BLG456E
Robotics
Motion Planning

- Framework.
- Reactive approaches.
- Transform-and-search approach.
- Iterative search approach.
- Optimisation for motion planning.
- Other considerations

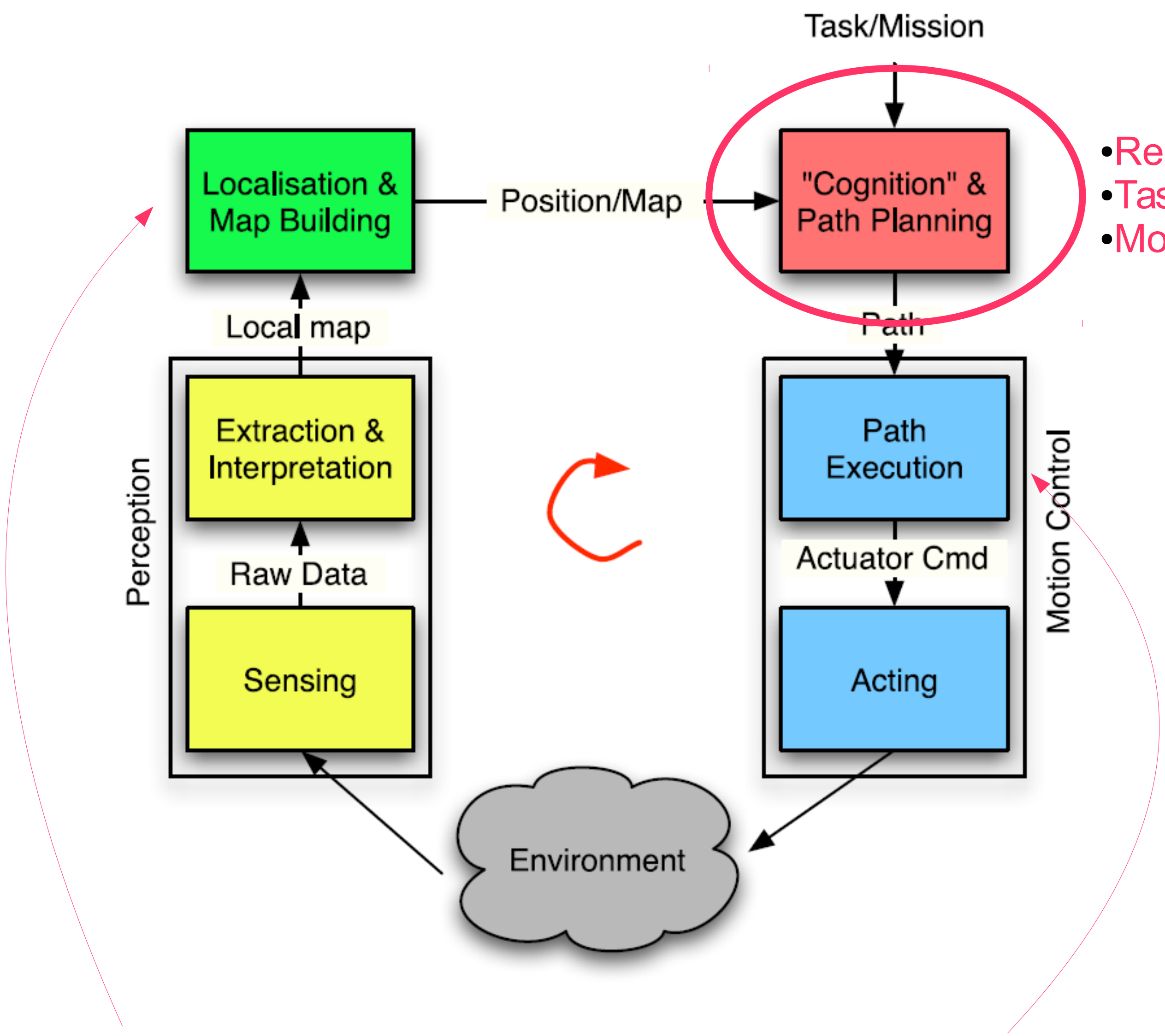
Lecturer: Damien Jade Duff
Email: djduff@itu.edu.tr
Office: EEBF 2316
Schedule: http://djduff.net/my-schedule
Coordination: http://ninova.itu.edu.tr/Ders/4709

Some slides adapted from work by Dr. Sanem Sarıel-Talay

Definition of Motion Planning

Motion planning: *Finding motions (paths, trajectories, movement policies…) to reach a goal while satisfying constraints.*

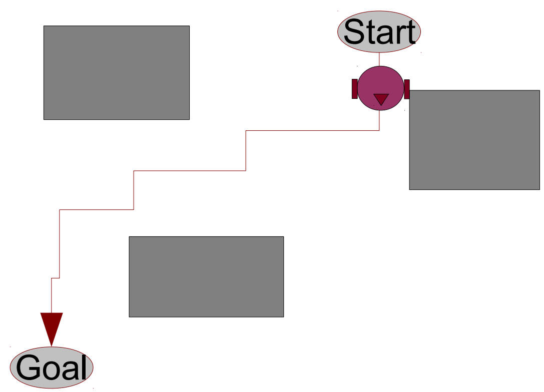
Standard model: Where motion planning fits



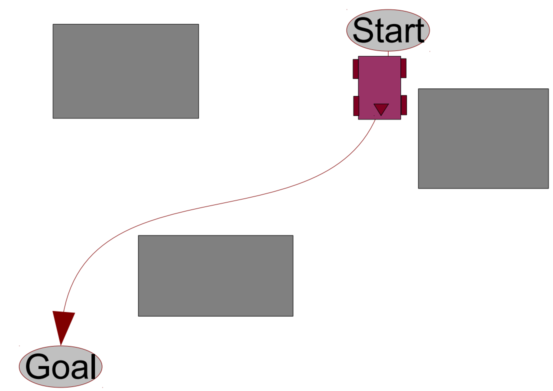
- Reasoning
- Task Planning
- Motion Planning

But sometimes can encompass many other parts (e.g. planning to explore, feedback planning).

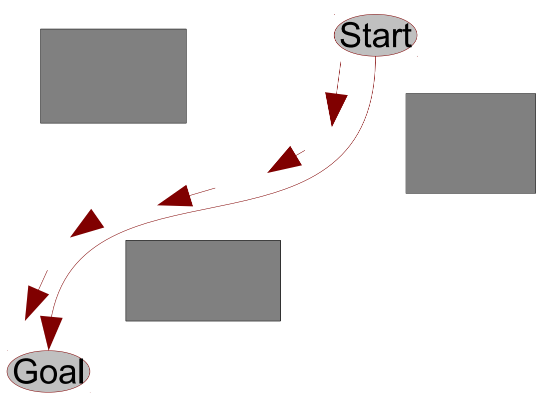
Kinds of abstraction in motion planning



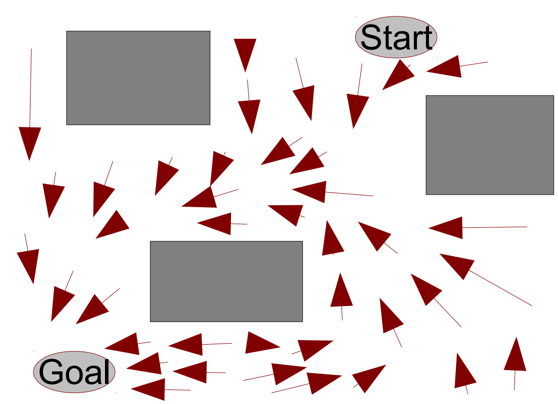
1. Path.



2. Path, conforming
to differential constraints.



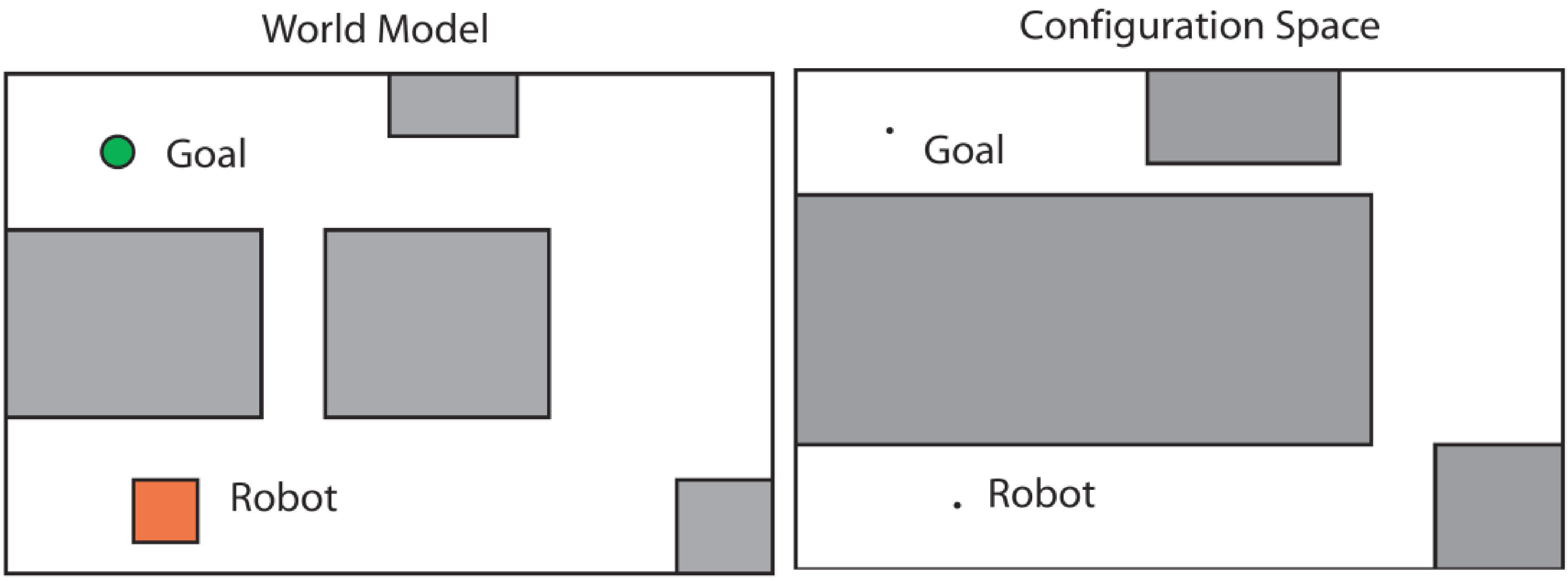
3. Trajectory in phase-space.



4. Feedback controller.

Workspace vs. configuration space

<u>Holonomic mobile robot</u>

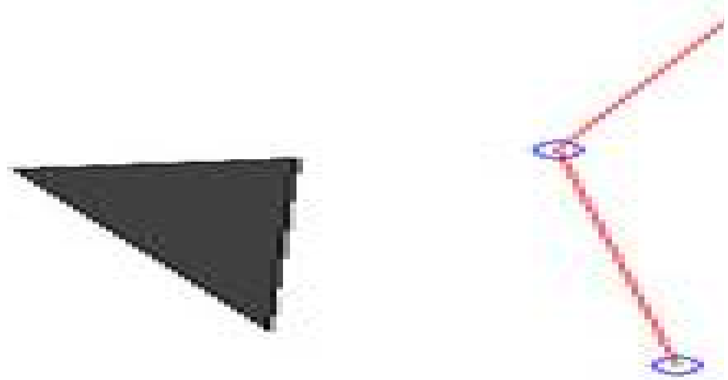


Configurations spaces are supposed to allow *general motion planning*.

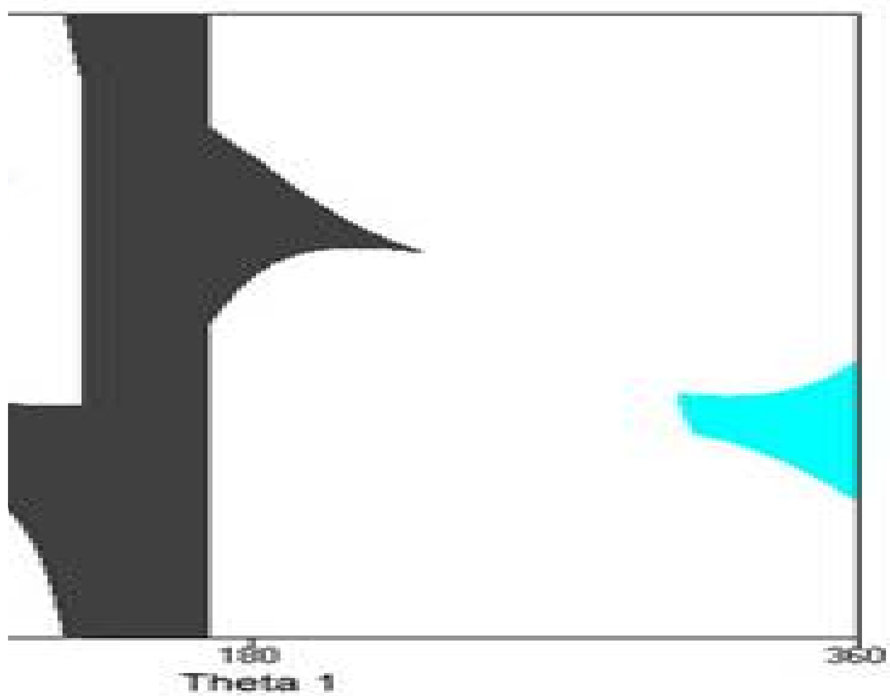
http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/current/exams/mid/2003/

Workspace vs. configuration space

Robot arm



World model



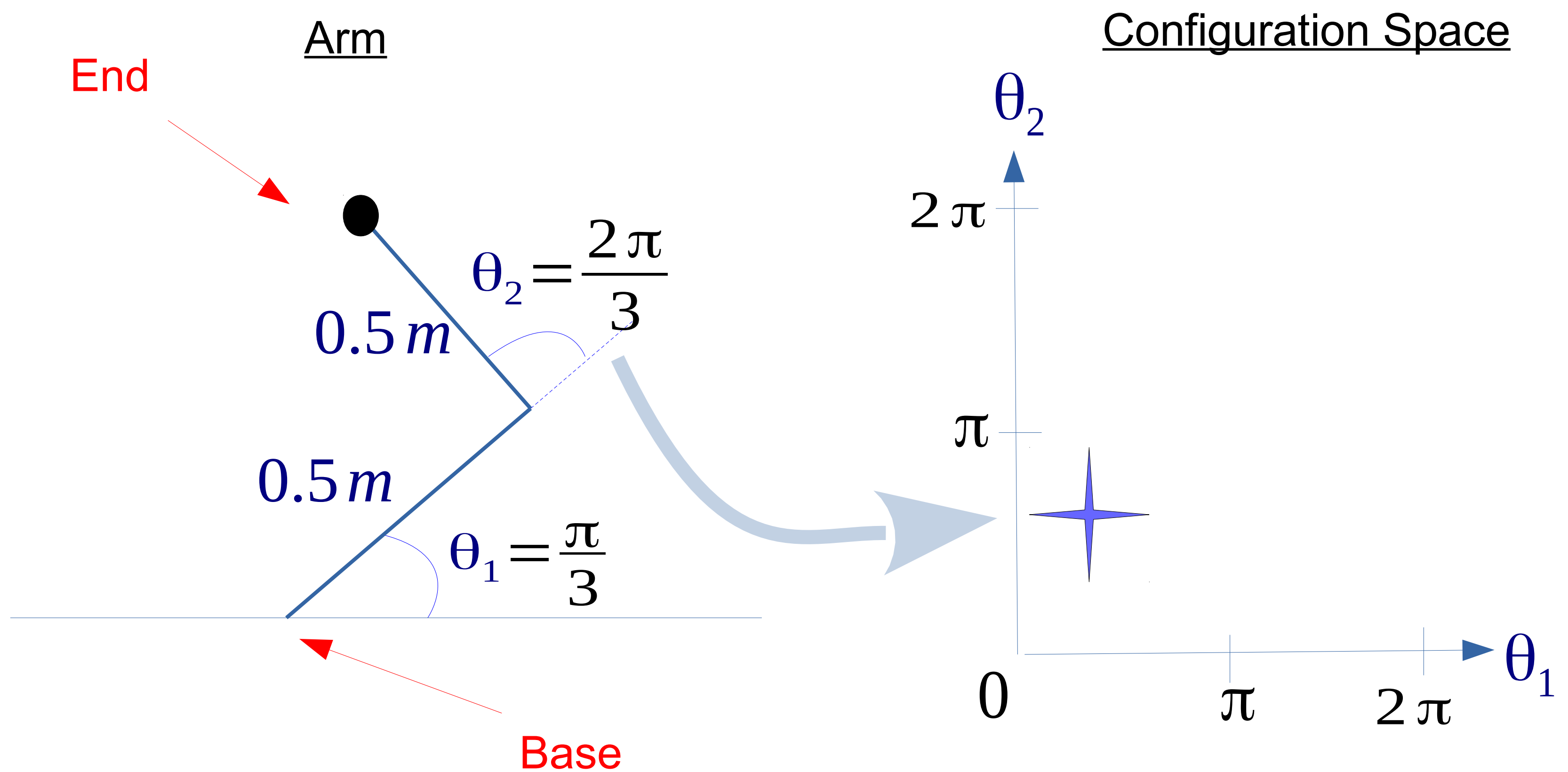
Configuration space

Configurations spaces are supposed to allow *general motion planning*.

http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/current/exams/mid/2003/

Exercise: configuration spaces

With the below robot arm, plot these configurations in configuration space and also draw the arm in its position:



Additional question: Plot a configuration that is impossible due to table-collision or self-collision.

Kinds of Motion Planning

- Workspace (Cartesian planning) vs. configuration space.
- Unconstrained vs. constrained.
- State-space vs. phase-space.
- Trajectory finding vs feedback planning.
- Off-line vs on-line planning.
- Continuous vs hybrid.
- Approach to search:
  - Reactive.
  - Transform and search.
  - Iterative search.

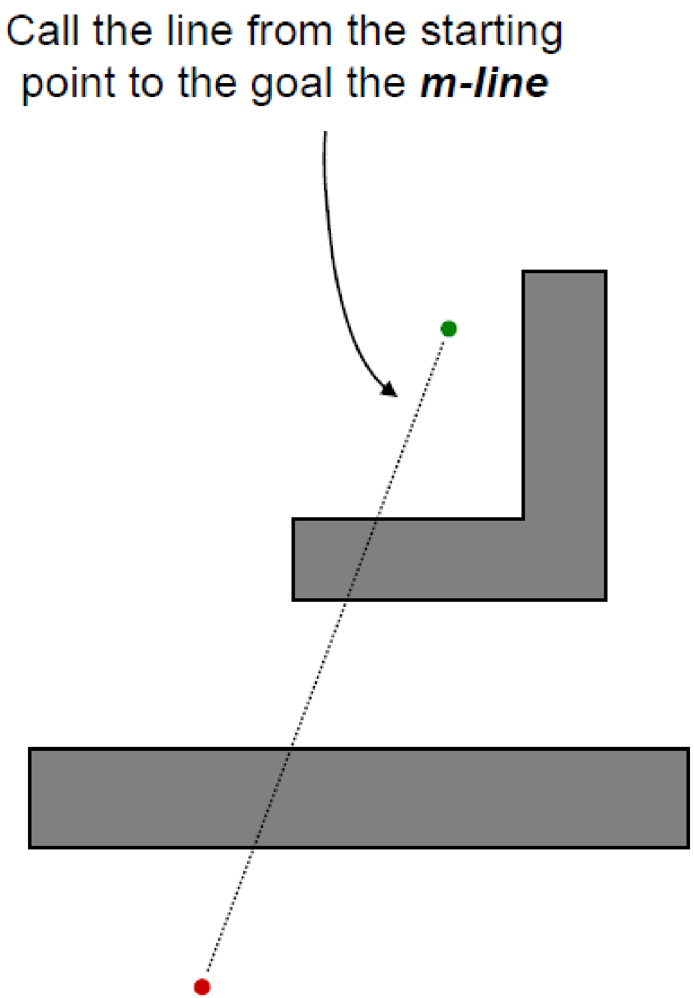
BLG456E
Robotics
Motion Planning

- Framework.
- Reactive approaches.
- Transform-and-search approach.
- Iterative search approach.
- Optimisation for motion planning.
- Other considerations

Lecturer: Damien Jade Duff
Email: djduff@itu.edu.tr
Office: EEBF 2316
Schedule: http://djduff.net/my-schedule
Coordination: http://ninova.itu.edu.tr/Ders/4709

Some slides adapted from work by Dr. Sanem Sarıel-Talay

Reactive algorithms I: Bug2



- IF no obstacle and would make progress THEN
  - Move towards goal on *m-line*.
- ELSE
  - Keep obstacle on right (left) side.
- LOOP

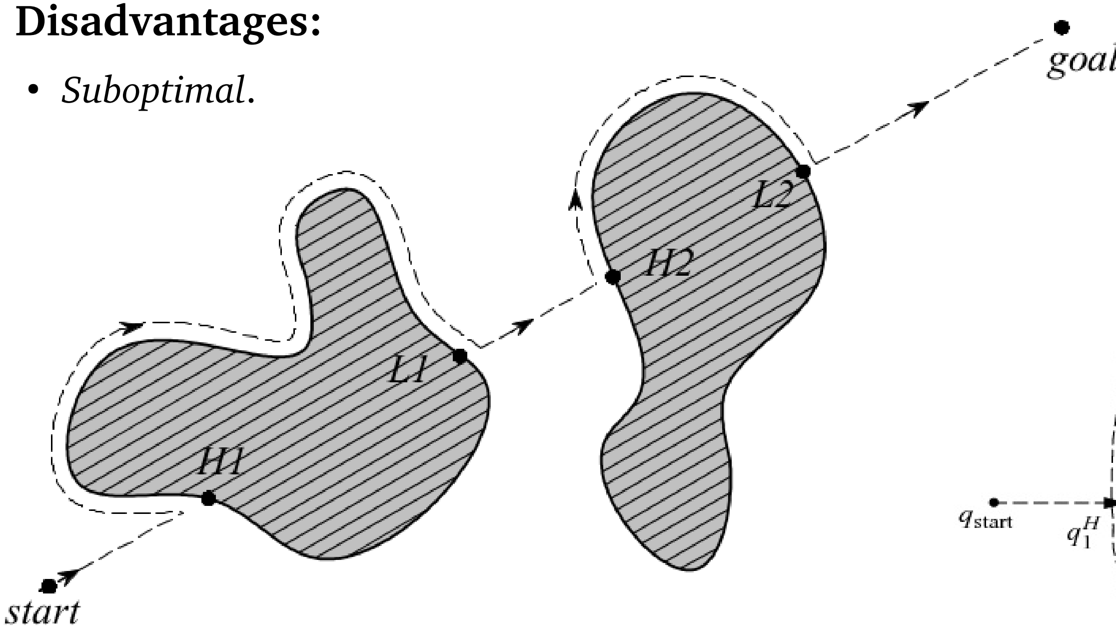
Reactive algorithms I: Bug2

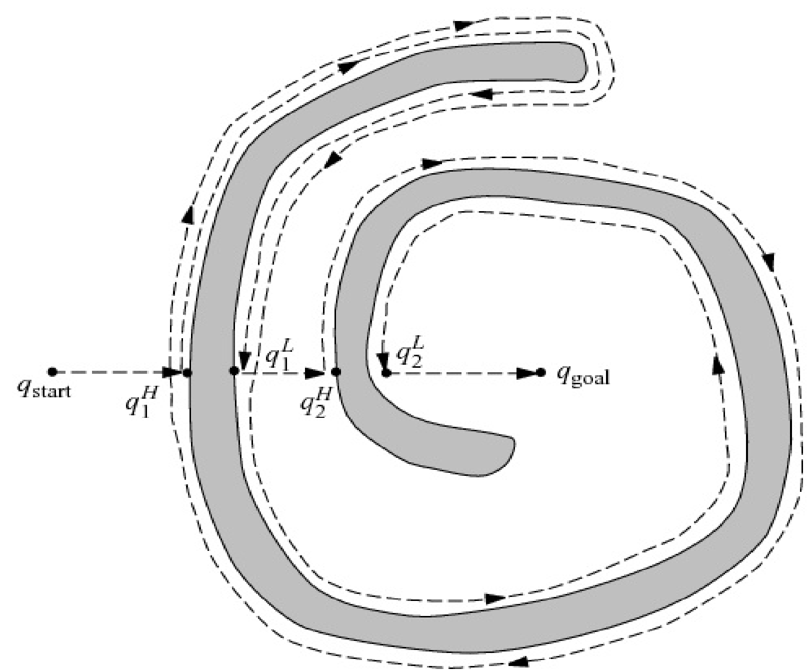
Advantages:

- No map required.
- Reactive (but needs localisation).
- *Complete*.

Disadvantages:

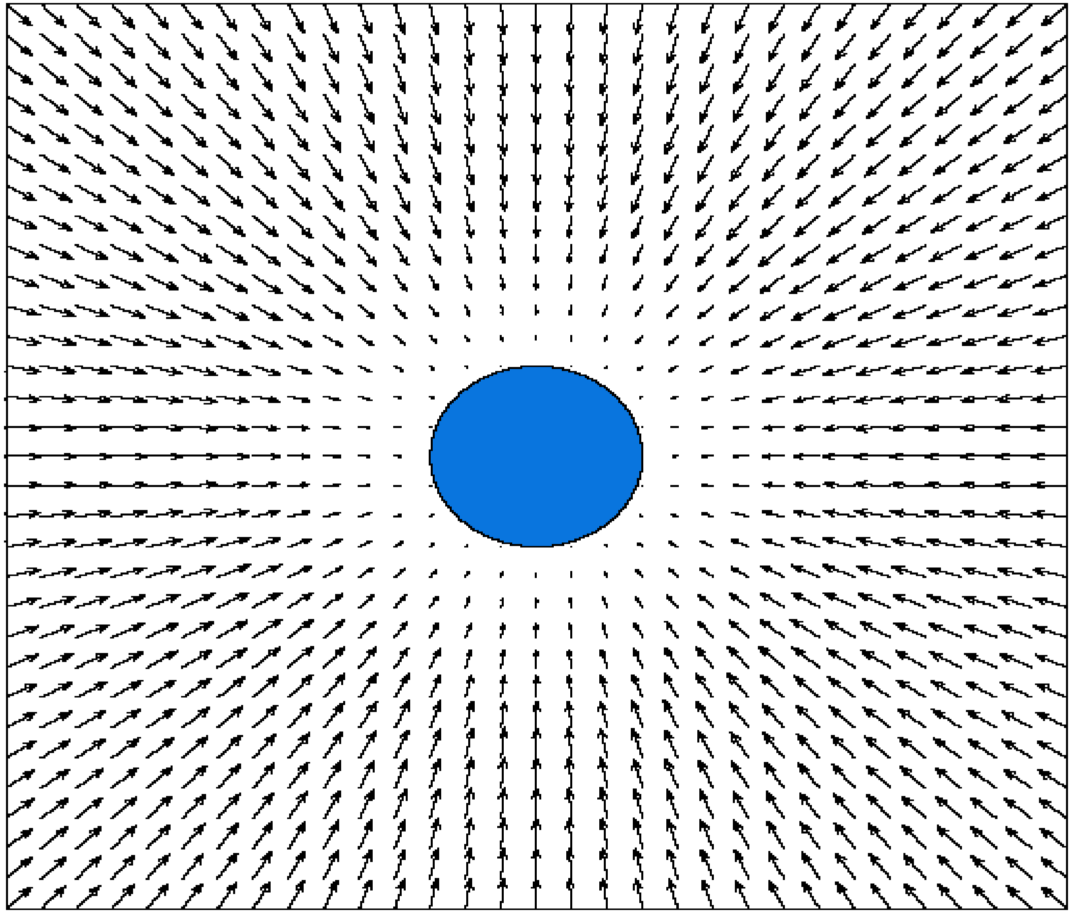
- *Suboptimal*.



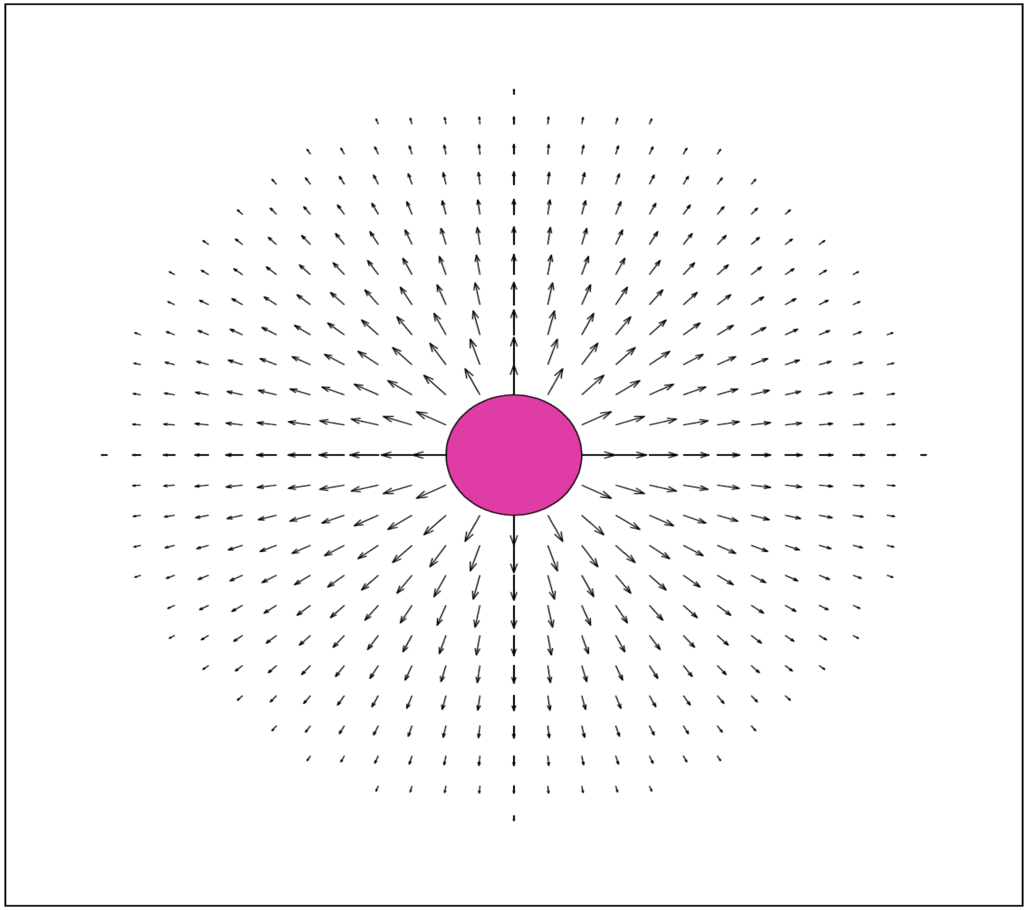


Reactive algorithms II: Potential field

Goal provides attractive field.

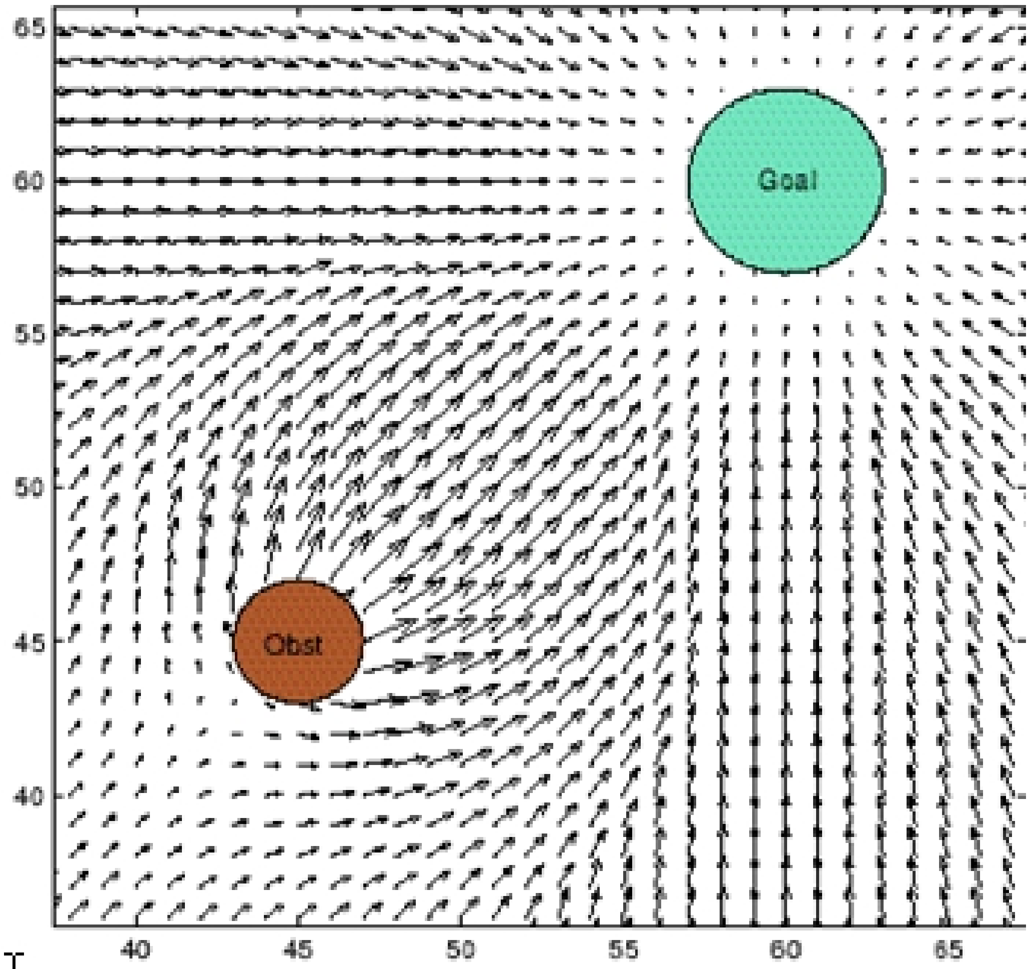


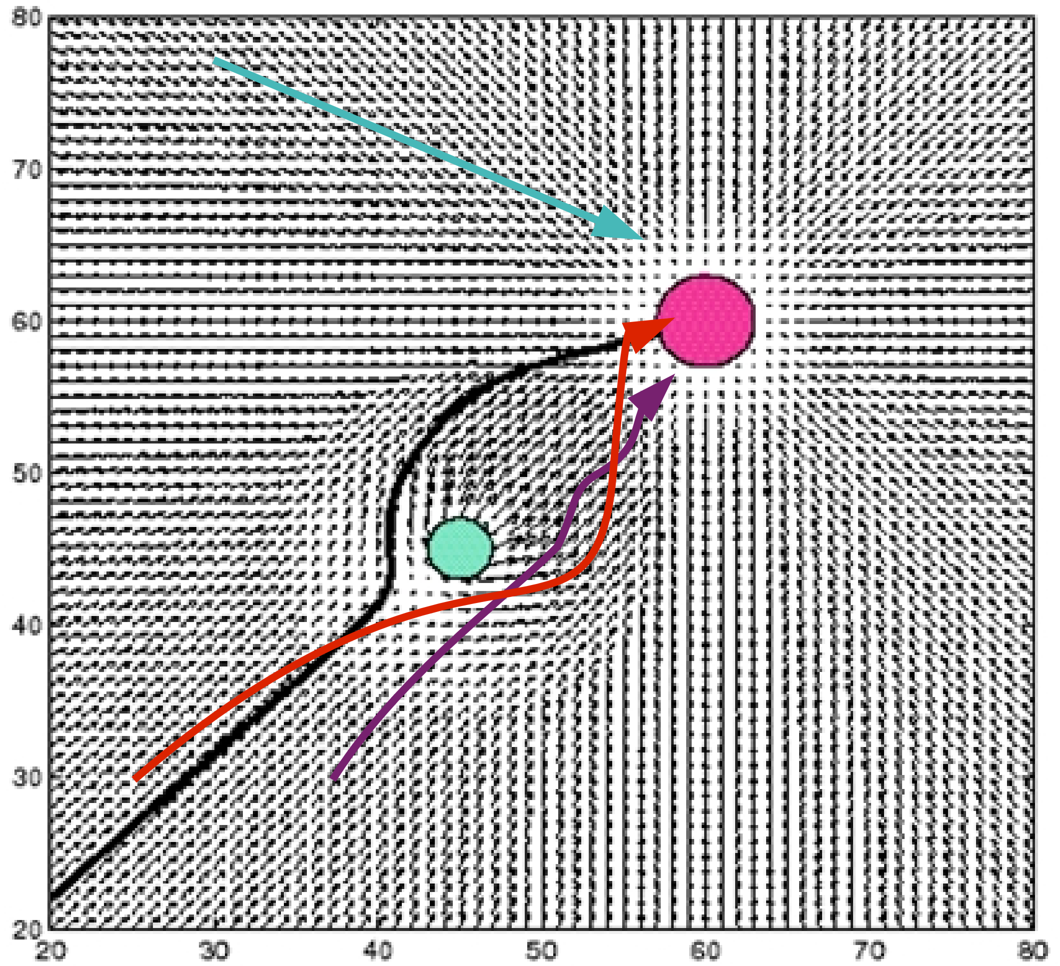
Obstacles provide repelling field.



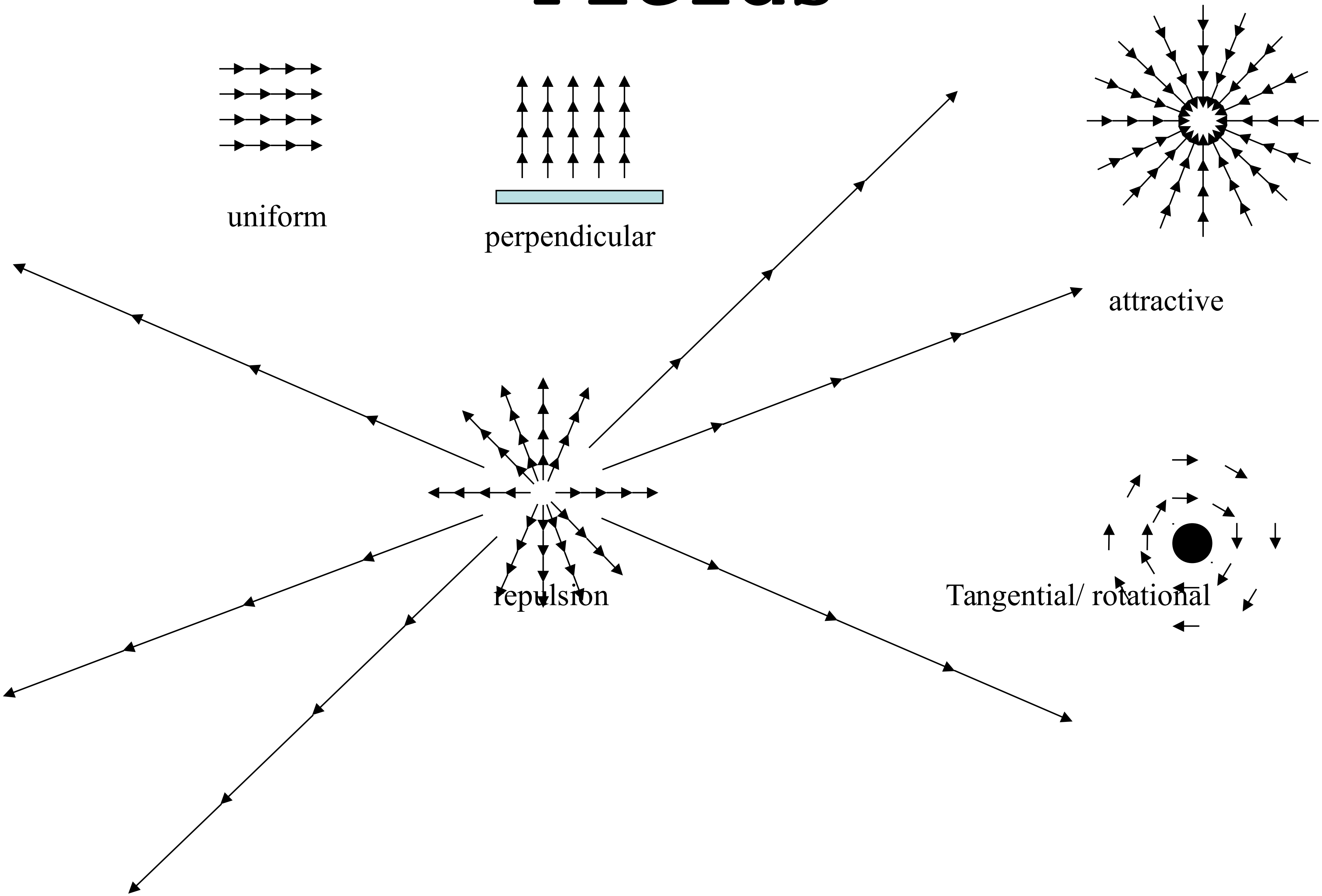
Reactive algorithms II: Potential field

Fields added (vector addition).





Primitive Potential Fields



Reactive algorithms II: Potential field

- Goal provides attractive field. e.g. $U_g(x,y)=k_g\left((x-x_g)^2+(y-y_g)^2\right)$
- Obstacles provide repelling field. e.g. $U_i(x,y)=\dfrac{k_{obs}}{(x-x_i)^2+(y-y_i)^2}$
- Fields added (vector addition). e.g. $U=(x,y)=U_i(x,y)+U_g(x,y)$

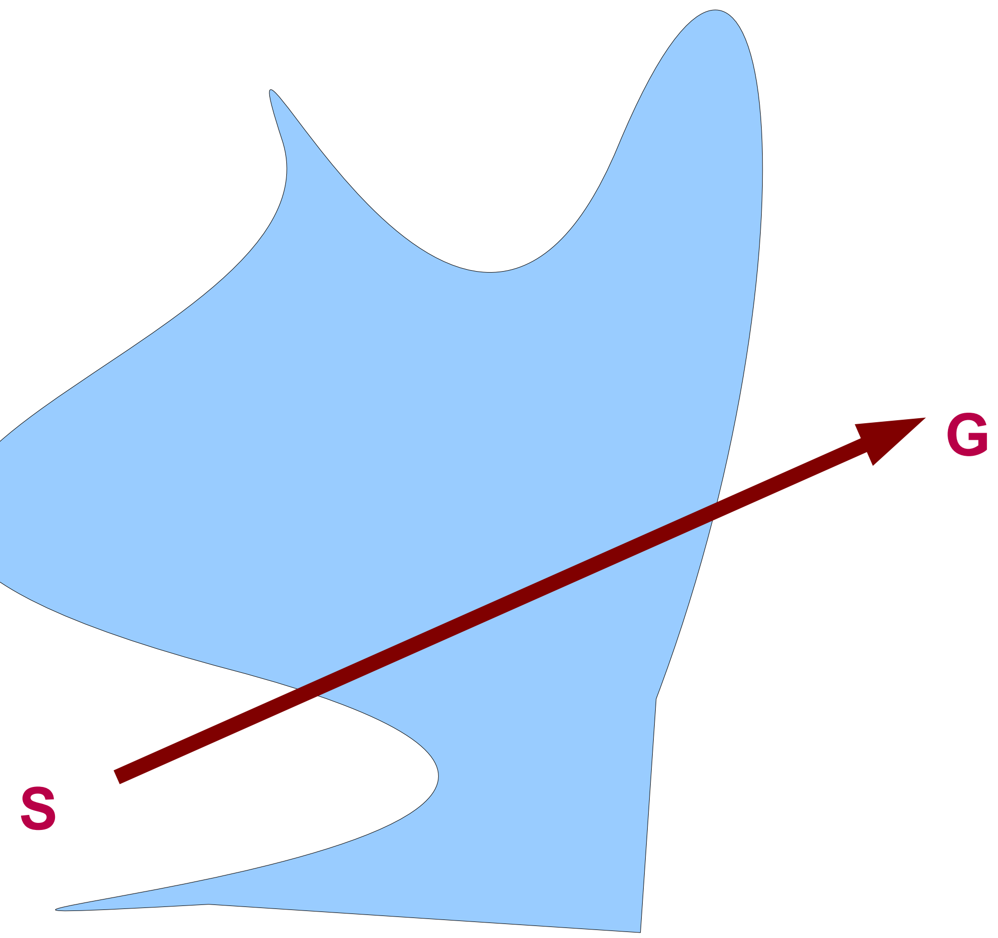
Differentiate these to find forces in X and Y direction.

$$\text{e.g. } \nabla U_i(x,y) = \begin{bmatrix} \dfrac{\partial U}{\partial x} \\ \dfrac{\partial U}{\partial y} \end{bmatrix} = \begin{bmatrix} \dfrac{2(x_i-x)k_{obs}}{((x-x_i)^2+(y-y_i)^2)^2} \\ \dfrac{2(y_i-y)k_{obs}}{((x-x_i)^2+(y-y_i)^2)^2} \end{bmatrix}$$

Reactive algorithms II: Potential field

- Advantages:
  - Can be faster.
  - Simple.
  - Adaptable to different domains/constraints.
- Disadvantages:
  - Suboptimal.
  - *Incomplete*.

    (won't always find a plan and won't know if there isn't one)



BLG456E
Robotics
Motion Planning

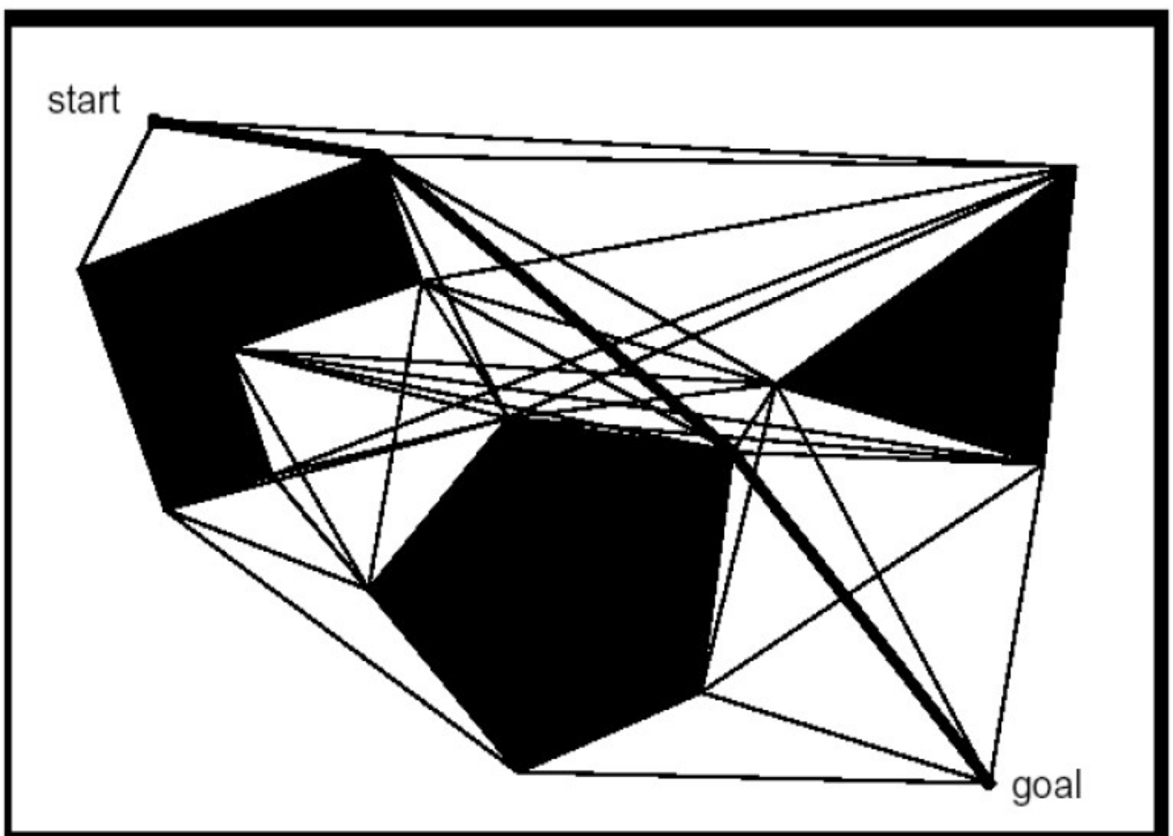
- Framework.
- Reactive approaches.
- Transform-and-search approach.
- Iterative search approach.
- Optimisation for motion planning.
- Other considerations

Lecturer:	Damien Jade Duff
Email:	djduff@itu.edu.tr
Office:	EEBF 2316
Schedule:	http://djduff.net/my-schedule
Coordination:	http://ninova.itu.edu.tr/Ders/4709

Some slides adapted from work by Dr. Sanem Sarıel-Talay

Transform-and-search algorithms I: Visibility graph

- Connect vertices (start & goal) with lines between.
- Do graph search (shortest path, cost=distance).



<u>Advantages?</u>
- Complete
- Geometrically optimal.

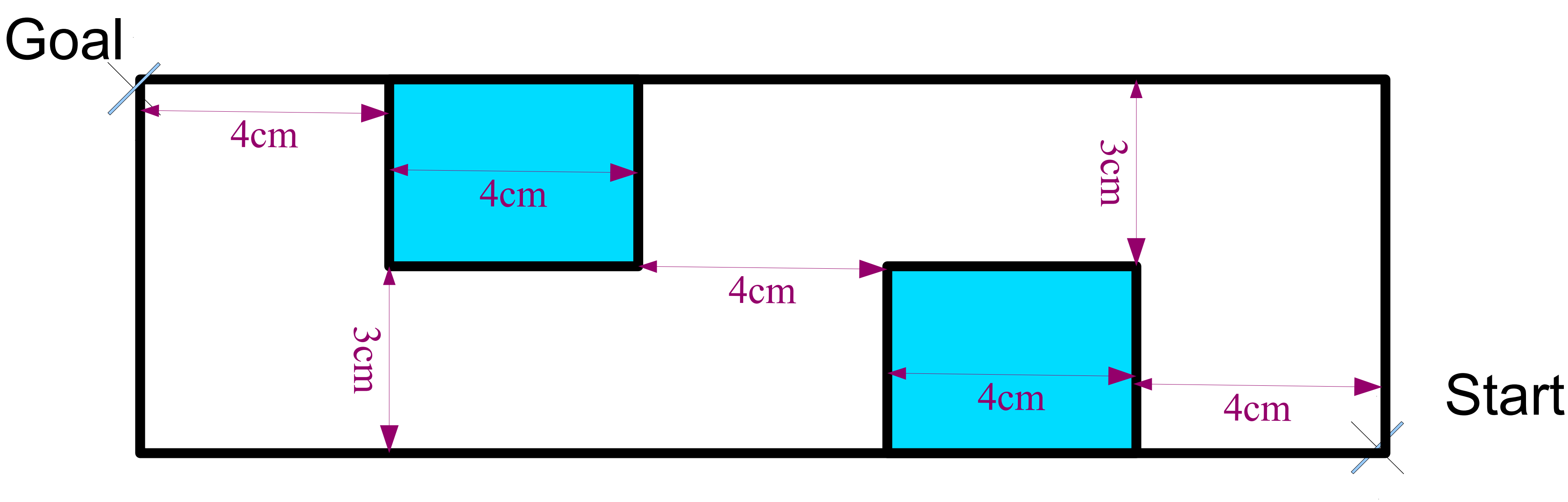
<u>Drawbacks?</u>
- Compuational complexity.
- Polygons required.
- Closeness to obstacles?
- Differential constraints?

Exercise: make a visibility graph

1. Make a visibility graph out of the below motion planning problem.

2. Calculate edge lengths.

3. Do a depth-first search, expanding nodes anti-clockwise, to find a path start→ goal.



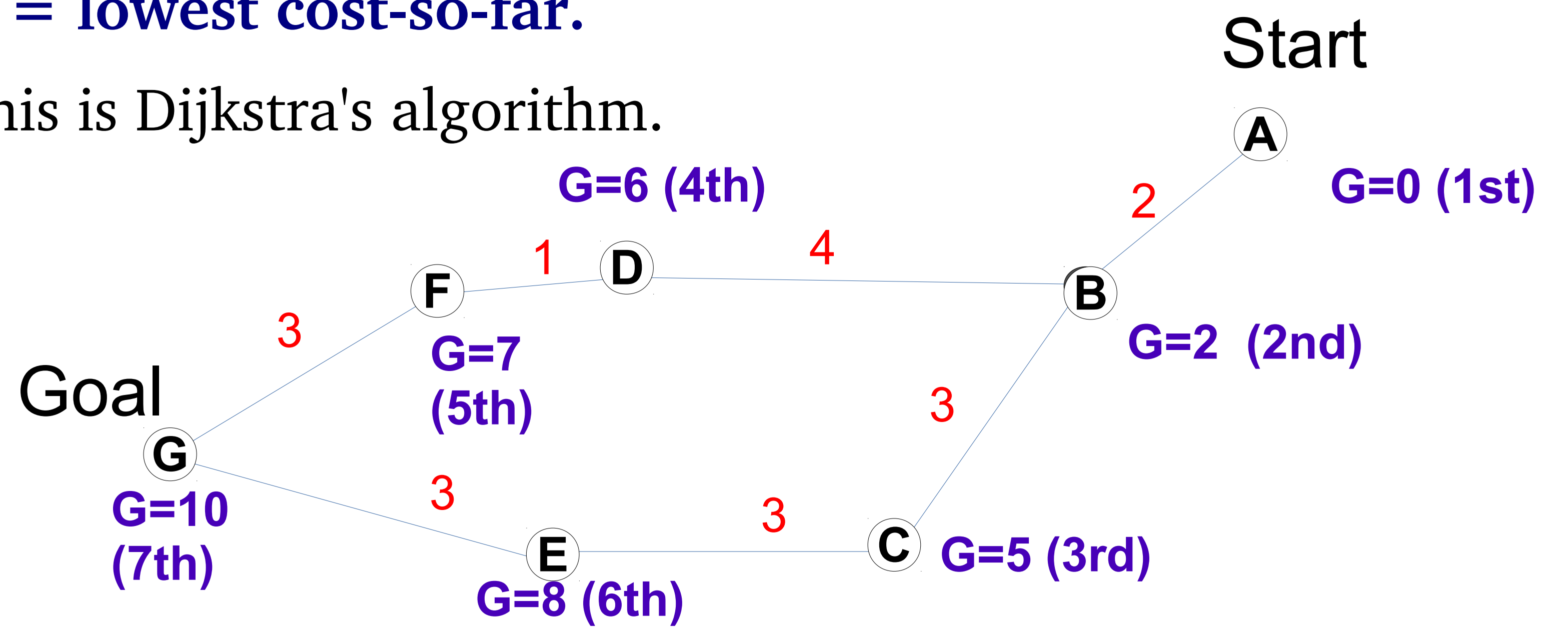
Better search in roadmaps: **Dijkstra's**

- Shortest path guaranteed.
- Order node expansion by "best first".

  Rather than FIFO or LIFO.
- Keep a "visited" list.
- Simplest example of "best first":

  G = lowest cost-so-far.

  This is Dijkstra's algorithm.

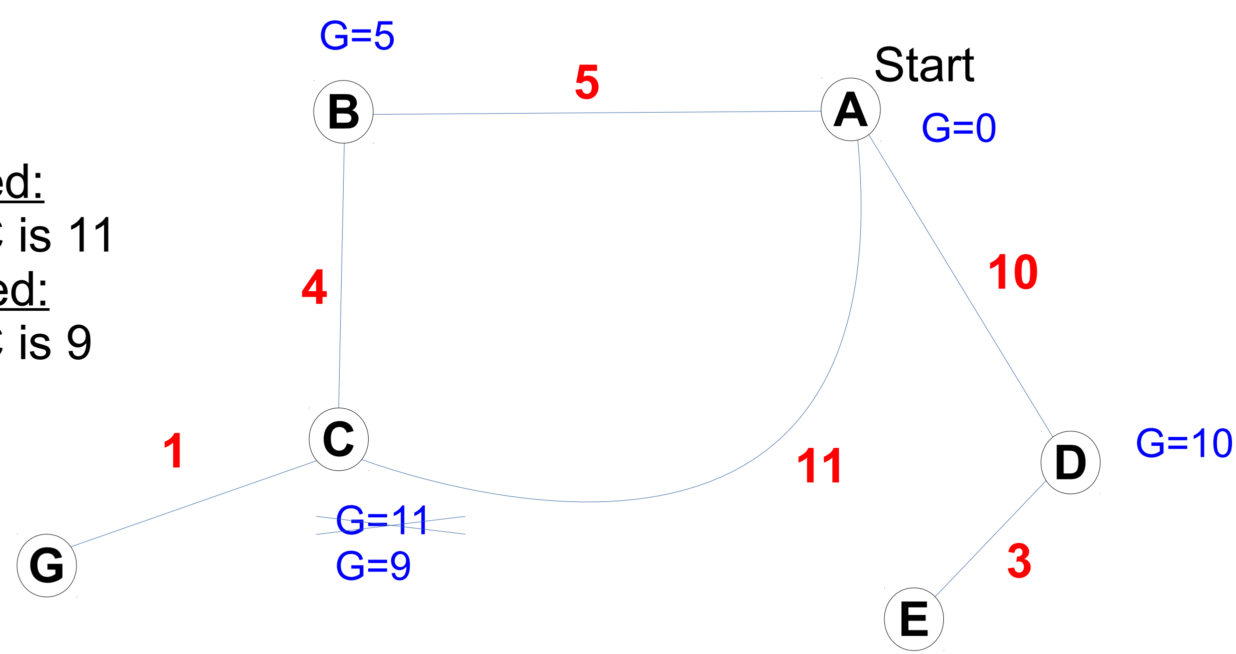


Multiple routes to a node

Sometimes need to update cost at a fringe node.

Fringe = set of candidate nodes to be expanded (unexpanded neighbours of expanded nodes)

<u>After A expanded:</u>
Cost-so-far at C is 11
<u>After B expanded:</u>
Cost-so-far at C is 9



Better search in roadmaps: **A***

- Can do better than cost-so-far.
- Add a heuristic cost-to-go.
  - Order node expansion by F = G + H

    G = Known cost-so-far (start to node).

    H = Heuristic cost-to-go (node to goal).
- Cost must be "admissible" for shortest-path guarantee.

  Admissible: Never overestimates cost.

General search algorithm

$FRINGE \leftarrow \{(initial\_node)\}$

$VISITED \leftarrow \{initial\_node\}$

loop while $|FRINGE| > 0$:

  $current\_path \leftarrow \text{Choose}(FRINGE)$

  for n in $\text{Neighbours}(\text{End}(current\_path))$:

    if $n \notin VISITED$:

      if $n \in GOAL$ return $current\_path + n$

      $FRINGE \leftarrow FRINGE \cup \{current\_path + n\}$

      $VISITED \leftarrow VISITED \cup \{n\}$

* FIFO
* LIFO
* cost-so-far
* heuristic

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

A* example

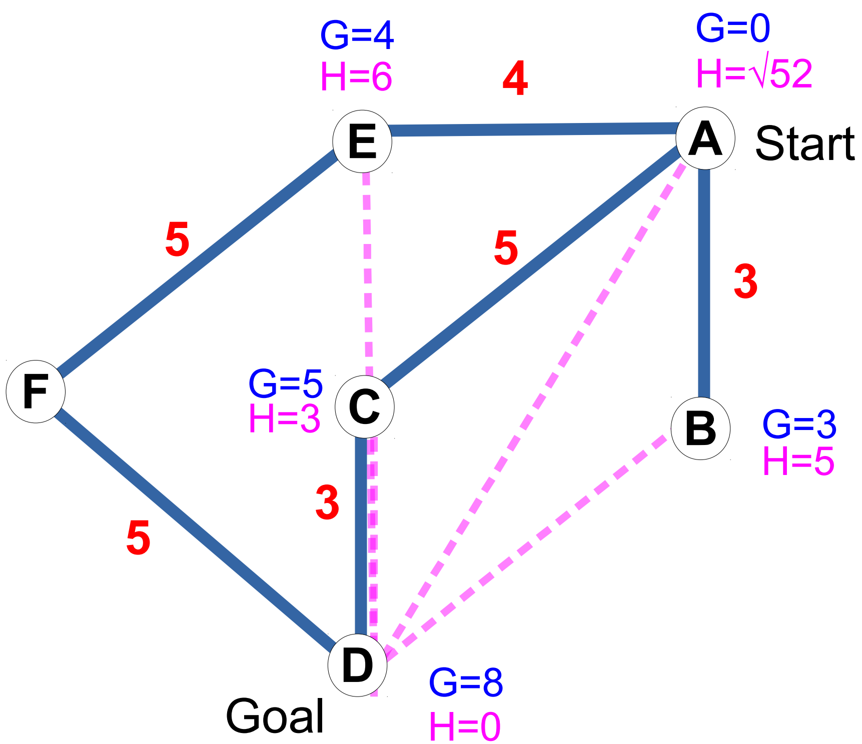
Euclidean distance:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Manhattan distance:

$$|x_1 - x_2| + |y_1 - y_2|$$

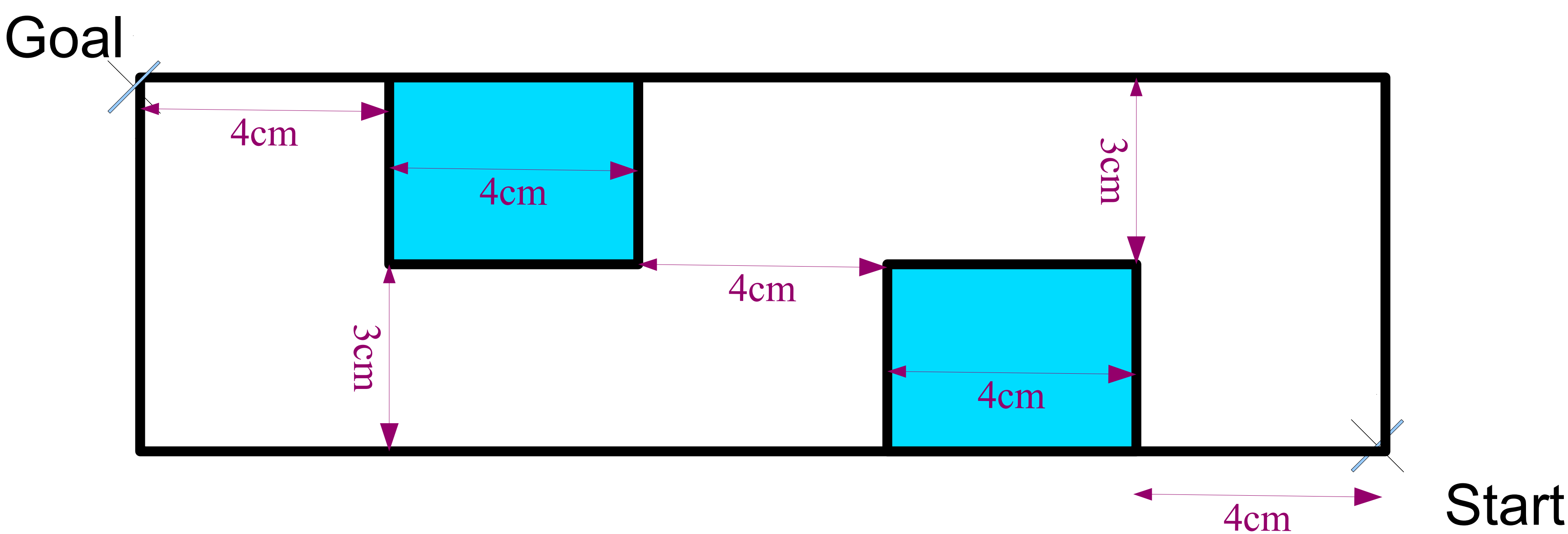
- G = cost so far.
- Let H be Euclidean distance to goal.
- What would happen if H was the Manhattan distance?



Fringe	Visited
Step 1: **A**: F=2√52 *Expand A*	Step 1:
Step 2: **B**: F=8 **C**: F=8 **E**: F=10 *Expand B*	Step 2: **A**
Step 3: **C**: F=8 **E**: F=10 *Expand C*	Step 3: **A, B**

Exercise: search a road-map

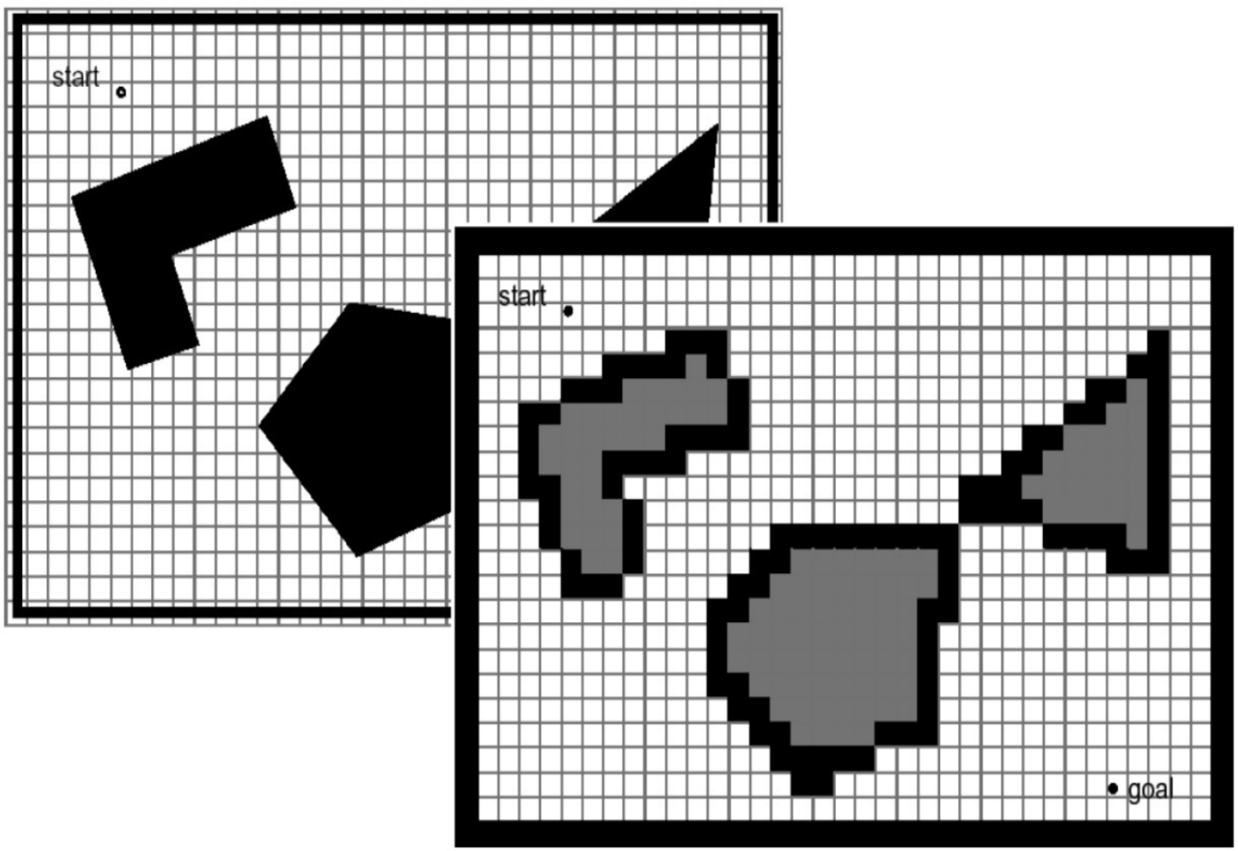
Calculate the visibility graph of the below problem, add edge lengths, and conduct an A* search using Euclidean distance as the heuristic.



Finding goal does not guarantee shortest path found.

Transform-and-search algorithms V: Approximate grid decomposition

- Graph nodes are grid entries.
- Do graph search (shortest path, cost=distance).

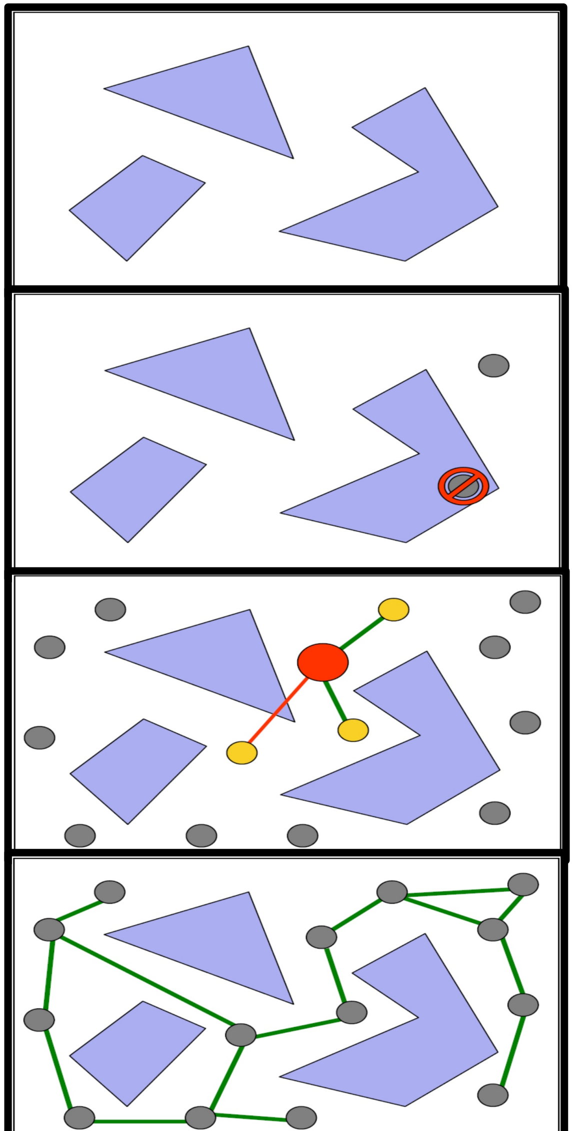


Advantages
- Efficient.
- More granular than simple decomposition.
- *Resolution complete.* (fine enough resolution → solution known)

Drawbacks?
- Approximation near edges (tight gaps)
- Robot shape?
- Differential constraints?

Diagram: Siegwart &

Transform-and-search algorithms III: Probabilistic Roadmaps

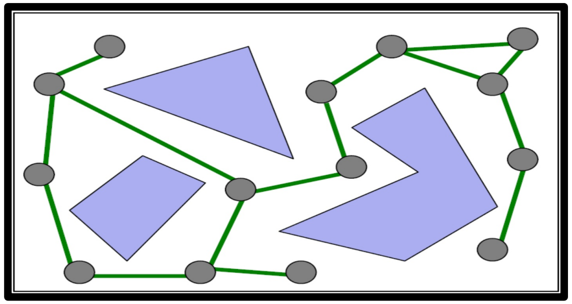


- Randomly sample N collision-free vertices.
- Put edges between k closest neighbours.
- Search graph: closest-point to start → closest-point to goal.

Transform-and-search algorithms III: Probabilistic Roadmaps

- <u>Advantages:</u>
  - *Probabilistically complete.*

    As sampled points $\rightarrow \infty$, correctness probability $\rightarrow 1$.
  - Efficient.
  - Not domain-specific (e.g. protein folding).
- <u>Disadvantages:</u>
  - Completeness depends on computation time.
  - Differential constraints & tight spaces:
    - How to move between nodes?
    - Start to a node, node to a goal?
    - Direction along edge important?
    - *Local planning* needed.



BLG456E
Robotics
Motion Planning

- Framework.
- Reactive approaches.
- Transform-and-search approach.
- Iterative search approach.
- Optimisation for motion planning.
- Other considerations

Lecturer: Damien Jade Duff
Email: djduff@itu.edu.tr
Office: EEBF 2316
Schedule: http://djduff.net/my-schedule
Coordination: http://ninova.itu.edu.tr/Ders/4709

Some slides adapted from work by Dr. Sanem Sarıel-Talay

Iterative search algorithms I: Incremental forward search

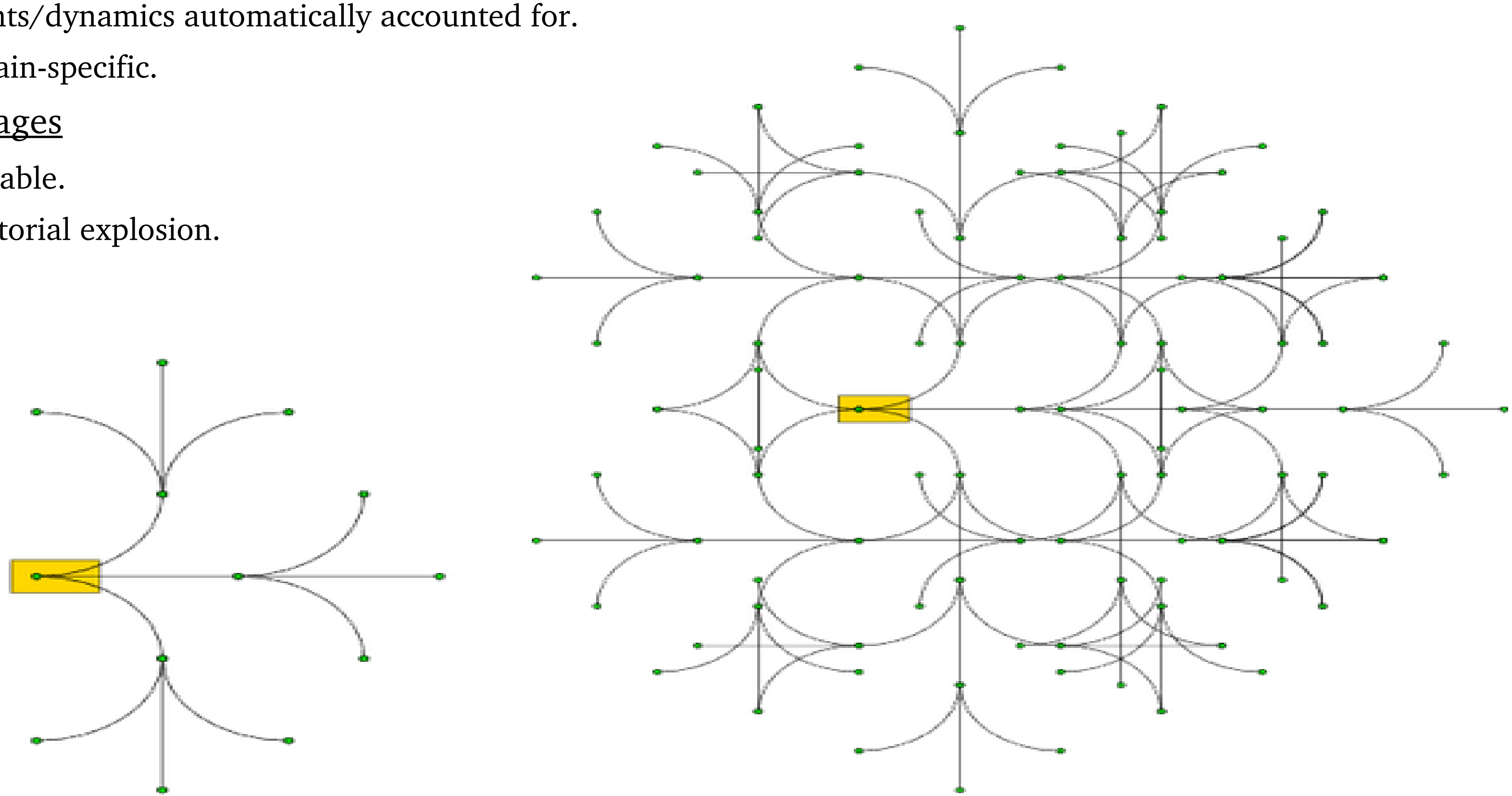
- Combine "motion primitives" – can be *any* motion!
- No graph, just search.

  Advantages

  - Constraints/dynamics automatically accounted for.
  - Not domain-specific.

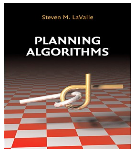
  Disadvantages

  - Non-reusable.
  - Combinatorial explosion.

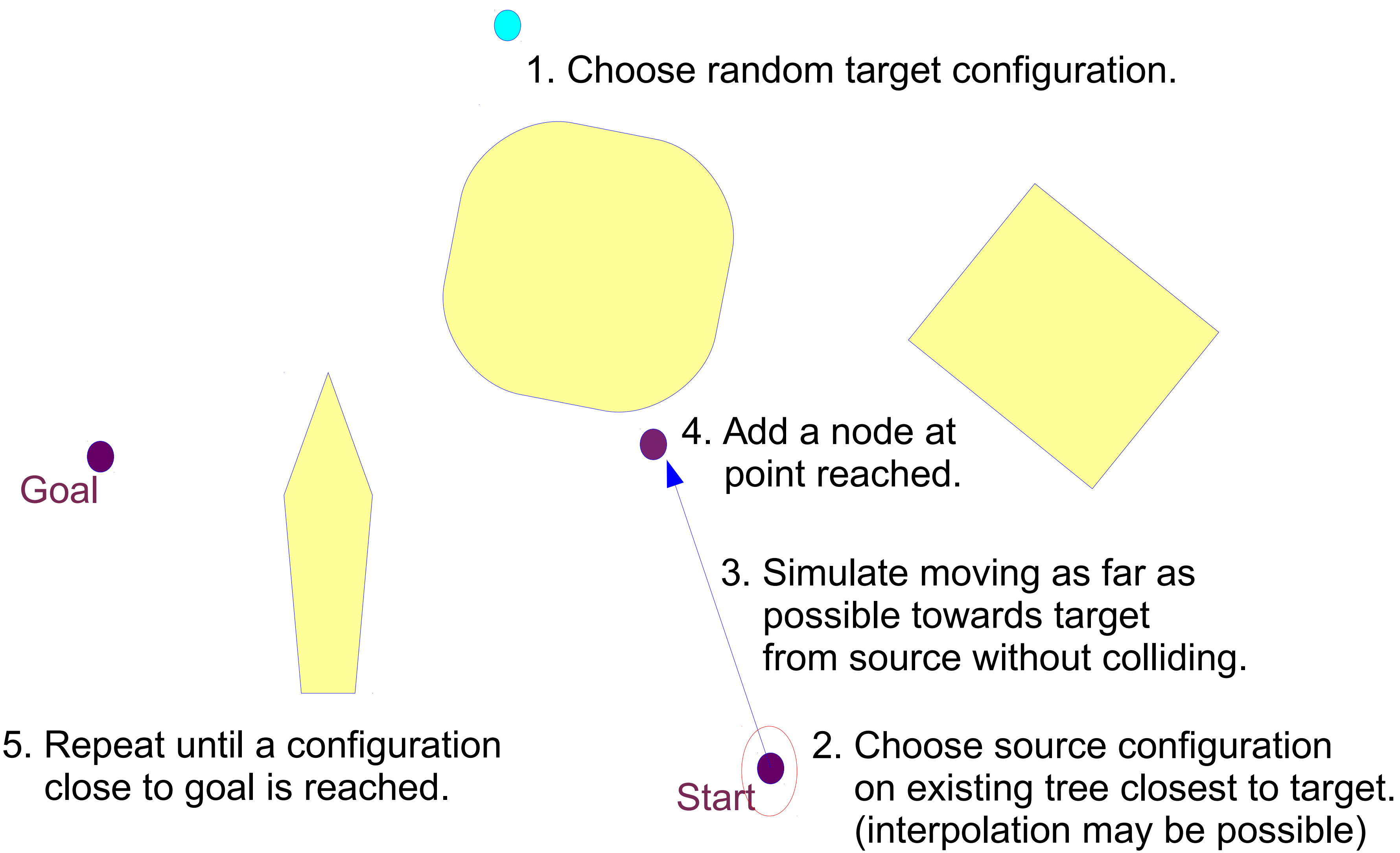


Two stages Four stages

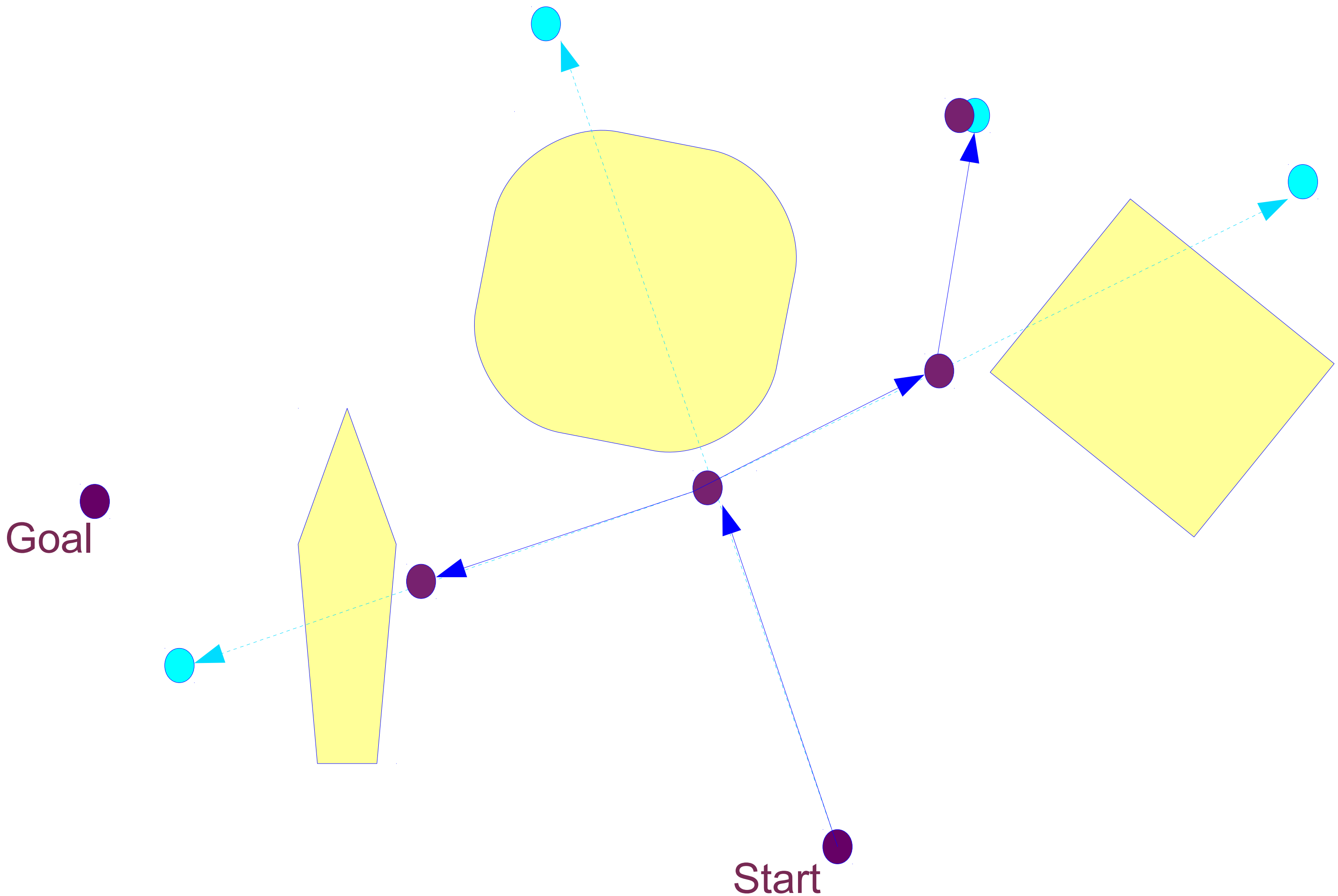
From LaValle "Planning Algorithms"



Iterative search algorithms II:
Rapidly exploring Random Trees (RRTs)



Iterative search algorithms II:
Rapidly exploring Random Trees (RRTs)



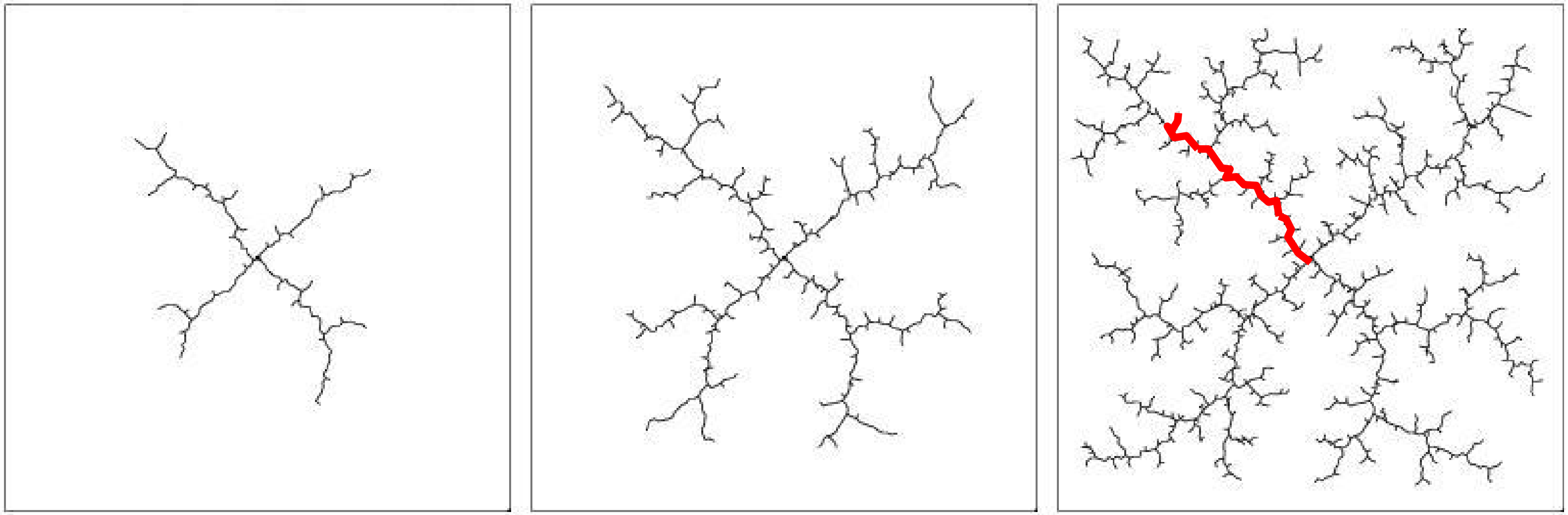
Iterative search algorithms II:
Rapidly exploring Random Trees (RRTs)

<u>Advantages</u>

- Fast.
- Not domain specific.
- Often probabilistically complete.

<u>Disadvantages</u>

- Can be suboptimal.
- Can fail.



See also http://msl.cs.uiuc.edu/rrt/gallery.html
and http://correll.cs.colorado.edu/?p=1623

BLG456E
Robotics
Motion Planning

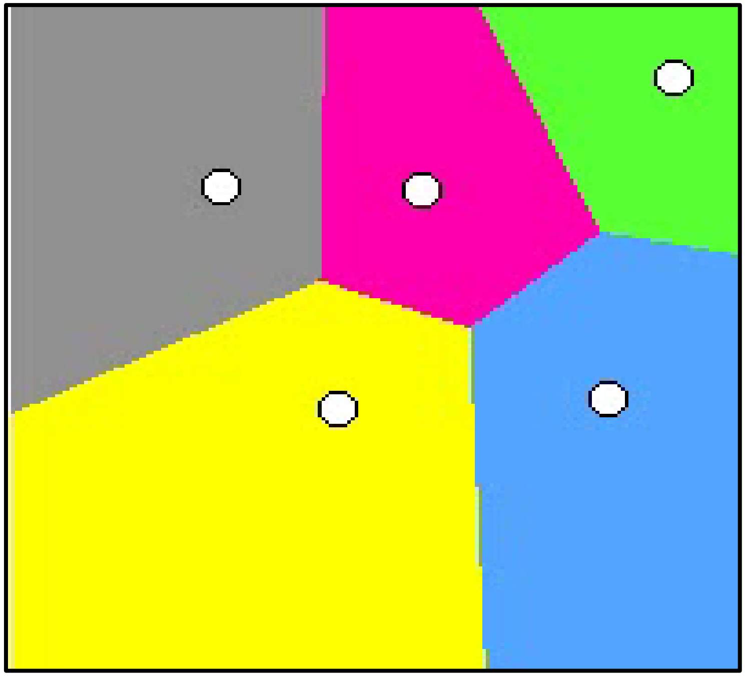
- Framework.
- Reactive approaches.
- Transform-and-search approach.
- Iterative search approach.
- Optimisation for motion planning.
- Other considerations

Lecturer: Damien Jade Duff
Email: djduff@itu.edu.tr
Office: EEBF 2316
Schedule: http://djduff.net/my-schedule
Coordination: http://ninova.itu.edu.tr/Ders/4709

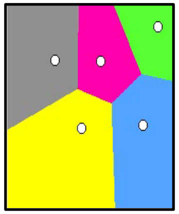
Some slides adapted from work by Dr. Sanem Sarıel-Talay

Voronoi Graph Preliminary: Voronoi decomposition

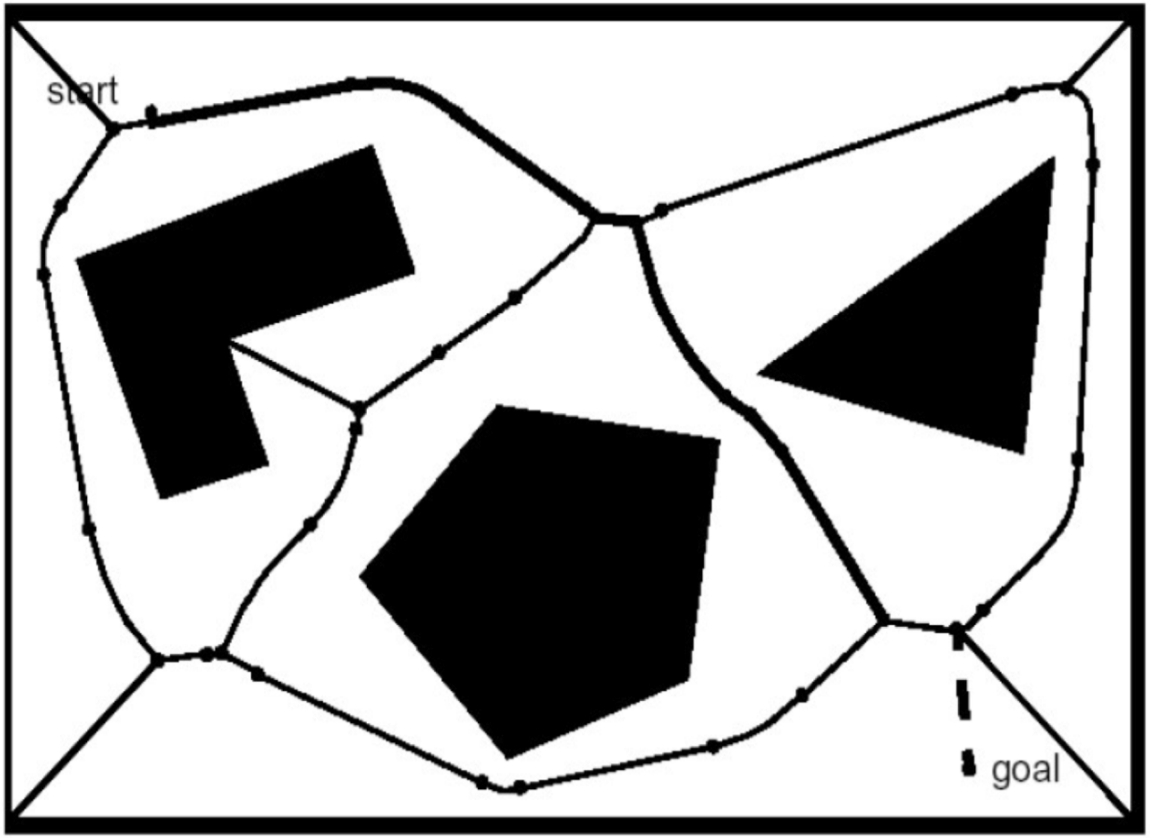
- Colour space according to closest obstacle point.
- *Edges are maximally distant from obstacle points.*
- Generalisations include distance from *edges*.



Transform-and-search algorithms II: Voronoi graph



- Graph nodes are Voronoi edge junctions.
- Do graph search (shortest path, cost=distance).



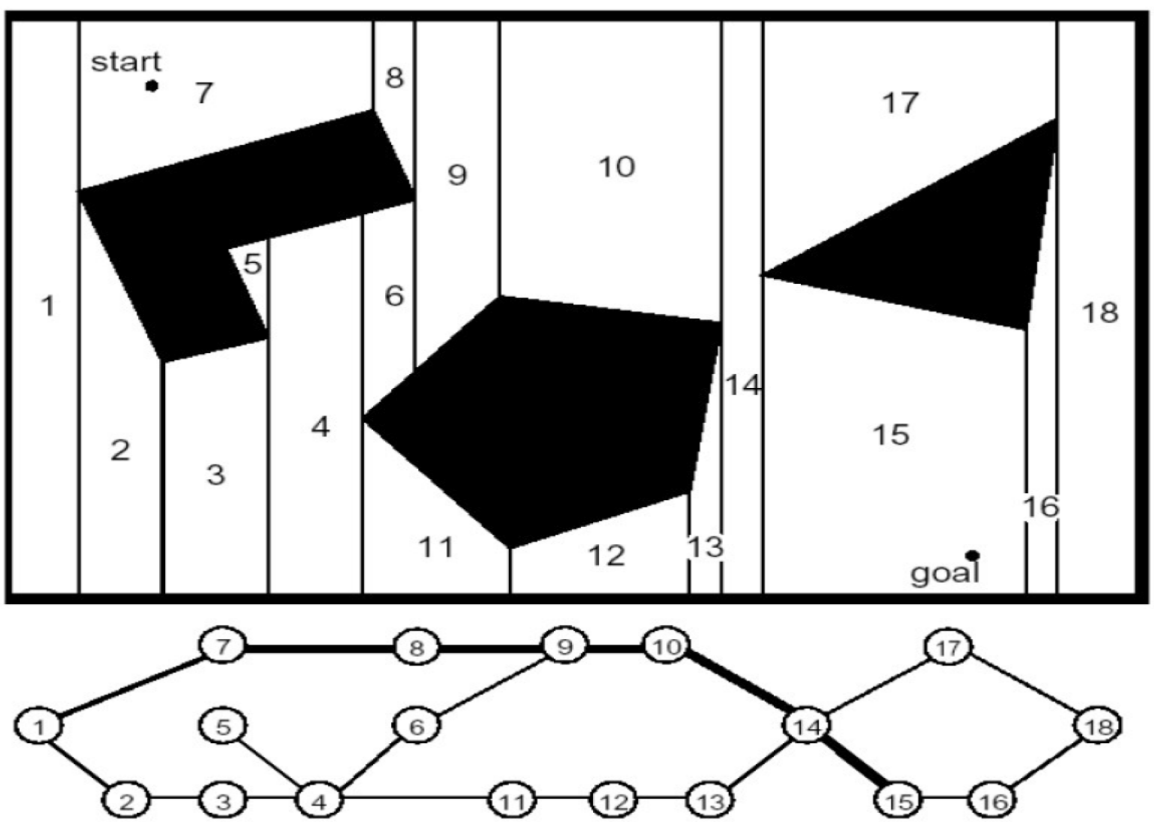
Advantages?
- Complete.
- Maximise clearance.
- Minimise accidental collision.
- Computational complexity.

Drawbacks?
- Geometrically suboptimal.
- Polygons needed.
- Differential constraints?
- Limited range sensors/
  localisation might fail.

Diagram: Siegwart &

Transform-and-search algorithms IV: Exact cell decomposition

- Graph nodes are cells.
- Do graph search (shortest path, cost=distance).



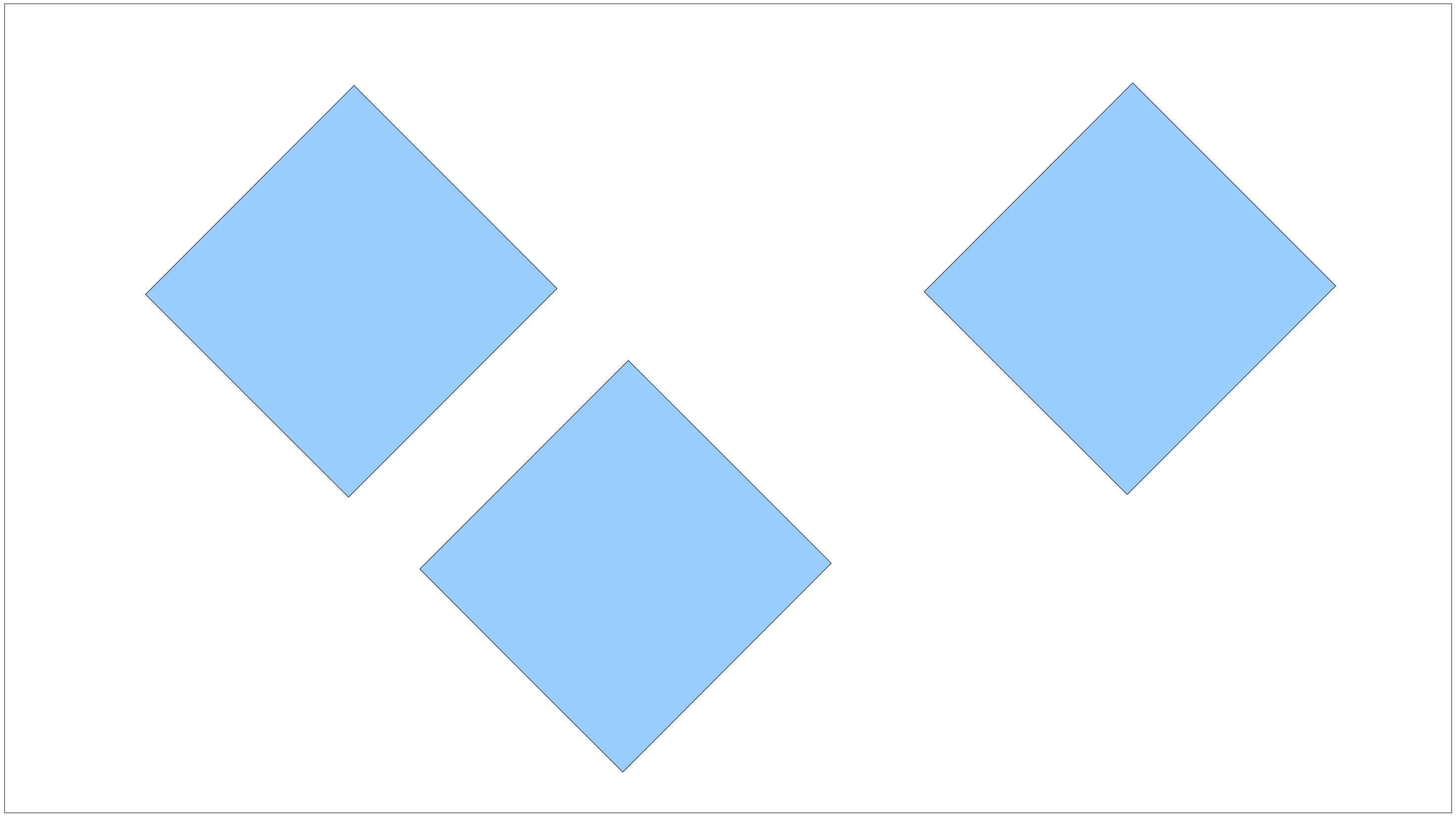
<u>Advantages</u>
- Graph is complete.

<u>Drawbacks?</u>
- Distance between nodes?
- Moving between nodes?.
- Robot shape?
- Differential constraints?

Diagram: Siegwart & Nourbakhsh

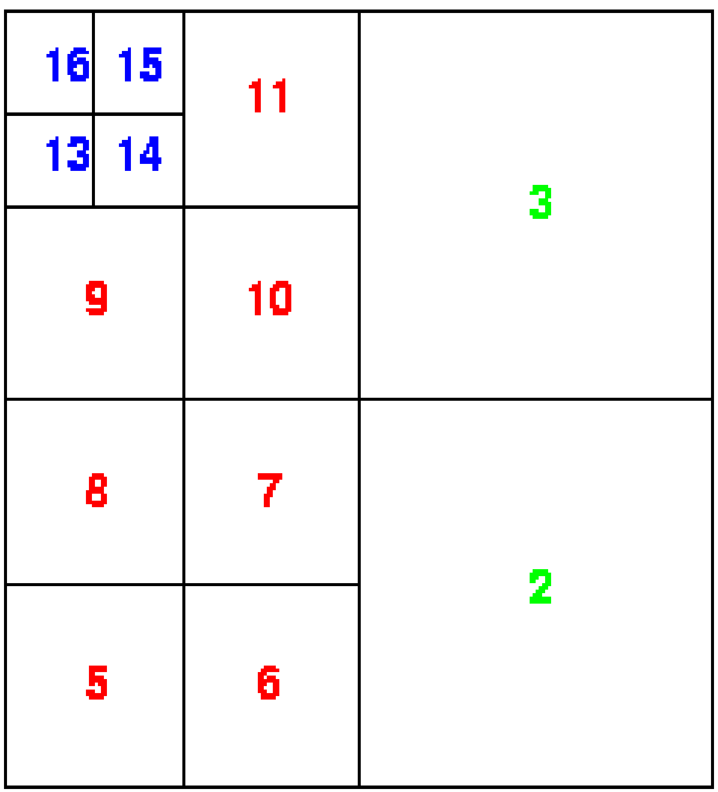
Exact cell decomposition exercise

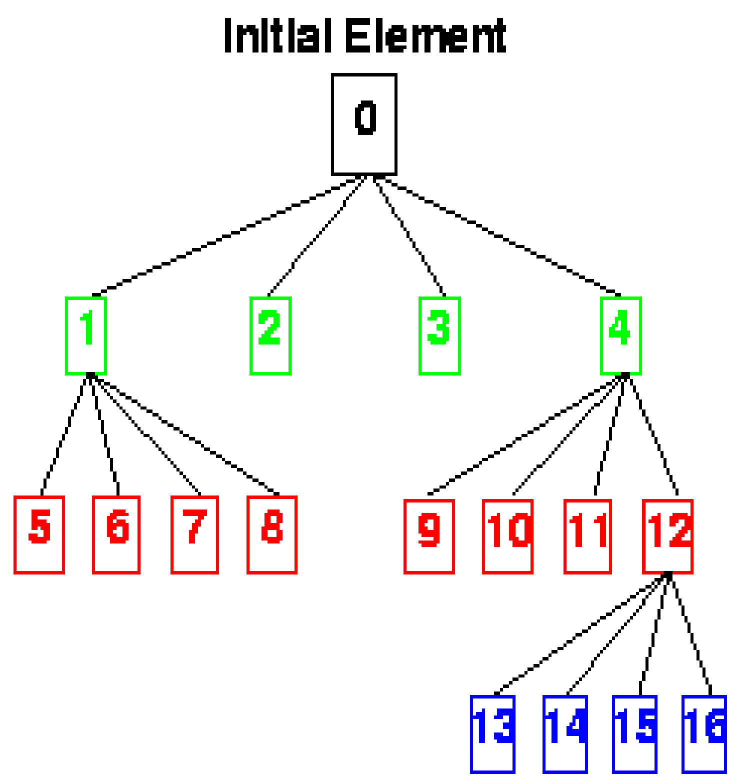
- Decompose the following map using vertical lines.



Adaptive cell decomposition preliminaries: Quad-tree

- **Quad-tree**: a data structure for efficiently indexing 2D space.

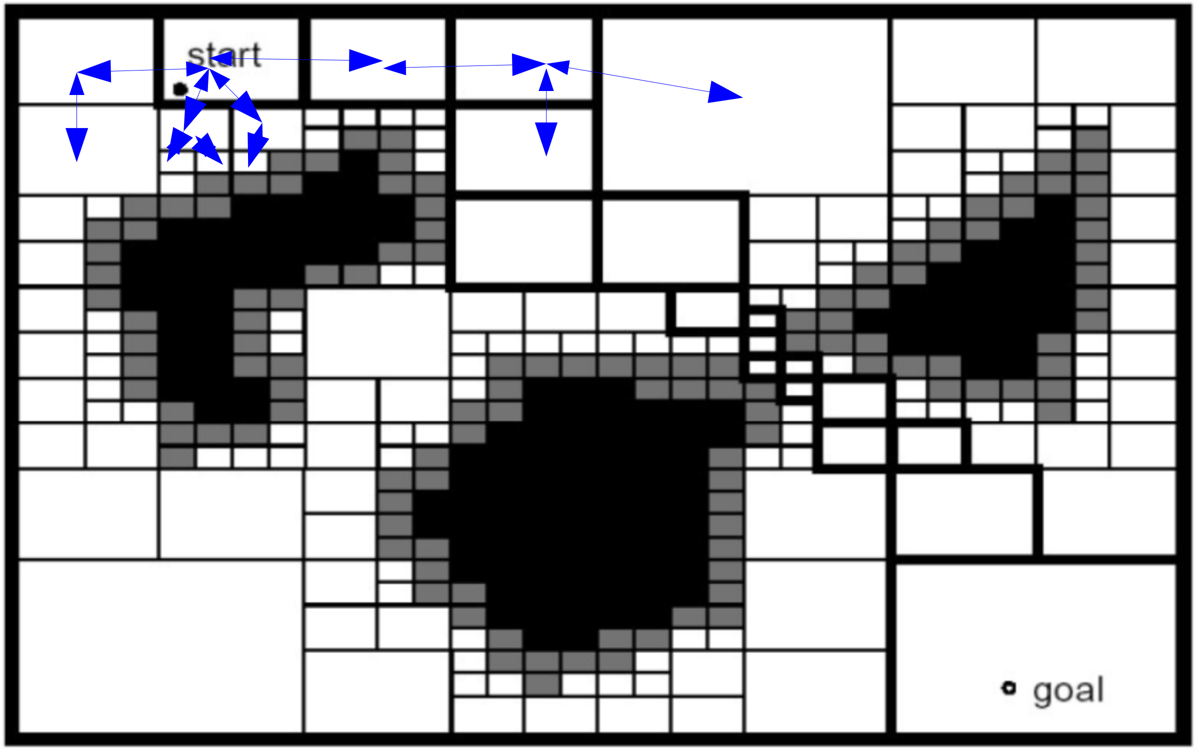




http://www.cs.sandia.gov/~kddevin/LB/quadtree.html

Transform-and-search algorithms V: Adaptive cell decomposition

- Quad-trees or oct-trees or kd-trees.
- Do graph search (shortest path, cost=distance).

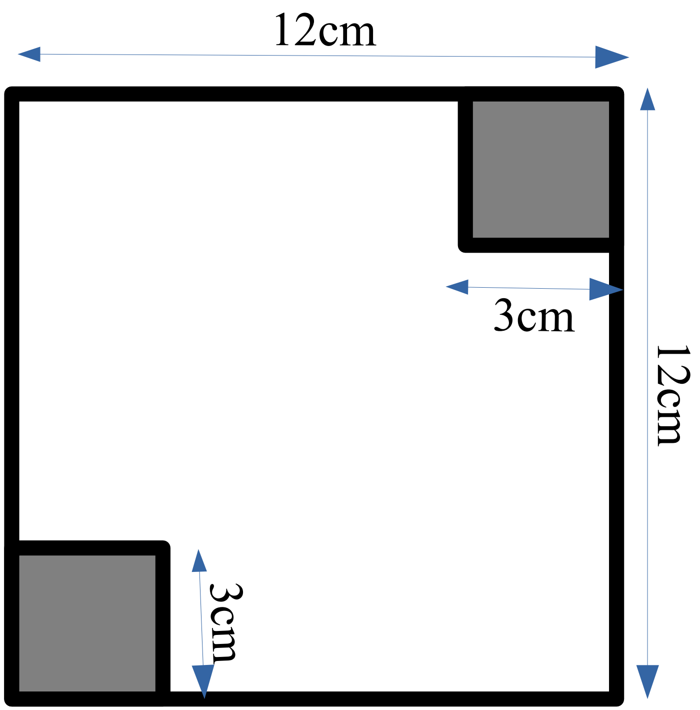


Advantages
- Efficient.
- Resolution complete.
- Higher granularity.

Drawbacks?
- Robot shape?
- Differential constraints?

Quad-tree exercise

Represent the following space with a quad-tree.



Some Variations

- Bug2 → Tangent Bug.
  - Better wall-following with laser scanner.
- Potential fields → Local minima repair.
  - Tricks for dealing with local minima.
- Visibility graph → Reduced visibility graph.
  - Reducing the size of the visibility graph.
- RRT → BiRRT.
  - Faster RRT with search from both goal and start.
- Decompositions → On-demand decomposition.
  - No transforming the full graph for search.
- ...

BLG456E
Robotics
Motion Planning

- Framework.
- Reactive approaches.
- Transform-and-search approach.
- Iterative search approach.
- Optimisation for motion planning.
- Other considerations

Lecturer:	Damien Jade Duff
Email:	djduff@itu.edu.tr
Office:	EEBF 2316
Schedule:	http://djduff.net/my-schedule
Coordination:	http://ninova.itu.edu.tr/Ders/4709

Some slides adapted from work by Dr. Sanem Sarıel-Talay

Path/trajectory optimisation

Path planning is an optimisation problem:

- Objective:
  - Minimize path-length.
  - Minimize danger (e.g. of rolling).
  - Minimize time.
- Constraints:
  - Do not collide.
  - Do not violate differential constraints.

- General optimisation may not be good for initialisation.

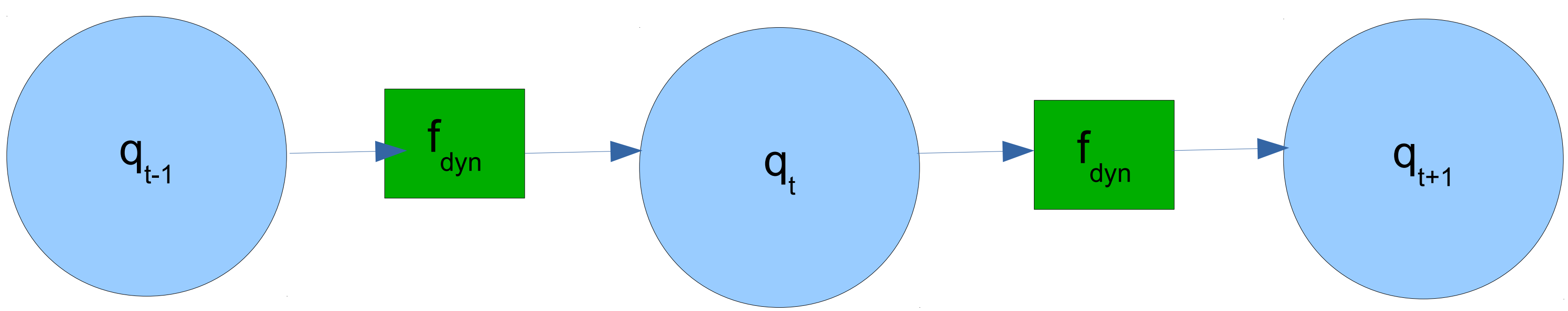
  → initialise with specialised planner (e.g. PRM).

Path/trajectory optimisation

- A trajectory is a sequence of states:

$$q_{1:T} = q_1, q_2, q_3 \ldots q_{T-2}, q_{T-1}, q_T$$

- Graphically:

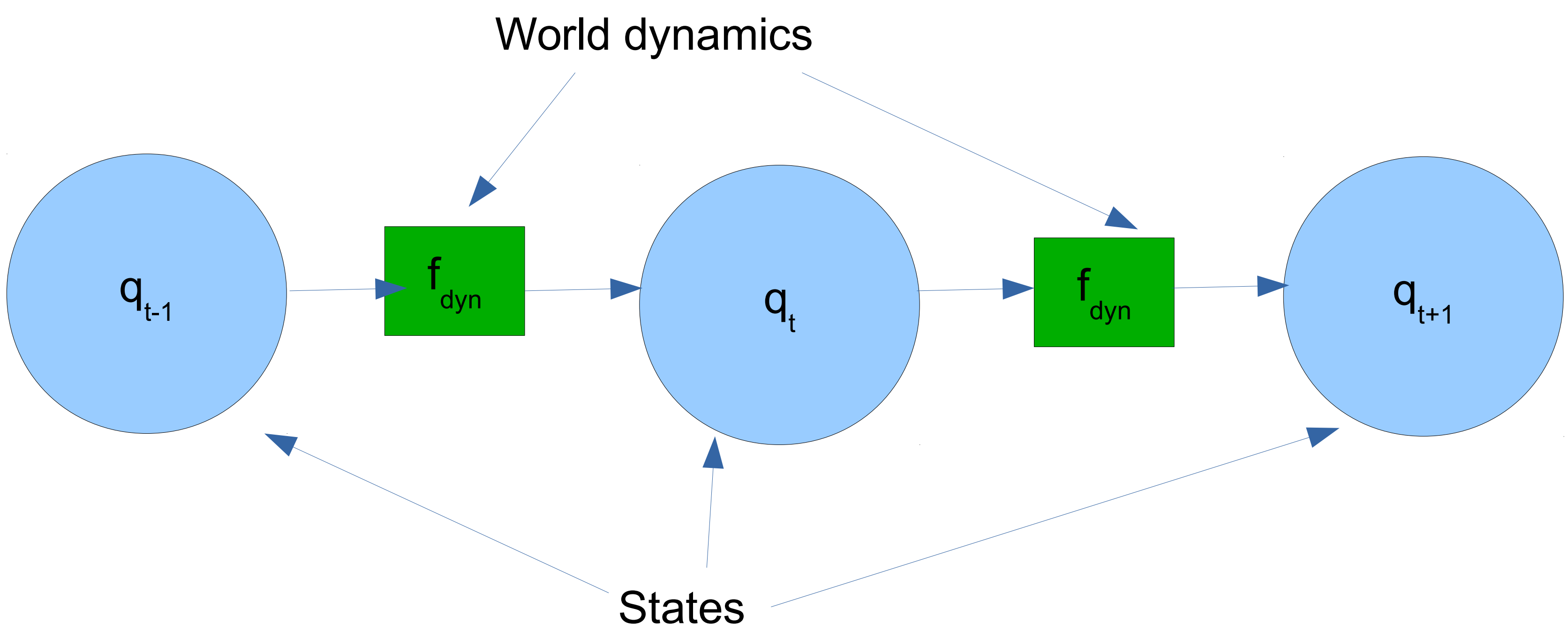


Path/trajectory optimisation

- A trajectory is a sequence of states:

$$q_{1:T} = q_1, q_2, q_3 \ldots q_{T-2}, q_{T-1}, q_T$$

- Graphically:

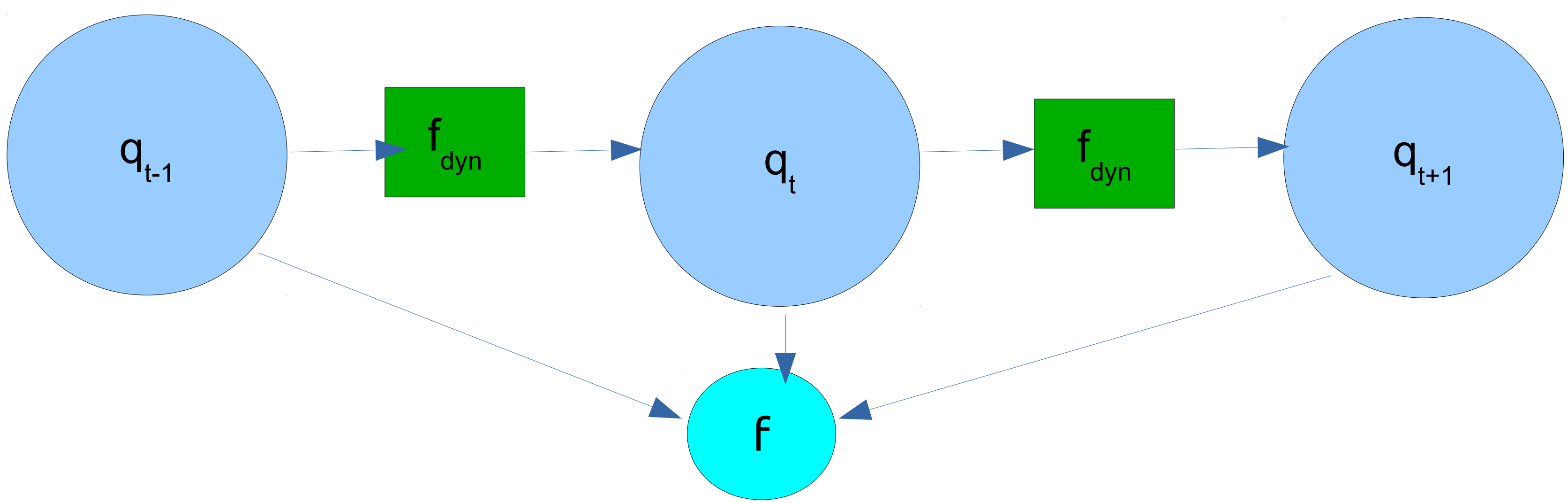


Path/trajectory optimisation

- We introduce a cost function f over trajectories. $f(q_{1:T})$

  Usually that cost function contains *task costs* and *control costs.*

$$f(q_{1:T}) = \sum_{t=0}^{T} \phi(q_t) + \sum_{t=1}^{T} \psi(q_{t-1}, q_t)$$

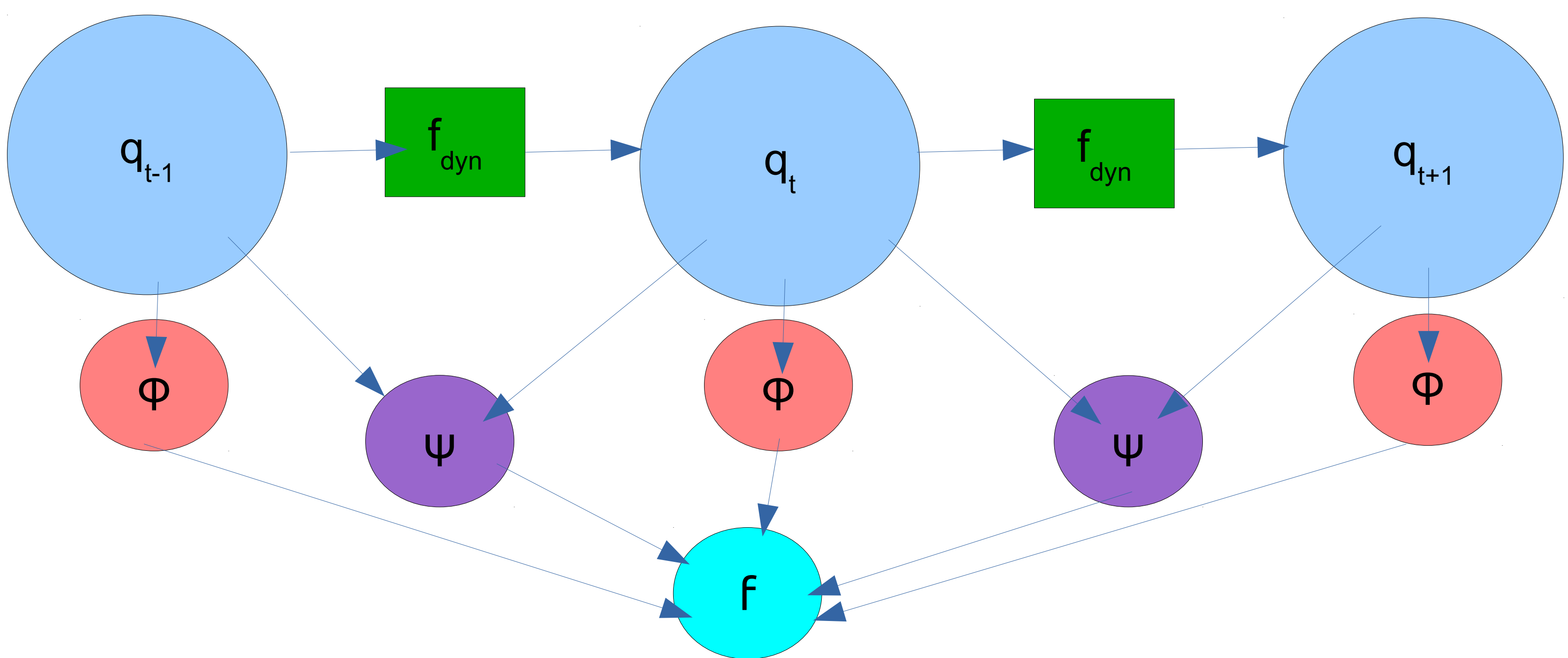


Path/trajectory optimisation

- We introduce a cost function f over trajectories.

  Usually that cost function contains *task costs* and *control costs*.

$$f(q_{1:T}) = \sum_{t=0}^{T} \phi(q_t) + \sum_{t=1}^{T} \psi(q_{t-1}, q_t)$$



Example task costs

Closeness to nearest obstacle (collision avoidance):

$$\phi_1(q_t) = \phi_1\left(\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}\right) = \frac{1}{(x_t - x_{obs})^2 + (y_t - y_{obs})^2}$$

Here $[x_{obs}, y_{obs}]$ is closest point on the nearest obstacle to q_t.

Example task costs

Path must end close to goal:

$$\phi_2(q_t) = \phi_2\left(\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}\right) = \frac{1}{\left(\left(x_t - x_G\right)^2 + \left(y_t - y_G\right)^2\right)^4}$$

Note: this produces a very "stiff" problem and could be handled better with a constraint.

Here $[x_G, y_G]$ is goal point.

Example task costs

Path must begin at start point:

$$\phi_3(q_t) = \phi_3\left(\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}\right) = \frac{1}{\left(\left(x_t - x_S\right)^2 + \left(y_t - y_S\right)^2\right)^6}$$

<u>Note</u>: this produces a very "stiff" problem and could be handled better with a constraint.

Here $[x_S, y_S]$ is start point.

Example control cost

Path should be as short as possible:

$$\psi_1(q_t, q_{t-1}) = \psi_1\left(\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}, \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix}\right) = (x_t - x_{t-1})^2 + (y_t - y_{t-1})^2$$

Note: by itself this will result in a 0-length path:
Need task costs too.

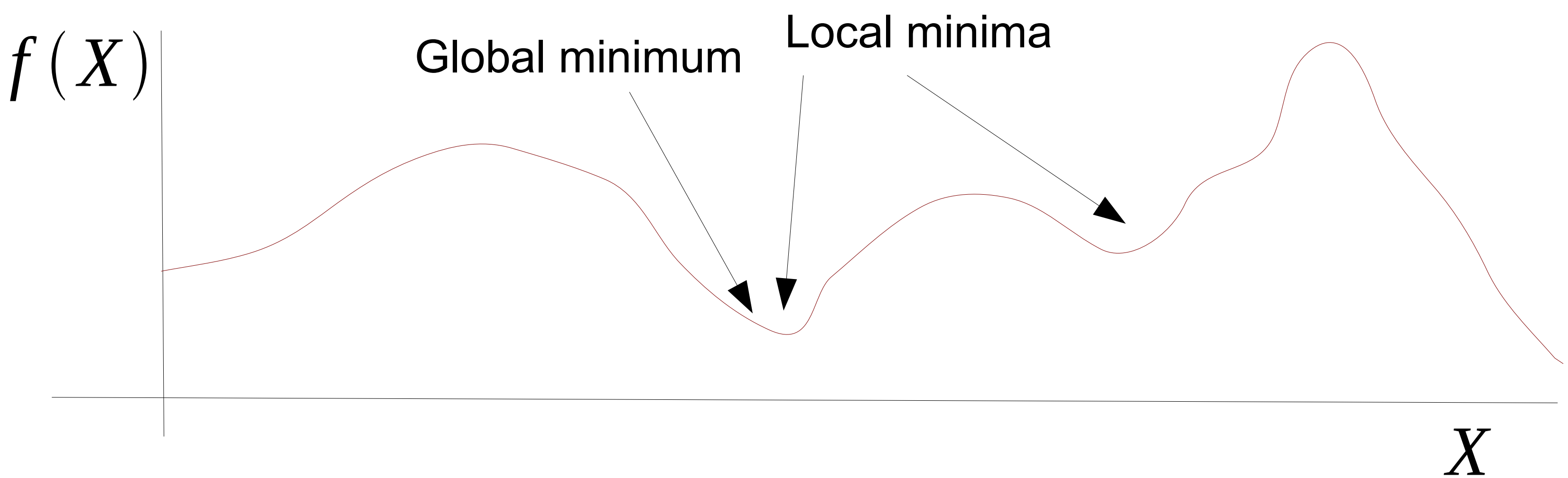
Example control cost

- Differential constraints should be observed as much as possible.

$$\psi_2(q_t, q_{t-1}) = \psi_2\left(\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}, \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix}\right) = \left|\cos(\theta_t)(x_t - x_{t-1}) + \sin(\theta_t)(y_t - y_{t-1})\right|$$

Optimisation

Once a cost function is designed, minimise it.



There are many methods to choose from, of varying quality.

- Gradient descent is one; it is generally slow.

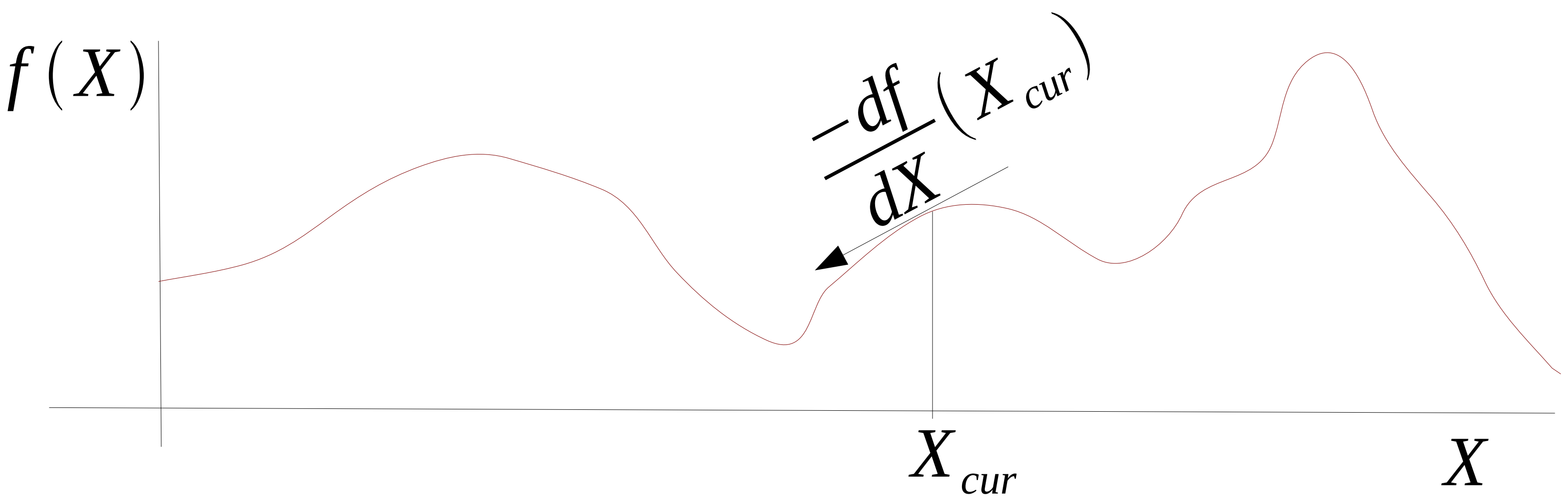
Gradient descent optimisation

Start with a guess X_{cur}

Calculate the gradient vector with respect to each variable $\frac{df}{dX}(X_{cur})$

Move in the opposite direction: $X_{new} = X_{cur} - \delta \frac{df}{dX}(X_{cur})$

(δ is the step-size constant).



Try to find *global* minima (not just local) by using random restart.

Calculating the gradient vector

Imagine I have the following cost function.

$$f(q_{0,}\cdots,q_T) = \sum_{t\in(1,T)} (x_t - x_{t-1})^2 + (y_t - y_{t-1})^2$$

Then the gradient vector is

$$\nabla f = \begin{bmatrix} \frac{df}{dx_0} \\ \frac{df}{dy_0} \\ \frac{df}{d\theta_0} \\ \vdots \\ \frac{df}{dx_T} \\ \frac{df}{dy_T} \\ \frac{df}{d\theta_T} \end{bmatrix} = \begin{bmatrix} 2(-x_1 + x_0) \\ 2(-y_1 - y_0) \\ 0 \\ \vdots \\ 2(x_T - x_{T-1}) \\ 2(y_T - y_{T-1}) \\ 0 \end{bmatrix}$$

- this is the direction to search in.

Other optimisation algorithms

- Linear programming.
- Quadratic programming.
- Genetic & Evolutionary Algorithms.
- Simulated Annealing.
- Coordinate descent.
- Newton's method.
  - Levenberg-Marquadt.
- Line-search methods.
- Trust-region methods.
- ...

Online planning

- Planning problem is always changing:
  - The environment moves.
    - e.g. when obstacles = people.
  - New things are discovered.
    - e.g. new ways, new obstacles.
- How to deal with this?
  - Contingency (feedback) planning.
  - Replanning.
  - Plan repair.
  - Continual planning.

Feedback & contingent planning

- What if no initial state given?
- Make a **policy**.
- "Under these circumstances do this, but if you see that do this other thing."

BLG456E
Robotics
Motion Planning

- Framework.
- Reactive approaches.
- Transform-and-search approach.
- Iterative search approach.
- Optimisation for motion planning.
- Other considerations

Lecturer: Damien Jade Duff
Email: djduff@itu.edu.tr
Office: EEBF 2316
Schedule: http://djduff.net/my-schedule
Coordination: http://ninova.itu.edu.tr/Ders/4709

Some slides adapted from work by Dr. Sanem Sarıel-Talay

Reading

- **Section 3.6**. Feedback control.
- **Chapter 6**. Planning & Navigation.