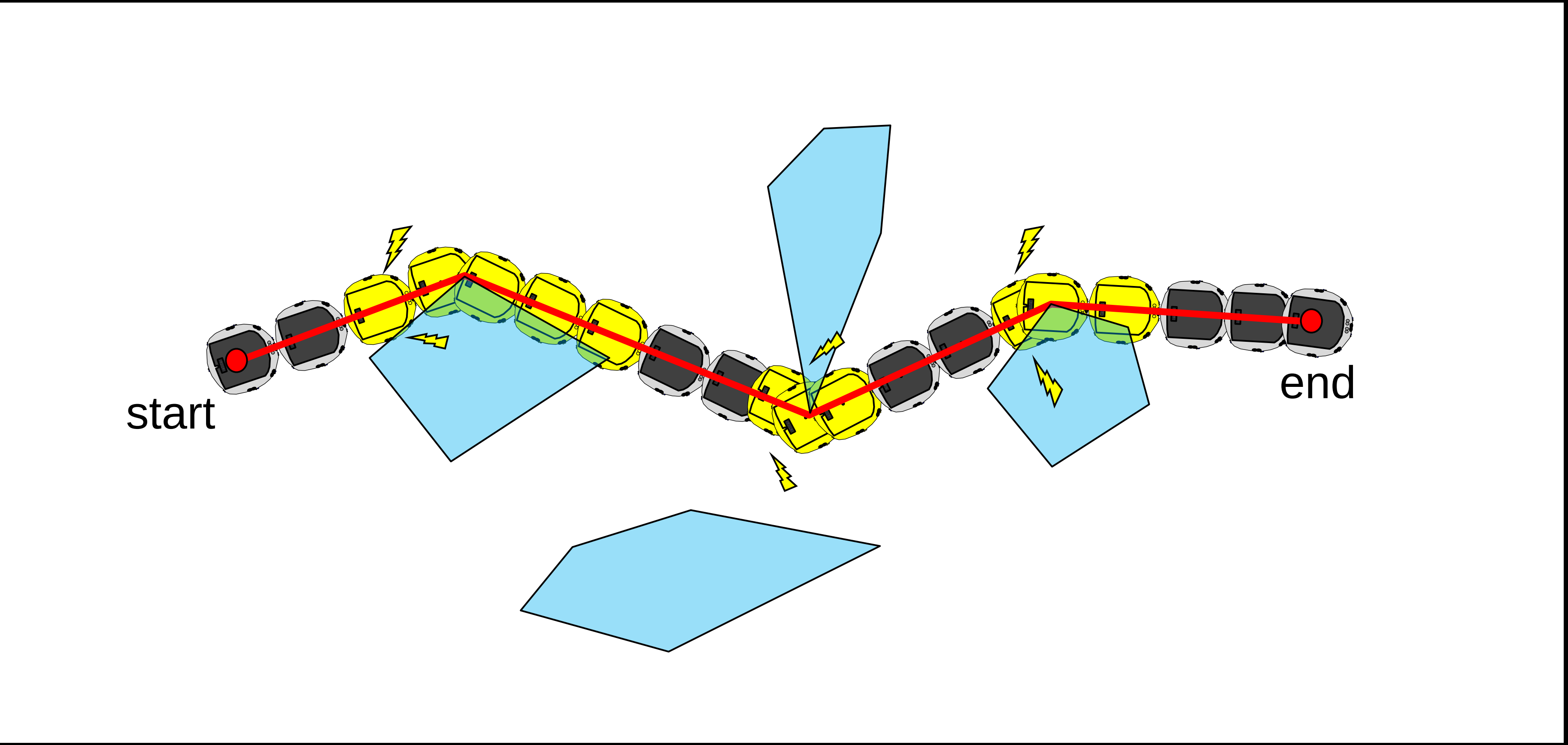
Grown Obstacle Space

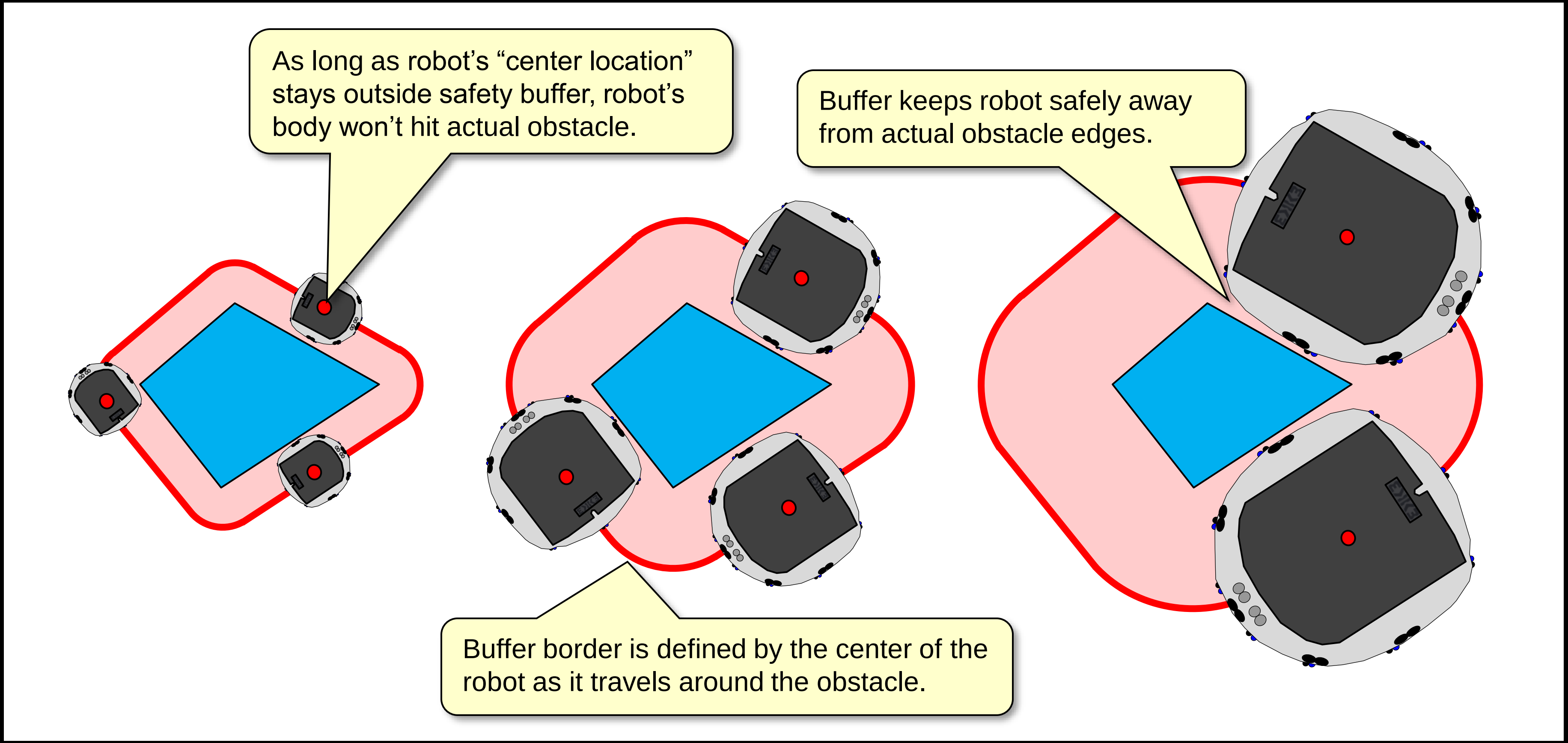
Shortest Path Problems

- Any real robot will collide with obstacles if it travels along our computed shortest path because we assumed that the robot was just a point, but a robot has a shape:



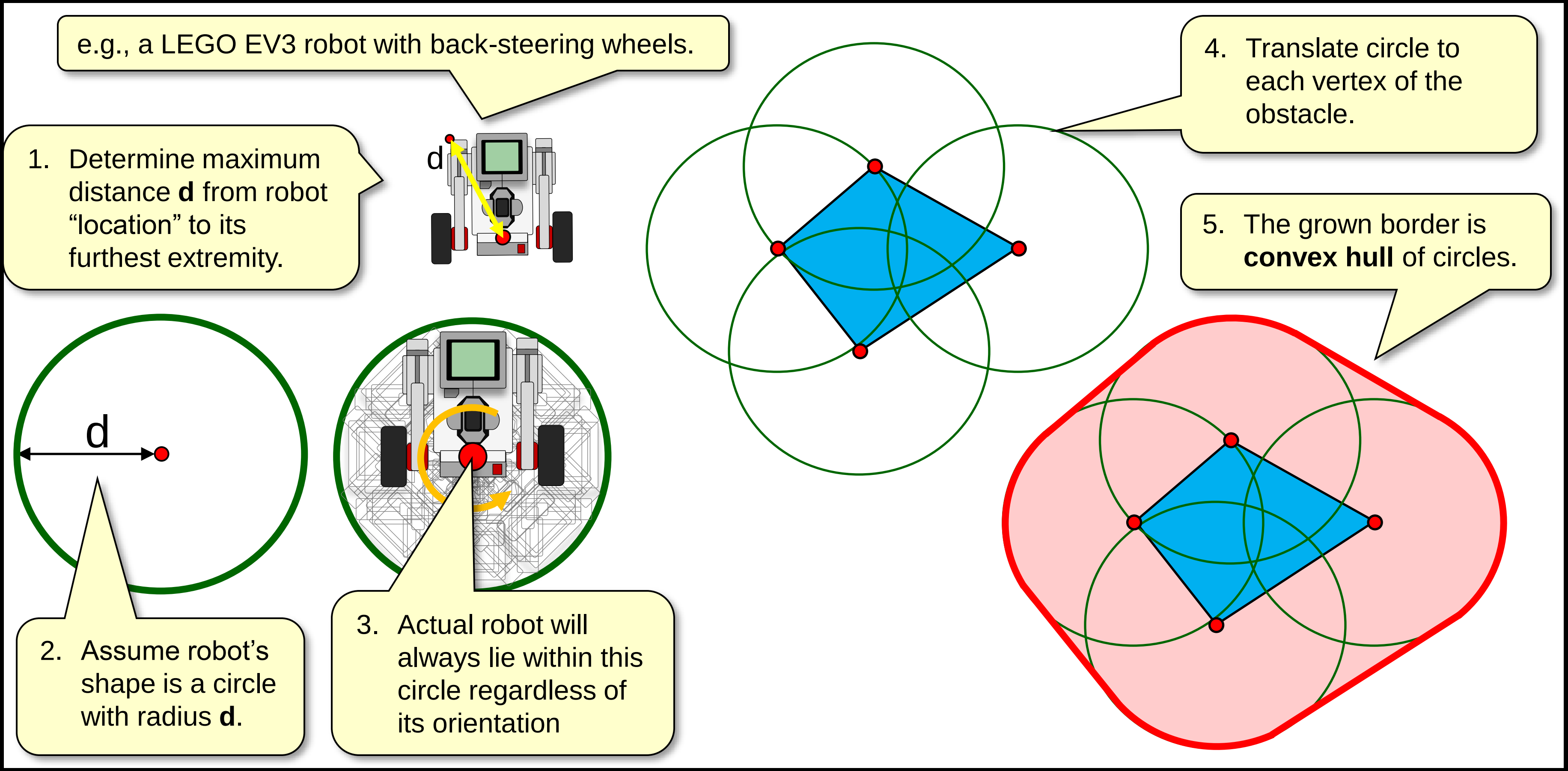
Safety Buffers

- Need to have a “safety buffer” around the obstacles which takes into account robot’s size and shape.



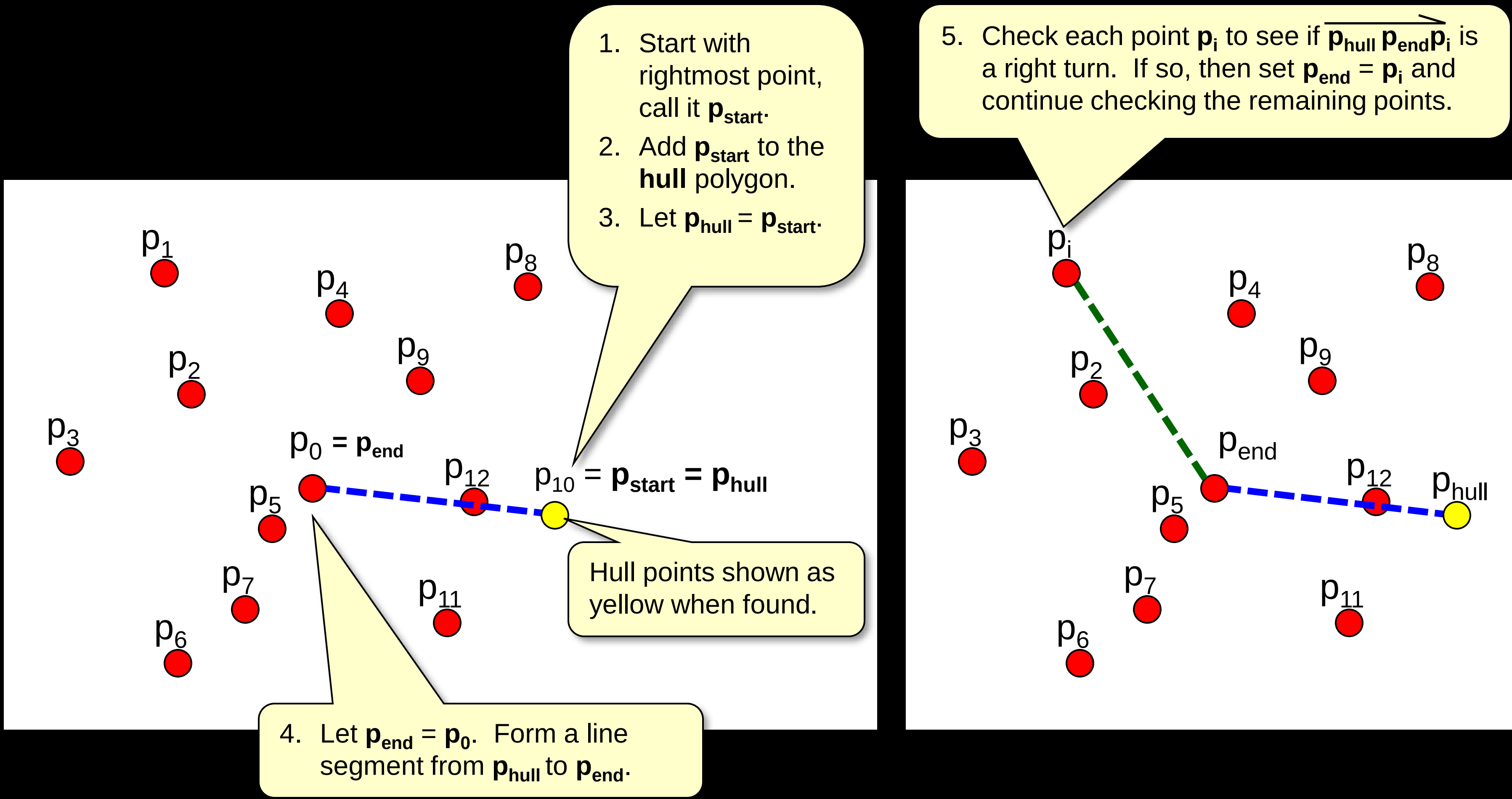
Grown Obstacle Space

- Must "grow" obstacles by amount that considers any orientation of robot at any point on obstacle border:

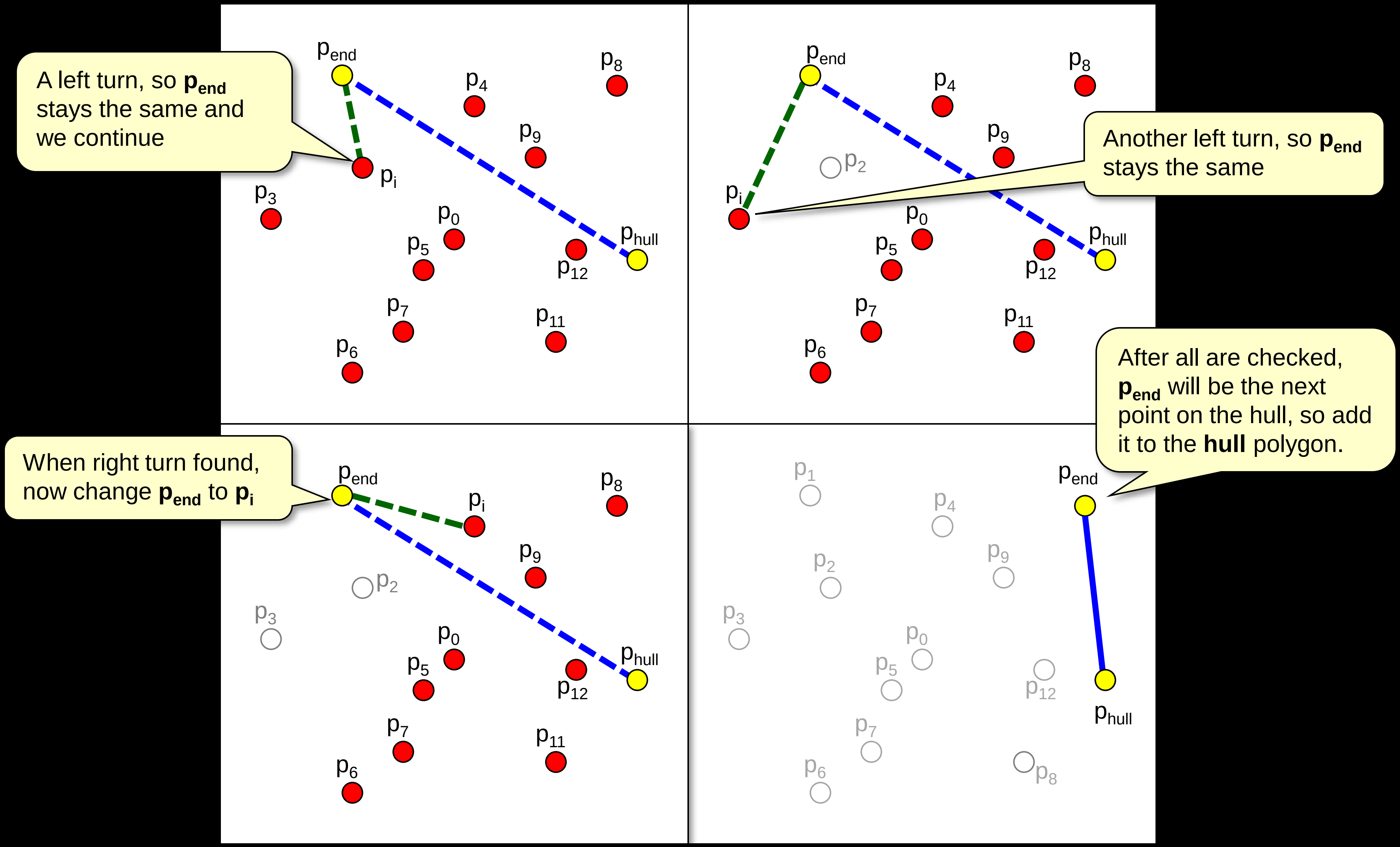


Computing the Convex Hull

- Algorithms exist to find convex hull of a set of points
  - Graham Scan (a.k.a. “Gift-Wrapping” method) is a simple one:

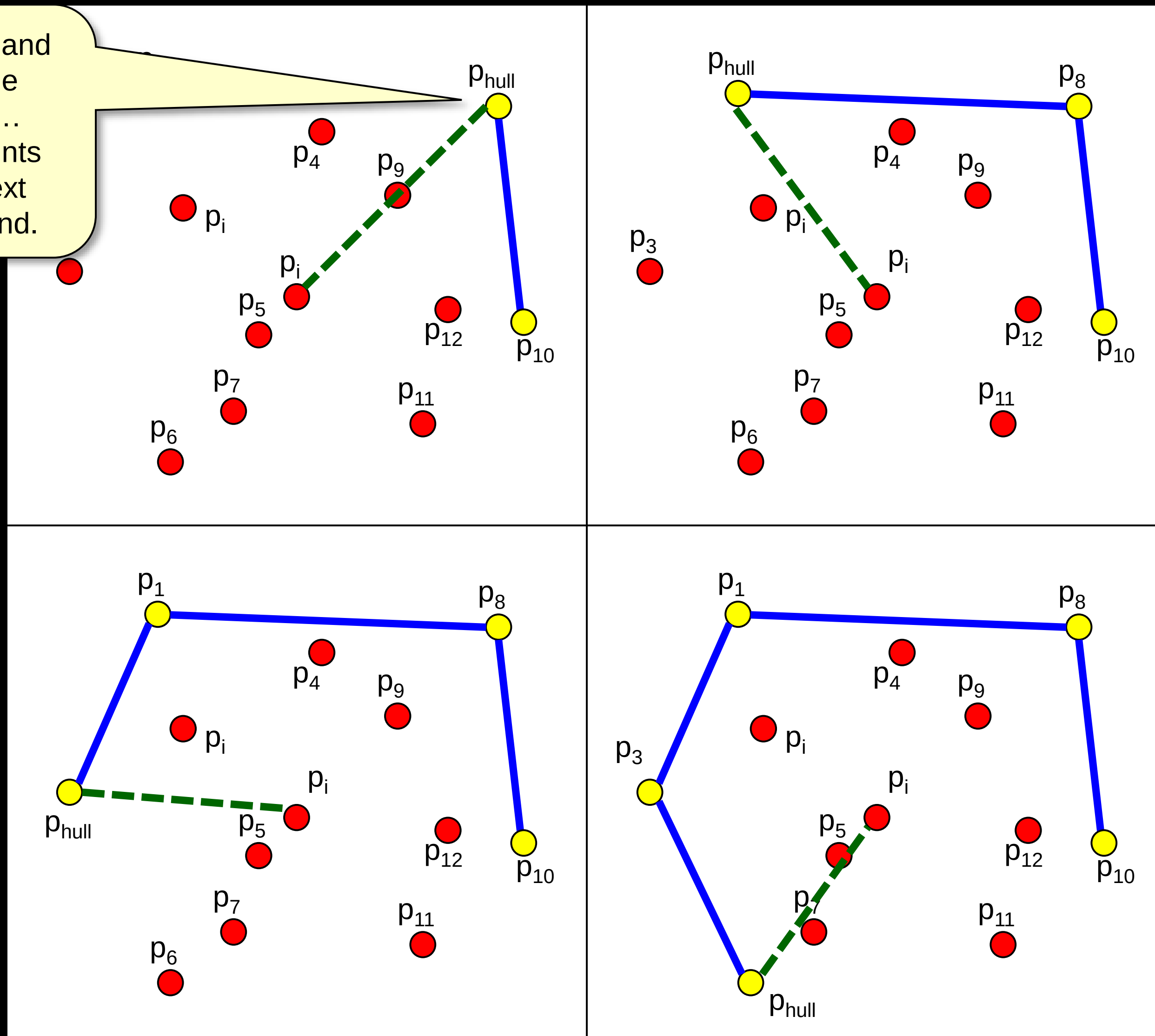


Computing the Convex Hull

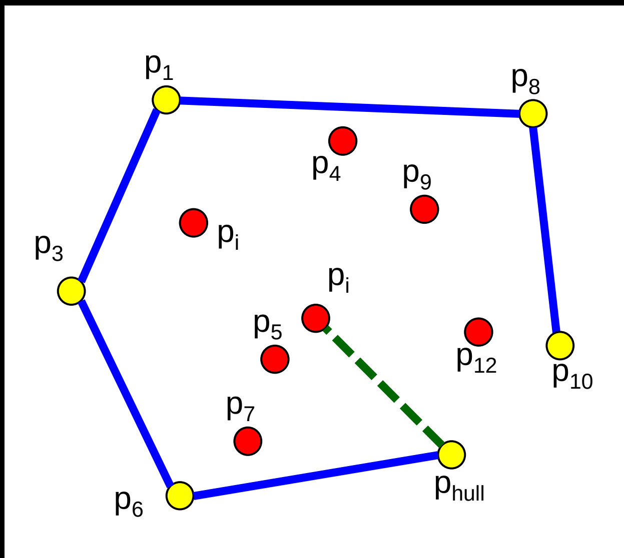


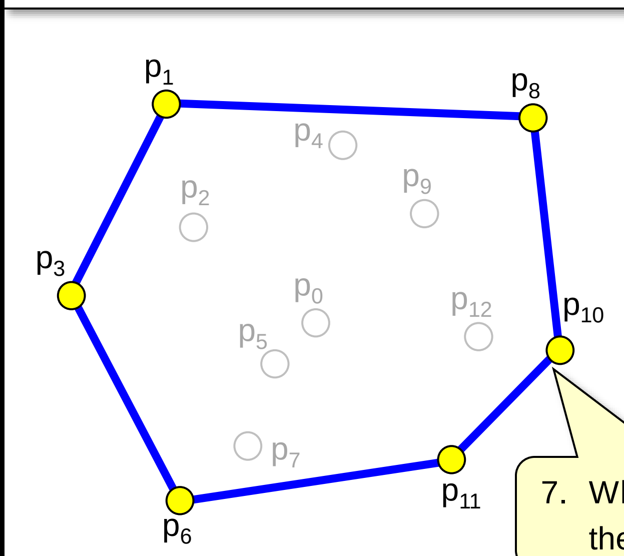
Computing the Convex Hull

6. Set $\mathbf{p_{hull}} = \mathbf{p_{end}}$ and repeat the whole process again … checking all points in order until next hull point is found.



Computing the Convex Hull






```
FUNCTION ConvexHull(PointList points)
    hull = an empty list
    pHull = the rightmost point in points
    pStart = pHull
    pEnd = NULL
    WHILE (pEnd is not equal to pStart) DO
        Add pHull to hull
        pEnd = the first point in points
        FOR (each point pi in points) DO
            IF (pHull is the same as pEnd) THEN
                pEnd = pi
            IF (angle pHull to pEnd to pi is a right turn) THEN
                pEnd = pi
        pHull = pEnd
    RETURN hull
```

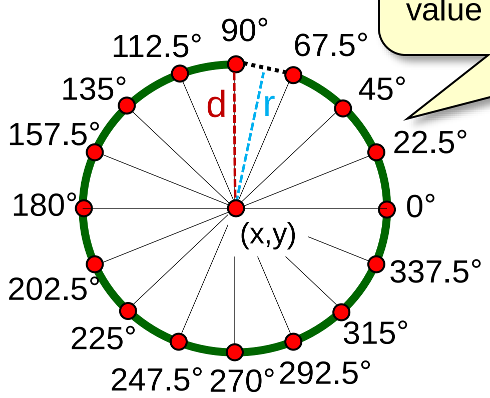
Special case

it is a right turn only if
$((pEnd_x - pHull_x)*(pi_y - pHull_y) - (pEnd_y - pHull_y)*(pi_x - pHull_x)) < 0$

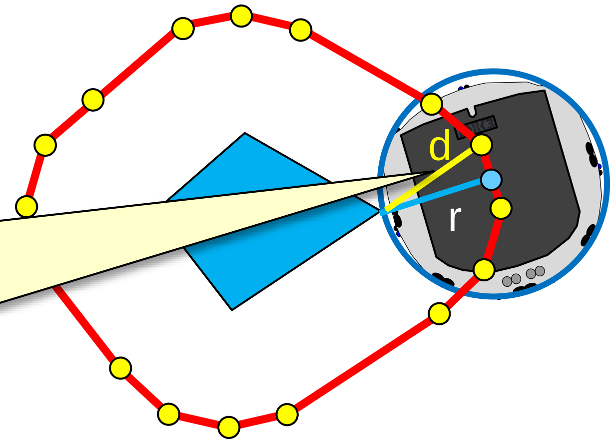
7. When $\mathbf{p_{end}} = \mathbf{p_{start}}$ then we are done.

Convex Hulls Around Obstacles

- Find Convex Hull of “points circles” around obstacle vertices:



Decide upon how many points you want on the circle by setting a **DEGREE_UNIT** value (e.g., 22.5° shown here).



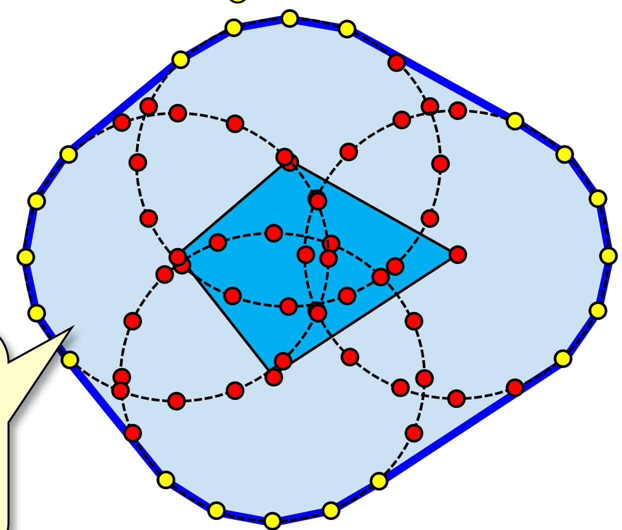
Given a robot radius of **r**, then points must be at least distance **d** from (x,y) so that robot does not collide when travelling from one vertex to another, where **d** is

$$\frac{r}{\cos\left(\frac{\text{DEGREE\_UNIT}}{2}\right)}$$

```
circlePoints = empty list

FOR (a=0; a<360; a+=DEGREE_UNIT) DO {
  xoff = d*cos(a)
  yoff = d*sin(a)
  if (xoff > 0) pX = x + ceil(xoff)
  otherwise pX = x + floor(xoff)
  if (yoff > 0) pY = y + ceil(yoff)
  otherwise pY = y + floor(yoff)
  add point (pX, pY) to circlePoints
}
```

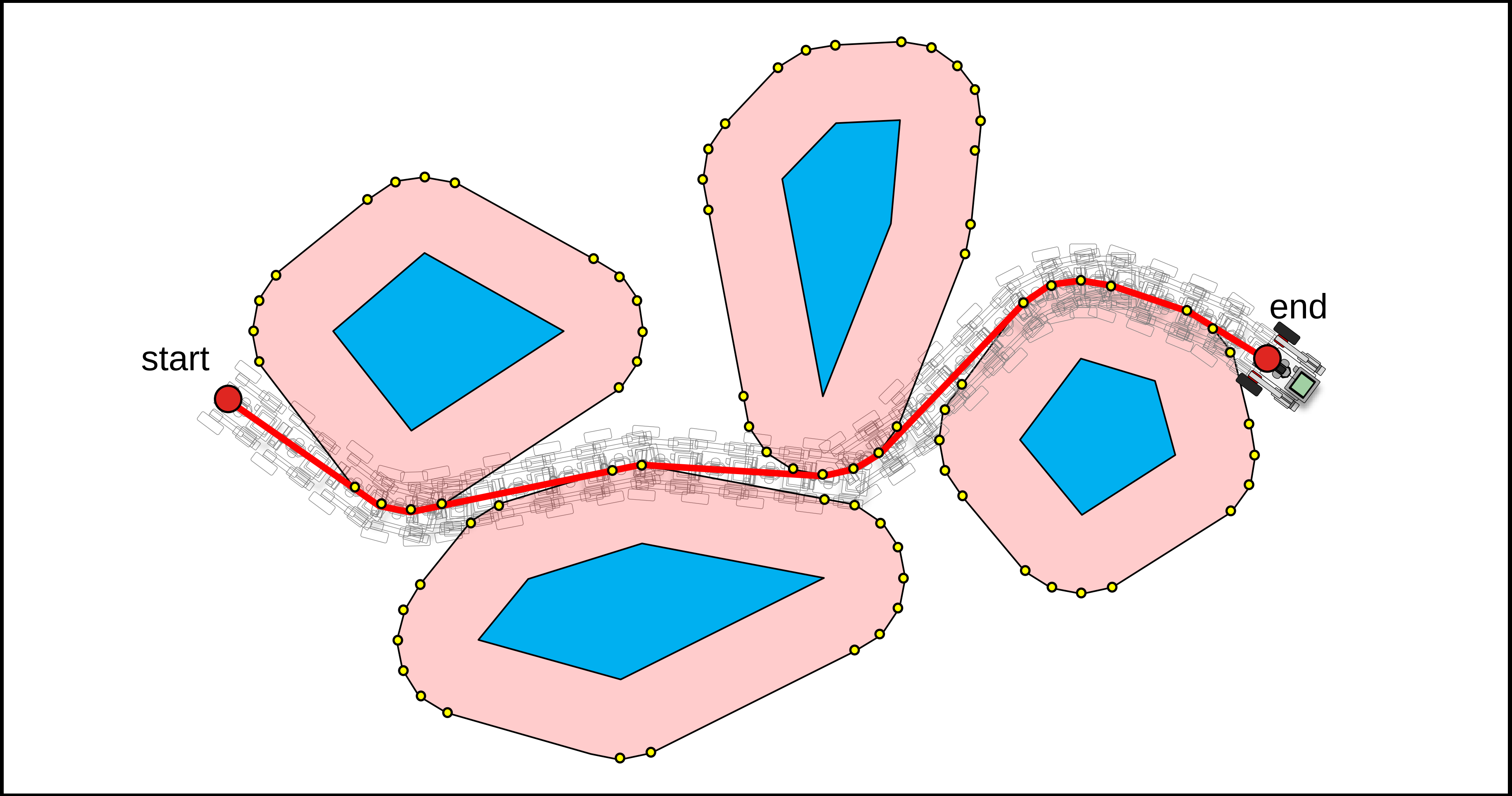
Create a list of **circlePoints** centered around a point (x, y) as shown here.



Merge all **circlePoints** lists from each vertex of the obstacle. Using this “merged” list of points … find the convex hull.

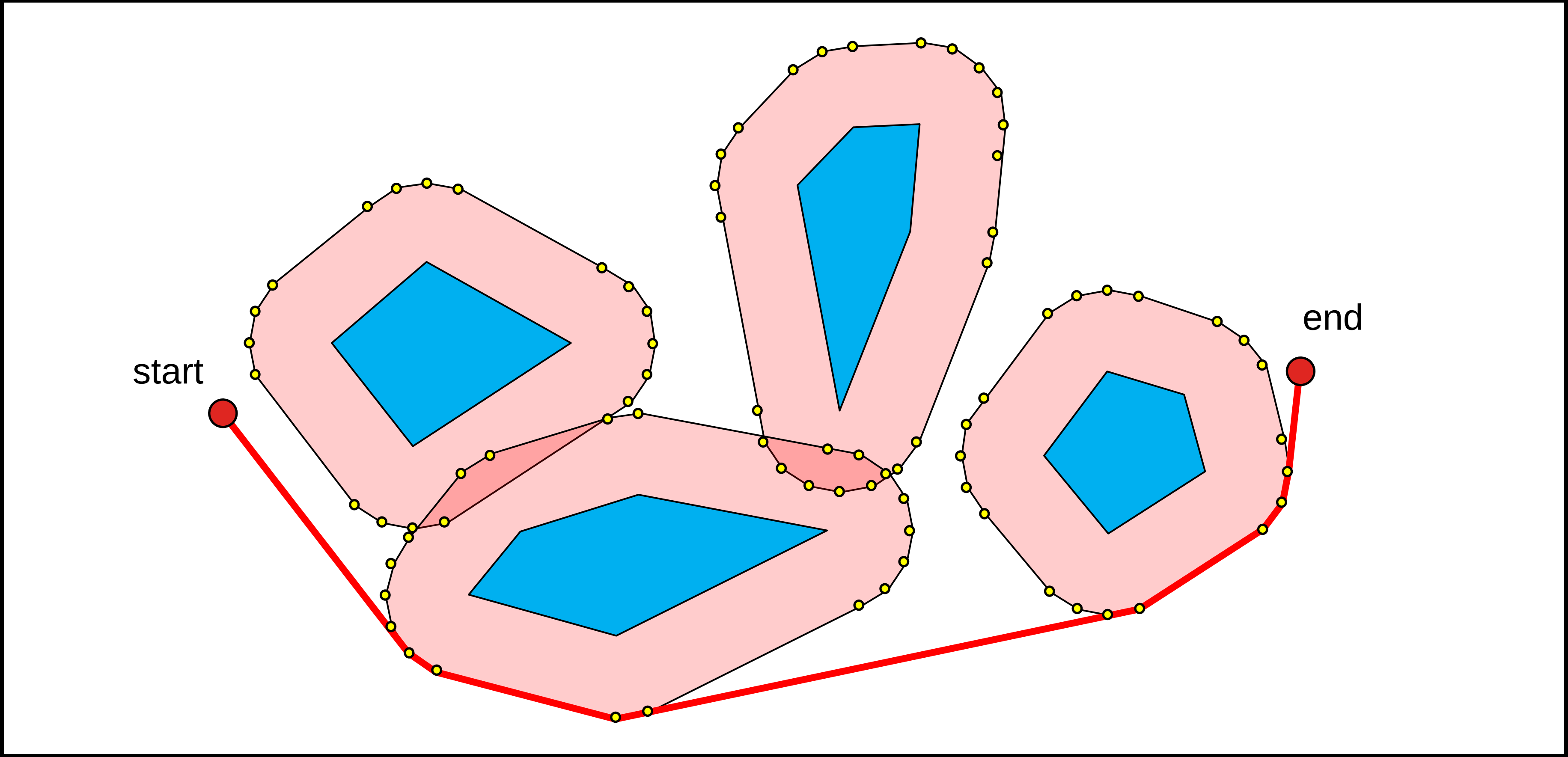
Real Robot Shortest Path

- Now the robot has collision-free travel



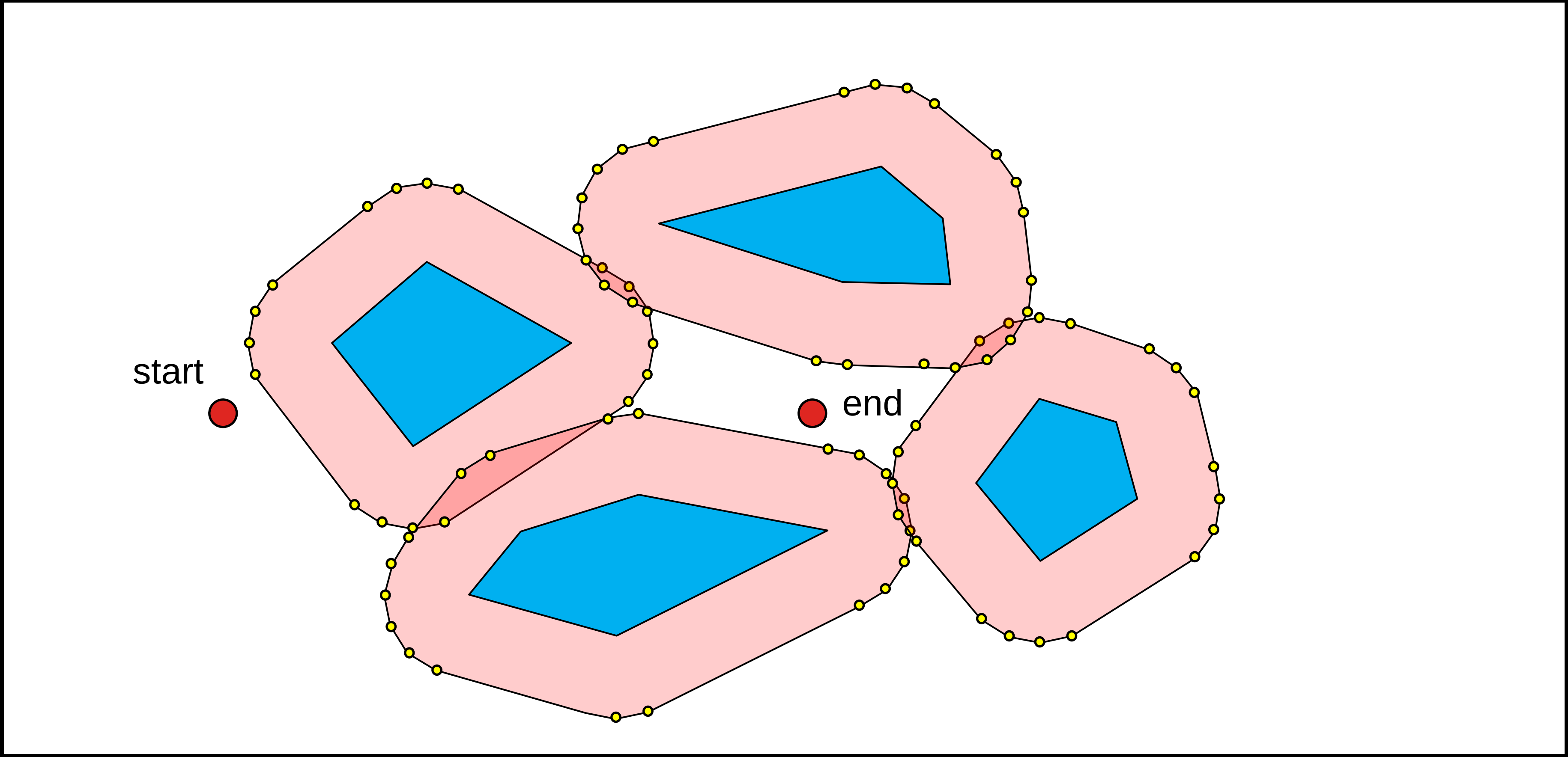
Real Robot Shortest Path

- In some cases, the grown obstacles will intersect, resulting in a different solution path.



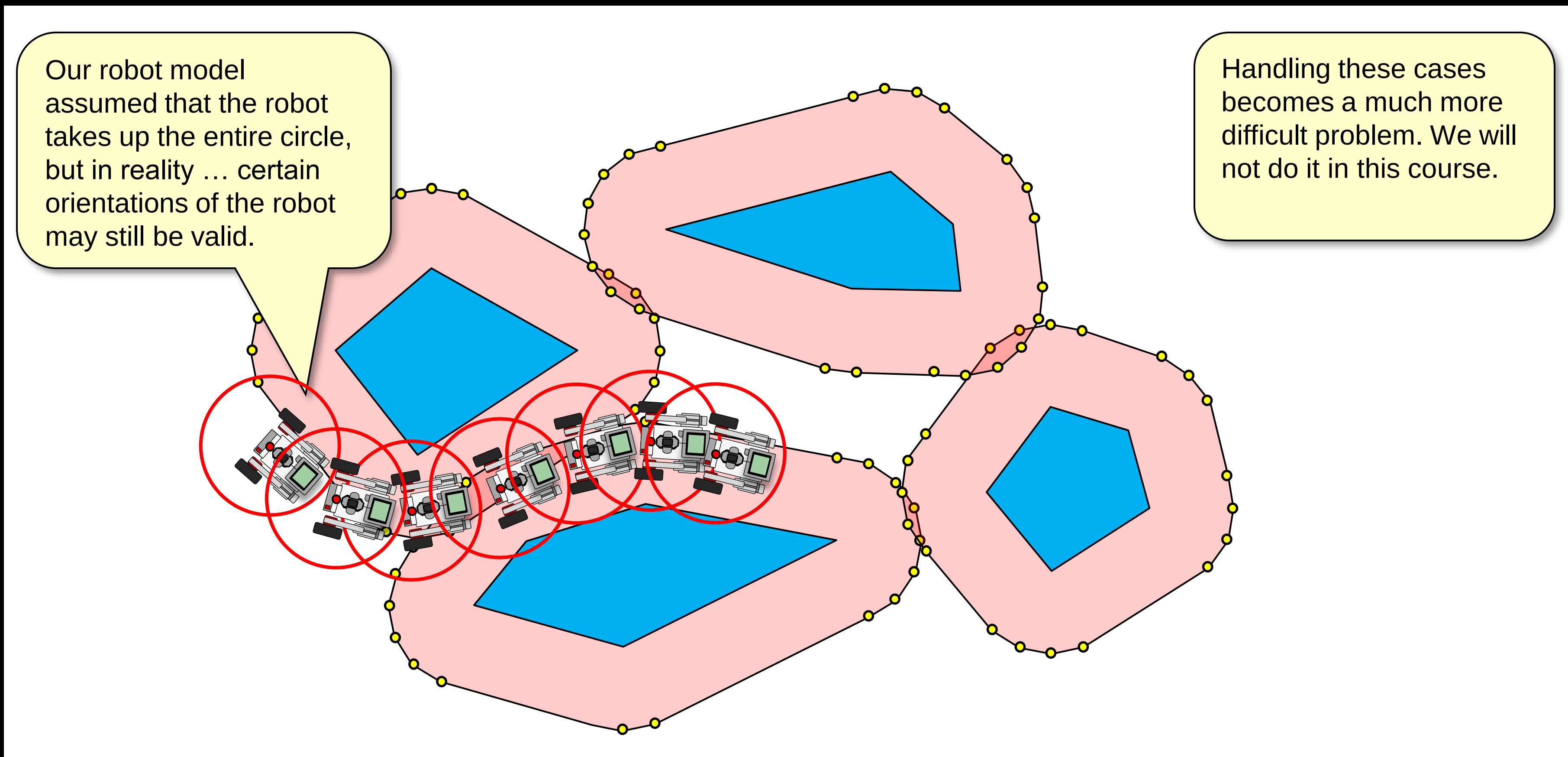
Real Robot Shortest Path

- In some cases, there may even be no solution !



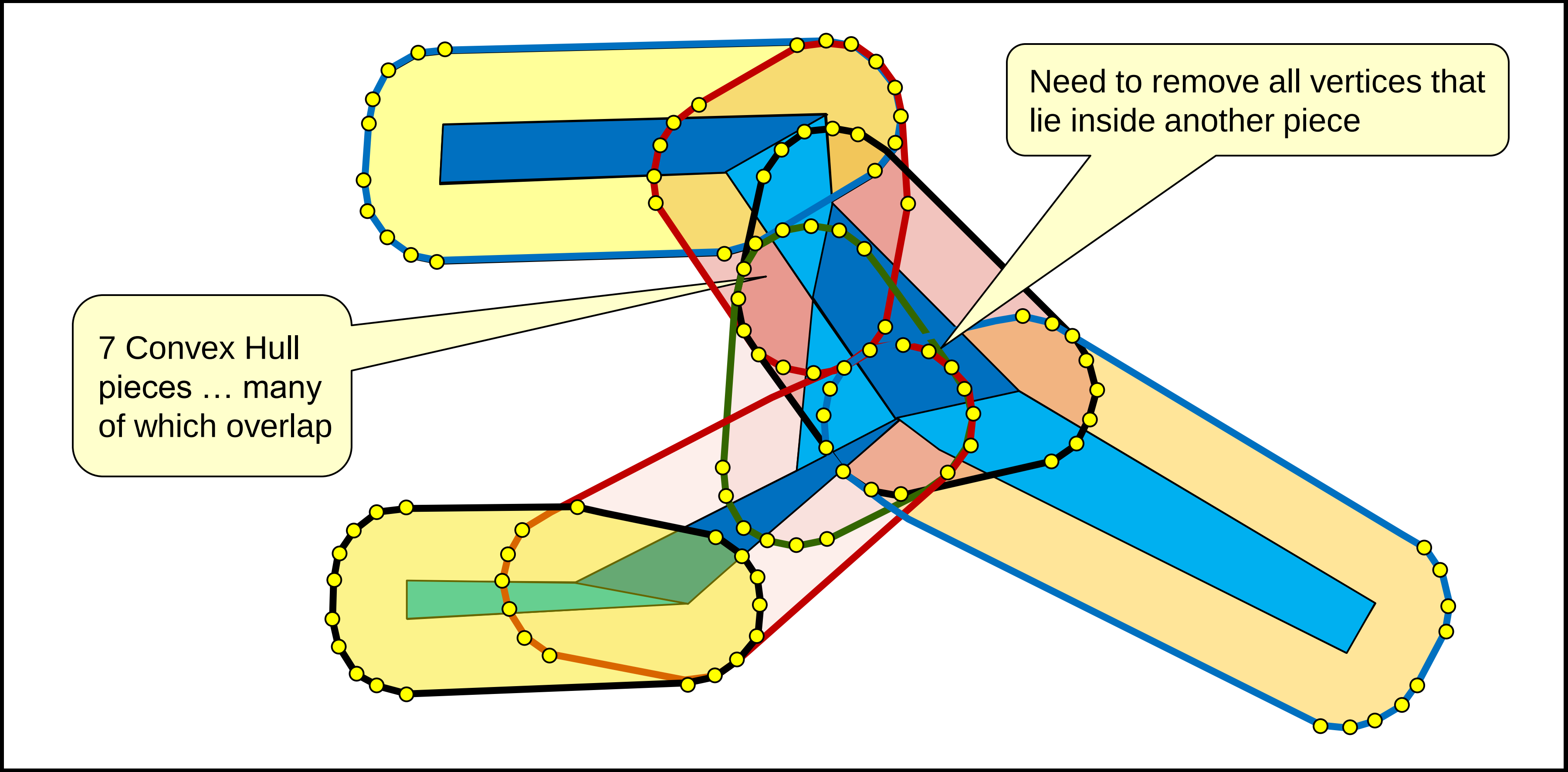
Real Robot Shortest Path

- However, this is a restriction on our robot model and algorithm, as actual robot can “fit”:



Non–Convex Solution

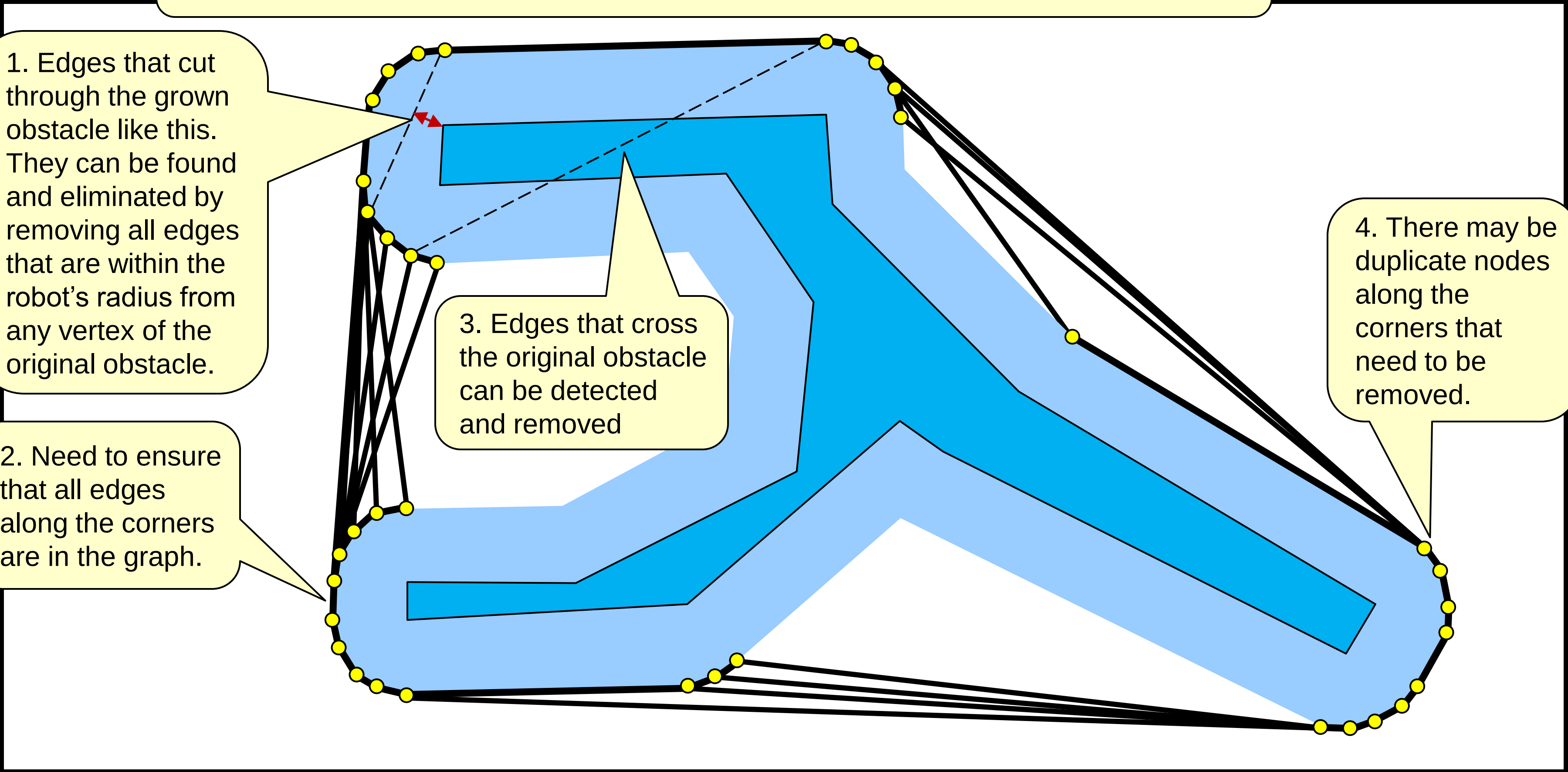
- Compute Convex Hull for each piece:



Non–Convex Challenges

- Compute visibility graph … but it is tricky:

Unfortunately, there are a lot of intersection-related problems that pop up due to the overlapping of the various convex pieces. This will require the coding of special cases.



1. Edges that cut through the grown obstacle like this. They can be found and eliminated by removing all edges that are within the robot's radius from any vertex of the original obstacle.

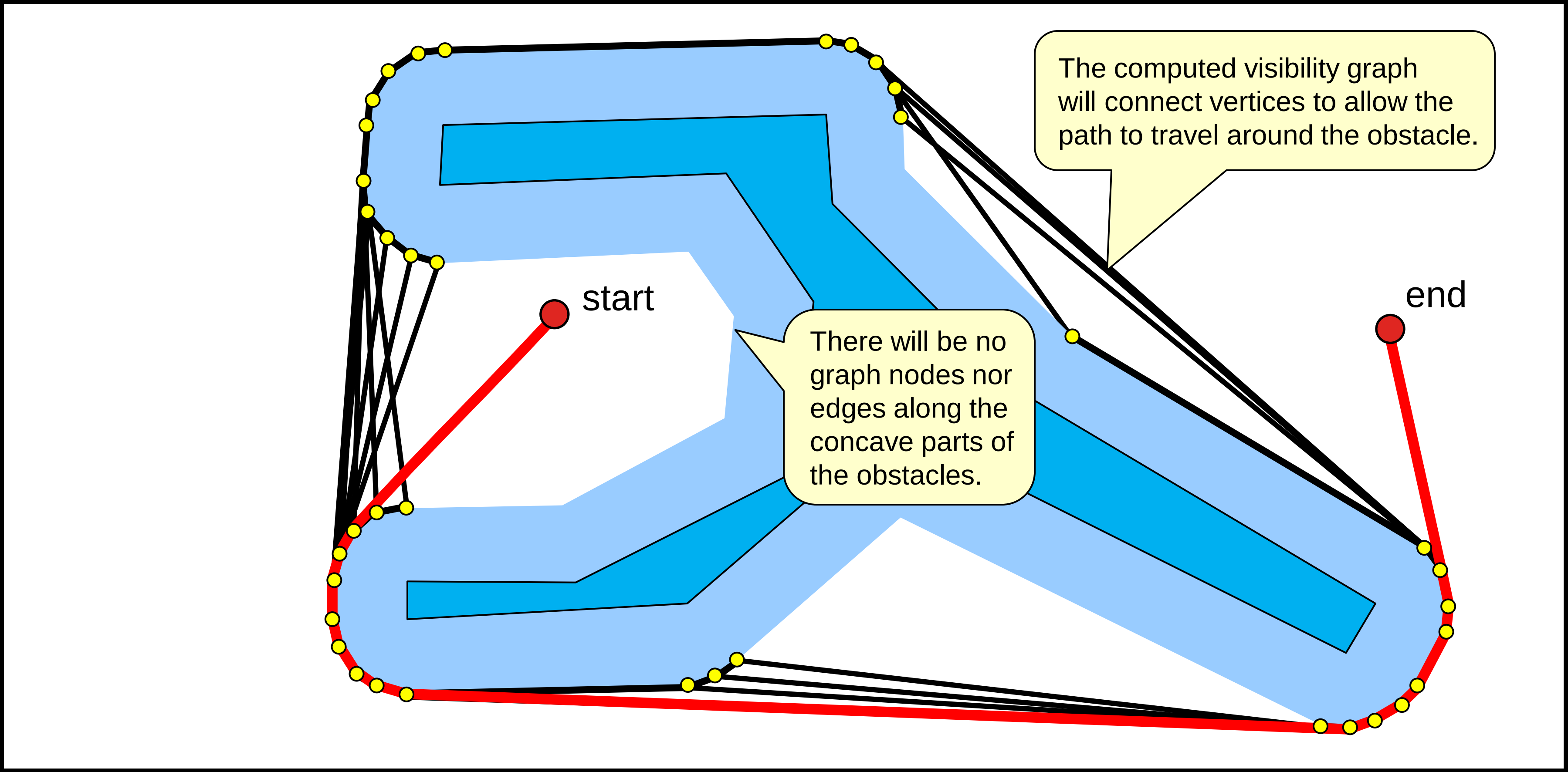
2. Need to ensure that all edges along the corners are in the graph.

3. Edges that cross the original obstacle can be detected and removed

4. There may be duplicate nodes along the corners that need to be removed.

Non–Convex Shortest Path

- If done correctly, the shortest path can be found:



Start the Lab …