The navigation stack can be visualized as an entity that takes in information from sensors like odometry and goal pose, and outputs safe velocity commands. Simply put, it performs the task of navigating a robot successfully. Information from sensors allows it to map the outside world and determine a safe path. These instructions are then sent to a mobile base like a robot.

Uncertainty creeps into navigation, primarily due to two reasons:

(i) Uncertainty in prediction
Although an action can determine motion of direction, the surrounding introduce error during execution

(ii) Uncertainty in sensing
Due to noise in sensor data and other factors, the state of the world cannot always be reliable estimated

BACKTRACKING

One way to refine the navigational stack is to algorithmically calculate an effective path, if the robot has access to perfect but uncertain information (can perceive all surroundings, but error is introduced).

Let us define a function f(P), that defines a region R for every point P, such that the robot can surely reach P from any point in R. Starting from the goal, we consider the region f(P), choose a point in R, and recursively repeat.

(the function f is commonly called 'backprojection')

We know that each point B is reached only from a point in f(B), that is, a point from which we can reach B even under uncertainty. Thus, by induction, we can prove that the path found can be traversed even under uncertainty.

We can call this method 'backtracking', as we start from the goal and make our way to the initial point.

IMPROVEMENTS:

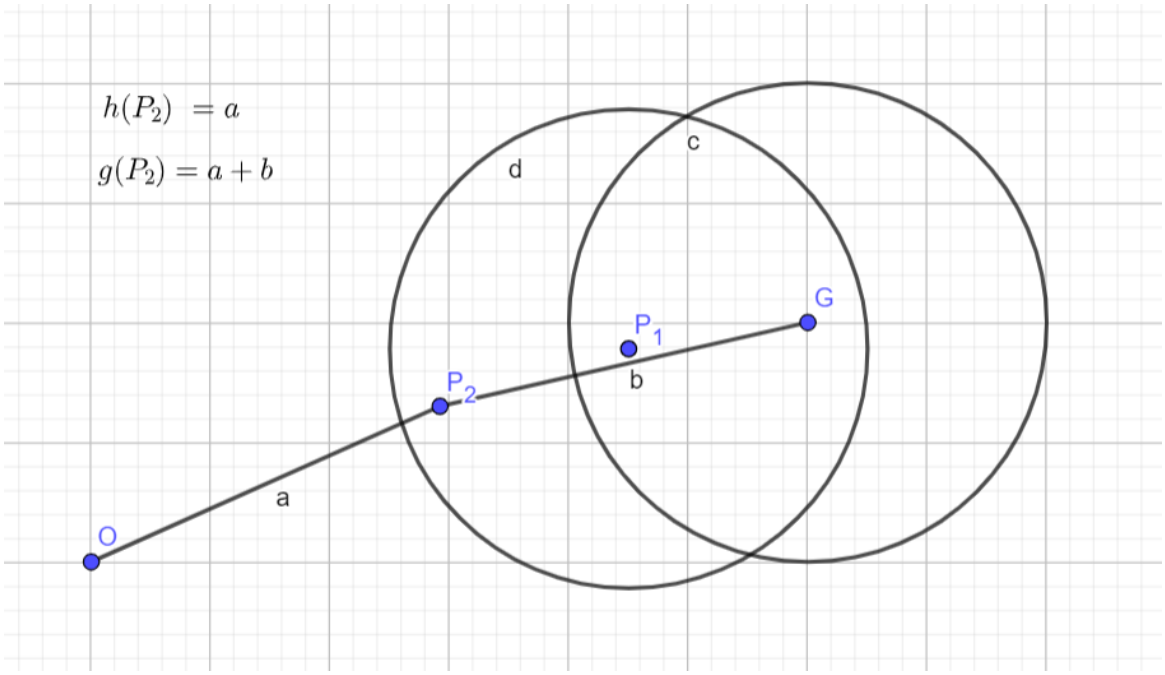
1. Incorporate heuristics in search

If we have information about surroundings (like coordinates of starting point and goal), we can also define heuristics like

(i) h(P) = distance of P from O

to implement greedy best first search

(ii) g(P) = distance of P from O + distance of P from G
to implement A* search



2. Although the sensing is uncertain, the surrounding will contain a few 'landmarks', positions that can be sensed for certainty (a large rock, for example). Hence, we can split the path into moving from one 'landmark' to the other, and perform backtracking on the sub-paths.
Since the algorithm, without any heuristic or improvement is of complexity $O(n^n)$, reducing distance of path for backtracking greatly reduces time taken.