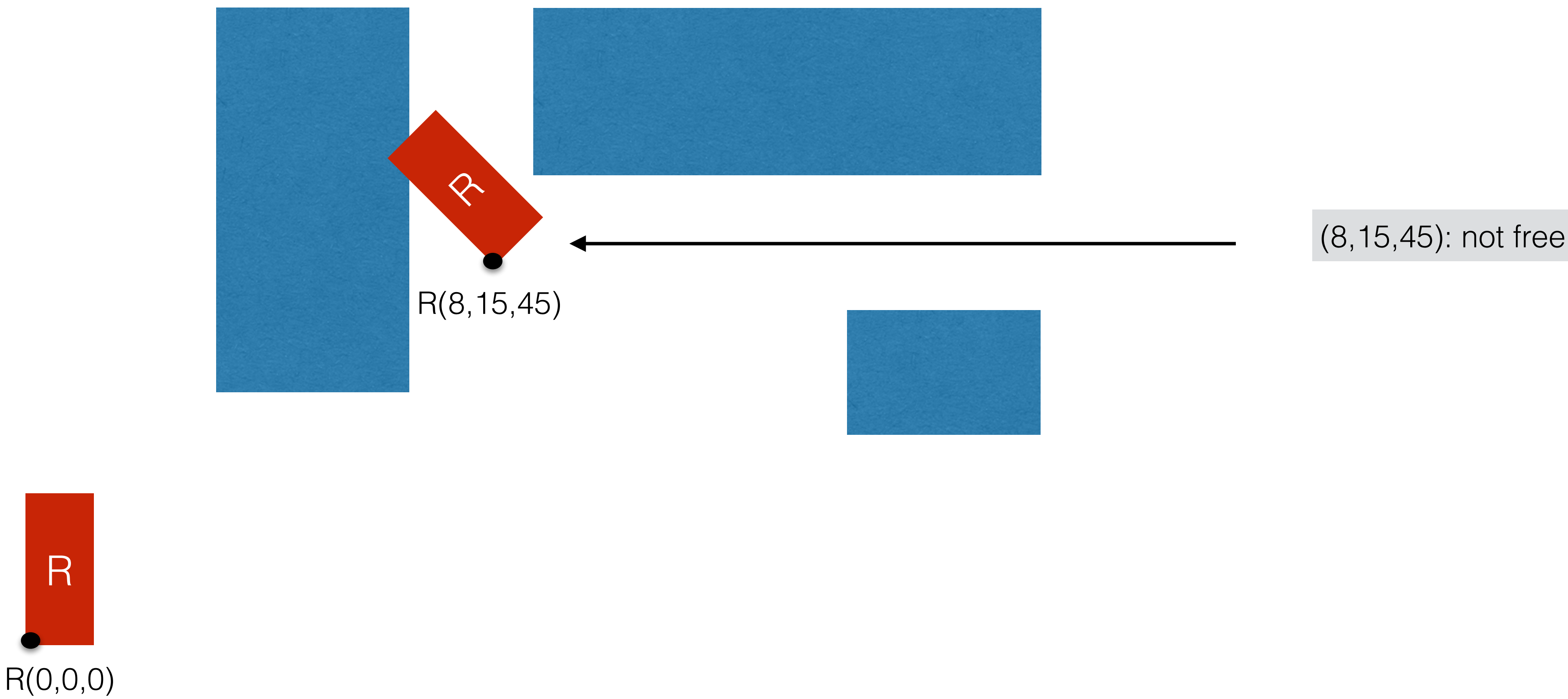
We need to write: `isFree` $(p = (x, y, \theta))$, `Robot R, Obstacles S)`

2D: robot can translate and rotate

C-space: 3D configuration p: (x, y, θ)



EXAMPLE



Dijkstra => A*

https://qiao.github.io/PathFinding.js/visual/

Dijkstra

- Evaluates vertices based on their distance to the start
  - priority(v) = $dist(v)$
- Dijkstra will explore a large portion of the graph before reaching the target. Would be nice if we could cut down the number of nodes traversed before reaching the goal

A*

- Idea: Steer the search towards the goal (while keeping solution optimal)
- priority(v) : $f(v) = dist(v) + h(v)$
  - dist(v): cost of getting from $start$ to v
  - h(v): estimate of the cost from v to $goal$
- Dijkstra is A* with $h(v) = 0$

Dijkstra => A*

Animations

- https://qiao.github.io/PathFinding.js/visual/
- https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstras_progress_animation.gif
- https://www.youtube.com/watch?v=DINCL5cd_w0
- https://www.google.com/search?client=firefox-b-1-d&q=dijkstra+ +vs+A*+algorithm&tbm=vid&sa=X&ved=2ahUKEwic2fXNvMr-AhV1FFkFHcSxB_kQ0pQJegQICxAB&biw=1313&bih=731&dpr=2#fpstate=ive&vld=cid:02bef27f,vid:g024lzsknDo
-

A*

- The heuristic $h(v)$ is called ***admissible*** if $h(v)$ is smaller than the true cost of getting from v to the target.
- Theorem: If $h(v)$ is admissible then A* will return an optimal solution.
- Put differently if $h(v)$ is too high, the algorithm loses optimality
    - h(v) = 0 will always work
    - higher $h(v)$ will steer the search more => more efficient
    - The closer h(v) is to the true cost of getting from v to goal, the more efficient
    - if $h(v)$ is too high => A* not optimal
- In many situations a safe admissible heuristic is

$$h(v) = \text{EuclidianDistance}(v, goal$$

admissible because the cost of getting from a to b is ≥ the Euclidian distance from a to b

(x, y) or (x, y, θ) or $(x, y, z, \theta_x, \theta_y, \theta_z)$ or …

generic A*(placement start, goal)

- initialize
  - for all v: dist[v] = ∞, f[v] = ∞, pred[v] = null, done[v]=false
  - dist[s] = 0, f[s] = h(s, target), PQ.insert(<s, f[s]>)
- while PQ not empty
  - <u, f[u]> = PQ.deleteMin()
  - for each neighbor v, if not done[v] and isFree(v)
    - alt = dist[u] + edge(u,v)
    - if alt < dist[v]
      - dist[v] = alt; f[v] = dist[v] + h(v, target);
      - pred[v] = u; PQ.decreaseKey(<v, f[v]>)

isFree(v): is v in C-free

Grid-based planners with A*

- The paths may be longer than true shortest path in C-space
- Not complete, but **resolution complete**
    - probability of finding a solution, if one exists —> 1 as the resolution of the grid increases
- While searching, it finds what points are in C-free, so it constructs C-free. Can interleave the construction with the search (ie construct only what is necessary). Or can construct it all at once (occupancy grid).

\+

- simple to understand/implement
- work in any dimension

−

- size and quality of path depends on the discretization of the problem
- not practical in high-d spaces
    - e.g. 6 dof: 1000 x 1000 x 1000 x 360 x 360 x 360

A* variants

- weighted A*
    - $c \cdot h()$ ==> solution is no worse than $(1+c)\times$ optimal
- anytime A*
    - use weighted A* to find a first solution ; then use A* with first solution as upper bound to prune the search
- real-time replanning
    - if the underlying graph changes, it usually affects a small part of the graph ==> don't run search from scratch
    - D*: efficiently recompute SP every time the underlying graph changes

- Finer resolution => better paths but slower
- C-free can be pre-computed (occupancy grid) or computed incrementally
- One-time path planning vs many times; static vs dynamic environment
- fixed resolution vs. multi-resolution techniques

Sampling-based planning

- Geometric planners:
  - ⊖ hard to construct C-obstacles except for simple cases (2d, no rotation)
- Grid-based planners:
  - ⊖ grid has uniform resolution and uses too much large for high #dof

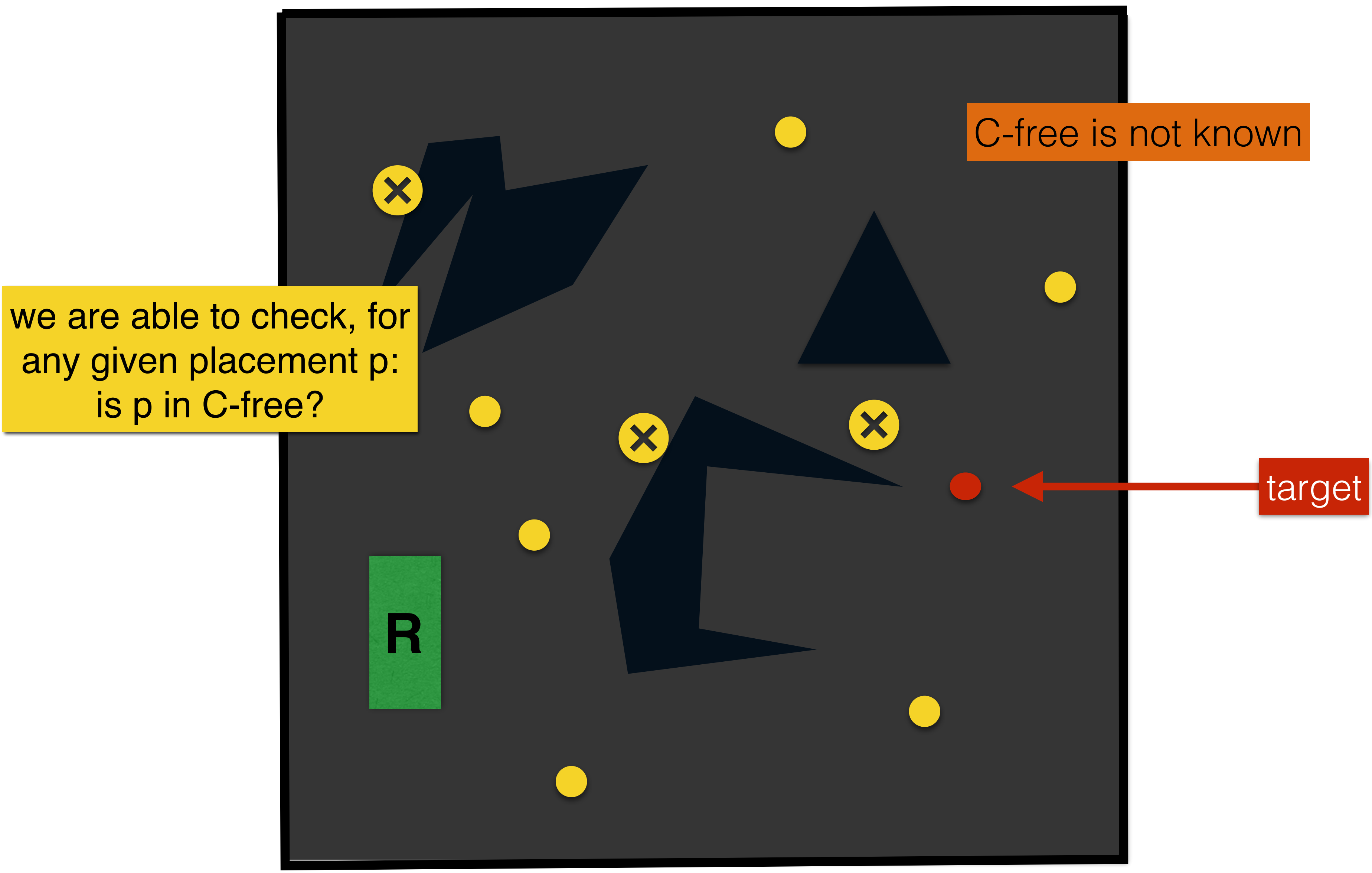
    e.g. DOF= 6: 1000 x 1000 x 1000 x 360 x 360 x 360

➡ Sampling-based planners
  - Sample and generate a sparse representation of C-free

- Potential field planners

Sampling-based planners

We don't know the C-obstacles, but we'll assume that we have a function that can check whether a given configuration is free.



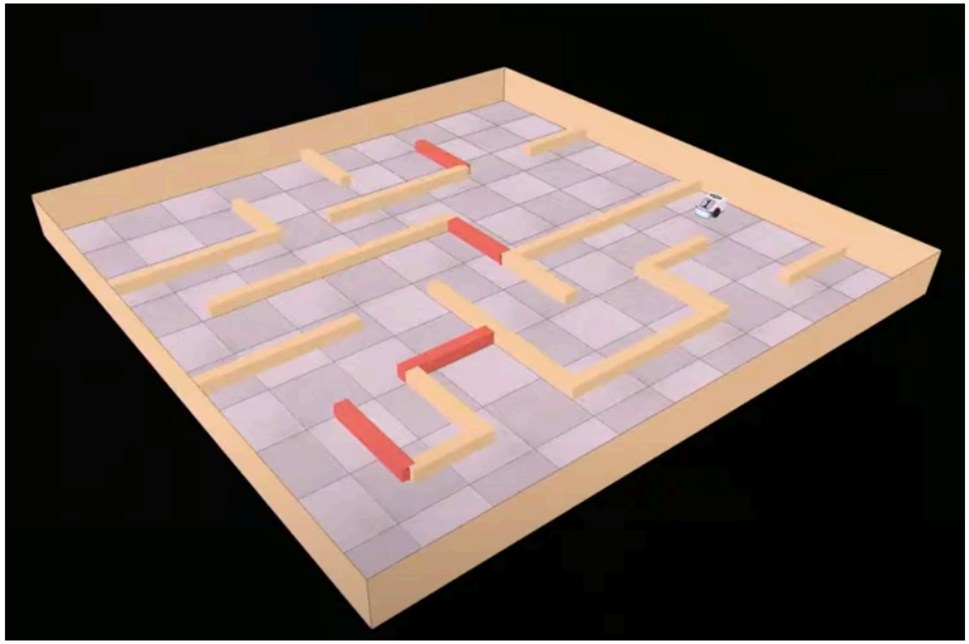
All planners need a collision detection function

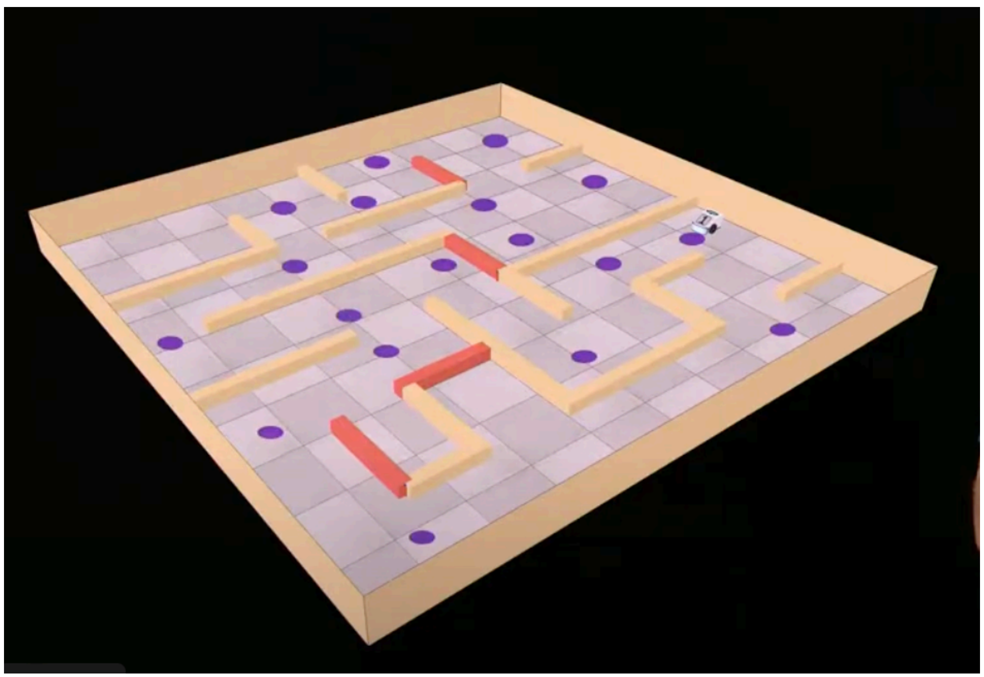
Would my robot, if placed at this point p, intersect any obstacle?

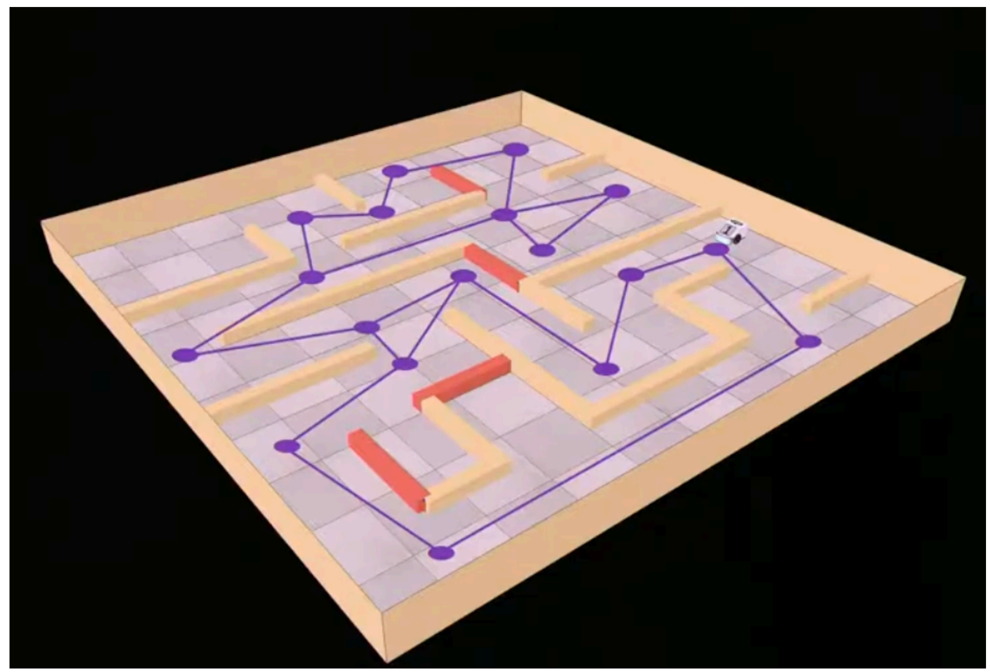
```
//return true if placement p is in C-free
//put differently, return true if placing the robot R at p does not intersect any obstacle
bool isFree(placement p,  the robot,   the obstacles)
```

- translate and rotate the robot to p
- check whether any edge of the robot intersects any of the obstacles

p is a point in C-space: (x, y) or (x, y, θ) or $(x, y, z, \theta_x, \theta_y, \theta_z)$ or …







Sampling-based planning

- Idea: Sample C-free and compute a roadmap that captures its connectivity
- Single-query, incremental search planners
  - Construct a graph/roadmap to connect $start$ and end
  - Reconstruct for different $(start, end)$ pairs
  - E.g. RRT (rapidly-exploring random tree) and variants
- Multiple-query planners
  - Construct a graph/roadmap and use it for **any** $(start, end)$ pairs
  - E.g. PRM (probabilistic roadmap) and variants

History

- Dijkstra 1950s
- A* 1960s
- PRM 1996
- RRT 1998
- RRT* 2010

Probabilistic Roadmaps and RRTs

- Efficient, easy to implement, applicable to many types of scenes
- Well-suited for high #dof
- Shown to be **probabilistically complete**
  - Finds a solution, if one exists, with probability —> 1 as the nb. of samples increases
- Leading motion planning technique, embraced by many groups, many variants, used in many type of scenes/applications.
    - PRM*, FMT* (fast marching tree), …
- No discretization (sample from a continuous space)
- But: Path not optimal, time may be unbounded

The RRT

(LaValle, 1998)

- Incrementally build a tree rooted at "start" outwards, while trying to determine if a path exists at each step
- Original paper:
  - https://www.cs.cmu.edu/~motionplanning/papers/sbp_papers/PRM/randtrees_02.pdf

NEW_CONFIG((q, q_{near}, q_{new})

- if q is not free, return false
- if segment $q_{near}q_{new}$ is not in C-free, return false

```
BUILD_RRT(q_init)
    T.init(q_init);
    for k = 1 to K do
        q_rand ← RANDOM_CONFIG();
        EXTEND(T, q_rand);
    Return T
```

```
EXTEND(T, q)
    q_near ← NEAREST_NEIGHBOR(q, T);
    if NEW_CONFIG(q, q_near, q_new) then
        T.add_vertex(q_new);
        T.add_edge(q_near, q_new);
        if q_new = q then
            Return Reached;
        else
            Return Advanced;
    Return Trapped;
```

Figure 2: The basic RRT construction algorithm.

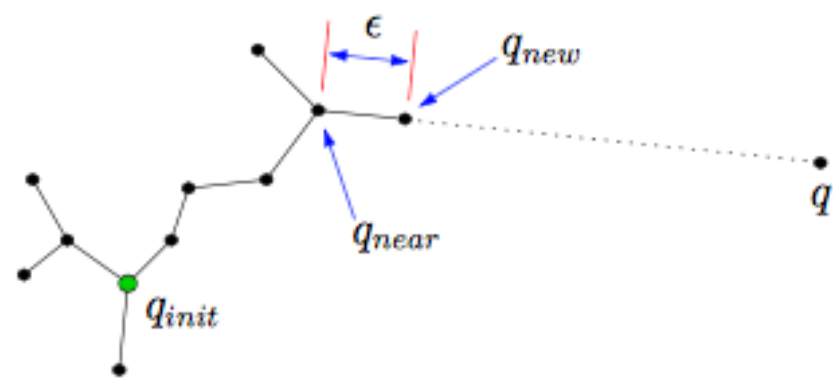
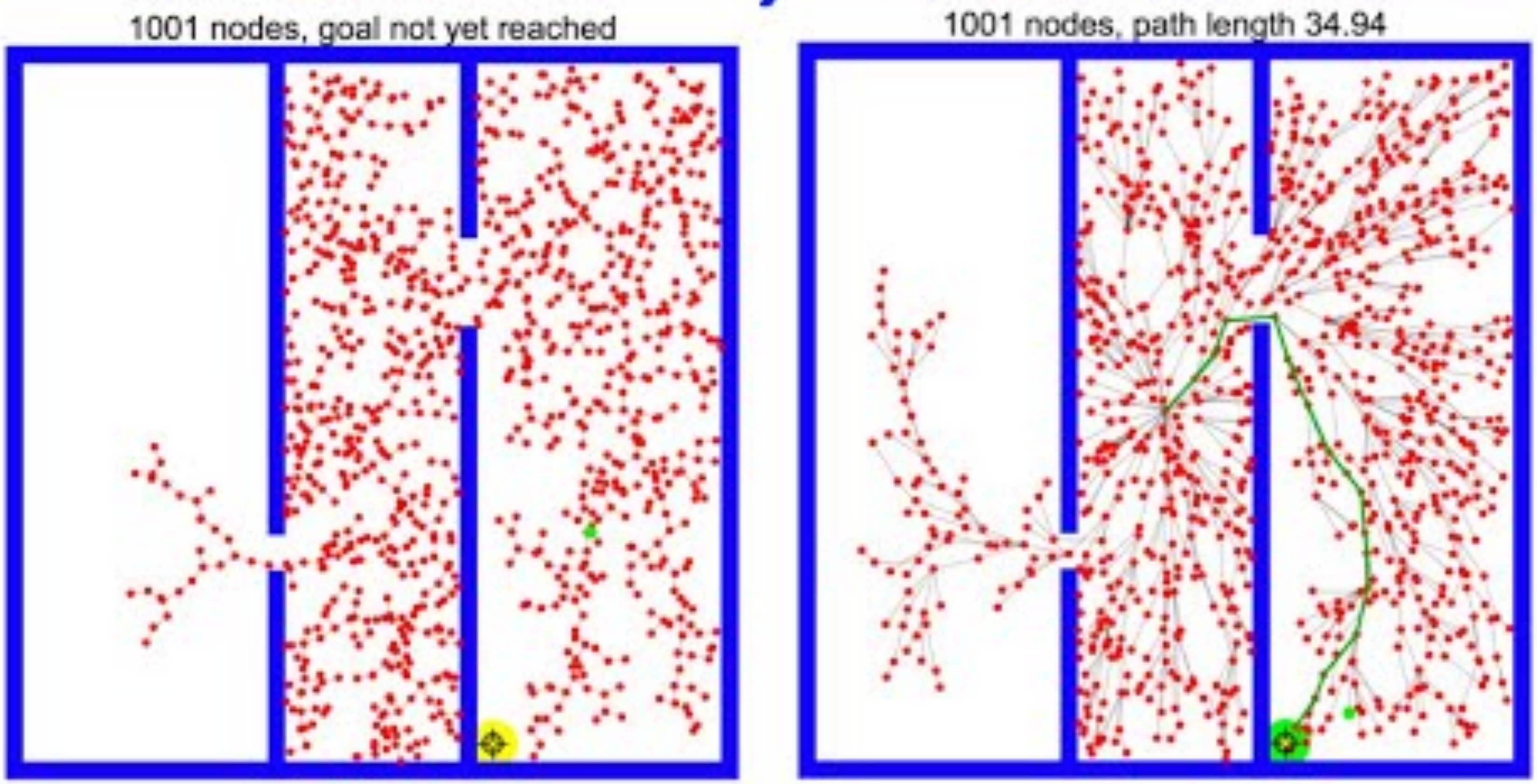


Figure 3: The EXTEND operation.

Random Trees, RRTs & RRT*



Probabilistic Roadmaps and RRTs

PRM

- Roadmap adjusts to the density of free space and is more connected around the obstacles
- Size of roadmap can be adjusted as needed
- More time spent in the "learning" phase ==> better roadmap
- Built once, re-used many times, used in static environments

RRT

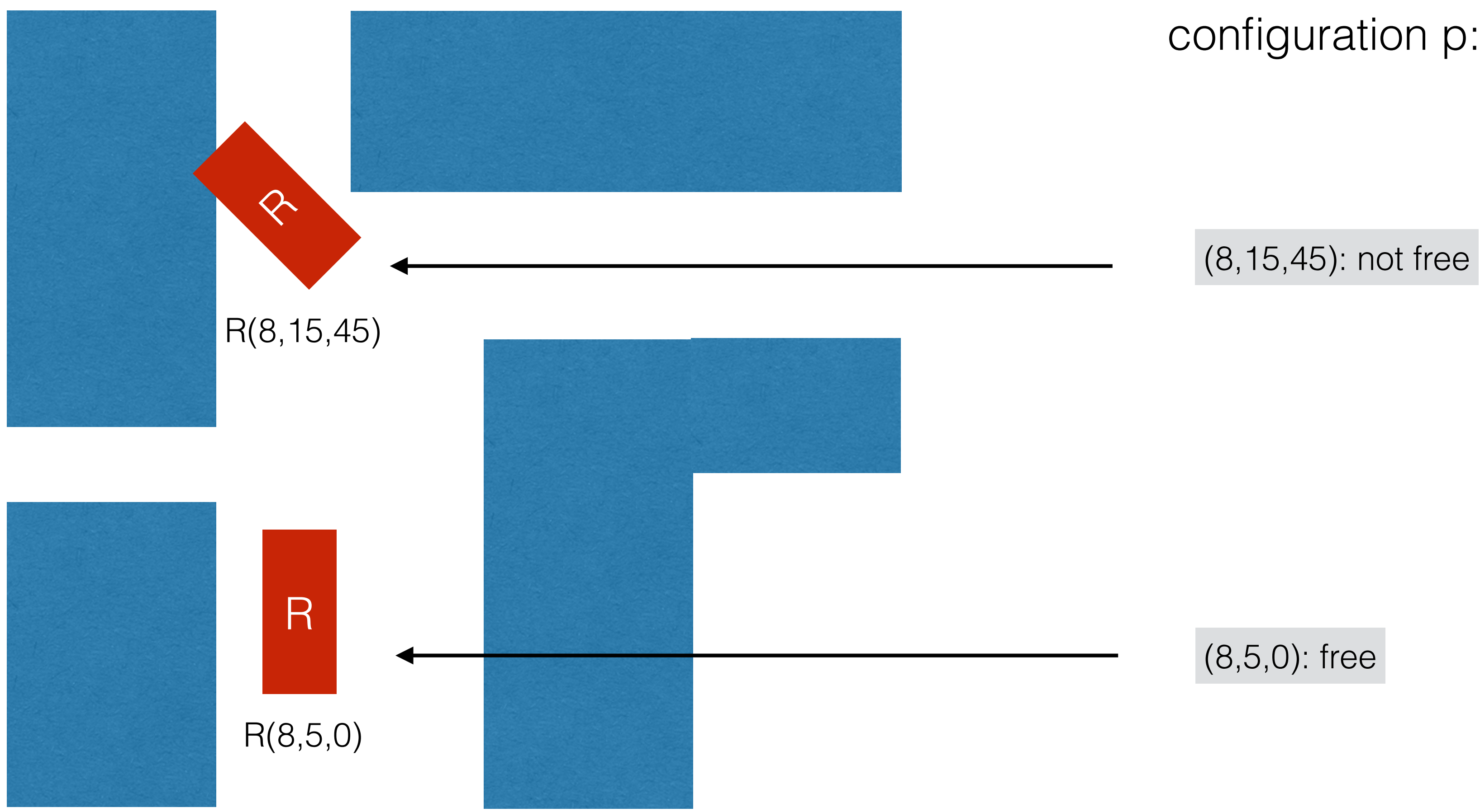
- Used in changing environments
- Faster to build than PRM

We need to write: `isFree` $(p = (x, y, \theta))$, `Robot R, Obstacles S)`

2D: robot can translate and rotate

C-space: 3D

configuration p: (x, y, θ)



Also need a segment collision detection function

Is segment pq in C-free?

```
//p, q are points in C-space: (x, y) or (x, y, θ) or (x, y, z, θx, θy, θz) or ...
bool localPlanner(placement p, placement q, the robot, the obstacles)
```

Probabilistic roadmaps

Probabilistic Roadmaps (Kavraki, Svetska, Latombe, Overmars et al , 1996)

- Roadmap construction phase
  - Start with a sampling of points in C-free and try to connect them
  - Two points are connected by an edge if a simple quick planner can find a path between them
  - This will create a set of connected components
- Roadmap query phase
  - Use roadmap to find path between any two points

Probabilistic Roadmaps

- Generic-Sampling-Based-Roadmap:
  - for i = 1 to N:
    - generate a random point p_i in C
    - if isFree(p), add p to R
  - add p_{start} to R
  - for each point p_i in R:
    - $N(p_i)$ = { closest neighbors of p_i in R}
    - for each neighbor q in $N(p_i)$:
      - if there is a collision-free local path from p_i to q and there is not already an edge from p_i to q then add an edge from p_i to q in the roadmap R

- Variants
  - how they select the initial n samples from C
    - e.g. return a set of n points arranged on a regular grid in C, random points, etc
  - how they select the neighbors
    - return the k nearest neighbors of p in V
    - return the set of points lying in a ball centered at p of radius r
- Often used: samples arranged in a 2-dimensional grid, with nearest 4 neighbors (2^d)

Probabilistic Roadmaps (Kavraki, Svetska, Latombe, Overmars et al , 1996)

(1) $N \leftarrow \emptyset$
(2) $E \leftarrow \emptyset$
(3) **loop**
(4) $c \leftarrow$ a randomly chosen free configuration
(5) $N_c \leftarrow$ a set of candidate neighbors of c chosen from N
(6) $N \leftarrow N \cup \{c\}$
(7) **for all** $n \in N_c$, in order of increasing $D(c,n)$ **do**
(8) **if** $\neg$*same_connected_component*$(c,n) \wedge \Delta(c,n)$ **then**
(9) $E \leftarrow E \cup \{(c,n)\}$
(10) update R's connected components

- Start with a *random* sampling of points in C-free
- Roadmap is stored as set of *trees* for space efficiency
    - trees encode connectivity, cycles don't change it. Additional edges are useful for shorter paths, but not for completeness
- Augment roadmap by selecting additional sample points in areas that are estimated to be "difficult"

the local planner delta(c,n): is segment cn in C-free?

Probabilistic Roadmaps

- Roadmap adjusts to the density of free space and is more connected around the obstacles
- Size of roadmap can be adjusted as needed
- More time spent in the “learning” phase ==> better roadmap

- Efficient, easy to implement, applicable to many types of scenes
- Well-suited for high #dof
- No discretization (sample from a continuous space)
- Shown to be probabilistically complete
  - finds a solution, if one exists, with probability —> 1 as the nb of samples increases
- Leading motion planning technique, Embraced by many groups, many variants of PRM’s, used in many type of scenes/applications (PRM*, FMT* (fast marching tree), …)

Sampling-based planning

- Geometric planners:
  - hard to construct C-obstacles except for simple cases (2d, no rotation)
- Grid-based planners:
  - grid has uniform resolution and uses too much large for high #dof
    e.g. DOF= 6: 1000 x 1000 x 1000 x 360 x 360 x 360
- Sampling-based planners
  - Sample and generate a sparse representation of C-free
- Potential field planners

Potential field methods   [Latombe et al, 1992]

- Define a potential field
  - Robot moves in the direction of steepest descent on potential function
- Ideally potential function has global minimum at the goal, has no local minima, and is very large around obstacles
- Algorithm outline:
  - place a regular grid over C-space
  - search over the grid with potential function as heuristic

https://www.youtube.com/watch?v=r9FD7P76zJs

Potential field methods

- Pro:
  - Framework can be adapted to any specific scene
- Con:
  - can get stuck in local minima
  - Potential functions that are minima-free are known, but expensive to compute

- Example: RPP (Randomized path planner) is based on potential functions
  - Escapes local minima by executing random walks
  - Successfully used to
    - perform riveting ops on plane fuselages
    - plan disassembly operations for maintenance of aircraft engines

Demos

DARPA challenges

- Fostered the development of self-driving vehicles
- 2004: noone finished the course
- 2005:
  - 132 mi course, in the desert in Nevada
  - 5 vehicles finished the race, with Stanford "Stanley" in first place, the first autonomous vehicle to ever finish a race (Stanley now at the Smithsonian Air & Space museum)
- 2007
  - Required teams to build an autonomous vehicle capable of driving in traffic and performing complex maneveurs such as merging, passing and parking
  - 5 vehicles finished the race,   with CMU "Boss" in first place, and Stanford "Junior" in second.

DARPA challenges

- Planners: Both graph search and incremental tree-based
  - **CMU**: lattice graph in 4D (x,y, orientation, velocity), search with D*
  - **Stanford**: incremental sparse tree of possible maneuvers, hybrid A*
  - **Virginia Tech**: graph discretization of possible maneuvers, search with A*
  - **MIT**: variant of RRT with biased sampling

- talk by Sertac Karaman in Darpa 2007 MIT team:

- *A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles, by Brian Paden, Michal Cáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli* https://arxiv.org/pdf/1604.07446.pdf

DARPA 2007, Stanford team

- uses hybrid A*

- Stanford’s A*-based planner in action

https://www.youtube.com/watch?v=qXZt-B7iUyw

http://robots.stanford.edu/papers/junior08.pdf

Junior: The Stanford Entry in the Urban Challenge

Michael Montemerlo[1], Jan Becker[4], Suhrid Bhat[2], Hendrik Dahlkamp[1], Dmitri Dolgov[1], Scott Ettinger[3], Dirk Haehnel[1], Tim Hilden[2], Gabe Hoffmann[1], Burkhard Huhnke[2], Doug Johnston[1], Stefan Klumpp[2], Dirk Langer[2], Anthony Levandowski[1], Jesse Levinson[1], Julien Marcil[2], David Orenstein[1], Johannes Paefgen[1], Isaac Penny[1], Anna Petrovskaya[1], Mike Pflueger[2], Ganymed Stanek[2], David Stavens[1], Antone Vogt[1], and Sebastian Thrun[1]

[1]Stanford Artificial Intelligence Lab, Stanford University, Stanford CS 94305
[2]Electronics Research Lab, Volkswagen of America, 4009 Miranda Av., Palo Alto, CA 94304
[3]Intel Research, 2200 Mission College Blvd., Santa Clara, CA 95052
[4]Robert Bosch LLC, Research and Technology Center, 4009 Miranda Ave, Palo Alto, CA 94304

Abstract

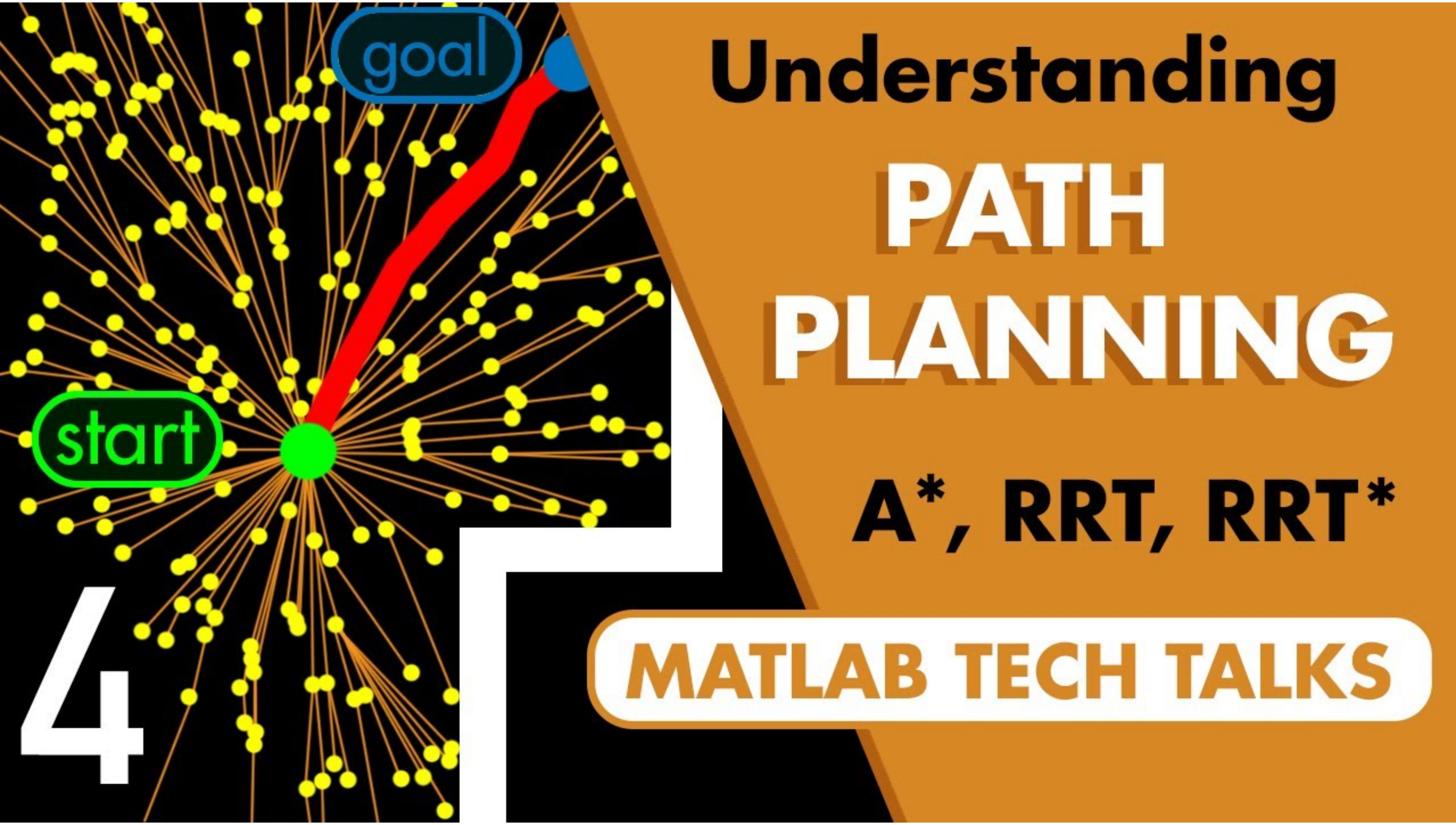
This article presents the architecture of Junior, a robotic vehicle capable of navigating urban environments autonomously. In doing so, the vehicle is able to select its own routes, perceive and interact with other traffic, and execute various urban driving skills including lane changes, U-turns, parking, and merging into moving traffic. The vehicle successfully finished and won second place in the DARPA Urban Challenge, a robot competition organized by the U.S. Government.

1 Introduction

DARPA

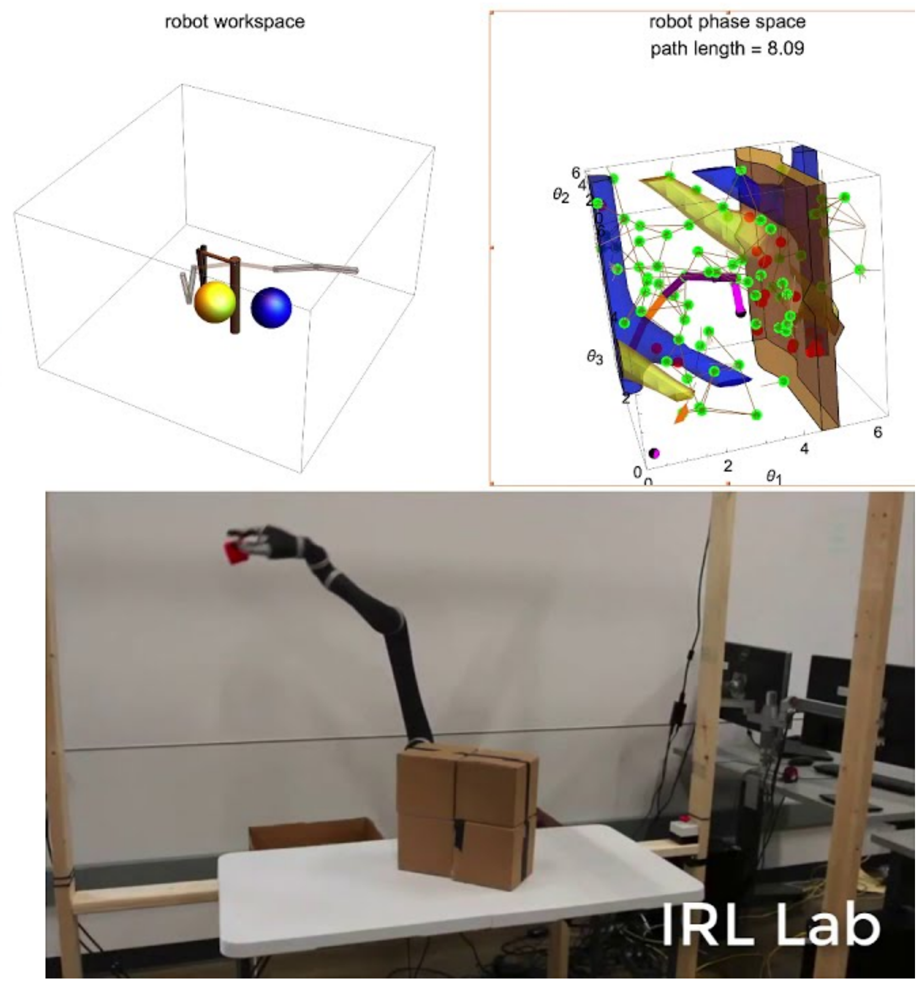
- Currently, RACER challenge for off-road autonomous vehicles

https://www.darpa.mil/news-events/2023-04-11



https://www.youtube.com/watch?v=QR3U1dgc5RE (start at min 10 for RRT)

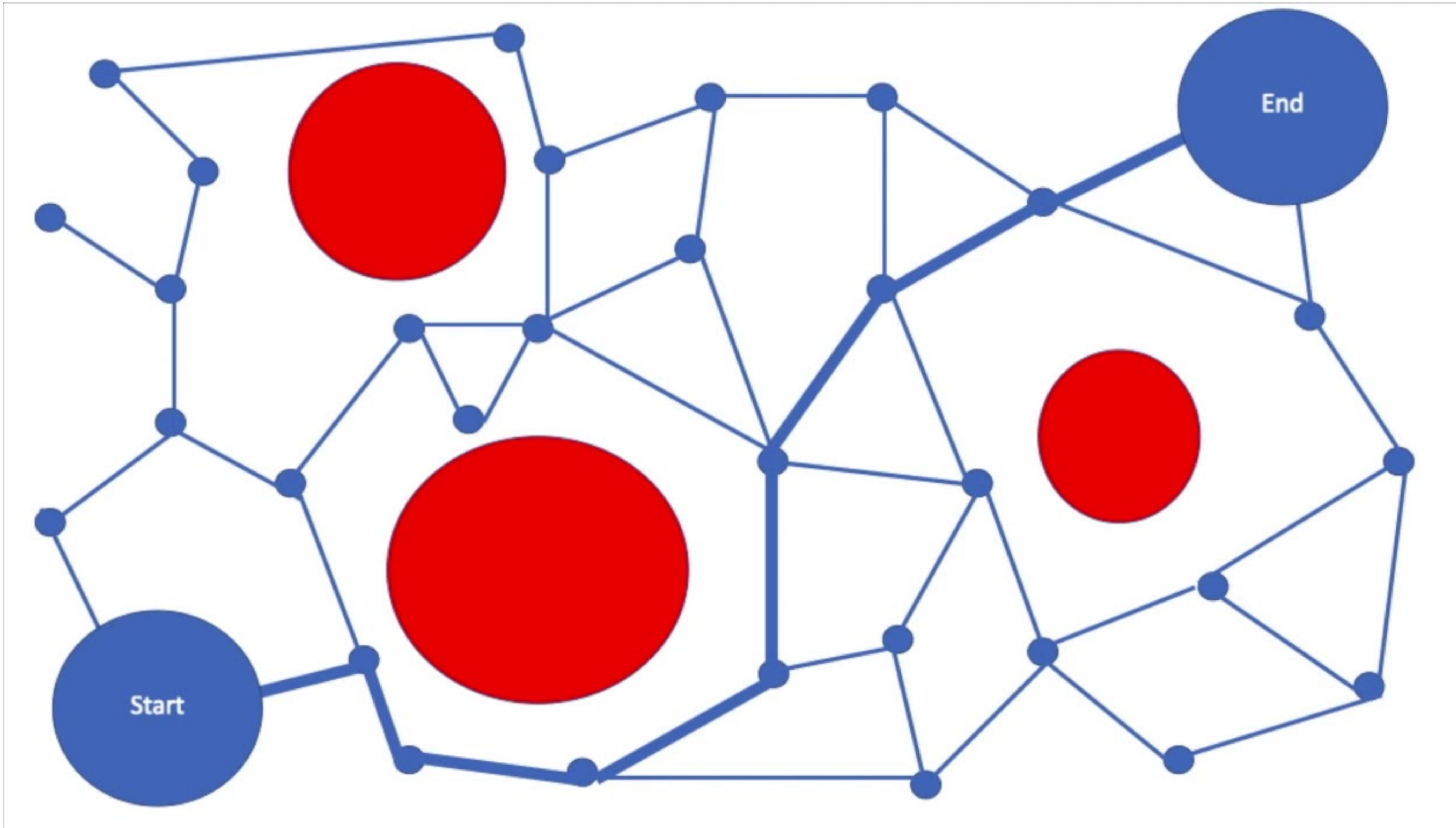
PRM: Probabilistic Roadmap Method for robotics



https://www.youtube.com/watch?v=tlFVbHENPCI

Comparison of RRT, PRM (MIT course project)

https://www.youtube.com/watch?v=gP6MRe_IHFo

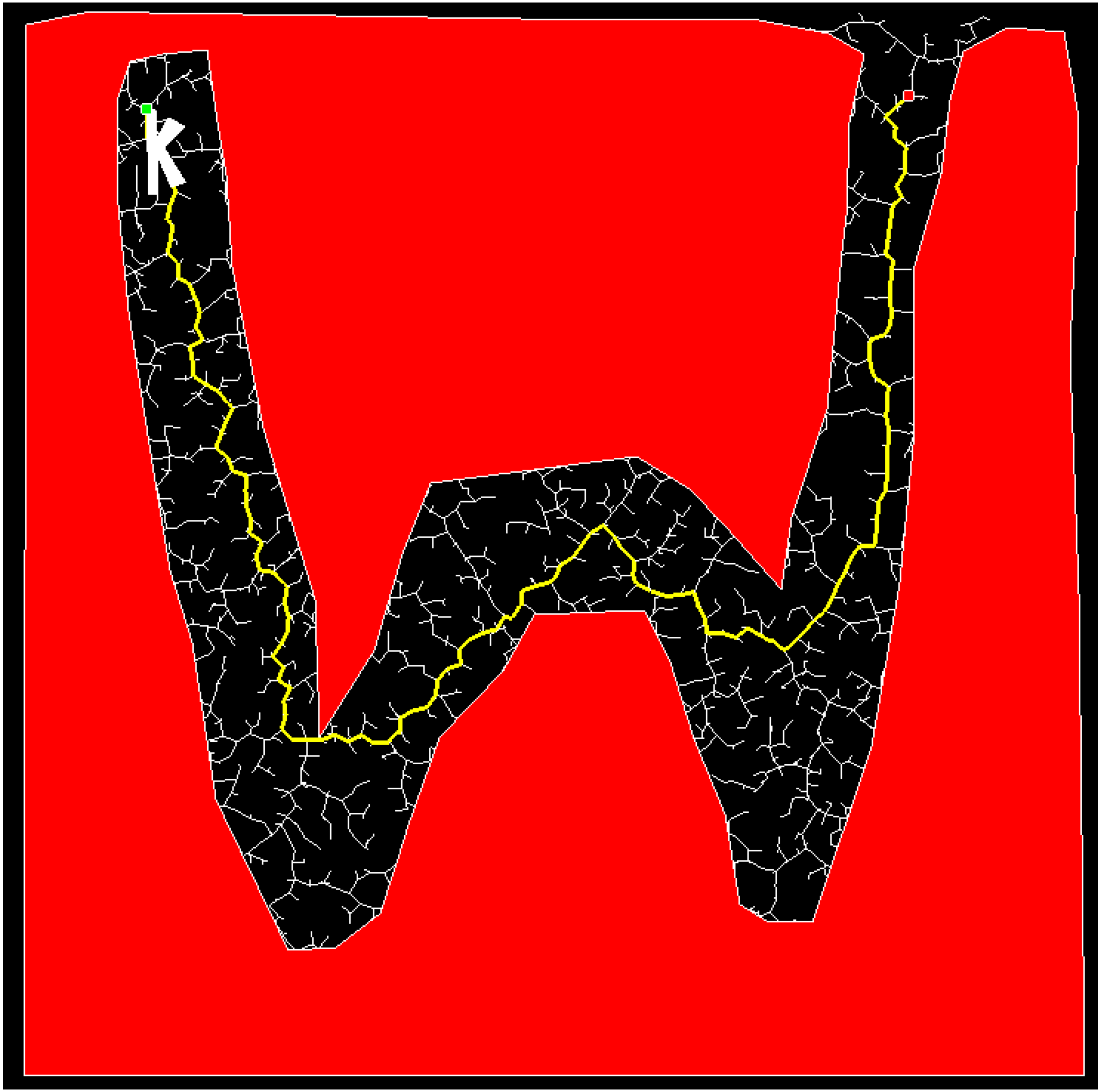


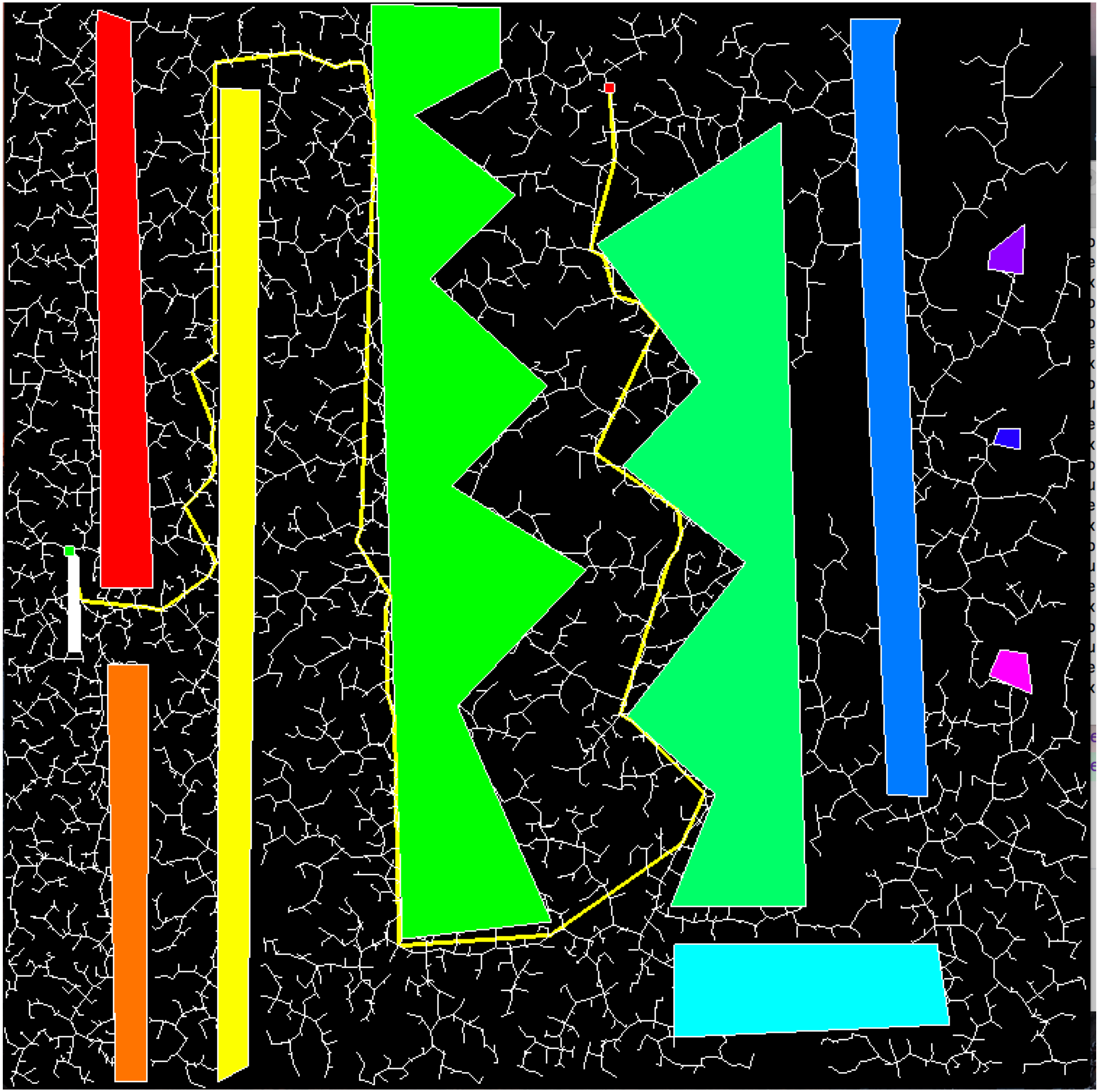
Project 7

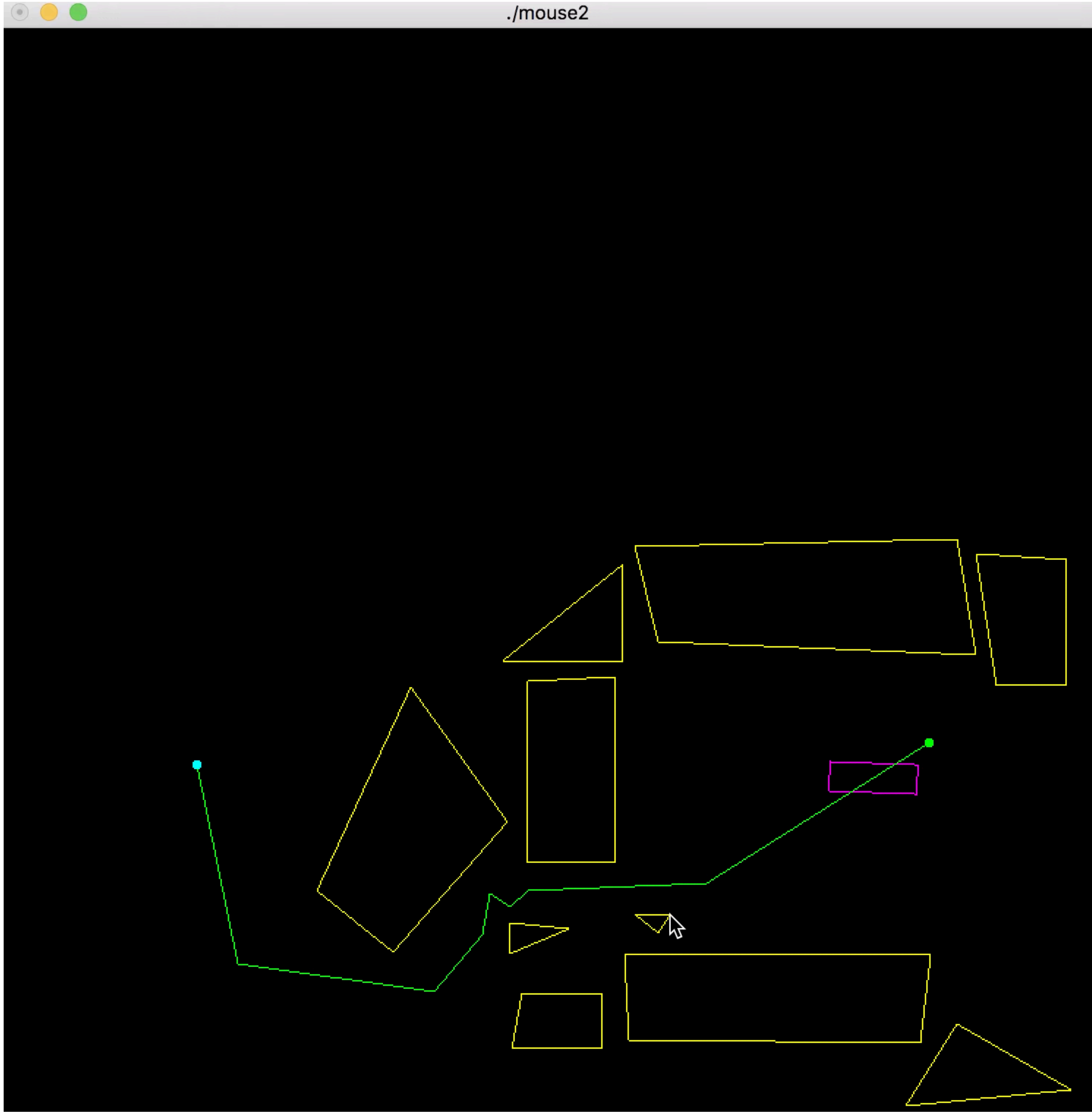
Heuristical motion planning

Implement `(x,y,theta)`-planning for a polygonal robot moving with translation and rotation in 2D using one of:

- A*
- A tree (RRT)
- A probabilistic roadmap (PRM)

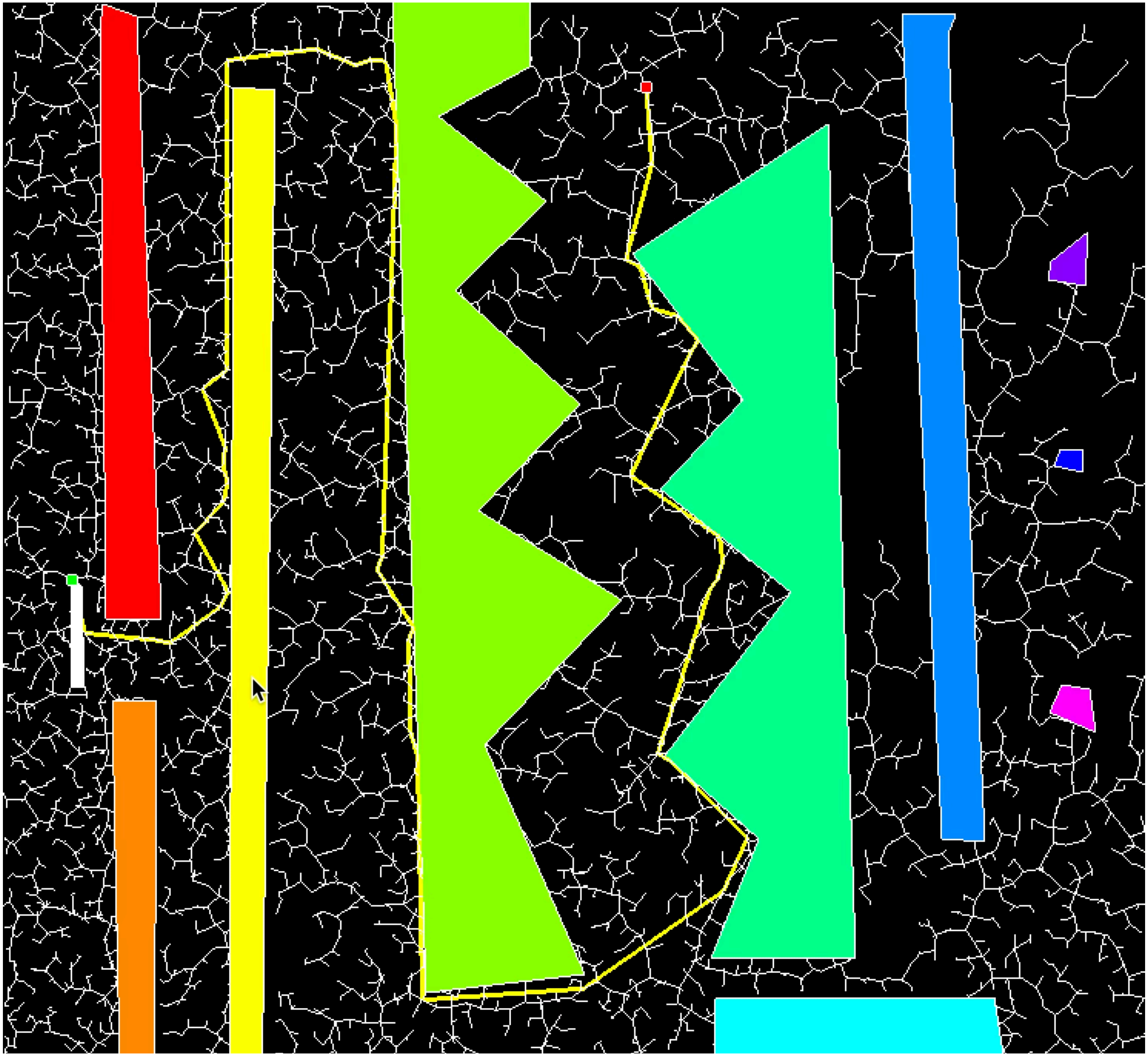

K

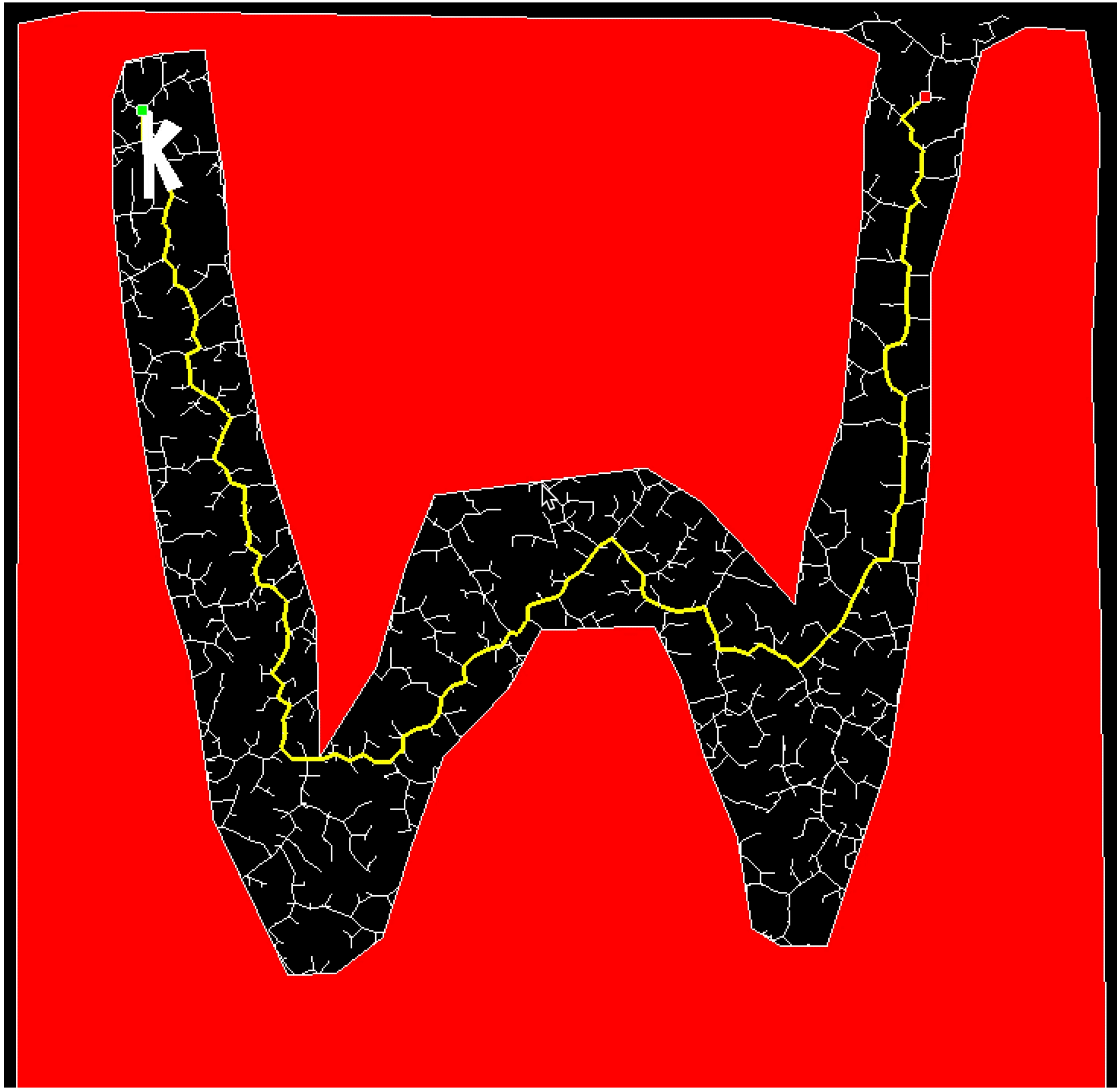




approx — mouse2 — 80×52

```
mouse click at (x=654, y=70)
mouse click at (x=732, y=25)
mouse click at (x=619, y=14)
BUILDING RRT...
RRT BUILD TIME = 0.046529 seconds
CALCULATING SHORTEST PATH...
SHRT PATH CALC TIME = 0.000084 seconds
OPTIMIZING PATH
PATH OPTIMIZATION TIME = 0.018946 seconds
mouse click at (x=667, y=391)
mouse click at (x=729, y=387)
mouse click at (x=729, y=302)
mouse click at (x=681, y=301)
BUILDING RRT...
RRT BUILD TIME = 0.012920 seconds
CALCULATING SHORTEST PATH...
SHRT PATH CALC TIME = 0.000008 seconds
OPTIMIZING PATH
PATH OPTIMIZATION TIME = 0.010743 seconds
mouse click at (x=357, y=91)
mouse click at (x=412, y=90)
mouse click at (x=411, y=53)
mouse click at (x=350, y=53)
mouse click at (x=426, y=384)
mouse click at (x=426, y=317)
mouse click at (x=344, y=318)
BUILDING RRT...
RRT BUILD TIME = 0.418595 seconds
CALCULATING SHORTEST PATH...
SHRT PATH CALC TIME = 0.010016 seconds
OPTIMIZING PATH
PATH OPTIMIZATION TIME = 0.030505 seconds
mouse click at (x=347, y=208)
mouse click at (x=269, y=119)
mouse click at (x=217, y=161)
mouse click at (x=281, y=300)
BUILDING RRT...
RRT BUILD TIME = 0.246371 seconds
CALCULATING SHORTEST PATH...
SHRT PATH CALC TIME = 0.001665 seconds
OPTIMIZING PATH
PATH OPTIMIZATION TIME = 0.019666 seconds
mouse click at (x=434, y=145)
mouse click at (x=450, y=132)
mouse click at (x=458, y=144)
BUILDING RRT...
RRT BUILD TIME = 1.057304 seconds
CALCULATING SHORTEST PATH...
SHRT PATH CALC TIME = 0.027927 seconds
OPTIMIZING PATH
PATH OPTIMIZATION TIME = 0.036774 seconds
```




K