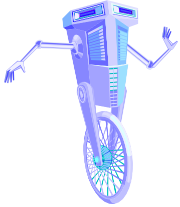


ECE497: Introduction to Mobile Robotics Lecture 5

Dr. Carlotta A. Berry

Spring 06 - 07

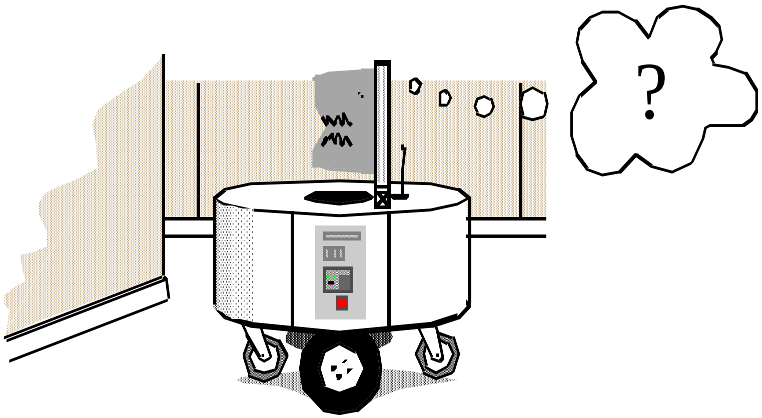
Quote of the Week

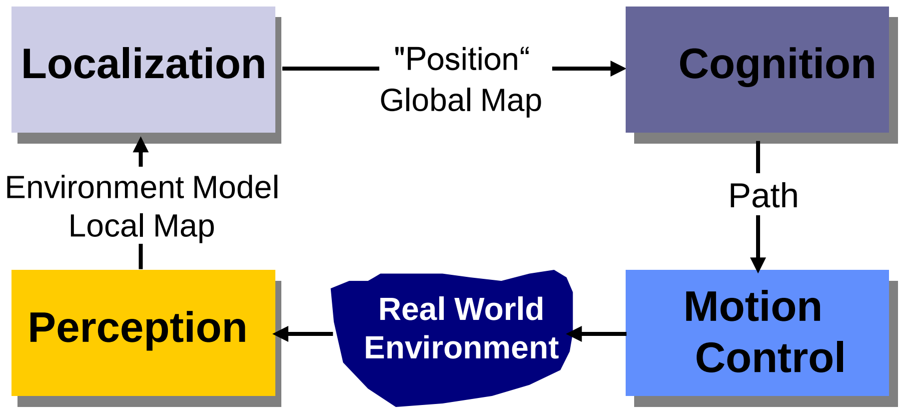
"Making realistic robots is going to polarize the market, if you will. You will have some people who love it and some people who will really be disturbed."

David Hanson, CNN.com, 11/23/06

Planning and Navigation (6.1):
Where am I going? How do I get there?

- So far we have discussed the elements of a mobile robot that are critical to robust mobility
  - *kinematics* of locomotion
  - *perception* for determining the robot's environmental context
  - *localization* with respect to the map
- the robot's *cognitive level* represents the purposeful decision-making and execution that a system utilizes to achieve its highest-order goals





Planning and Navigation (6.1):
Where am I going? How do I get there?

- In mobile robotics, the specific aspect of cognition directly linked to robust mobility is *navigation competence*
- given partial knowledge about is environment and a goal position or series of positions
  - *navigation* encompasses the ability of the robot to act based on its knowledge and sensor values so as to reach its goal positions as efficiently and as reliably as possible
- the key difference between various navigation architectures is the manner in which they decompose the problem into smaller subunits
- there are 2 competences for mobile robot navigation:
  - given a map and goal location, *path planning* involves identifying a trajectory that will cause the robot to reach the goal location when executed
  - given real-time sensor readings, *obstacle avoidance* involves modulating the trajectory of the robot in order to avoid collisions

Competencies for Navigation: Planning and Reaction (6.2)

- The navigation task for a robot
  - involves executing a course of action *(plan)* to reach its goal position
  - during execution, the robot must *react* to unforeseen events (obstacles) in such a ways to still reach the goal
- Cognition / Reasoning :
  - is the ability to decide ***what actions are required*** to achieve a ***certain goal*** in a ***given situation (belief state)***.
  - decisions ranging from ***what path to take*** to what ***information in the environment to use***.
- Today's industrial robots can operate without any cognition (reasoning) because their environment is static and very structured.
- In mobile robotics, cognition and reasoning is primarily of geometric nature, such as picking a safe path or determining where to go next.

Competencies for Navigation: Planning and Reaction (6.2)

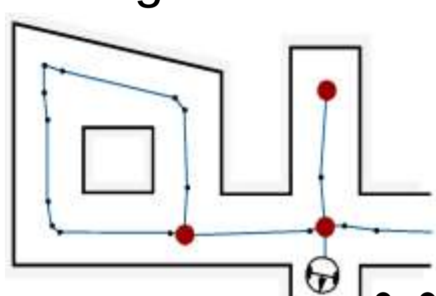
- The robot must incorporate new information gained during plan execution
- The planner must also incorporate the new information as it is received in order to correct a planned trajectory
- When a planner incorporates every new piece of information in real time, instantly producing a new plan that reacts to the new information and the concept of planning and reacting merge is called *integrated planning and execution*
- Robot control can usually be decomposed in various behaviors or functions
  - e.g. wall following, localization, path generation or obstacle avoidance.
- You can generally distinguish between (*global*) path planning and (*local*) obstacle avoidance.

Competencies for Navigation: Global Path Planning (6.2.1)

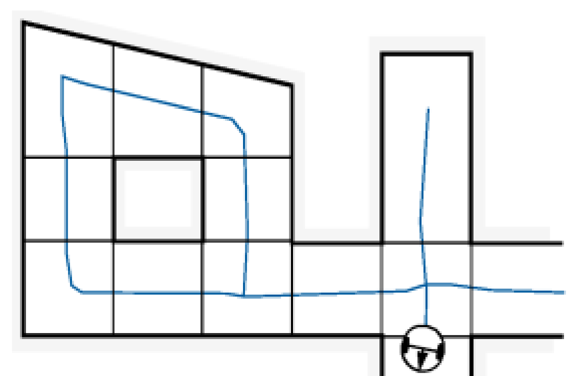
- The robot's environment representation can range from a continuous geometric description to a decomposition-based geometric map or a topological map
- Assumption: there exists a good enough map of the environment for navigation.
- Three general strategies for decomposition
  - *road map* - identify a set of routes within the free space
  - *cell decomposition* – discriminate between free and occupied cells
  - *potential field* – impose a mathematical function over the space

Competencies for Navigation:
Global Path Planning (6.2.1)

- *Road Map, Graph construction*
  - Identify a set of routes within the free space
  - Where to put the nodes?
  - Topology-based:
    - at distinctive locations
  - Metric-based:
    - where features disappear or get visible



- *Cell decomposition*
  - Discriminate between free and occupied cells
  - Where to put the cell boundaries?
  - Topology- and metric-based:
    - where features disappear or get visible

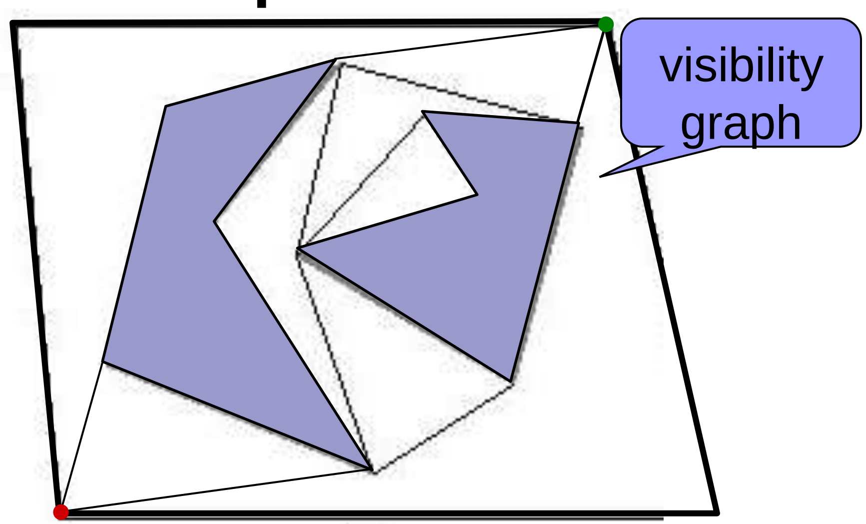


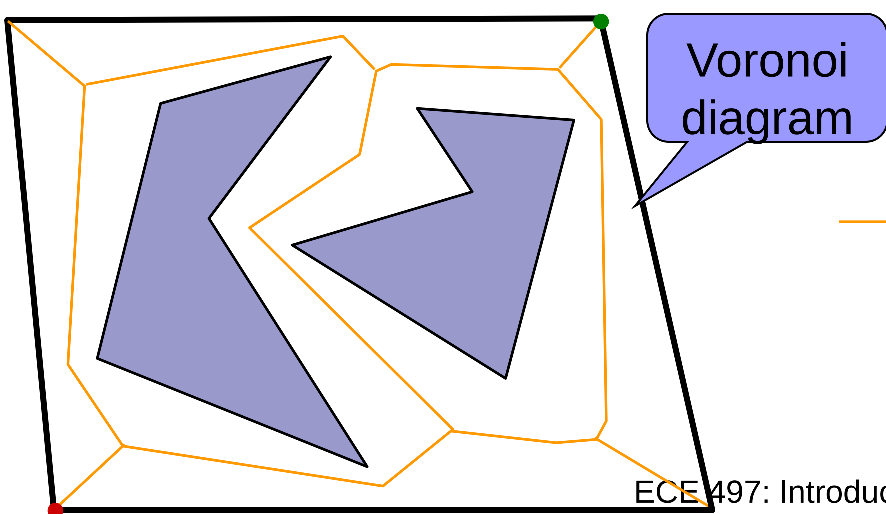
Competencies for Navigation: Road map path planning (6.2.1.1)

- Road maps capture the connectivity of the robot's free space in a network of 1D curves or lines, or *road maps*
- Path planning is reduced to connecting the initial and goals position of the robot to the road network
- the road map is a decomposition of the robot's configuration space based on obstacle geometry
  - *visibility graph* – roads come as close as possible to obstacles and resulting paths are minimum-length solutions
  - *Voronoi diagram* – roads stay as far away as possible from obstacles

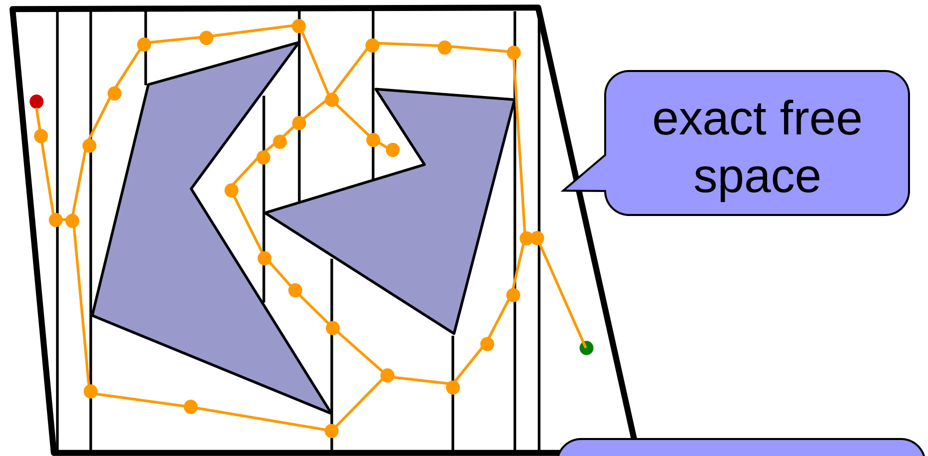
Full-knowledge motion planning

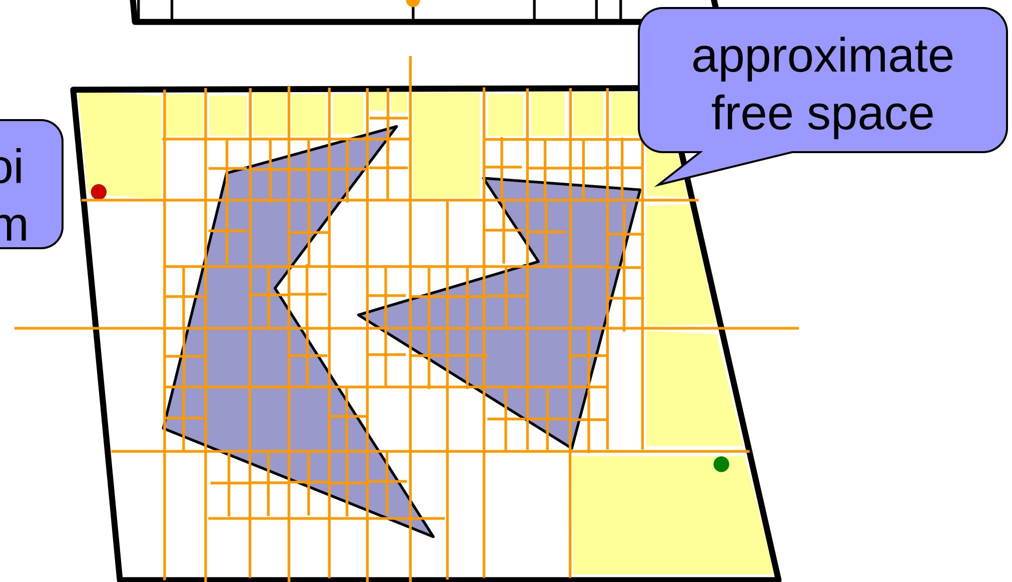
Roadmaps



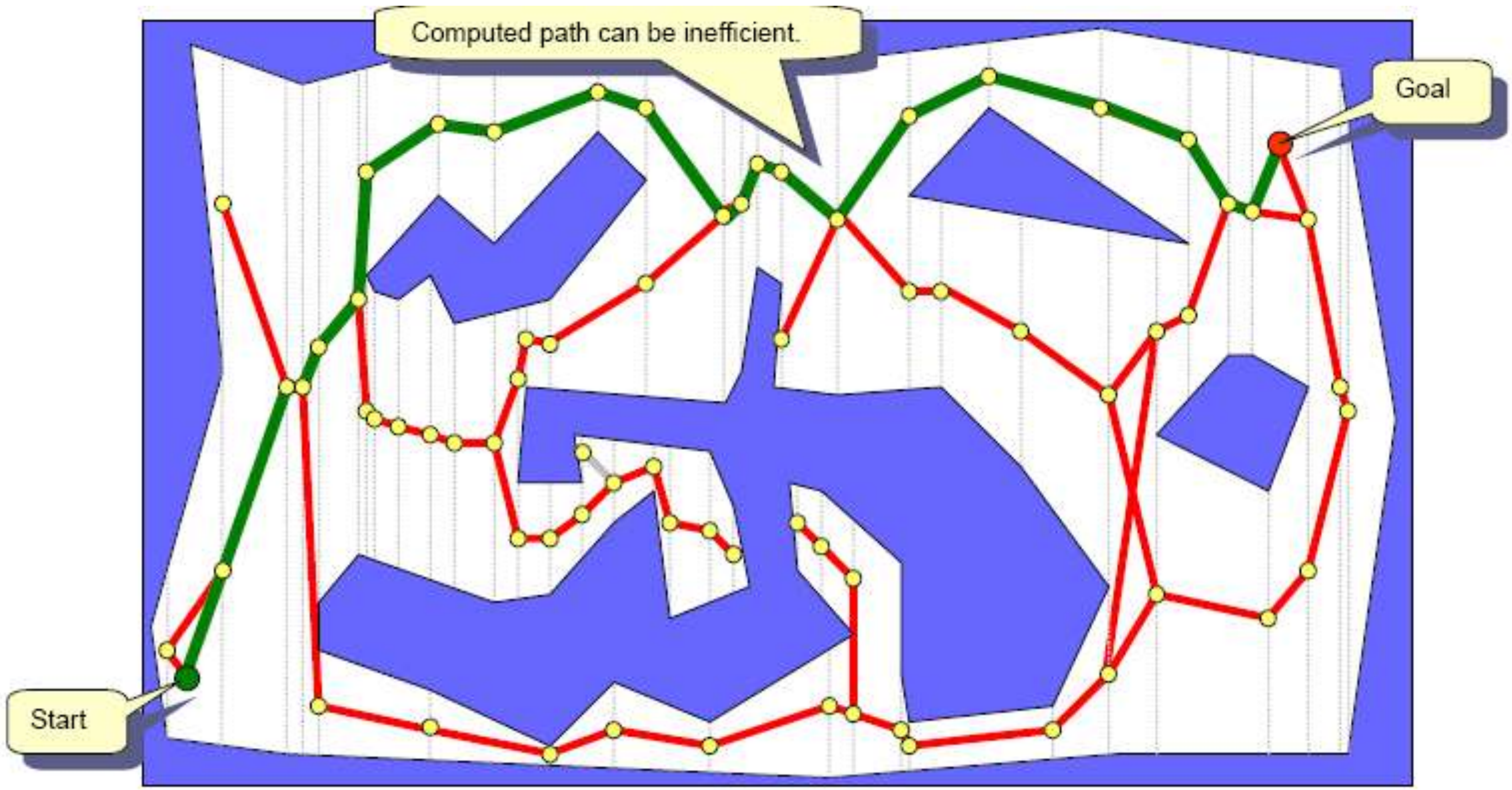


Cell decompositions

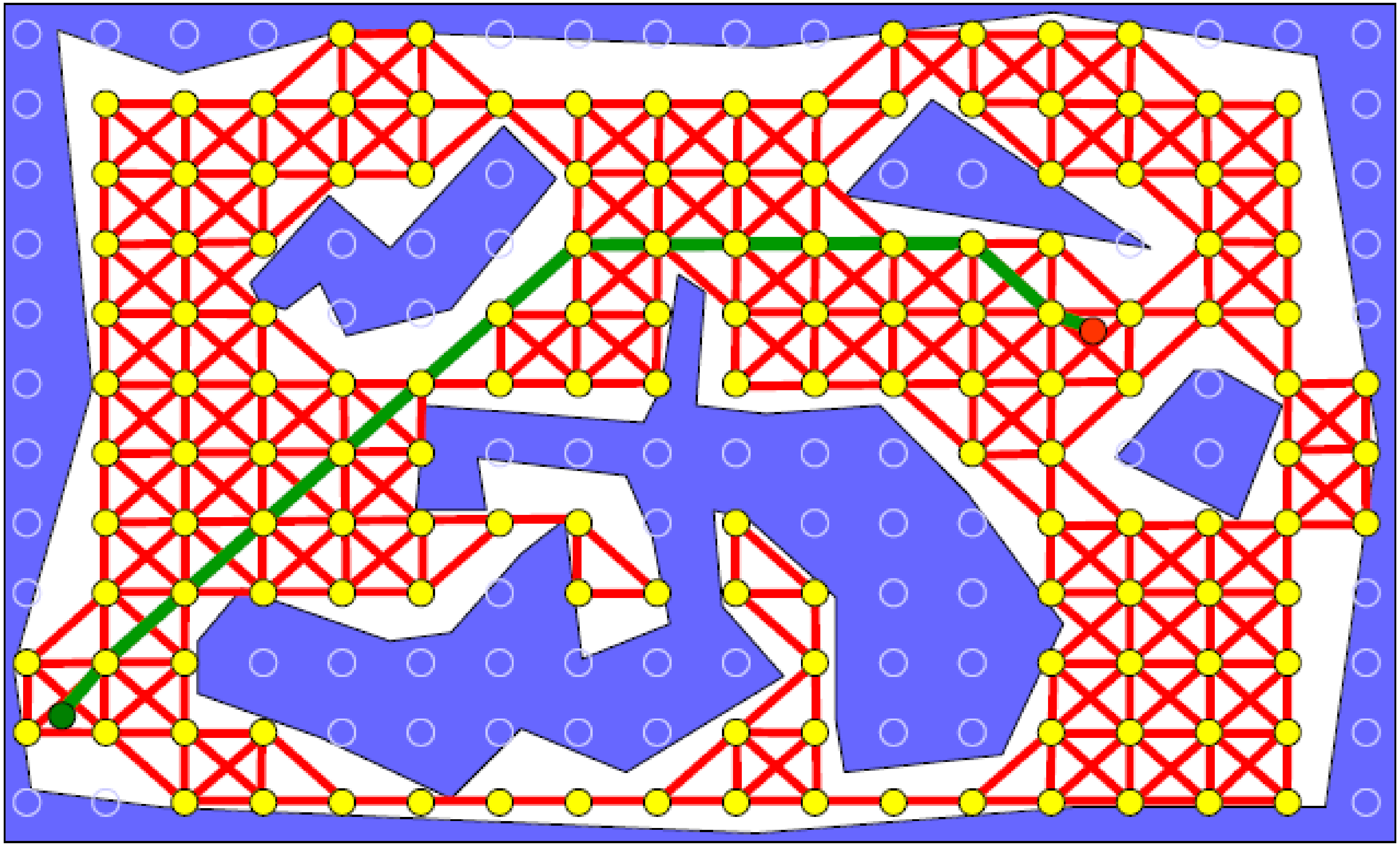




Road map-based path planning

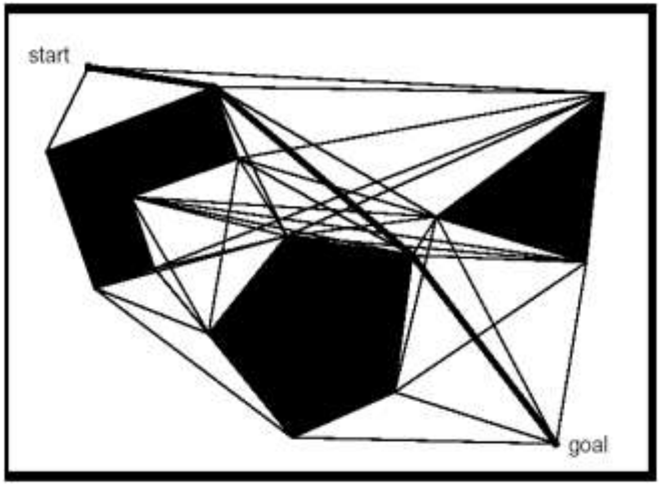


Roadmap-based path planning



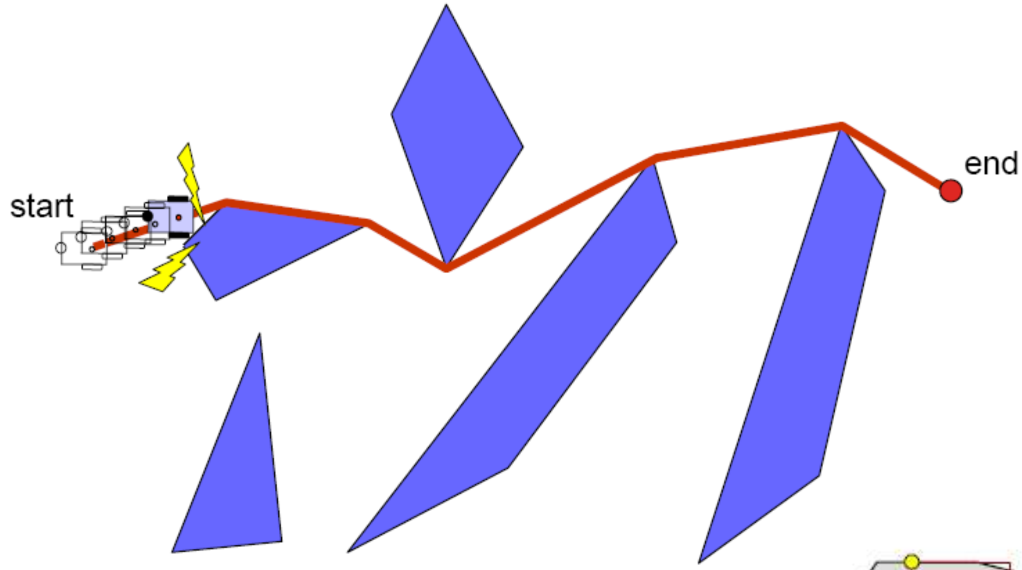
Competencies for Navigation:

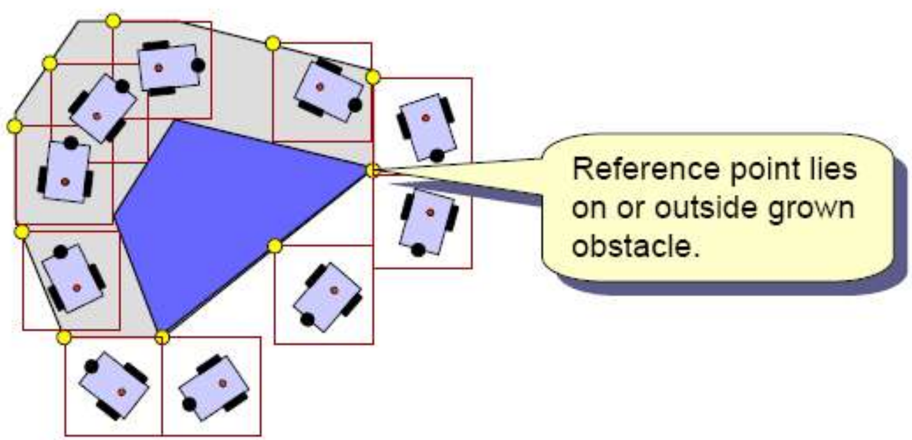
Visibility graph (6.2.1.1)



- the *visibility graph* consists of all edges joining vertices that can see each other
- objects in the environment are polygons in either discrete or continuous space
- the size of the representation and the number of edges and nodes increase with the number of polygons
- paths take the robot as close as possible to obstacles on the way to the goal
- the length of the solution path is *optimal*
- sense of safety from obstacles is sacrificed for this optimality
- one solution is to grow obstacles by the robot's radius or modify the solution path

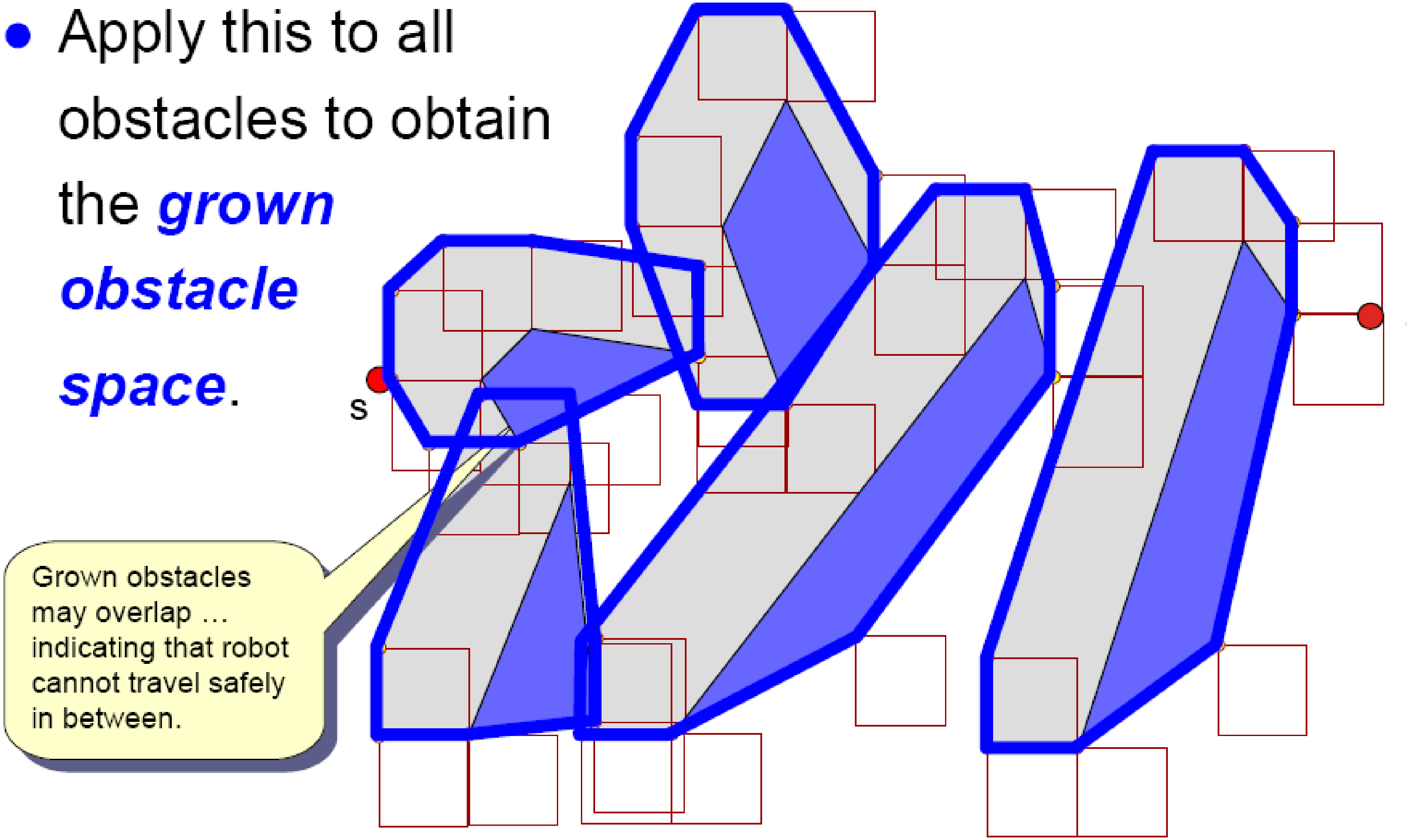
Visibility Graph Paths



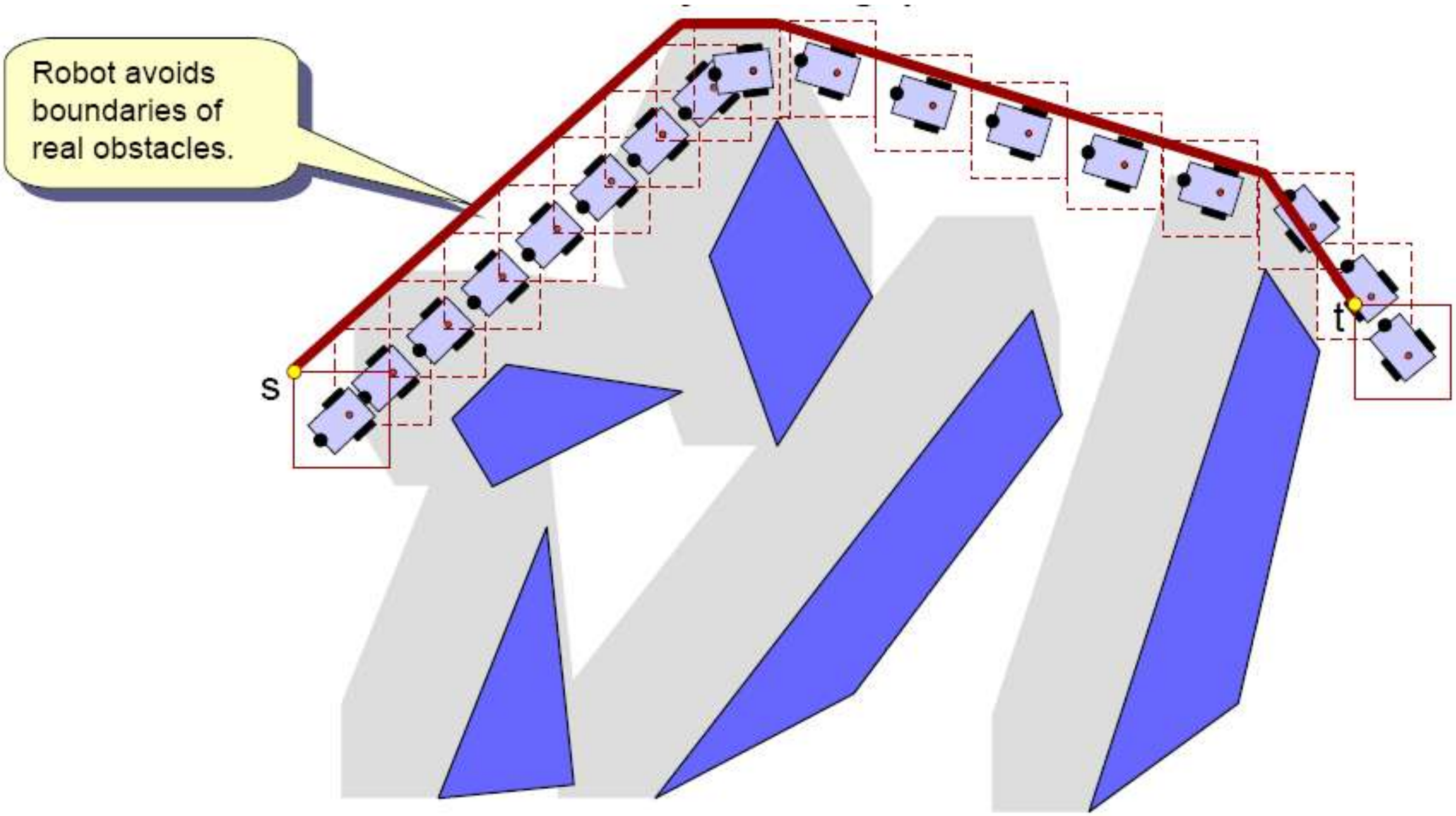


Visibility Graph Paths

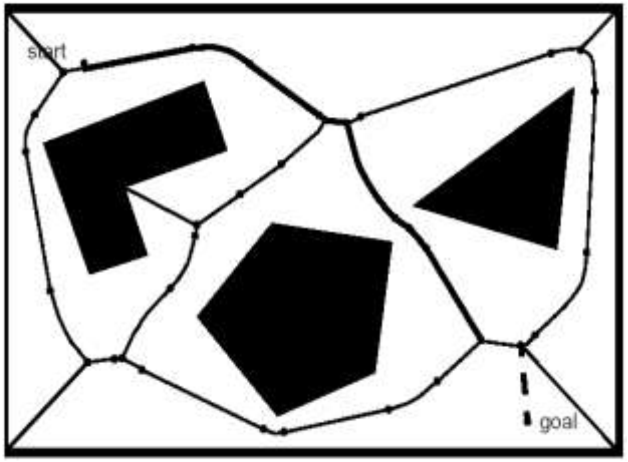
- Apply this to all obstacles to obtain the ***grown obstacle space***.



Visibility Graph Paths

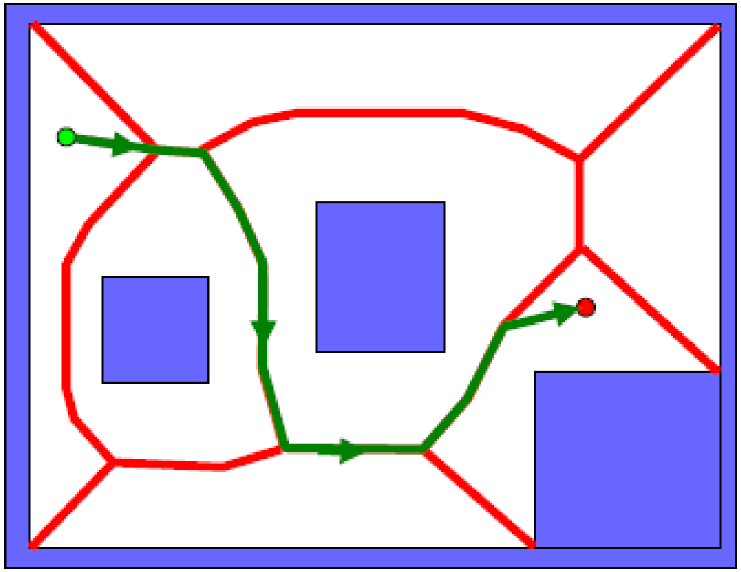


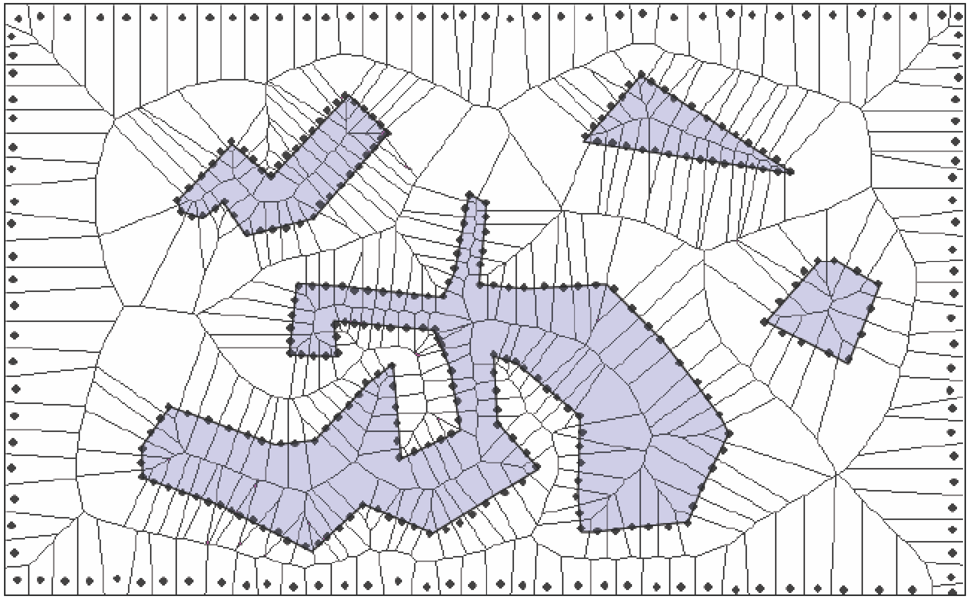
Competencies for Navigation: Voronoi diagram (6.2.1.1)



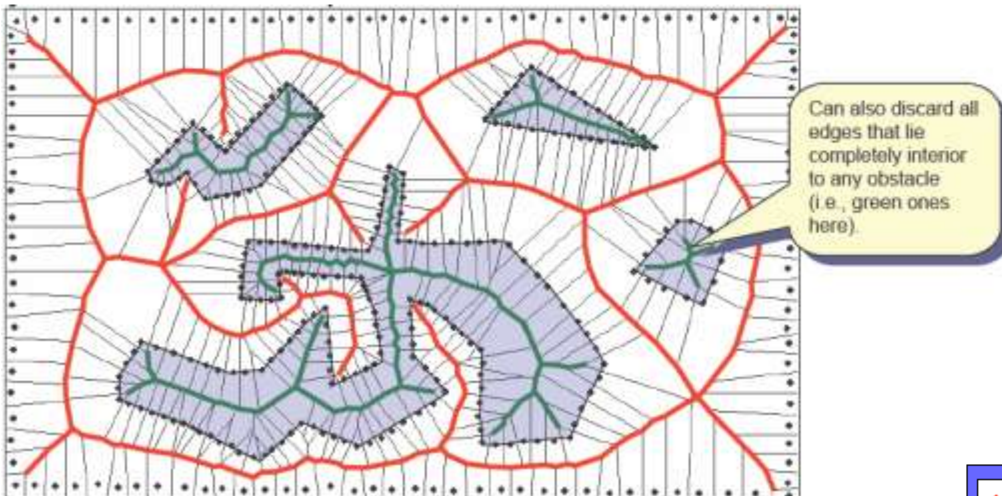
- a *Voronoi diagram* is a complete road map method that tends to maximize the distance between the robot and obstacles
- paths on the Voronoi diagram are usually far form optimal in the sense of the total path length
- one important weakness is in the case of limited range localization sensors. these sensors will be in danger of sensing its surroundings
- the *Voronoi diagram* has the advantage in *executability*
- the robot maximizes the readings of the local minima in its sensor values
- this has been used to conduct automatic mapping by finding and moving on unknown Voronoi edges and then constructing a consistent Voronoi map of the environment

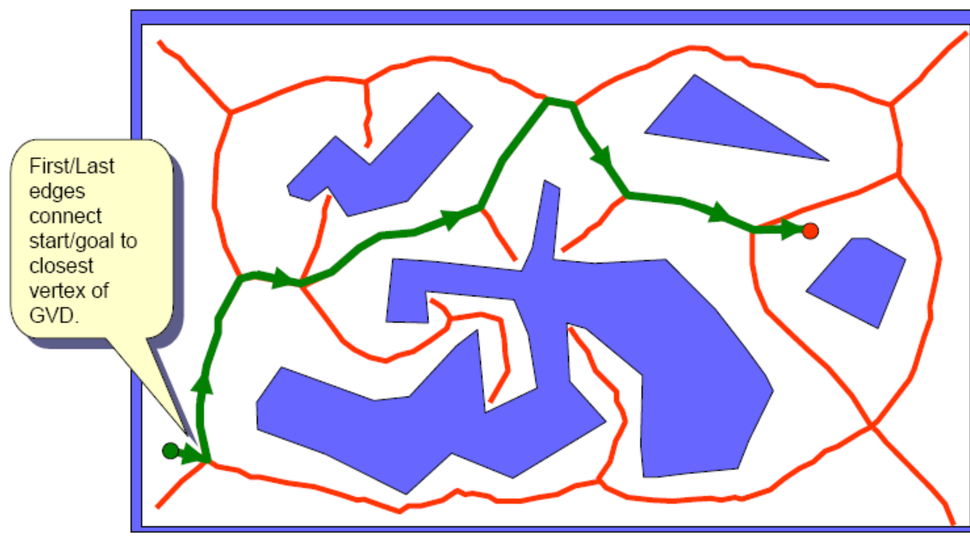
Voronoi Diagram



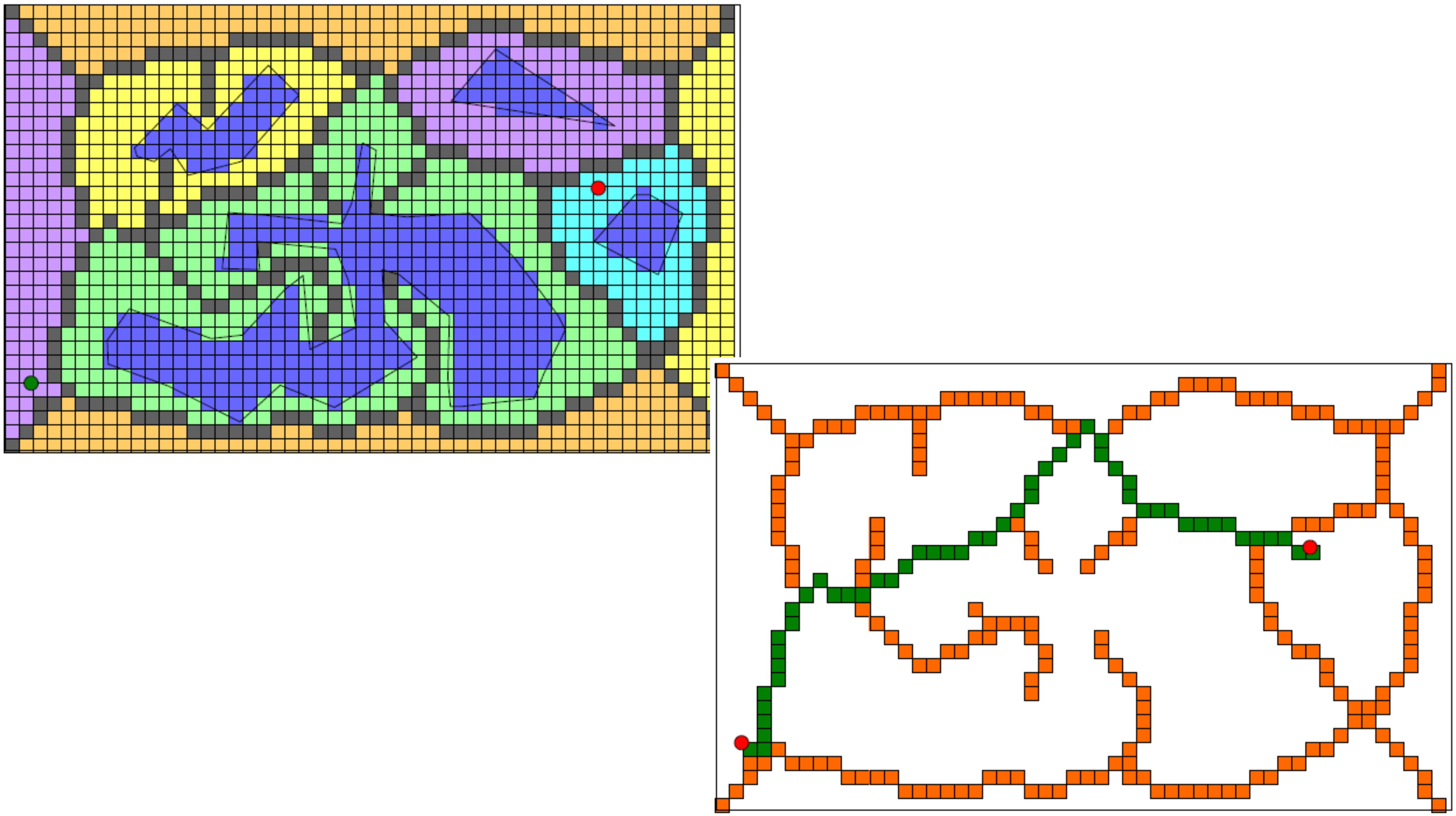


Voronoi Diagram





Discretized Voronoi Diagram

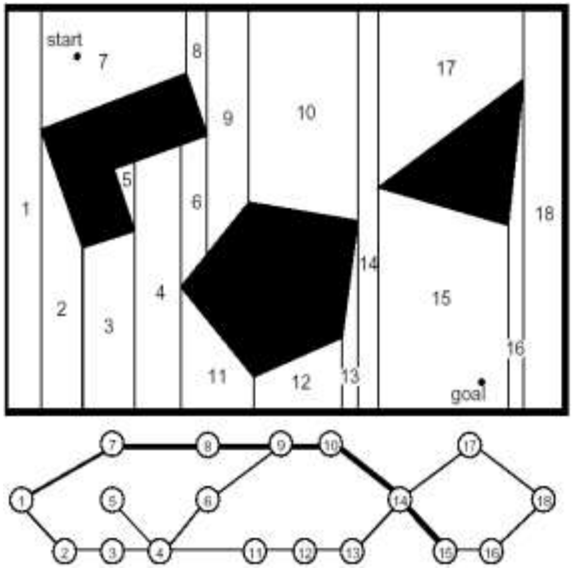


Competencies for Navigation: Cell decomposition path planning (6.2.1.2)

- Use cell decomposition to discriminate between geometric areas, or cells that are free and those that are occupied by objects
- Divide space into simple, connected regions called *cells*
- Determine which open cells are adjacent and construct a *connectivity graph*
- Find cells in which the initial and goal configuration (state) lie and search for a path in the connectivity graph to join them.
- From the sequence of cells found with an appropriate search algorithm, compute a path within each cell.
  - e.g. passing through the midpoints of cell boundaries or by sequence of wall following movements.

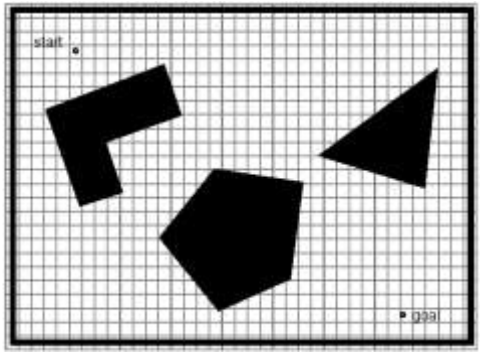
Competencies for Navigation:
Cell decomposition path planning (6.2.1.2)

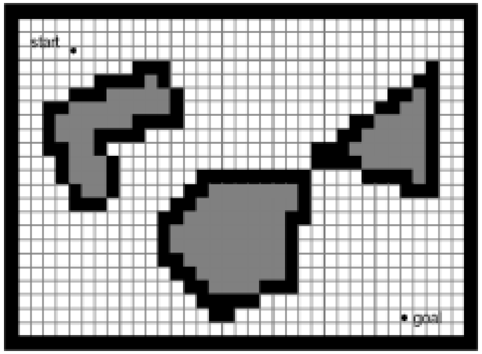
- An important aspect of *cell decomposition* is the placement of the boundaries between the cells
- if the boundaries are placed as a function of the structure of the environment then the method is *exact cell decomposition*
- if the decomposition is an approximation of the actual map, the system is an *approximate cell decomposition*



Competencies for Navigation:
Exact cell decomposition (6.2.1.2)

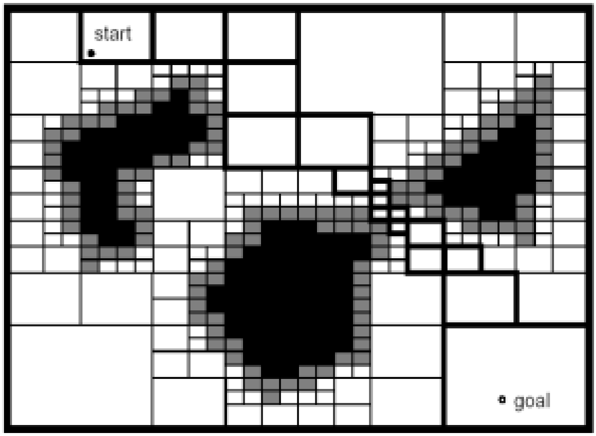
- the boundaries of cells is based on geometric criticality
- the cells are completely free or occupied
- what matters is the robot's ability to traverse from each free cell to adjacent free cells
- efficient computation in that case of large, sparse environment
- used rarely in mobile robot applications due to complexities in implementation





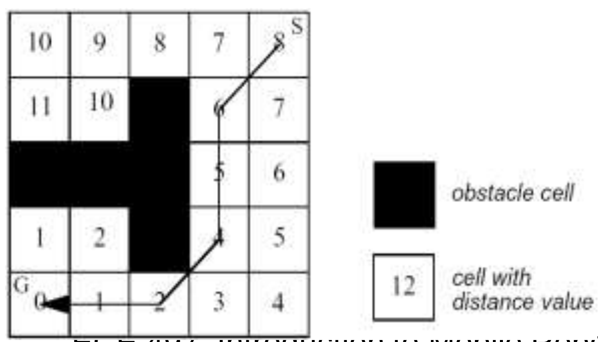
Competencies for Navigation:
Adaptive cell decomposition (6.2.1.2)

- one of the most popular techniques for mobile robot path planning
- cell size is not dependent upon objects in an environment so narrow passageways may be lost
- low computational complexity for path planning
- the fundamental cost is memory because the grid must be represented in entirety
- sparse environments contain few cells consuming dramatically less memory

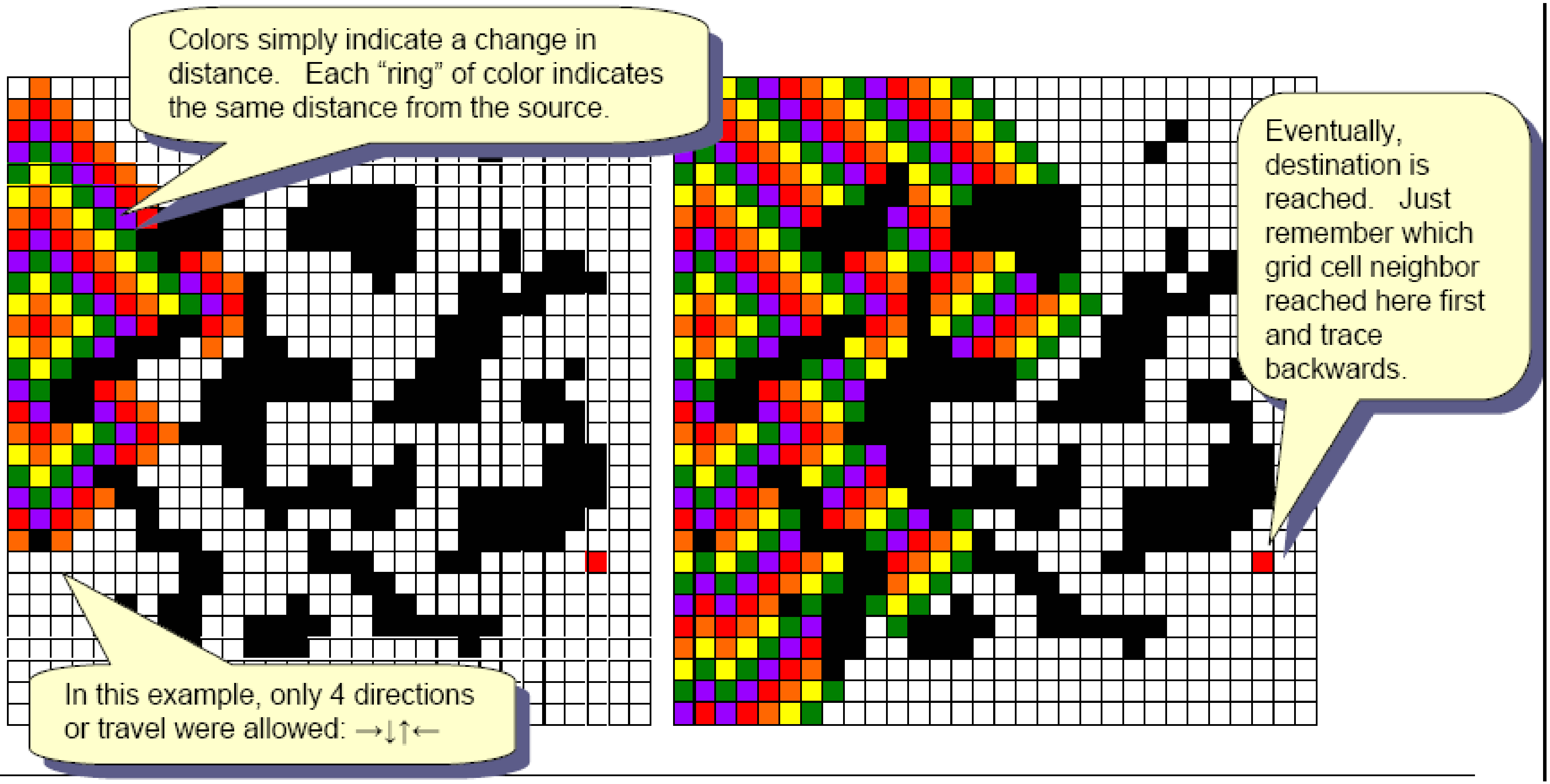


Competencies for Navigation: Approximate cell decomposition (6.2.1.2)

- *Wavefront expansion* or *grassfire* is an efficient and simple to implement technique for finding routes in fixed-size cell arrays
- employs wavefront expansion from the goal position outward, marking each cell's distance to the goal
- this continues until the wave reaches the initial position
- the planner can then estimate the robot's distance to the goal as well as recover a specific solution trajectory by linking together adjacent cells that are always closer to the goal



Wavefront propagation

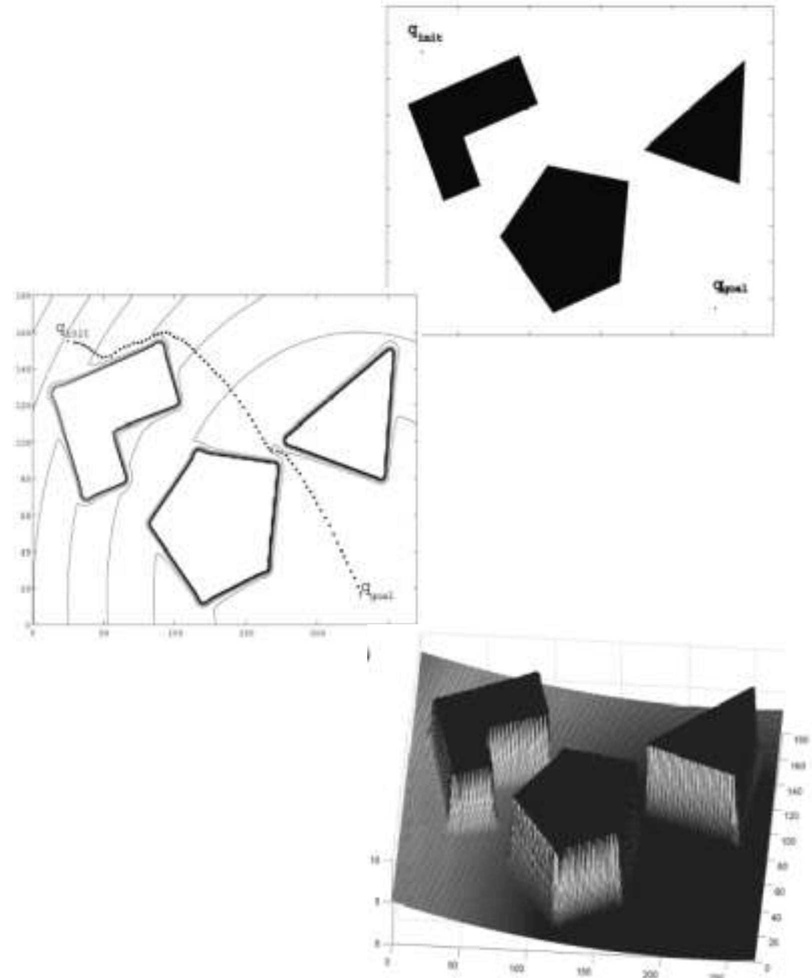


Competencies for Navigation: Potential field path planning (6.2.1.3)

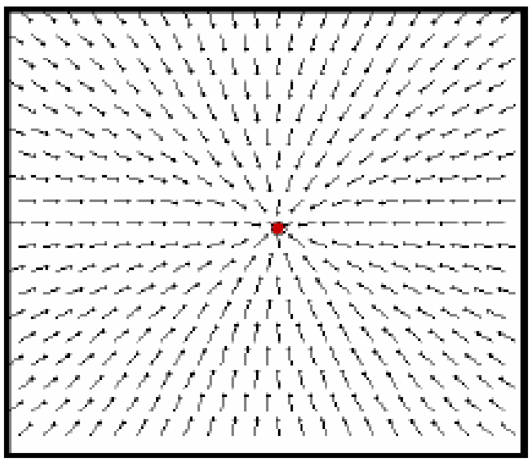
- *Potential field path planning* creates a field, or gradient, across the robot's map that directs the robot to the goal position from multiple prior positions
- Robot is treated as a *point under the influence* of an artificial potential field.
  - Generated robot movement is similar to a ball rolling down the hill
  - Goal generates attractive force
  - Obstacles are repulsive forces
  - the superposition of all forces is applied to the robot
  - artificial potential field smoothly guides the robot toward the goal while simultaneously avoiding obstacles

Competencies for Navigation:
Potential field path planning (6.2.1.3)

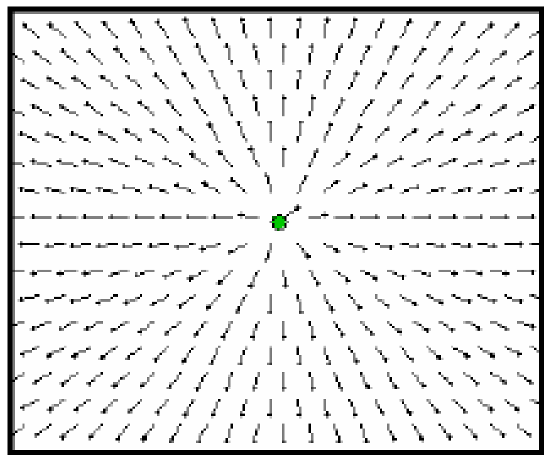
- The resulting potential field is not just for path planning but also a control law for the robot
- assuming the robot can localize itself, it can always determine the next required action based on the field
- the potential field is 2D and as new obstacles appear, the field is updated
- the repulsive potential generates a stronger force when the robot is closer to the object
- under ideal conditions, the robot's velocity vector is proportional to the field force vector
- there is a local minima problem dependent upon the obstacle shape and size



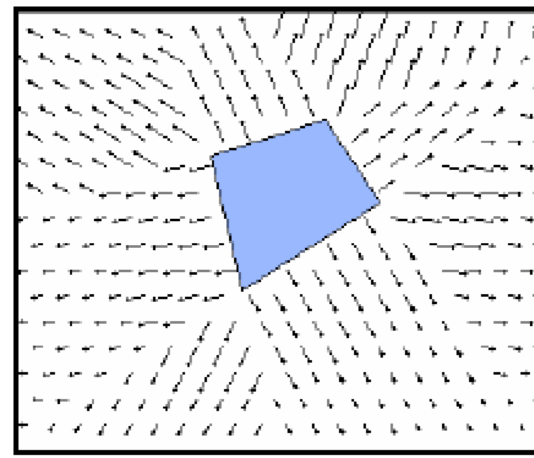
Competencies for Navigation:
Potential field path planning (6.2.1.3)



Attract to goal

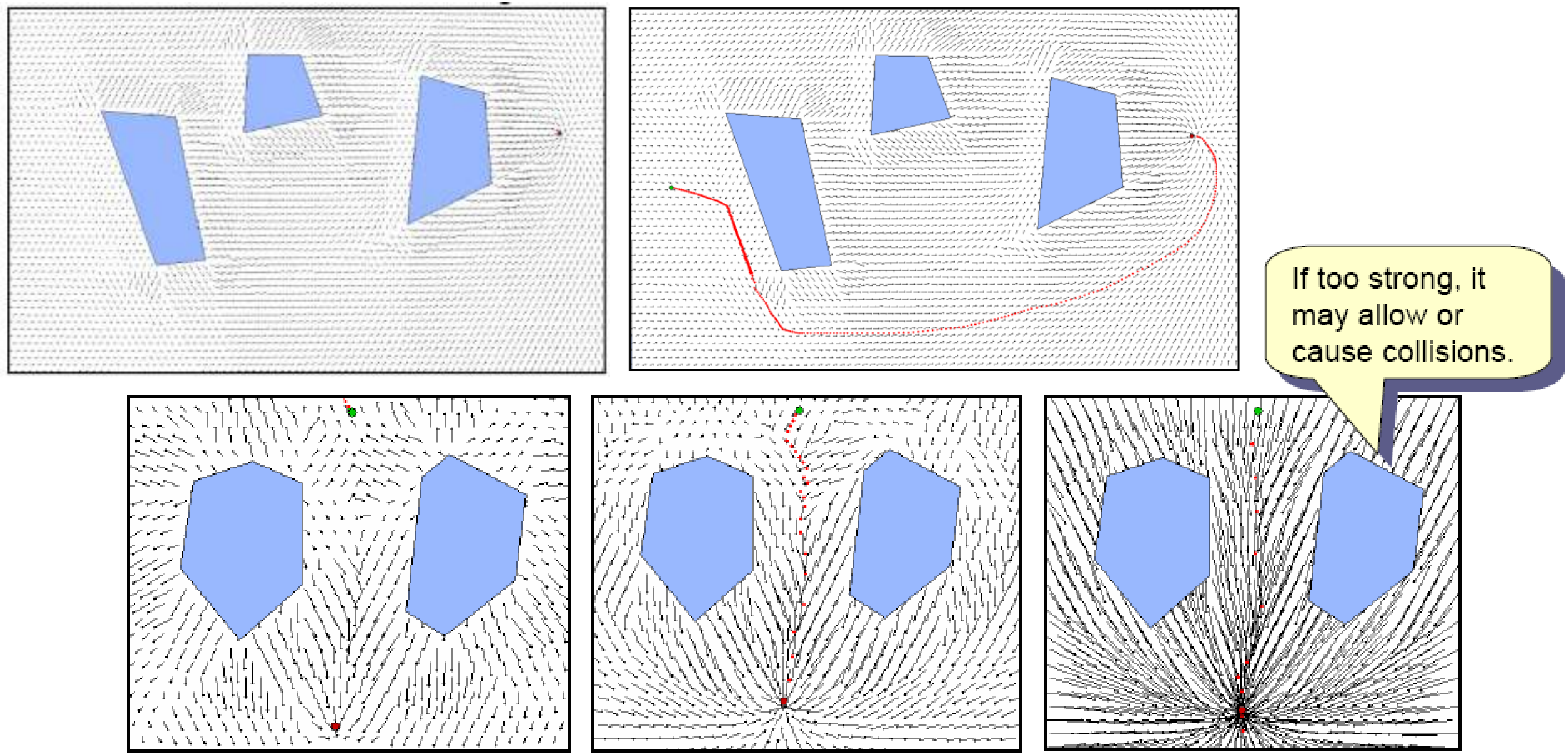


Repel from source



Repel from obstacle

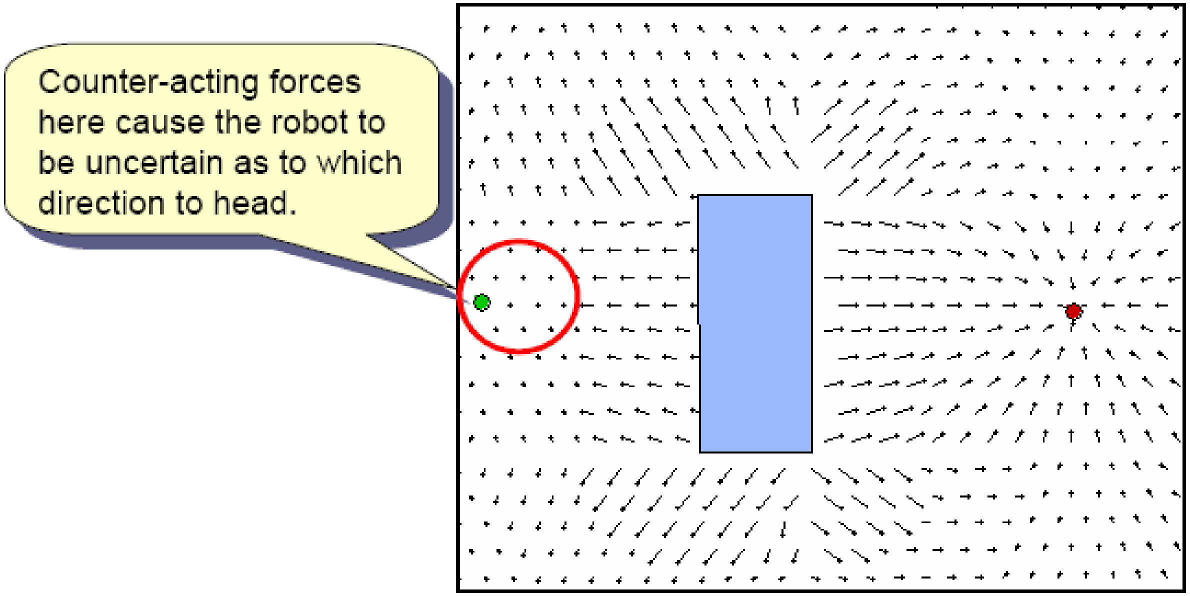
Competencies for Navigation:
Potential field path planning (6.2.1.3)



weak goal attraction | **medium** goal attraction | **strong** goal attraction

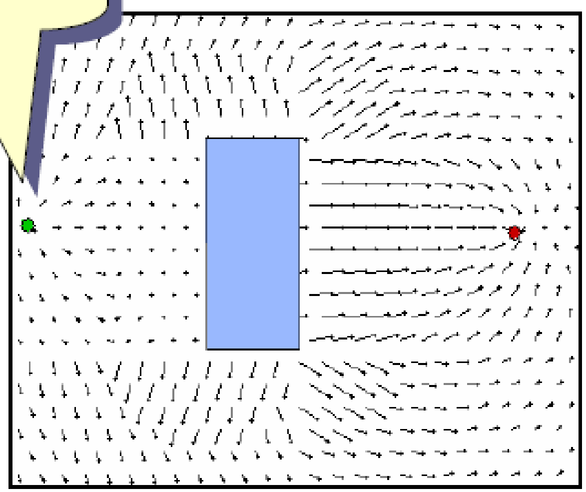
Local Minima

- In some cases, there may be a local minimum problem where the robot gets stuck due to counter-acting forces:

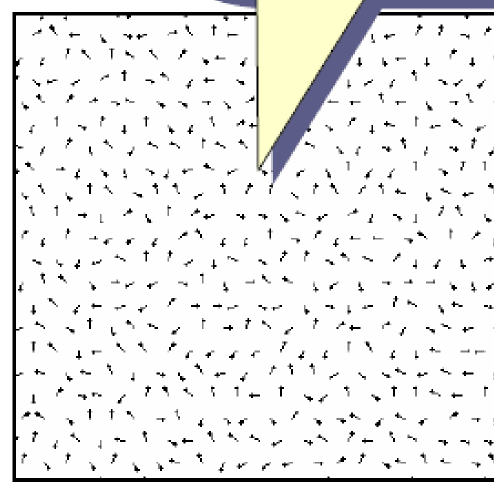


Local Minima

- To overcome such local minima problems by:
  - adding field from source (i.e., to "push" away from it)
  - introduce ***noise*** into the environment



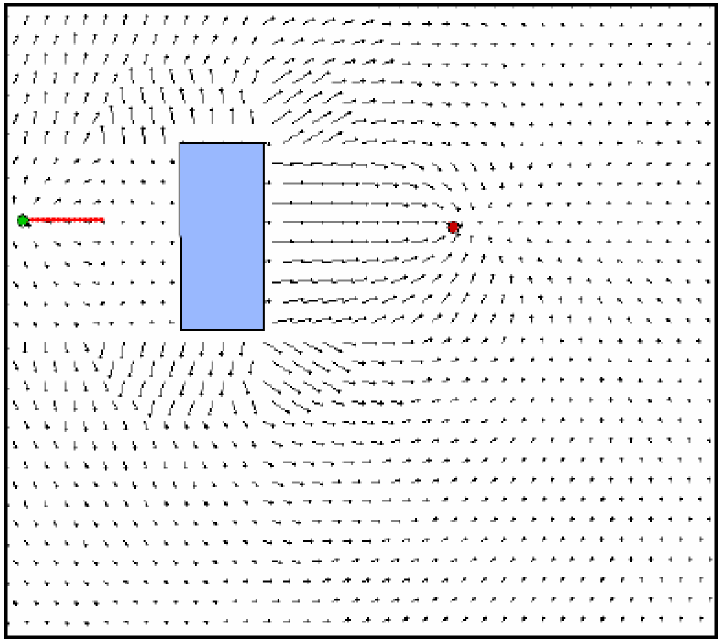
source field added



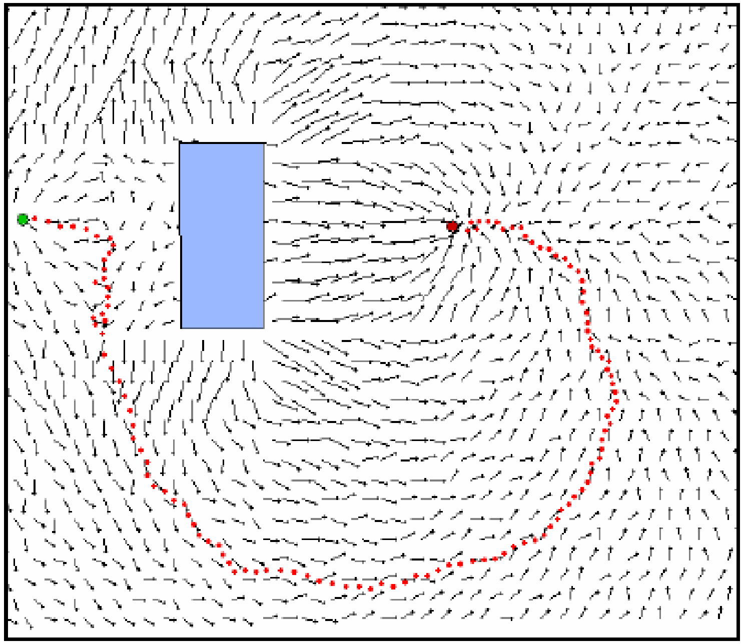
noise (i.e., random vectors)

Local Minima

- The addition of the outwards source field will likely still lead to a local minimum, but added noise often overcomes minimum problem (but no guarantee):



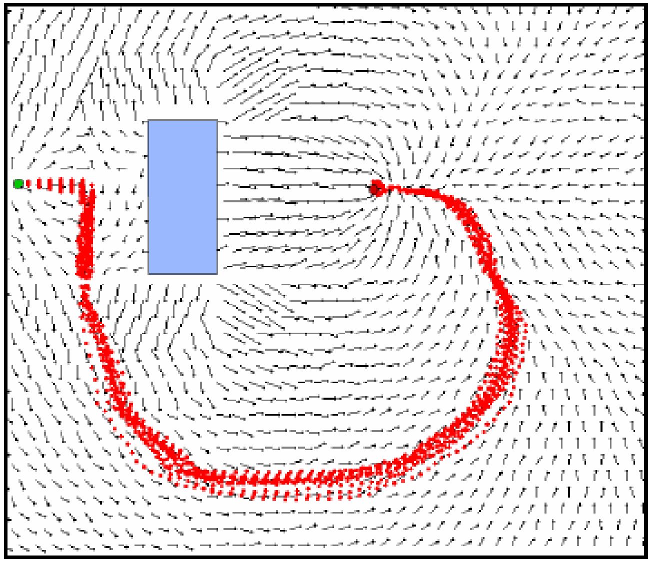
without noise, no path



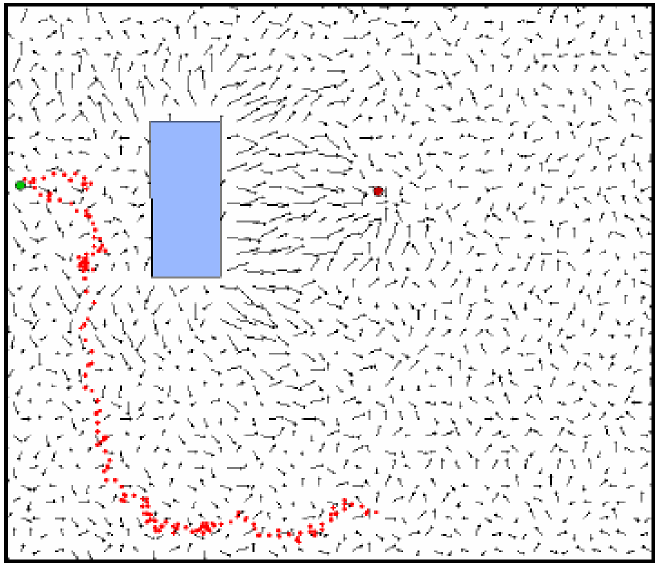
with noise, path found

Local Minima

- During multiple attempts, the path will vary, depending on the random values of the noise vectors.
- Too much noise will not work.

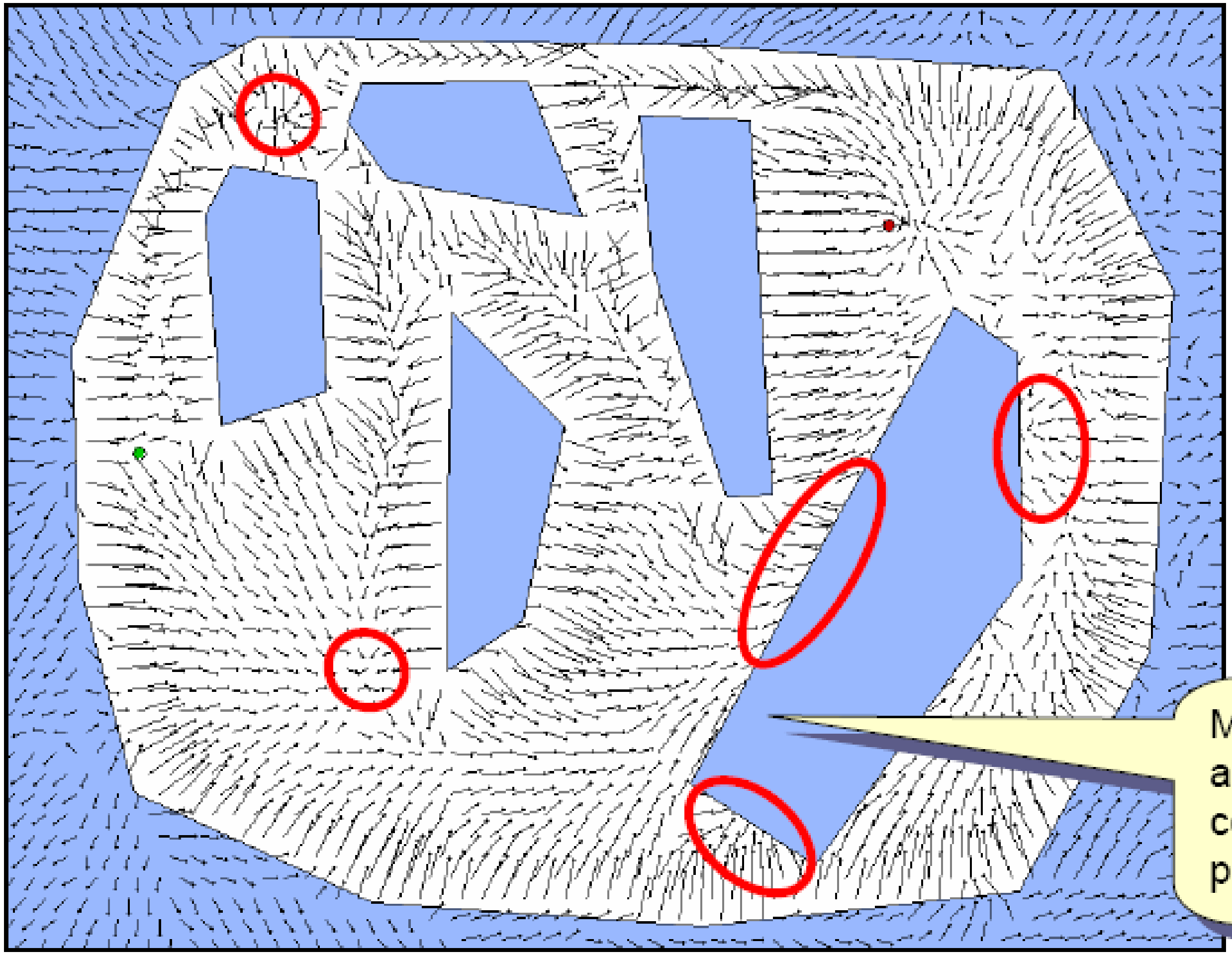


multiple iterations



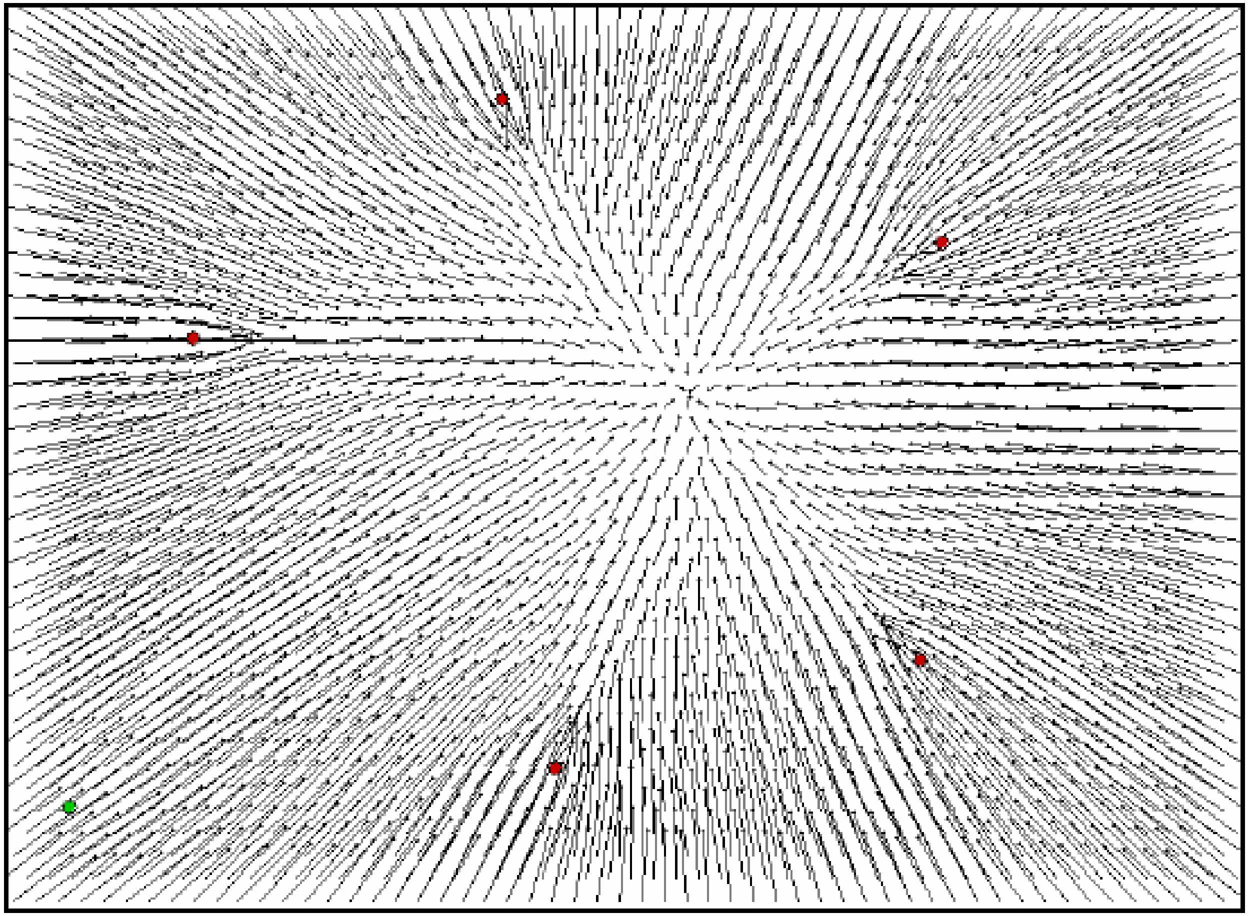
too much noise, no path found

Counteracting fields (environment boundary and obstacles)



May introduce additional counter-acting problems.

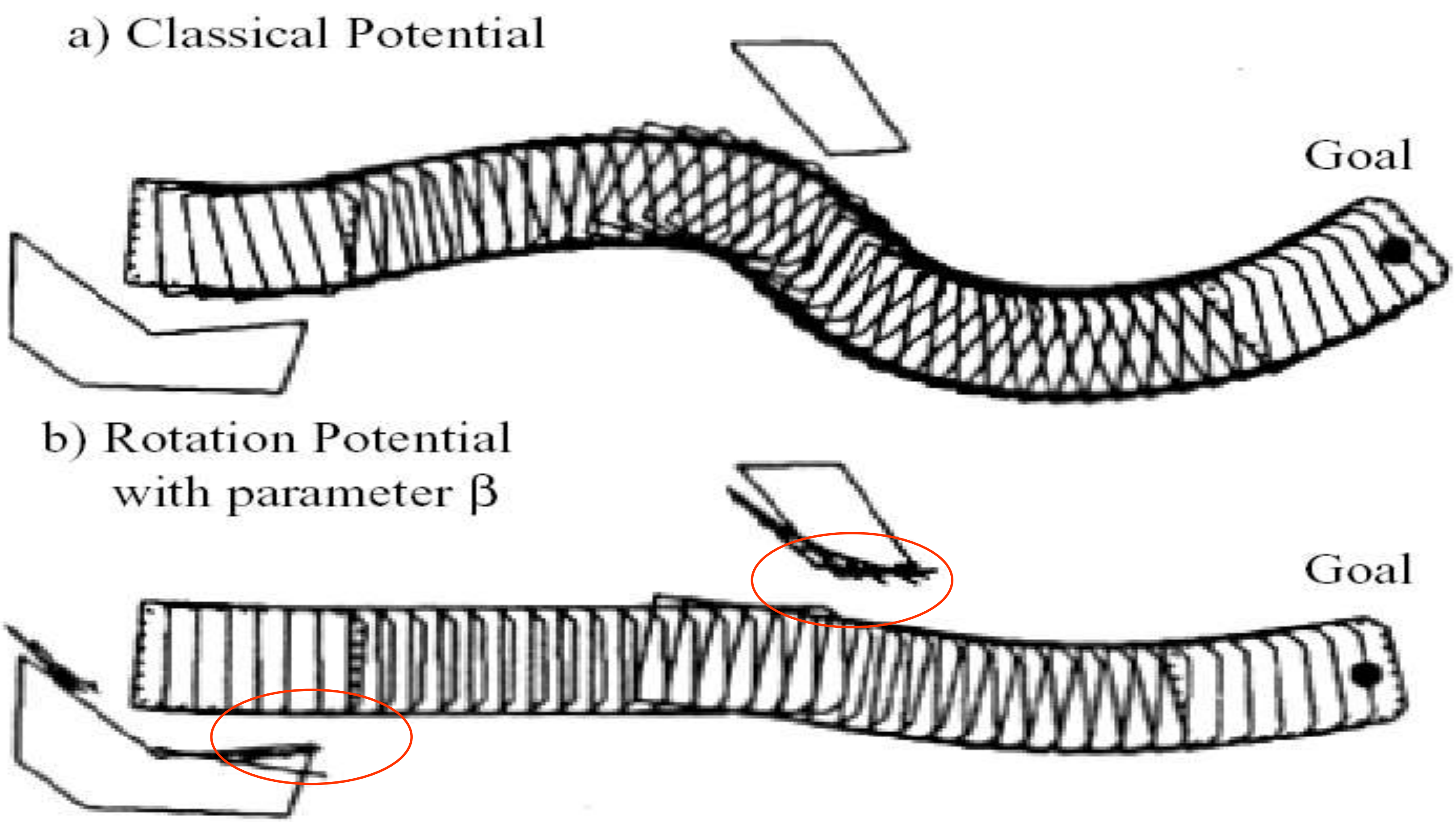
Potential fields cannot handle multiple goals



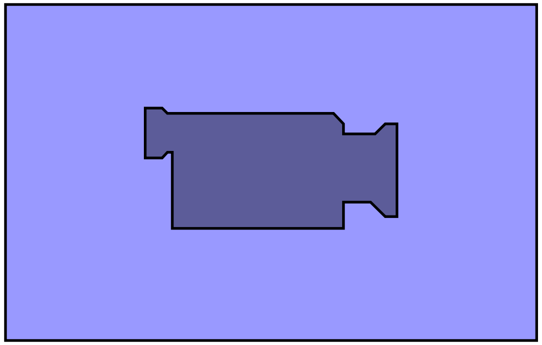
Competencies for Navigation: Extended Potential field method (6.2.1.3)

- Additions to the basic potential field are:
  - *rotation potential field*
  - *task potential field*
- Rotation potential field
  - force is a function of the distance from the obstacle
  - force is also a function of robots orientation to the obstacle
  - reduces a gain factor when the obstacle is parallel to the robot's direction of travel
  - enhanced wall following
- *Task potential field*
  - considers the present robot velocity
  - Filters out the obstacles that should not influence the robots movements, i.e. only the obstacles in the sector Z in front of the robot are considered
  - the sector Z id defined as the space which the robot will sweep during the next movement

Competencies for Navigation: Extended Potential field method (6.2.1.3)



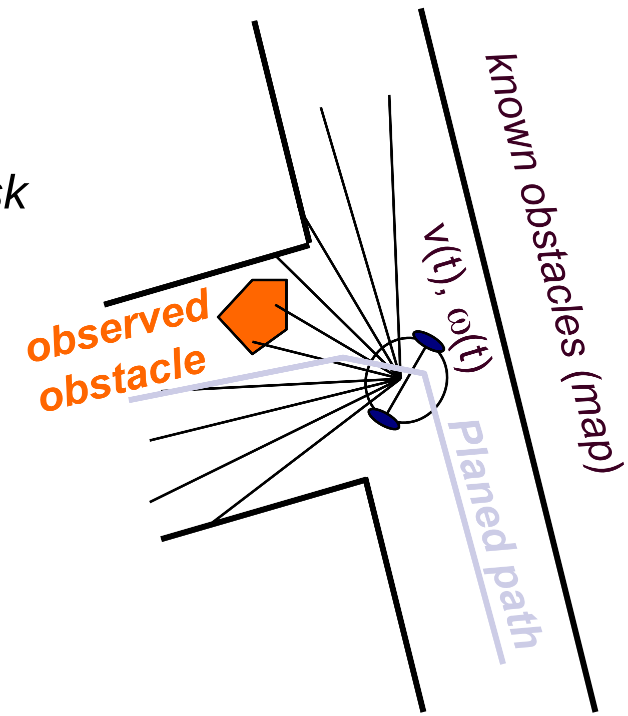
Potential Field Path Planning: Sysquake Demo



- Notes:
  - Local minima problem exists
  - problem is getting more complex if the robot is not considered as a point mass
  - If objects are convex there exists situations where several minimal distances exist → can result in oscillations

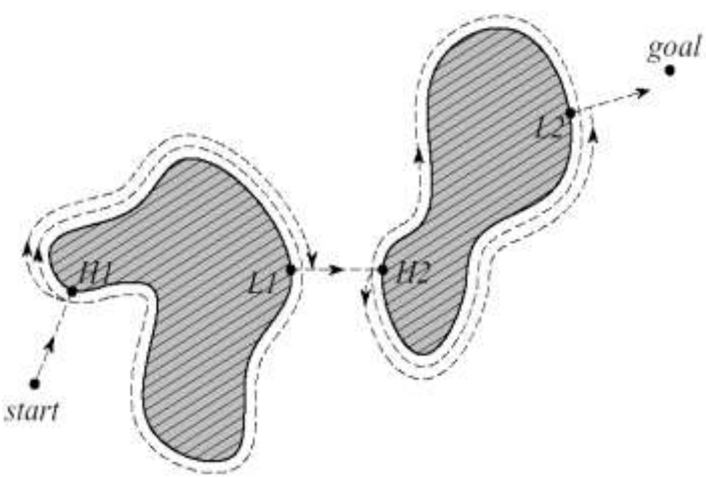
Obstacle Avoidance (6.2.2):
Local Path Planning

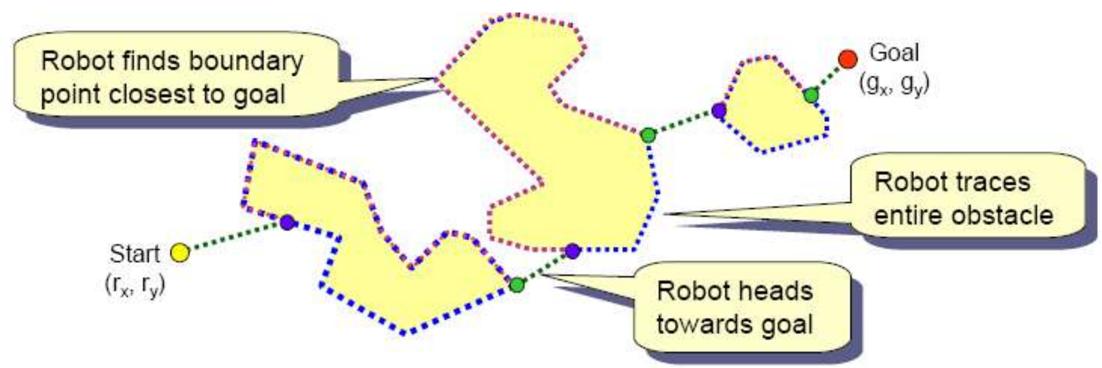
- The goal of the obstacle avoidance algorithms is to avoid collisions with obstacles
- local obstacle avoidance focuses on changing the robot's trajectory as informed by its sensor readings and its goal position and relative location to the goal position
- It is usually based on *local map*
- Often implemented as a more or less *independent task*
- However, efficient obstacle avoidance should be optimal with respect to
  - the overall goal
  - the actual speed and kinematics of the robot
  - the on boards sensors
  - the actual and future risk of collision



Obstacle Avoidance (6.2.2.1): Bug algorithm

- simplest obstacle avoidance algorithm
- follow the contour of each obstacle in the robot's way and circumnavigate it
- Each encountered obstacle is once fully circled before it is left at the point closest to the goal
- very inefficient but guarantees that the robot will reach any reachable goal

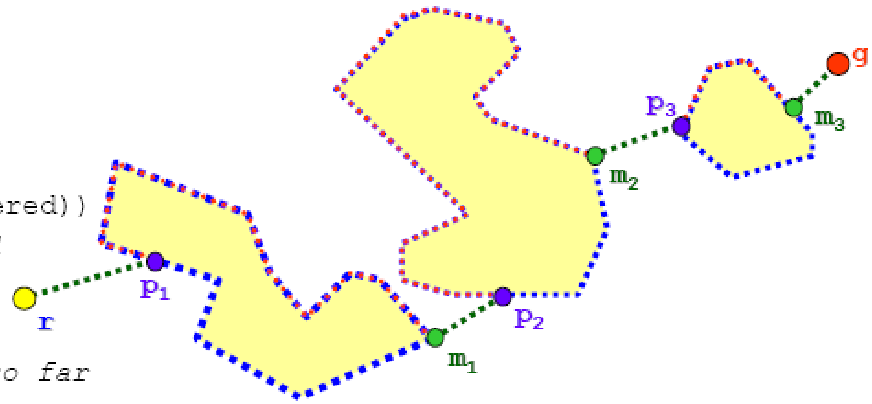


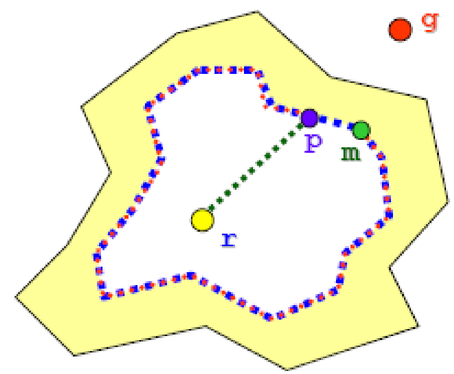


The Bug1 Algorithm

- Here is the pseudo code for the algorithm:

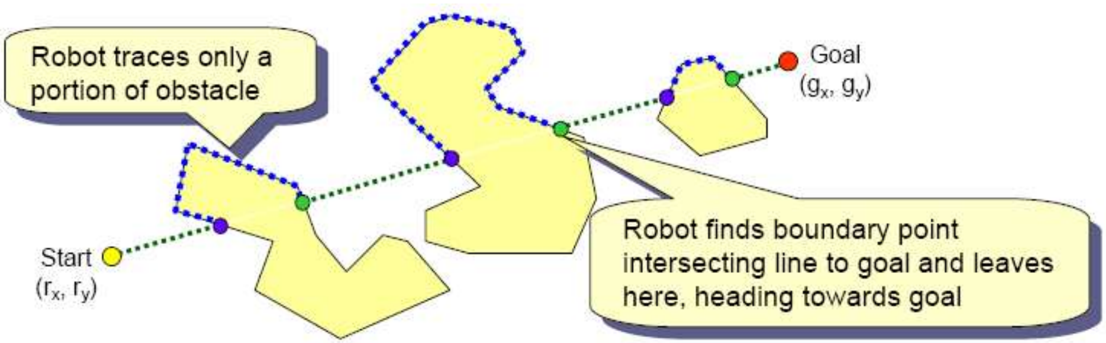
```
WHILE (TRUE)
   REPEAT
     Move from r towards g
     r = robot's current location
   UNTIL ((r == g) OR (obstacleIsEncountered))
   IF (r == g) THEN quit  // goal reached

   LET p = r   // contact location
   LET m = r   // location closest to g so far
   REPEAT
     Follow obstacle boundary
     r = robot's current location
     IF ((distance(r,g) < distance(m,g)) THEN m = r
   UNTIL ((r == g) OR (r == p))
   IF (r == g) THEN quit  // goal reached

   Move to m along obstacle boundary
   IF (obstacleIsEncountered at m in direction of g)
     THEN quit   // goal not reachable
ENDWHILE
```

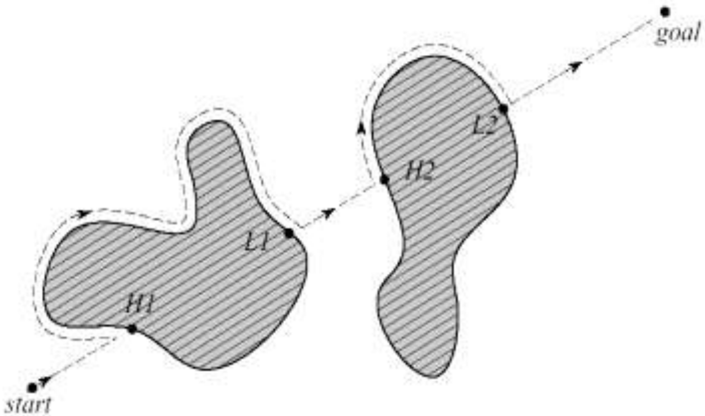




Obstacle Avoidance (6.2.2.1): Bug2 algorithm

- Follows the obstacle always on the left or right side
- Leaves the obstacle if the direct connection between start and goal is crossed
- has significantly shorter total robot travel
- the Tangent Bug adds range sensing and a local environmental representation to termed the *local tangent graph (LTG)*
- LTG approaches globally optimal paths

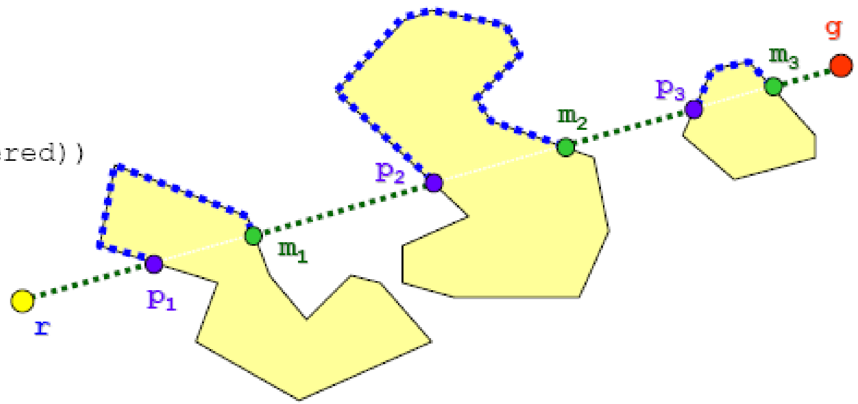




The Bug2 Algorithm

- Here is the pseudo code for the algorithm:

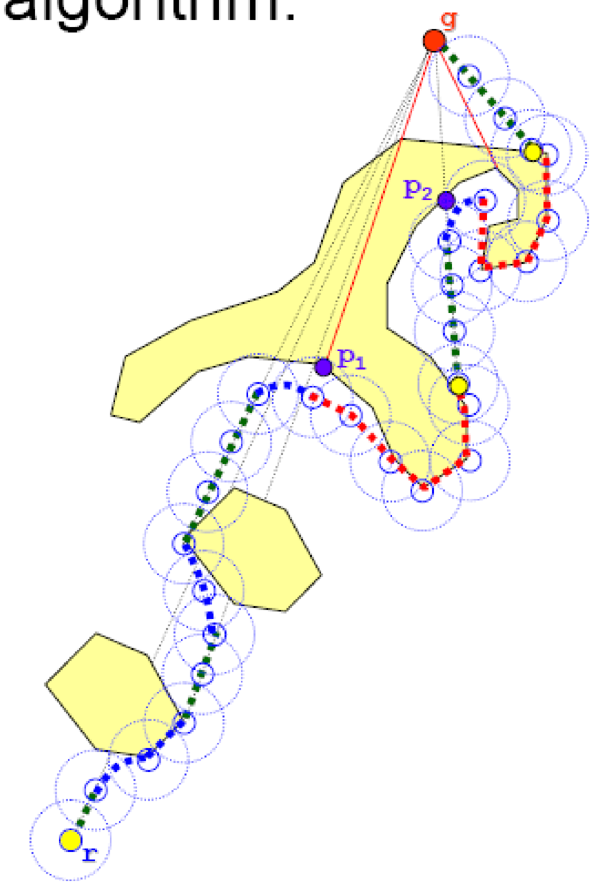
```
WHILE (TRUE)
    LET L = line from r to g
    REPEAT
      Move from r towards g
      r = robot's current location
    UNTIL ((r == g) OR (obstacleIsEncountered))
    IF (r == g) THEN quit  // goal reached

    LET p = r   // contact location
    REPEAT
      Follow obstacle boundary
      r = robot's current location
      LET m = intersection of r and L
    UNTIL (((m is not null) AND (dist(m,g) < dist(p,g)) OR (r == g) OR (r == p))

    IF (r == g) THEN quit  // goal reached
    IF (r == p) THEN quit  // goal not reachable
ENDWHILE
```



Tangent Bug Algorithm

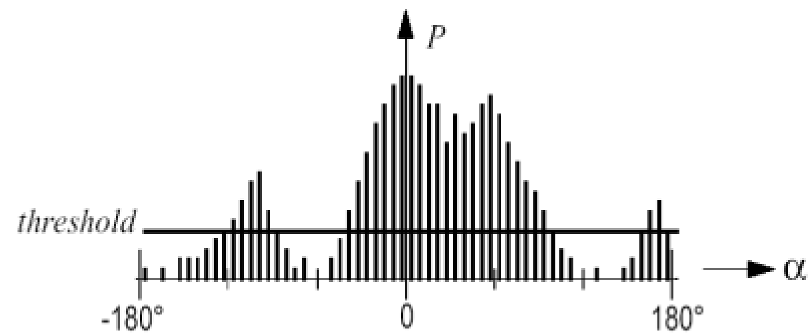
- Here is the pseudo code for the algorithm:

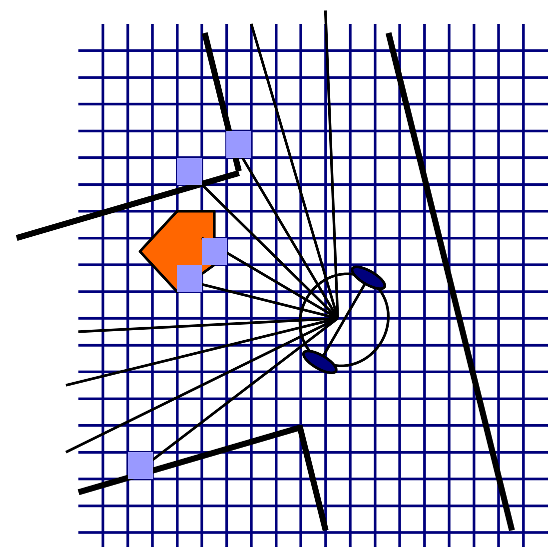
```
WHILE (TRUE)
    LET w = g
    REPEAT
      r' = r    // robot's previous location
      update r by moving towards w
      IF (no obstacle detected in direction w) THEN w = g
      ELSE
         LET eL and eR be discontinuity points
         IF ((dist(r, eL)+dist(eL,g)) < (dist(r, eR)+dist(eR,g)))
            THEN w = eL ELSE w = eR
    UNTIL ((r == g) OR (dist(r',g) < dist(r,g)))
    IF (r == g) THEN quit       // goal reached

    LET p = m = r // contact location
    LET dir = direction of continuity point (L or R)
    REPEAT
      LET w = the discontinuty point in direction dir
      IF (dist(r,g) < dist(m,g)) THEN m = r
      update r by moving towards w
    UNTIL ((dist(r,g) < dist(m,g)) OR (r == g) OR (r == p))

    IF (r == g) THEN quit       // goal reached
    IF (r == p) THEN quit       // goal not reachable
ENDWHILE
```



Obstacle Avoidance (6.2.2.2): Vector field histogram (VFH)

- VFH uses potential fields and creates a local map of the environment around the robot
- it is an occupancy grid populated only by relatively recent sensor range readings
- Environment represented in a grid (2 DOF)
  - cell values equivalent to the probability that there is an obstacle
- Reduction in different steps to a 1 DOF histogram
  - calculation of steering direction
  - all openings for the robot to pass are found
  - the one with lowest cost function *G* is selected

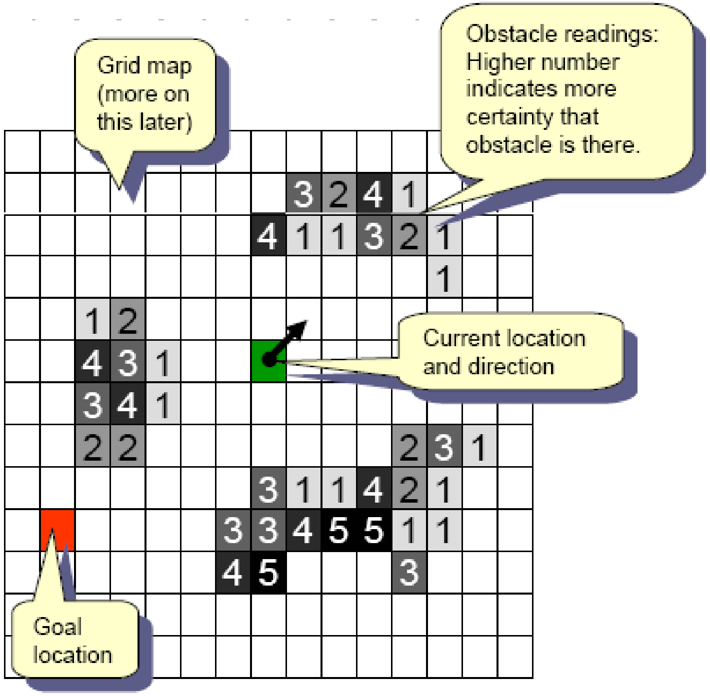




$$G = a \cdot \text{target\_direction} + b \cdot \text{wheel\_orientation} + c \cdot \text{previous\_direction}$$

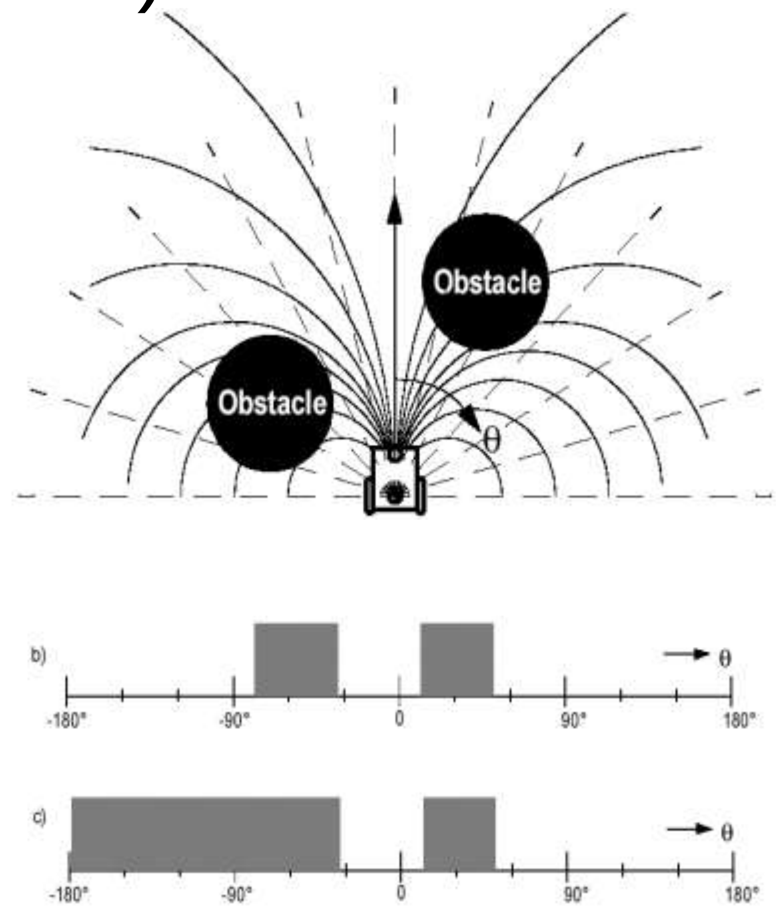
Obstacle Avoidance (6.2.2.2): Vector field histogram (VFH)

- Notes:
  - Limitation if narrow areas (e.g. doors) have to be passed
  - Local minimum might not be avoided
  - Reaching of the goal can not be guaranteed
  - Dynamics of the robot not really considered



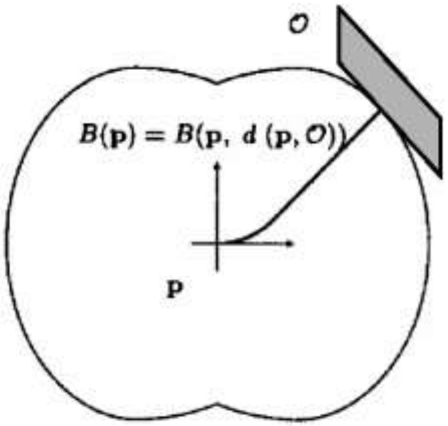
Obstacle Avoidance (6.2.2.2): Vector field histogram+ (VFH+)

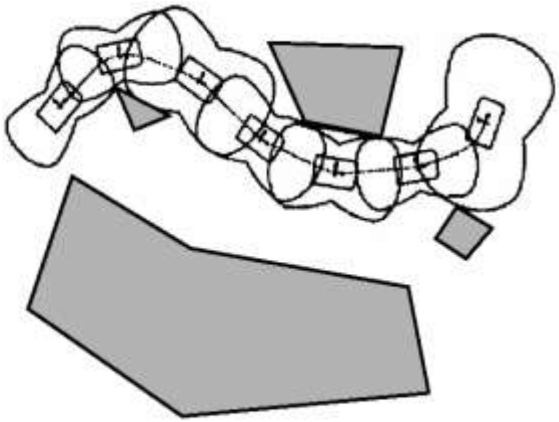
- takes into account a simplified model of the moving robot’s possible trajectories based on its kinematic limitations (dynamics)
- the robot is modeled to move in arcs or straight lines
- an obstacle blocks all of the allowable trajectories which pas through the obstacle



Obstacle Avoidance (6.2.2.3): The bubble band technique

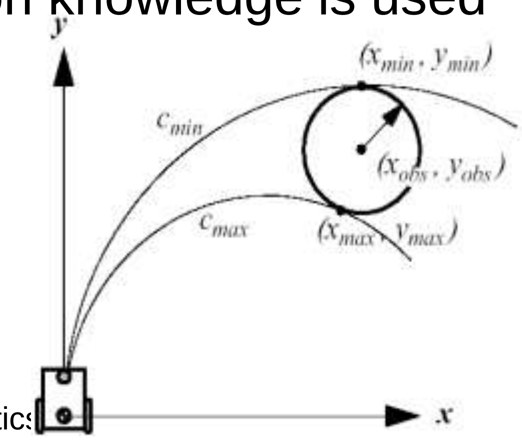
- A *bubble* is defined as the maximum local subset of the free space around a given configuration of the robot which can be traveled in ay direction without collision
  - generated using the distance to the object and a simplified model of the robot
  - bubbles are used to form a band of bubbles which connects the start point with the goal point
  - requires a global map and global path planner
  - accounts for the actual dimensions of the robot
  - most valid when the environmental configuration is well known ahead of time





Obstacle Avoidance (6.2.2.4):
Curvature velocity techniques (CVM)

- Adding *physical constraints* from the robot and the environment on the *velocity space* (v, ω) of the robot
- Assumption that robot is traveling on arcs ($c = \omega / v$)
- There are acceleration and speed constraints
- Obstacle constraints: Obstacles are transformed in velocity space
- obstacles are approximated as circular objects
- Objective function to select the optimal speed
- also suffers from local minima since no a priori knowledge is used by the system



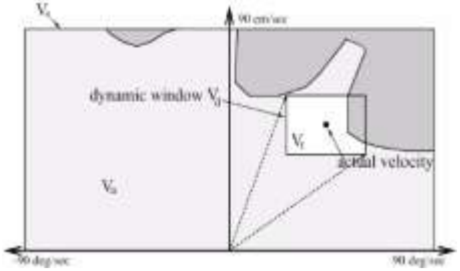
Obstacle Avoidance (6.2.2.4):
Lane curvature method

- Improvement of basic CVM
  - there was difficulty driving the robot intersections of corridors
  - Not only arcs are considered
  - lanes are calculated trading off lane length and width to the closest obstacles
  - Lane with best properties is chosen using an objective function
- Note:
  - Better performance to pass narrow areas (e.g. doors)
  - Problem with local minima persists

Obstacle Avoidance (6.2.2.5): Dynamic window approaches

- The kinematics of the robot is considered by searching a well chosen velocity space
  - velocity space -> some sort of configuration space
  - robot is assumed to move on arcs
  - ensures that the robot comes to stop before hitting an obstacle
  - objective function is chosen to select the optimal velocity
- given current robot speed, the algorithm selects a *dynamic window* of the velocity space that can be reached within the next sample period, taking into account the acceleration capabilities of the robot and the cycle time
- the *dynamic window* is reduced by keeping only those tuples that ensure that the vehicle can come to a stop before hitting an obstacle
- the window is rectangular following from the approximation that the translation and rotation are independent

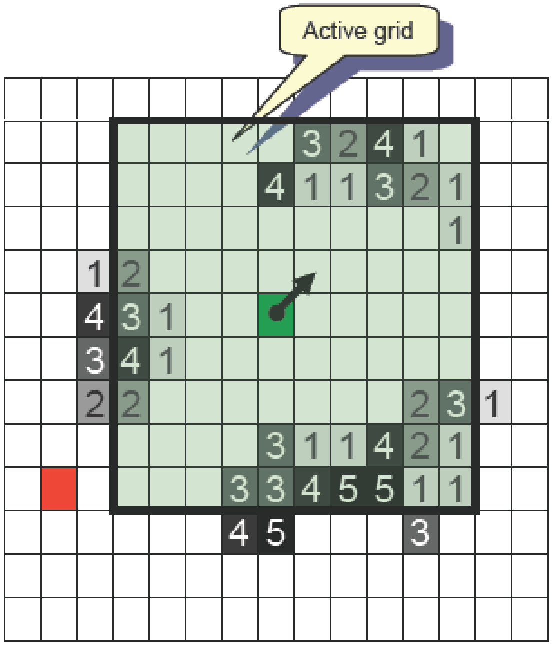
$$O = a \cdot heading(v, \omega) + b \cdot velocity(v, \omega) + c \cdot dist(v, \omega)$$



- heading = measure of progress toward goal
- velocity = forward velocity of the robot
- dist = distance to the closest obstacle in the trajectory

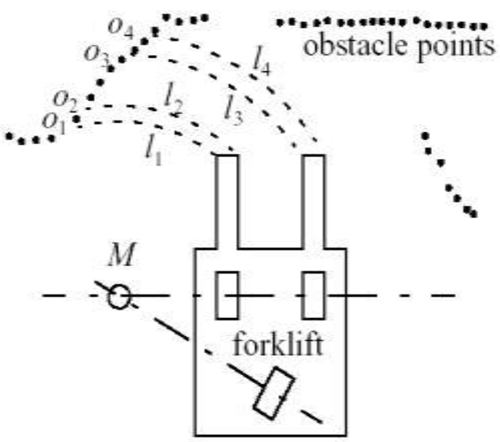
Obstacle Avoidance (6.2.2.5): Global Dynamic window approach

- adds global thinking to the algorithm
- adds wavefront propagation to the objective function for the dynamic window approach
- calculates the wavefront only on a selected rectangular region which is directed from the robot toward the goal
- the width of the region is enlarged and recalculated if the goal cannot be reached within the constraints of this chosen region
- updated from range measurements as the robot moves in the environment



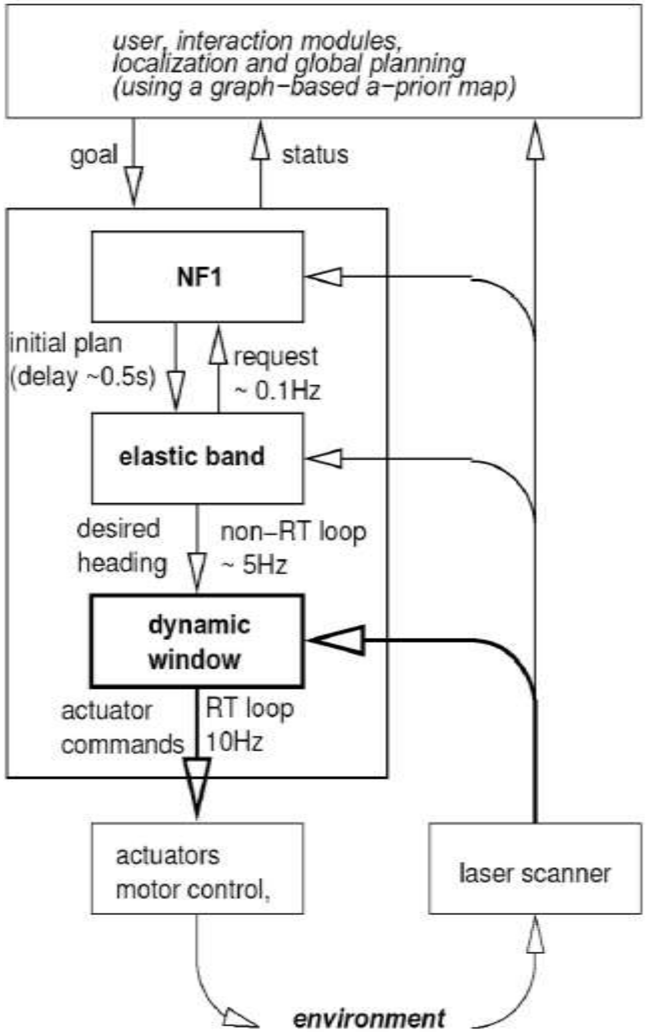
Obstacle Avoidance (6.2.2.6): Schlegel approach

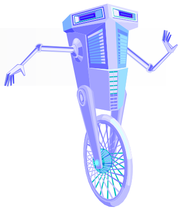
- Some sort of a variation of the dynamic window approach
  - approach considers the dynamics as well as the actual shape of the robot
  - for use with raw laser data and sensor fusion using a Cartesian grid to represent obstacles
  - uses a wavefront planner
  - motion of circular arcs
  - real time performance achieved
    by use of precalculated table to find distance to collision with obstacle



Obstacle Avoidance (6.2.7): ASL approach

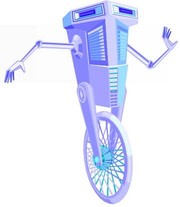
- Dynamic window approach with global path planning
  - Global path generated in advance
  - local path planning performed by wavefront
  - resulting path converted to an elastic band that does not take into account kinematics
  - Path adapted if obstacles are encountered
  - dynamic window considering also the shape of the robot
  - real-time because only max speed is calculated





Navigation Architectures (6.3)

- given techniques for path planning, obstacle avoidance, localization and perceptual interpretation, how do we combine all these into one complete robot system for real world application?
- the study of *navigation architectures* is the study of principled designs for the software modules that constitute a mobile robot navigation system



Navigation Architectures (6.3)

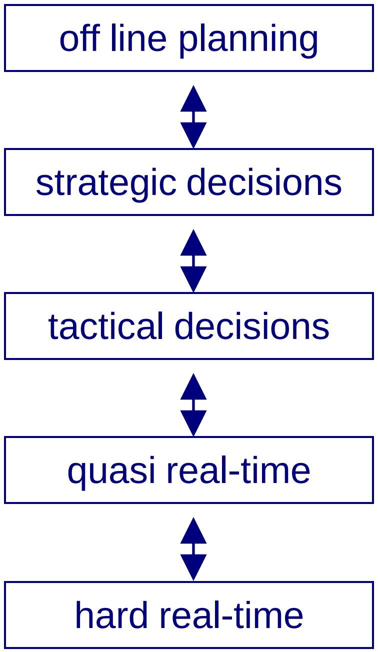
- concrete advantages of a well-designed navigation architecture:
  - modularity for code reuse and sharing
  - control localization
- an architecture can be characterized by its decomposition of the robot's software
- for navigation competence, *temporal* and *control* decompositions are particularly relevant
- *behaviors* are used for implementing control decomposition

Navigation Architectures:
Techniques for decomposition (6.3.3)

- *temporal decomposition*
  - distinguishes between real-time and non real-time demands on mobile robot operation
- *control decomposition*
  - identifies the way in which various control outputs within the mobile robot architecture combine to yield the robot’s physical actions

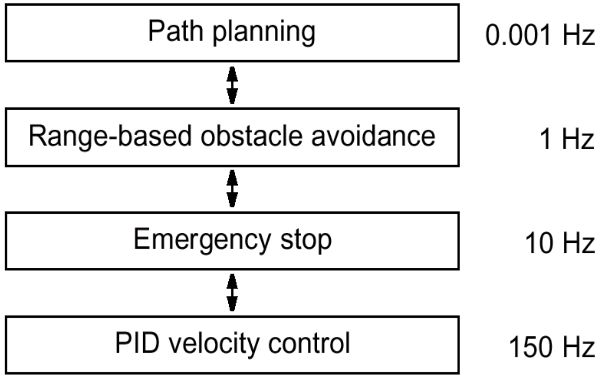
Navigation Architectures:
Generic temporal decomposition (6.3.3.1)

- the most real-time processes are shown at the bottom of the stack
- the highest category are processes with no real-time demands
- *sensor response time* is the amount of time between the acquisition of a sensor-based event and a corresponding change in the output of the module
- sensor response time increases as one moves up the stack
- for the lowest-level, the sensor response time is limited by the raw processor and sensor speed
- for the highest level, the sensor response can be limited by slow deliberate decision making
- *temporal horizon* is the amount of look ahead used by a module during the process of choosing an output
- *temporal memory* is the amount of historical time span of sensor input that is used by the module to determine the next output



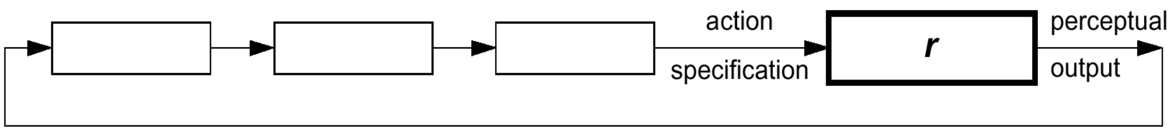
Navigation Architectures:
Generic temporal decomposition (6.3.3.1)

- the spatial impact of layers increases dramatically as one moves from low level to high level modules
- real time modules control wheel speed and orientation, controlling spatially localized behavior
- high level strategic decision making have little or no bearing on local position, but information global positions far into the future
- low level modules tend to produce outputs directly from immediate sensor inputs and are not context sensitive
- high level modules exhibit very high context specificity

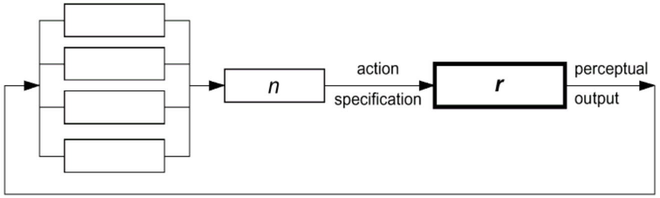


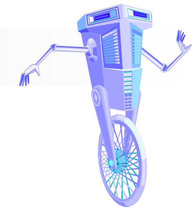
Navigation Architectures:
Control decomposition (6.3.3.2)

- Control decomposition identifies the way in which each module's output contributes to the overall robot control outputs
- Pure serial decomposition
  - advantages relating to predictability and verifiability



- Pure parallel decomposition
  - control flow contains a combination step at which each point the result of multiple modules may impact the robot in arbitrary ways

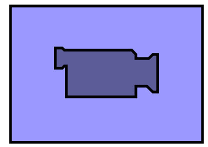


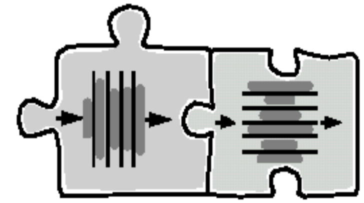


Navigation Architectures:
Parallel decomposition (6.3.3.2)

- The output of each component to inform the overall decision can be *switched parallel* or *mixed parallel*
- both of these architectures are popular in the *behavior-based robotics* community
  - motor-schema architecture – behaviors map sensor value vectors to motor value vectors
  - behavior network – a behavior is chosen discretely by comparing and updating activation levels for each behavior
  - subsumption architecture – the active model is chosen via a suppression mechanism rather than an activation level

Our basic architectural example (6.3.4)





Mixed Approach