

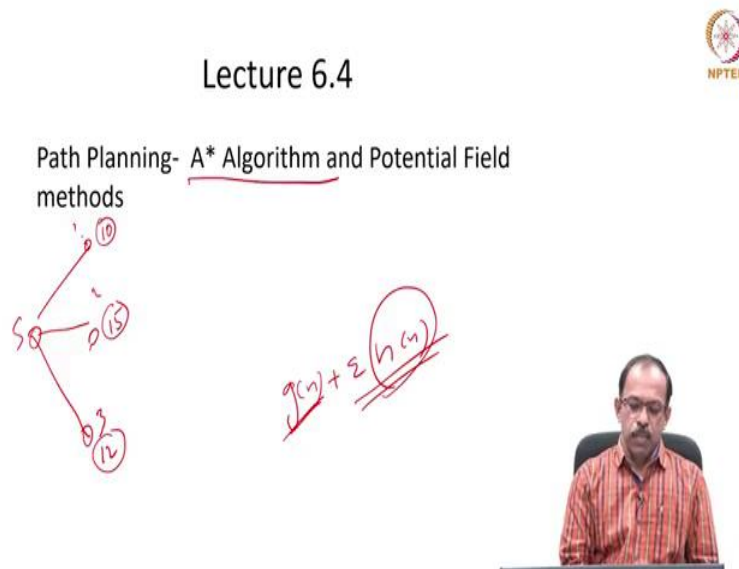
**Wheeled Mobile Robots**  
**Prof. Asokan Thondiyath**  
**Department of Engineering design**  
**Indian Institute of Technology, Madras**

**Path Planning and Obstacle avoidance**  
**Lecture - 6.4**  
**Path Planning - A\* Algorithm and Potential Field methods**

(Refer Slide Time: 00:13)

Lecture 6.4

Path Planning- A\* Algorithm and Potential Field methods



Hello everyone, welcome back to the discussion on Path Planning of Mobile Robots. So, in the last class, we discussed about some algorithms for used for graph search in order to find out the optimal path for the robots. So, we talked about two method – one is the grassfire algorithm which assumes that the cost of travel from one node to the other node is equal.

And then if you have this breadth first approach, then you will be able to find out the cost of travel from the start to the goal node and you can find out the lowest cost of the path which actually provides you the lowest cost. And if the travel cost from one node to the other node is not constant, then we can go for the Dijkstra algorithm which will follow the breadth first approach.

But try to find out the lowest cost based on the individual node cost and expand those that lowest cost node, and then continue the breadth first approach, and then reach the

goal node, and find out the lowest cost which with which the goal node can be reached from the start nodes, so that is basically the Dijkstra's algorithm.

So, today we will look into a few other methods one is known as the A\* algorithm, and then we look at the other methods of potential field method for planning also. So, one of the when we discussed about this graph search, I talked about this cost function. So, cost function was given as  $g(n) + \epsilon \times h(n)$  ok. So, when there is this h is basically the heuristic cost. And in the previous two methods we did not really go for any heuristic cost, we just simply looked at the travel cost from one node to the other nodes.

Now, if you want to add a heuristic cost to the planning to the graph search algorithm, then you will be able to get a better method of cost estimation of the paths. So, why we need this is to ensure that we have a better way of expanding the nodes. Because in the Dijkstra algorithm what we do is, we look at the lowest cost node and then try to expand the lowest cost nodes.

But if you have some way if we know that lowest cost node is not the one which gives you the lowest I mean the lowest cost to the goal, then we can actually expand the node which actually will be potentially will be the best node to expand, so that is basically we need to have some kind of a heuristic estimate of the travel cost from all these nodes to the goal node, and then we will be able to find a better way of search.

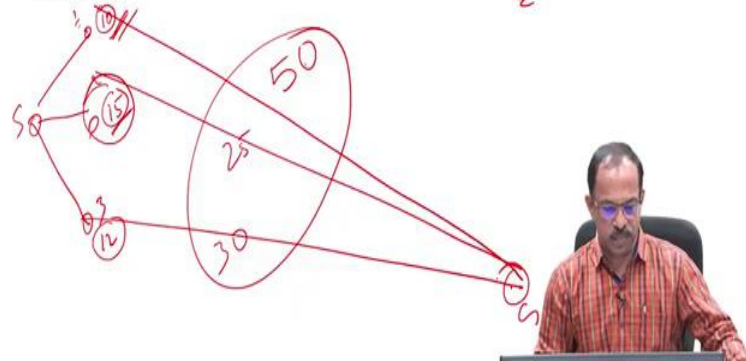
To explain that, suppose you have the start node here, and you are expanding all the neighbouring nodes 1, 2, 3. And you find out the cost of travel to 1, 2, 3 in the Dijkstra method. And you find out that this cost may be 10, this may be 15 and this may be 12. So, what we do is we simply expand this 10. But we do not know whether this 10 will be the best to reach there or 10, if you take the 10, this node one the cost of reaching the goal may be slightly more.

(Refer Slide Time: 03:43)

## Lecture 6.4



### Path Planning- A\* Algorithm and Potential Field methods



So, if you have some kind of an estimate to say that ok. So, this is the goal. This is the goal point. If you have some kind of an estimate saying that ok, from here to from 3 to goal, or from 2 to goal, or from 1 to goal, if we have some idea of which one will be the shortest path from here to reach the goal, then we can expand that one first instead of expanding the this one.

Suppose, this from 1 to a goal is the cost of travel will be 50, and this may be 25, and this may be 30. Some, if you have some kind of an estimate like that, then we can look at if we are here the total cost will be  $15 + 25 = 40$ , this will be  $10 + 50 = 60$ , this is  $12 + 30 = 42$ .

So, we have some kind of an idea that this goal may be the this node may be the best to explore further, because the total cost of travel from start to goal seems to be less from this route compared to this route, so that is the heuristic that we try to add. So, we can have different kinds of heuristics.

One may be the distance from there to here, or one may be looking at the obstacles in that path or some kind of heuristics can be included, and then we can expand the node, so that is basically the principle of A\* algorithm ok.

(Refer Slide Time: 05:07)

## A\* algorithm



Expected total cost is a function of path cost and heuristic cost. Heuristic cost is taken as the distance of any cell from the goal in absence of obstacle cells.

A\* search begins by expanding the start node and placing all of its neighbors on a heap. In contrast to Dijkstra's algorithm, the heap is ordered according to the smallest cost value that includes the heuristic function. The lowest cost state is then extracted and expanded. This continues until the goal node is explored. The lowest cost solution can again be backtracked from the goal.

What A\* Search Algorithm does is that at each step it picks the node according to a value 'f' which is a parameter equal to the sum of two other parameters – 'g' and 'h'. At each step it picks the node/cell having the lowest 'f', and process that node/cell.



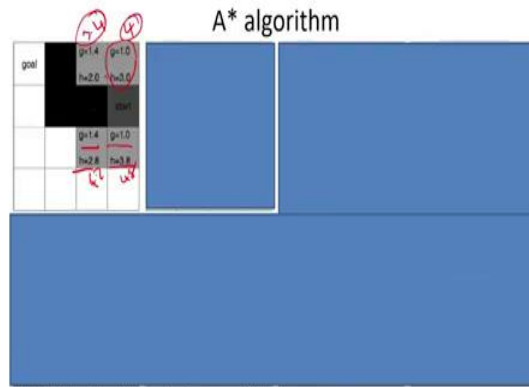
So, the expected total cost is a function of path cost and heuristic cost. So, its not only the path cost to that node we will add, we will add a heuristic cost also and then try to expand it that is basically the A\* algorithm. So, it begins by expanding the start node and placing all of its neighbours on a heap. Similar to the Dijkstra, it will do the same thing.

But in contrast to Dijkstra's algorithm, the heap is ordered according to the smallest cost value that includes the heuristic function also. So, it is not only check the cost of travel from the start to the next node, you look at the cost of travelling from that node to the goal node or a heuristic cost also will be considered when doing this.

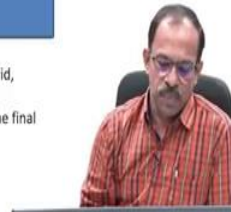
And then the lowest cost is extracted and expanded further, so that is the way how the A\* algorithm works. So, the basic principle remains same. The breadth first approach, the cost of travel may be different from node to node, but the only thing is that you add an heuristic cost to make the search much more faster and more efficient that is the A\* algorithm.

So, what that does is that at each step, it picks the node according to a value f that is the cost which is parameter equal to the sum of two other parameters – that is the g and h. So, g is the node cost and h is the heuristic cost. And it picks the cell having the lowest f and process it further, further expansion. So, we will take an example and then explain this.

(Refer Slide Time: 06:42)



We define 'g' and 'h' as simply as possible below  
**g** = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.  
**h** = the estimated movement cost to move from that given square on the grid to the final destination.



So, look at this as the graph that we have. And we need to start from here. So, this is the start nodes, and this is the goal nodes. And we assume that it can actually travel in whatever way you want the direction and we can calculate the cost of travel based on the movement. And it need not be the same cost also; it can be different cost also, ok.

So, now, it will see what it will do it will expand all the nodes to look at the all the neighbouring nodes and see what is the cost of travel to that. So, first one is g. So, g will be the cost of travel from start node to the next node. So, we will say  $g = 1$ ;  $g = 1$ , because it can travel from here with the one cost of 1. And then it can travel from here with the cost of 1.4 ok, because it is a square root – the root 2 is the 1.4. And from here this is 1.

So, we assume that if the robot is in this node, it will take 1.4, the cost is travelled to from start to sorry if it start from start the robot is in start the cost of travel to this node will be 1.4, this will be 1, this will be 1, this will be 1.4 that is the g. But in addition to g, it will try to calculate a heuristic cost of travel from that node to the goal nodes that is the heuristic cost.

So, it will take a heuristic cost of 3 that is if the robot is in this node, how much will be the cost of travel from this node to the goal? So, you do not take care of the obstacles and all it will simply look at how much will be the distance or the travel cost if there are no obstacles or anything I mean that is one kind of heuristic that you can use. You can have

your own heuristics to decide how do you want to use it ok. So, this is  $h = 3$ , because it has to travel 1, 2, 3 to reach the goal.

Similarly, from here it has to travel 2 point  $1.4 + 1.4$ . So, 1.4 one point this is from here it is 1.4, another 1.4, then 1. So, 2 point, 3.8 will be the heuristic cost. Yes, same way this will be 2.8, because it has to travel 1.4 then 1.4, so 2.8 will be the heuristic cost. So, this way it will calculate the heuristic cost for all the nodes that is what is shown here  $g(1)$ ,  $h(3)$ ;  $g(1.4)$ ,  $h(2.0)$ ;  $g(1.4)$ ; 1.0, 3.8. So, we have a  $g$  and  $h$ .

Now, it will find out what is  $g + h$  for each node. So, it will find out what is  $g + h$  for this will be 4, this will be 3.4, this will be 4.8, and this will be 4.2. So, it will find out what is  $g + h$  for each node, and then find out which one is the smallest one. So, it will look at this is the smallest one is 3.4. And 3.4 cannot be expanded further because there are no neighbouring nodes which are free, there is no travel node. So, it cannot be expanded.

So, look at which one is the next one, so next one is 4. So, 4 cannot be expanded, again all the neighbouring nodes are explored. So, the next one is 4.2. Now, it will try to find out 4.2. Expand 4.2, and find out what will be the cost of travel from that node to the neighbouring nodes ok.

(Refer Slide Time: 10:25)

**A\* algorithm**

We define 'g' and 'h' as simply as possible below  
**g** = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.  
**h** = the estimated movement cost to move from that given square on the grid to the final destination.

Dijkstra is a special case of A\* Search Algorithm, where  $h = 0$  for all nodes.

So, this is the ok. So, this is the 4.2. So, now, this node will be explore further ok. So, it will try to explore this, I mean this node, this node, this, all these four nodes will be

explored further, and we will try to find out what is the  $g$  and  $h$ . So, this  $g$  is basically the total cost of travel from start to this node, ok. So, this 1.4 up to here and from here it is 1, so 2, it will be 2.4. 1.4 up to here, another 1.4 here, so it will be 2.8;  $g$  will be 2.8. So, 1.4 plus 1,  $g$  will be 2.4.

This will be 1.4 and again 1.4, so 2.8. So, all the  $g$  will be calculated. So, the  $g$  is the total cost of travel from the start to that node the total node cost. Then again same principle we go for the  $h$  also heuristic cost, how much will be the cost of travel from these nodes to the goal nodes. So, you need to go like this and reach. So, 3 into 1.4, so it will be 4.2 will be the heuristic cost. Same way we can calculate the heuristic cost for each one of these 3.4, 2.4.

So, this 2.4 is coming because of 1.4 from here to here, and then 1 here, so 2.4. So, this way you will be able to get all the heuristic cost and the  $g$  here. So, the movement cost and heuristic cost. Now, you look at which one is the smallest one. So, this is 4.8, this is 4.2, so this is 4.2, this is 7, 7.2, this is 6.2, this is 6.2, 4.8 ok. So, you will be expanding all these four point. So, this one is already expanded. So, you do not.

So, now you look at which one is the smallest and try to expand and as you can see here these are all expanded nodes. So, you cannot do further. So, you will expand the 4.8 node, and then find out what is this. So, this node will be sorry this node is now further explored, so all the neighbouring nodes.

So, all these neighbouring nodes will be explored to find out the travel cost and the heuristic cost. So, you will see that all those things are obtained. So, here this  $g = 3.8$  and  $h = 1$ . And finally, you explore this because the this will be the last one because you have this 4.8, it will be expanded to the next one.


So, you will get  $g = 4.8$  and goal is  $h = 0, 0$ . And finally, you get that is the shortest I mean the lowest cost that you have. So, this way the node exploration becomes much more faster and you will be able to find a behind the shortest path much faster. So, you will be finding that this is the shortest path that you can have in order to reach the goal from the start point ok. So, this is basically known as the A\* algorithm.

So, after the A\* algorithm was introduced, we had many more algorithms to explore further. So, we have modified A\*, and A\*\*, many, many algorithms are there if you want to search, you will find that there are many such algorithms existing for path plan.

So, we will not go into the details I just want to give you a brief overview of some of the methods available for graph search alright. So, we can say that Dijkstra is a special case of A\* where  $h = 0$ . So, otherwise it will be almost the same ok. When  $h$  is equal to when it is not equal to 0, you have the A\* algorithm ok. So, that is about the A\* algorithm.

(Refer Slide Time: 14:43)


Potential field path planning



The potential field method treats the robot as a point under the influence of an artificial potential field.

The robot moves by following the field, just as a ball would roll downhill.

- The goal acts as an attractive force
- Obstacles act as repulsive forces.
- The superposition of all forces is applied to the robot, which, in most cases, is assumed to be a point in the configuration space.



So, let us look at the potential field path planning also. So, I mentioned that there are many methods for path planning. So, potential field planning is one methods used for path planning and in many cases for obstacle avoidance also. So, the point here is that the potential field method treats the robot as a point under the influence of an artificial potential field.

So, assume that there are some artificial potential fields are there in the arena. And assume that the robot moves by following the field just as a ball would roll downhill. So, wherever there is a gradient, it will try to follow, so that is basically the principle here. So, the goal acts as an attractive force.


So, we assume that the goal has got some kind of a potential to attract the robot towards that. So, it has a attractive force. And then obstacle has got something called repulsive



forces. So, the robot will be repulsed by the obstacles, and attracted by the target, so that is the way how it is assumed.

And the superposition of all forces is applied to the robot and in both cases assumed to be a point in the configuration space. So, we assume apply these forces into the robot, and then find out which whichever there is a wherever there is a gradient of force or there is a smallest force existing, it will try to move along that path, so that it will reach the goal. So, that is the basic principle of a potential field path planning ok.

(Refer Slide Time: 16:23)



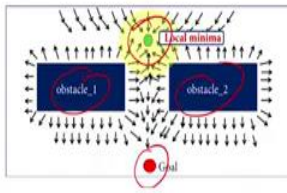

$$f_{total} = f_{att} + f_{rep}$$

$$f_{att} = k_{att} \frac{r_{goal} - r}{|r_{goal} - r|}$$

$$f_{rep} = \begin{cases} -k_{rep} \sum_{i=1}^n \left( \frac{1}{d_i} - \frac{1}{d_{max}} \right) s_i, & \text{if } d_i < d_{max} \\ 0, & \text{otherwise} \end{cases}$$

where  $s_i = (r - o_i) / |r - o_i|$ .

$r_{goal}$  is the position vector of the goal point and  $r$  is the position vector of the robot  
 $o_i$  is the position vector of each obstacle.

So, you have an obstacle. And you have an obstacle 1, and you have an obstacle 2. This is the goal. And then there is a robot here somewhere the robot is here, then it will try to get repulsed by the obstacle and then attracted by the goal. So, it will try to move, so that is basically the principle. So, we assume that the total force is attraction plus repulsion. And attraction potential can be defined by some relationship.

So, there are some standard relationship to define these kind of forces because that depends on the distance to the goal, or from the robot how far the robot is from the goal. So, we based on that, we will define this function. And similarly a repulsion potential also can be defined based on the distance that is the obstacle is within the within a particular radius of the robot, then we consider that as a repulsive force.

If it is far away, we do not consider, so that is why we have a relationship like this, where  $k$  will be the coefficient can be or a constant, that can be a decided by the designer depending on how fast or how close you want this to be, or how fast you want the robot to reach the goal, all those things you can actually define a constant.


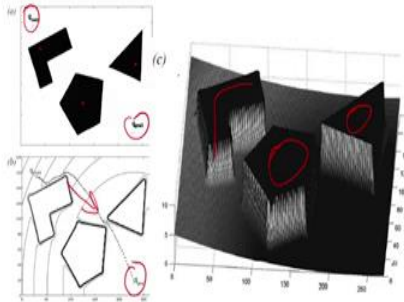
So,  $r_{goal}$  is the position vector of the goal point and  $r$  is the position vector of the obstacle ok, so that is basically how we define. And then we will have some kind of a function potential function and then we look at the total potential acting at the on the robot and based on that, we can actually move ok. So, we need to find out the points where actually the robot can take the next position or we will try to find out the minimal point and then the robot will move to that minimal point.

So, one problem with that is sometimes it will get into a local minima. So, robot will see that there is no other point which can actually move because the current position is the minimal. So, it is not able to move forward, so that is one problem with the potential field function where actually a potential field method where the robot may get into a local minima.

But there are many strategies to avoid the local minima or to overcome the local minima, so if you go through the literature if you are interested you can further go through the literature and then try to find out various methods of potential field application for robot path planning.

(Refer Slide Time: 18:55)

Potential field path planning



The slide titled "Potential field path planning" features three diagrams labeled (a), (b), and (c). Diagram (a) shows a 2D environment with several black polygonal obstacles and a red path starting from a goal point and navigating around the obstacles. Diagram (b) shows a similar 2D environment with a different set of obstacles and a red path. Diagram (c) is a 3D surface plot representing the potential field, where the height of the surface indicates the potential energy. The surface has peaks at the locations of the obstacles and a valley leading to the goal point. A red path is shown on the surface, starting from the goal point and moving towards the obstacles, illustrating how the robot might get stuck in a local minimum.

So, to represent graphically you can actually show that there is a goal here there is a start point and these are all the obstacles. So, we assume that the obstacles are having some field like this. These are the potential fields. And then there is an attraction potential for the from the goal towards this for the robot. And it will check all this field – total field, it will be robot will calculate based on its position what is the total field acting, and then go for the minimal point and then move towards that point ok.

(Refer Slide Time: 19:29)

**Multipoint potential field method for path planning and Obstacle avoidance in 3D space**

NPTEL

1/13/2021 Intelligent Service Robotics: Volume 6, Issue 4 (2013), Page 211-224, 51

This is just for your information because this is some of the work that we did using potential field method for path planning of mobile robots. This is a we actually developed a method called multipoint potential field methods for planning path planning and obstacle avoidance. So, I can actually see this paper in this journal Intelligent Service Robotics.

So, in this, we assume the robot to have instead of having a single point, there is robot has a single point, we assume it is a it is a 3D, it has got a 3D shape, and then we divide this into multiple points and then identify the potential field in the space. And even the obstacles will be contrast having multiple points, and then solve the problem. So, this is just for your information. If you are interested, you can actually refer to the paper, and then learn more about the multipoint potential field methods.

(Refer Slide Time: 20:29)

$\psi_{obs} = \psi_s \pm 90$   
 $\theta_{obs} = \theta_s \pm 90$

$\psi_{obs}$  - horizontal beam width of sensor  
 $\theta_{obs}$  - vertical beam width of sensor

Repulsive potential  $U_{obs}$  at  $q_i$  due to the obstacle point  $p_j$ :

$$U_{obs_j}(q_i) = \begin{cases} \frac{1}{2} \eta \left( \frac{1}{d_{obs_j}} - \frac{1}{d_i} \right) d_{obs_j}^2, & \text{if } d_{obs_j} \leq d_i \\ 0, & \text{if } d_{obs_j} > d_i \end{cases}$$

[Ge and Cui (2000)]

$d_i = |q_i - p_j|$  is the distance between  $q_i$  point and  $p_j$  point on the obstacle,  $d_i$  is the distance influence threshold  
 $\eta$  is a positive scaling factor

1/13/2021 SSK,IITM 52

(Refer Slide Time: 20:35)

### Simulation results

Static obstacle environment

SP (0,0,0) GP (90,90,90)

53

So, let me skip this. And now we talk about, these are some of the simulation results from the multipoint potential field method. So, you can see that whatever may be the dimension of the obstacle, whatever maybe the size of the obstacle, you still will be able to get a safe path from the start to the goal using this multipoint potential field method.

Even moving obstacles can actually be considered in this case, but it is more of a offline path planning method where we can actually if we know the size of the obstacle and the distance, then we will be able to calculate the potential field.

(Refer Slide Time: 21:15)



## Obstacle Avoidance

Local obstacle avoidance focuses on changing the robot's trajectory as informed by its sensors during robot motion. The resulting robot motion is both a function of the robot's current or recent sensor readings and its goal position and relative location to the goal position.



So, as a last topic let us talk about the obstacle avoidance also. So, far we talked about path planning. Now, we will talk a little bit about obstacle avoidance in for mobile robots. So, the local obstacle avoidance focus on changing the robots trajectory as informed by its sensors during robot motion.

So, there is when you need to want to you want to do an online obstacle avoidance, then the robot needs to change its trajectory based on the information it gets from the sensors. So, the resulting motion is both a function of the robots current or recent sensor readings and its goal position and relative location of the goal position.

So, the robot needs to plan its path and then or the robot needs to replan its trajectory based on the position of the obstacle, its target the goal position as well as the current position of the robot, so that is basically the requirement in obstacle avoidance. So, it requires kind of planning the path and as well as avoiding the obstacle. Again there are multiple methods available in the literature. We will not go through all those methods, just want to tell you one or two methods normally employed ok.

(Refer Slide Time: 22:36)

## Bug Algorithm

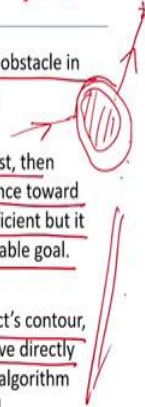
Bug 1



The basic idea is to follow the contour of each obstacle in the robot's way and thus circumnavigate it.

With Bug1, the robot fully circles the object first, then departs from the point with the shortest distance toward the goal. This approach is, of course, very inefficient but it guarantees that the robot will reach any reachable goal.

With Bug2 the robot begins to follow the object's contour, but departs immediately when it is able to move directly toward the goal. In general this improved Bug algorithm will have significantly shorter total robot travel



So, one of the earliest and commonly used one is known as the Bug algorithm. So, Bug algorithm is used to avoid the obstacles. And the basic idea is to follow the contour of each obstacle in the robots way and then circumnavigate it. So, that is one strategy proposed very early that is if the robot is moving in one direction, and there is an obstacle. So, you what you do and you know the goal position is somewhere here.

And this was the trajectory, it was following. Then what it will do? It will try to follow like this, and then go all the way and then find out all the distance from here each point it will try to find out how far it will be to reach the goal. And then it will find out the minimal point. And then from that minimal point, it will try to jump to the next path, so that is basically known as the Bug algorithm.

So, in there are different Bug algorithms whether we talk about the Bug1 algorithm which was initially first proposed Bug1 algorithm. So, the Bug1 algorithm the robot fully circles the object first, then depart from the point with the shortest distance toward the goal.

So, it will try to circumnavigate the obstacle and then from each point as it moves around the goal, it will try to find out how far will be the distance from this point to the goal point. So, that will keep calculating and find out the minimal point, and then it will jump from that minimal point to the target. So, that is basically the Bug1 algorithm.

It is very inefficient, but it guarantees that the robot will reach the reachable goal. So, always ensure that the robot can reach the goal by this method. But it is very inefficient

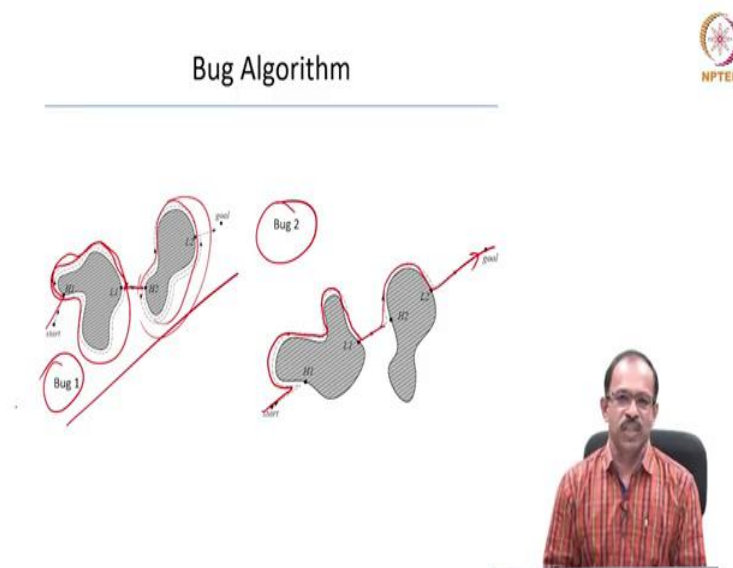
because it has to circumnavigate the whole obstacle which may be a very difficult task. So, it is not commonly used, but that was proposed initially.

Now, to overcome that a new algorithm was proposed which is Bug2 algorithm. In Bug2, the robot begins to follow the objects contour, but departs immediately when it is able to move directly toward the goal. So, it would not check for whether it is a minimal distance or not. So, any point it is able to go directly to the goal it will start moving towards that goal without completely navigating around the object, so that is basically the Bug2 algorithm ok.

So, it will be significantly shorter than the Bug 1 algorithm. So, these are the Bug1 and Bug2 are one of the algorithms I mean one of these two algorithms which are proposed for obstacle avoidance. And there are again you can avoid the obstacle using the multi potential field method also.

Because you can actually if you know the obstacle distance and size, you can create and also potential field artificial potential field can be created and that can be used for online obstacle avoidance. So, there are many methods. So, Bug1 and Bug2 are some one of the or Bug1 and Bug2 are the initially proposed algorithms for obstacle avoidance.

(Refer Slide Time: 25:46)



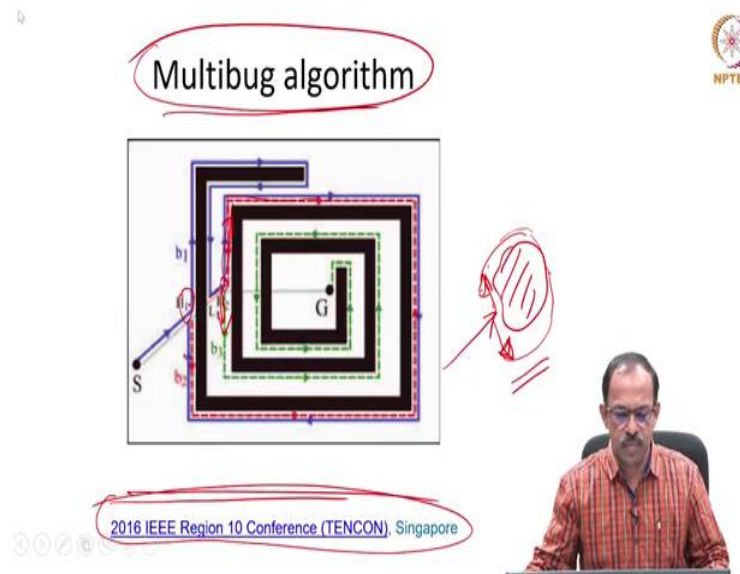
So, this is the Bug 1. So, it goes like this all the way one full round, and then comes, and then find out the minimal path and then try it. Again it passes if there is a, it will again go



like this ok. And in Bug 2, it will start from here go around the obstacle and it will see whether we can actually travel to the path in the it reaches into the direct entry to the goal, it will go here, and again it will just go through this, and then call this.

So, it need not circumnavigate the obstacle. So, this is the Bug 2 algorithm; commonly found algorithms very proposed very early. But there are now advanced methods are there, you can go through literature if you are interested more about it.

(Refer Slide Time: 26:34)



So, before closing let me tell you about an algorithm which we developed we called this as the Multibug algorithm. So, in Bug 1 and Bug 2 we had only one Bug and that Bug will be going around the robot, and then about the obstacle and around the obstacle, and then find out the paths that is basically considered as the Bug algorithm.

Now, we actually propose though this was proposed in presented in one of the conferences. So, we actually proposed a Multibug algorithm. So, in this case, what the proposal is that if you have an obstacle, if you have an obstacle, and if the robot is actually reaching here, so instead of going all the way one robot one bug going and finding out this, we actually try to create that there will be two bugs. So, one bug moves along this and then this is more like an offline obstacle avoidance.

So, we will have one this bug divided into two bugs will go around, and whenever it if they reach together at one point, then one of the bugs would be killed, and then the other




bug will continue otherwise it will look at easiest path from where it can go then it will keep moving like this.

So, this way and whenever there is an obstacle, it will actually divide into multiple bugs, and then move forward and then try to find out the path to reach the goal. So, that is basically the Bug algorithm. So, you can see here it divides into two, and then again here it divides into two, and it goes like this.


So, you will be having two bugs moving and reaching them. So, that is basically the Bug algorithm. So, this Bug algorithm if you are interested to know more you can actually refer to this paper, and you will be able to know more about it ok.

(Refer Slide Time: 28:25)

**Summary**



- Navigation of mobile robots
  - Path Planning
    - Graph Construction
      - Visibility Graph
      - Voronoi Diagram
      - Cell Decomposition
    - Graph Search
      - Breadth First
      - Depth first
    - Algorithms: Grass fire, Dijkstra, A\*
  - Obstacle Avoidance
    - Bug algorithm



So, that is all from I said regarding path planning and obstacle avoidance ok. So, we talked about the navigation of mobile robots. Primarily we looked at the path planning method. In path planning, we talked about graph construction and then graph search. And in graph search, we looked at the two methods, breadth search and breadth first and depth first. And then we talked about few algorithms – grass fire, Dijkstra, and A\* algorithm. And finally, we talked briefly about the Bug algorithm also.

So, with this I complete my teaching for this course. So, we talked about sensors and perception, we talked about localization, then we talked about a few algorithms which can be used for simultaneous localization and mapping also. And then finally, we

discussed about the path planning of mobile robots. We talked about the navigation of mobile robots which included the path planning and obstacle avoidance.

Hope you enjoyed these lectures. In case you have you want to know more or if you have any questions, please feel free to contact me. My contact details will be available to you from the SWAYAM site. So, please feel free to contact me for any clarification.

Thank you very much, and all the best. Bye.