Robot Motion Planning

Prof. Ashish Dutta
Department of Mechanical Engineering
Indian Institute of Technology – Kanpur

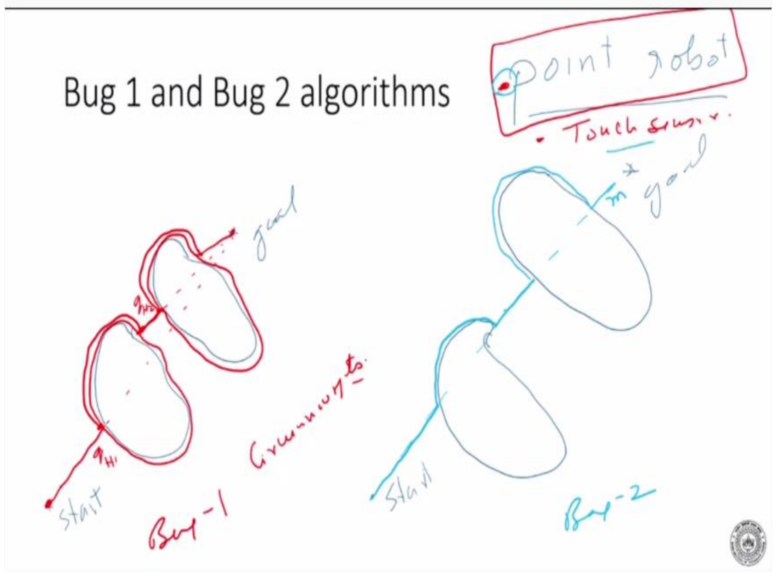
Lecture – 07
Configuration Space

Hello and welcome to Lecture number 7 of the course Robot Motion Planning. In the last class, we have looked at the bug algorithm in which we saw that the bug algorithm essentially is for a point robot where the robot moves towards the goal and in the process if it hits an obstacle it basically circumnavigates the obstacle or it finds the m line and avoids the obstacle and then goes to the goal.

So, today, we will move on to the next step. That is we have a bug now or a robot which also has a range sensor. So, from some distance, it can figure out where is the obstacle and then take corrective action. After that we will move on to the topic of configuration space. Now, in robot motion planning, this is something very basic that the robot path is planned in the configuration space and not in the Cartesian space.

So, today, let us very quickly revise what we are doing in the last class and then move on to the next topic which is configuration space. So, in the last class, we were looking at bug algorithm.

(Refer Slide Time: 01:17)



Bug 1 and Bug 2 algorithms

And, we saw the Bug 1 algorithm which is shown on the left hand side. And, we have a start point and we have a goal point. Now, couple of points to remember here, first is that the robot is a point robot. So, it is a point which is moving in space. Number 2, it has some kind, a sensor like a touch sensor. So, it has to touch the obstacle to understand that. That is an obstacle. And, how does the Bug 1 work? So, this is my Bug 1 algorithm.

So, the Bug 1 algorithm essentially starts by connecting the start point with the goal point. And then, the bug follows the straight line because it has to move towards the goal. And, once it touches the obstacle or hits the obstacle what it does is it circumnavigates the obstacle again comes back to this point. Next, it has to find the leave point. So, this point is called the hit point or we can call it q hit 1. Now, it has to find a leave point.

Now, the leave point is the point which is closest to the next obstacle or is nearest to the goal. So, in case, there is no obstacle it can go to the goal straight or if there is an obstacle, then go to the obstacle. So, let us say it has circumnavigated and come back to the hit point. Now, it will start this way again and go and find another point which is nearest to the next obstacle let us say, it is this point.

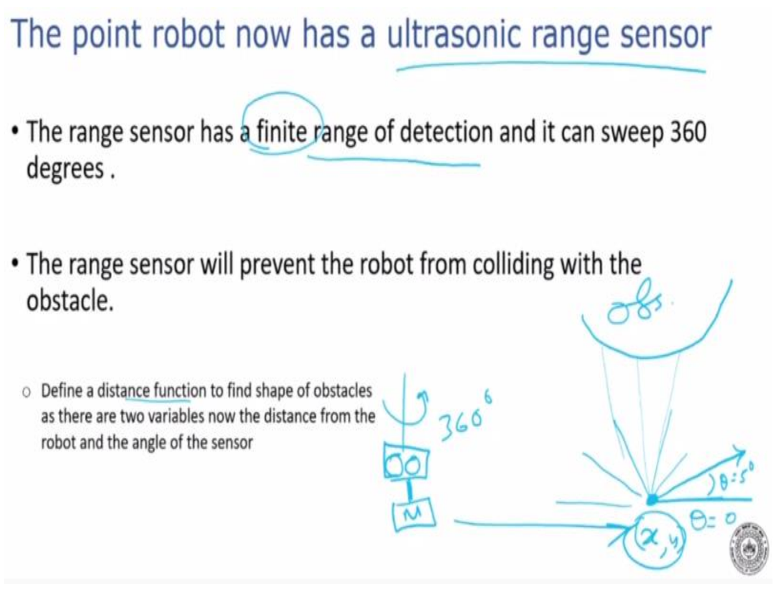
So, what it will do now is it will try and go towards the goal again. And then, it leaves from here and in a straight line towards the goal. It hits the next obstacle. Now, it is circumnavigates again comes back to here. And then, what it has to do is it has to find a point to leave. So, this is my q_{hit_2} . So, it has to find a point where it has to leave the obstacle and go towards the goal. So, maybe it finds a point here. It goes like this.

Finds this point and then goes there. Now, something about the Bug 1 is that it circumnavigates the obstacle. Now, in Bug 2 is a modification of Bug 1 which appears to be faster is that we have a start point and a goal point. So, this is my Bug 2. So, what it does is it connects the start and the end point and the goal point by means of a straight line which we call the m line here. And, it follows the m line until it hits an obstacle. Then, what it does.

It goes around the obstacle till it finds the m line again. The moment it finds the m line, it follows the m line. Goes around the obstacle like this, finds the m line and goes to the goal. Now, it would appear that Bug 2 algorithm is faster than Bug 1 algorithm. But this depends on the shape of the obstacle. This is something we have seen in the last class that in the case of a spiral very often the Bug 1 will succeed in finding a path and reaching the goal whereas the Bug 2 might fail.

So, it is very much dependent on the geometry of the obstacle. Now, 2 things that we talked about here, number 1 is that the robot is a point robot. The next, it has a touch sensor. So, that is how it finds out where is an obstacle.

(Refer Slide Time: 04:05)



The point robot now has a ultrasonic range sensor

- The range sensor has a finite range of detection and it can sweep 360 degrees .
- The range sensor will prevent the robot from colliding with the obstacle.
  - Define a distance function to find shape of obstacles as there are two variables now the distance from the robot and the angle of the sensor

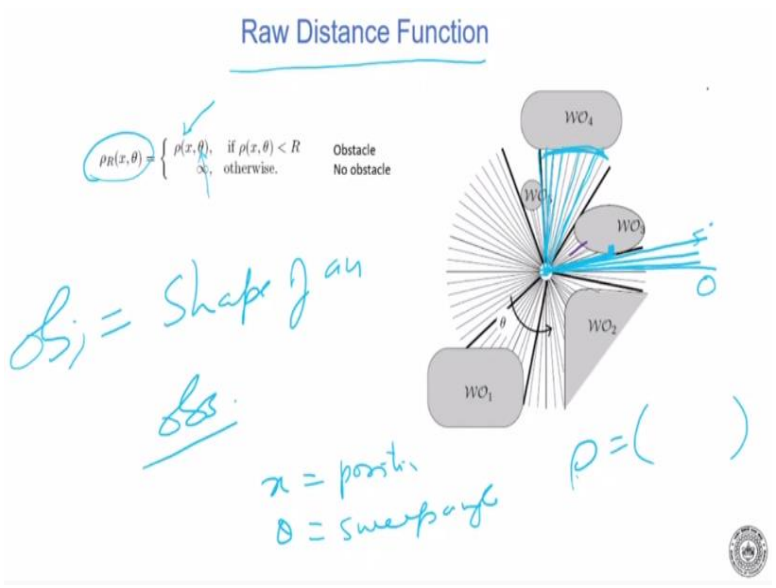
Now, suppose, the robot now has an ultrasonic range sensor that means it can find out how far it is from the obstacle. So, let us say we have an obstacle here. And, we have the robot which is here now, the point robot. But, it has an ultrasonic sensor. Now, the range sensor has a finite range of detection that we know and it can sweep 360^0. So, if you can imagine that if I can mount a range sensor, so, I have a motor here and I mount the range sensor.

So, this is my range sensor and it can rotate about this axis by 360^0. So, suppose if you can imagine this, this is my range sensor. And, in this direction, it is my 0 degree is here. $\theta = 0$ here. And, it can sweep. As the motor rotates, what will happen? It will sweep and it will give me the distance. So, first of all, there is a range sensor. It has a finite range. And, the range sensor will prevent the robot from colliding with the obstacle.

So, in the previous case, it had to touch the obstacle. Here, it will not touch the obstacle because it knows the obstacle is there. Now, when the robot is standing at this point x say x y is the coordinate of the robot at this point. Now, it also has a sweep angle. For example, whether it is seeing this side or it is seeing that side. So, there is a θ. Let us say this is equal to 5^0 when it is seeing this side.

So, there are 2 variables here when the robot is standing at this point. One is the distance it is from the obstacle. So, this is my obstacle which is given by x and the angle at which the sensor is currently with respect to the 0. So, using these 2, what we do is we compute a distance function which gives us the exact information as to how far the obstacle is and it is in which direction compared to the 0 of the robot.

(Refer Slide Time: 05:49)



Raw Distance Function

$$\rho_R(x,\theta) = \begin{cases} \rho(x,\theta), & \text{if } \rho(x,\theta) < R \\ \infty, & \text{otherwise.} \end{cases}$$

Obstacle

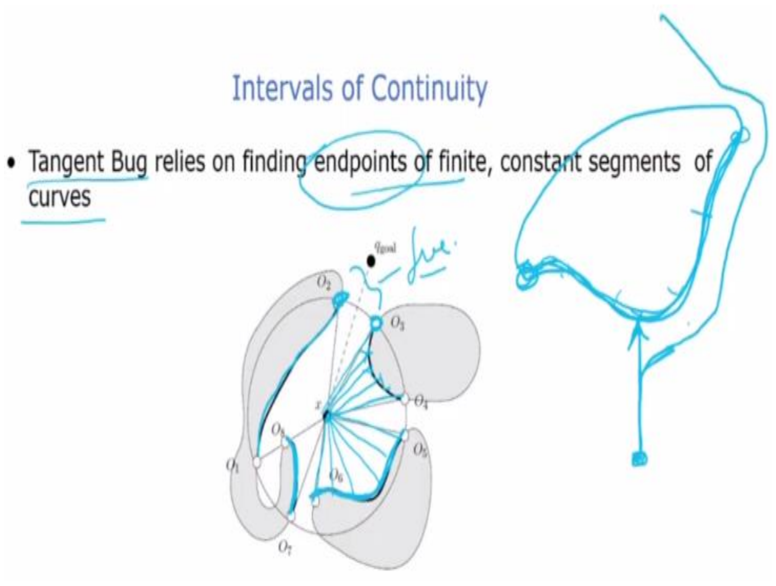
No obstacle

So, we define a distance function. Now, what is the objective of this distance function? The objective of the distance function is to find out the shape of an obstacle. We want to find the shape of an obstacle. So, the distance function essentially consists of ρ which consists of x and θ. x is the, where the robot is in the workspace and θ is the sweep angle. So, x is my position and θ is my sweep angle of the range sensor.

Say, for example, if this is my 0^0, next is going to be 5^0, 6^0. So, now, at any point here, and we are going to find out this distance function ρ. So, when we are going to get a ρ, ρ will be a set of points. No, set of distances which will actually tell me that at 0 there is nothing here. At 2^0, 3^0, 4^0, 5^0 here now, there is an obstacle here. So, this is the distance function which will give me this distance now.

Similarly, for different θ's, it is going to give me these distances. And, these distances will indicate that there is an obstacle there. So, now, basically with respect to the position of the robot and the sweep angle, it knows exactly what is the position of the obstacles. So, the objective of this is that the robot should be able to find out where it is with respect to the obstacles. And then, find a path. And, to do that, we have to first find the raw distance function.

(Refer Slide Time: 07:21)



Intervals of Continuity

- Tangent Bug relies on finding endpoints of finite, constant segments of curves

Now, as we have seen that the objective is basically to find the profile or the, let us say the curves of the object. So, in this particular case, the robot is at point x. It is sweeping from θ = 0. And, wherever it gets the obstacle you get a distance there. As we have seen the way ultrasonic sensor works is basically if it hits the obstacle, the ray will come back and then it knows, what is the distance of the object?

Now, what it has done is basically it has given us this distance functions. And, we know that this is the profile of the object now because I know the corresponding distances of every point on that object with respect to x and because I know, what is the sweep angle θ? I also know, what is the orientation? So, now, I have completed my the range sensor has given me what.

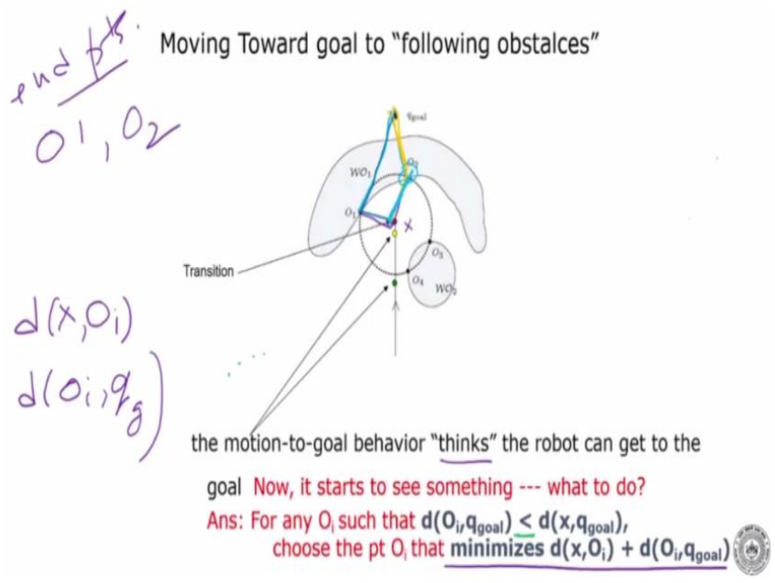
It has given me more information that it is also telling me, what is the shape of the object here? Now, the Tangent Bug algorithm basically relies on finding endpoints of finite constant segments of the curves. Now, why I need to find the end points? Because, in between, it does not matter. So, if I have an obstacle like this. This is my obstacle. And suppose, I find that the range sensor is giving me this profile. This is the profile of the obstacle.

It is enough for me to know where is the endpoints. I do not need to know what is in between. It may not be important for me because I am finding free path. So, if I know this end point and I know this end point that in between, here and here is free path now. So, I am basically interested in finding the free path. That is why for all these curves we can find the, it is important to find the endpoints of these curves also.

Now, we set the objective of this is not to go and hit an obstacle but to keep a safe distance. And, that is what the Tangent Bug does. So, if I have an obstacle like this and this is my robot, it knows that the obstacle is here from the range sensor reading and the angle of sweep. So, basically, what it will do? It will not go and hit because it knows the obstacle is there. It will go to some point and then it will turn right.

It will become tangential to that point there. And, that is why it is called a Tangent Bug. So, it behaves like a bug algorithm which is tangential to the surface at that point. So, basically, it will go like this, like this and then go off. So, this is what we call the Tangent Bug algorithm which works using the sensor data.

(Refer Slide Time: 09:45)



Moving Toward goal to "following obstalces"

the motion-to-goal behavior "thinks" the robot can get to the goal Now, it starts to see something --- what to do?

Ans: For any O_i such that $d(O_i, q_{goal}) \leq d(x, q_{goal})$, choose the pt O_i that minimizes $d(x, O_i) + d(O_i, q_{goal})$

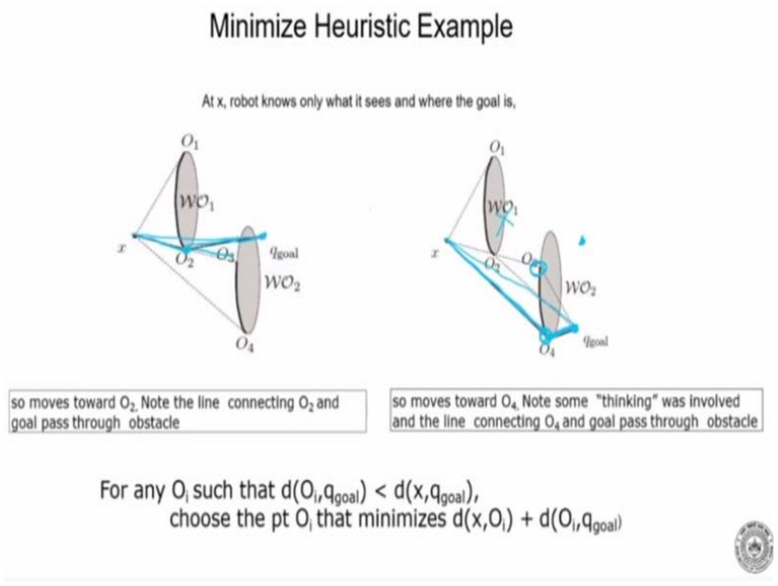
We see that the sensor gives the information of the shape of the obstacle to the robot. So, now, we can see that we have a robot which is at a point this red point which we can call x. Now, at the point x, it has got the information that the shape of the obstacle is like this. So, now, it has to decide whether it goes to point 1 or it goes to point 2. So, we have point 1, you have point 2. There are many points in between of course but it does not matter.

As we said that we are interested in the end points. So, the end points are O_1 and O_2. So, at the point x, the robot has to think and decide whether it will go towards 1 or 2, point 1 or point 2. So, in that case, basically, what it does? It minimizes this function. Now, what is this function d (x O_i) is the distance of the robot from the point x to the point 1 and the point 2. So, there are 2 distances here. So, there is one distance here. There is one distance here.

And, there is another distance there. That is one distance. The other one is d (O_i q_{goal}). That is the other distance which is the distance of the 2 points O_1 and O_2. That is one distance here, another distance there. So, what it basically does is it minimizes the sum of d (x_iO_i)+ d(O_iq_i). That means it is going to basically measure this distance. This one plus this one is longer or the other one is longer.

This one plus this one is longer. So, it will minimize that which means it will go towards the point where this distance is less. So, in this particular case, we can see that this distance is less. So, the right side distance point is less. So, it will go towards this side. We will go here. So, that is how it decides which side to go.

(Refer Slide Time: 11:44)



Minimize Heuristic Example

At x, robot knows only what it sees and where the goal is,

so moves toward O_2. Note the line connecting O_2 and goal pass through obstacle

so moves toward O_4. Note some "thinking" was involved and the line connecting O_4 and goal pass through obstacle

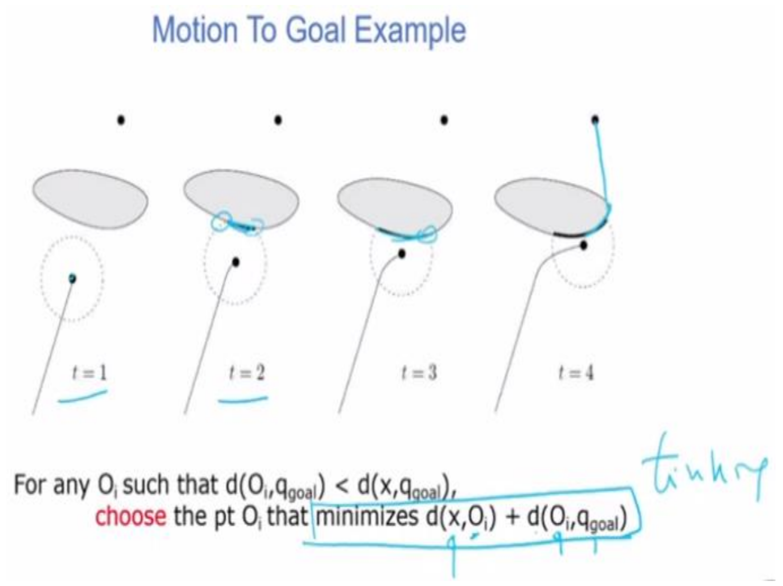
For any O_i such that $d(O_i,q_{goal}) < d(x,q_{goal})$,
choose the pt O_i that minimizes $d(x,O_i) + d(O_i,q_{goal})$

Now, next, we see that this depends very much on where the position of the goal is also. For example, in this figure, on the left side, the goal is here. And, this point is this line is going straight through because this distance was going straight through the goal now. So, it is going to go from this point to here, the corner of the obstacle. Then, it has to choose where it has to go. So, whether it is going to go like this and like this or it is going to go this side that has to choose.

Now, in the case on the right, you can see that the goal is right here. In the earlier case, the goal was there. Now, the goal is here. So, it does not have to go to the obstacle one at all because very easily it can see that this is my straight line connecting the goal and one end is here, the other end is here. So, which distance will be less? This distance will be less. So, it will go here and then it will go in that direction.

So, it is not getting the obstacle one at all. So, we can see that the path is very much dependent on geometry of the obstacles.

(Refer Slide Time: 12:40)



Motion To Goal Example

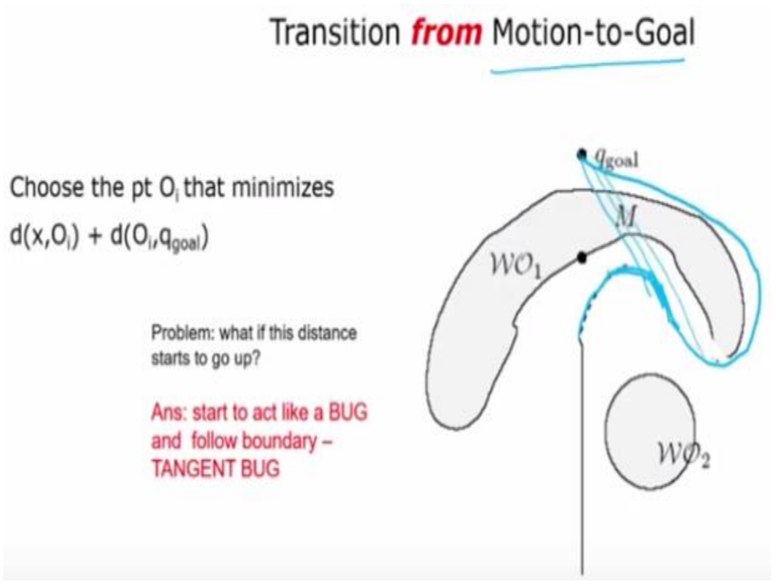
For any O_i such that $d(O_i,q_{goal}) < d(x,q_{goal})$,
choose the pt O_i that $\boxed{\text{minimizes } d(x,O_i) + d(O_i,q_{goal})}$

Now, the motion to goal is basically by executing the bug algorithm and by minimizing this distance which is giving the distance between the position of the robot at x and the point where it is hitting the obstacle. That is O_i plus the distance between the rest corresponding point and the goal point. So, it will always try to minimize that. And, trying to minimize that, it will move. So, you can see that at time is equal to 1 second it is coming near.

The range sensor is such that it does not see the obstacle at point time is equal to 2. It is seeing the obstacle. Now, it is deciding which side to go whether this point to this point and it decides it goes towards the right. So, as it is going towards the right, we are seeing more of the obstacle. And, it is going more and more towards the right using the bug algorithm till it comes here. And then, it finally goes there.

So, what it was basically doing is it is minimizing this function. And, this is something like it is thinking. The robot is thinking which side to go.

(Refer Slide Time: 13:35)


Transition *from* Motion-to-Goal

Choose the pt O_i that minimizes $d(x,O_i) + d(O_i,q_{goal})$

Problem: what if this distance starts to go up?

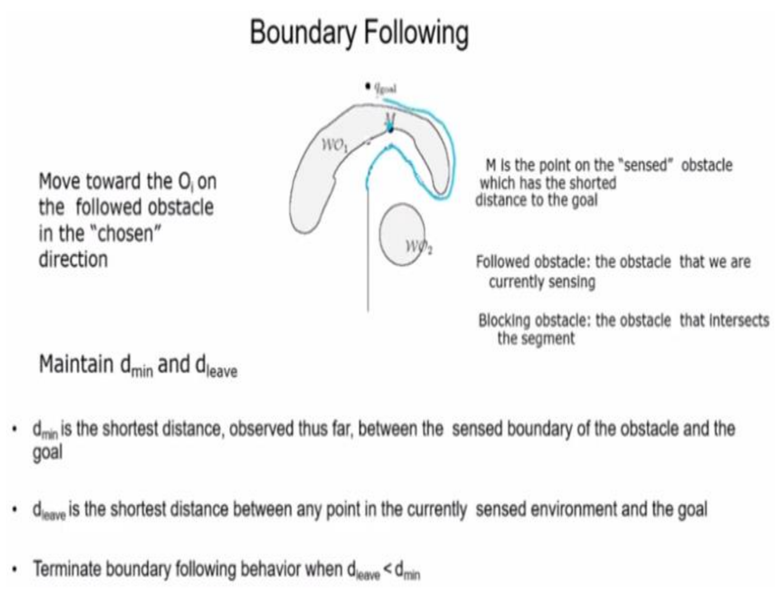
Ans: start to act like a BUG and follow boundary – TANGENT BUG

Now, transition from motion to goal is exactly the same way. Now, when it is going straight towards an obstacle here, basically, it will try and transition and then it will follow the bug algorithm. So, when it is going towards the goal, what is happening in this case is that you can see that as it is going backwards the distance is increasing actually. So, what? Here, it will switch. It will transition from motion to goal. That is basically the bug algorithm.

And, try and go around the obstacle and then go towards the goal because you can see in all the cases, it go it tries to go towards the goal. It does not go away from the goal. But, in obstacles of this nature, what will happen is as the robot is pushed away just from the surface of the obstacle, it is following the obstacle, the profile of the obstacle. So, the distance towards the goal is increasing here now.

So, now, it is transitioning to the bug algorithm again following the surface and then it is going towards the goal.

(Refer Slide Time: 14:31)



Boundary Following

Move toward the O_i on the followed obstacle in the "chosen" direction

M is the point on the "sensed" obstacle which has the shorted distance to the goal

Followed obstacle: the obstacle that we are currently sensing

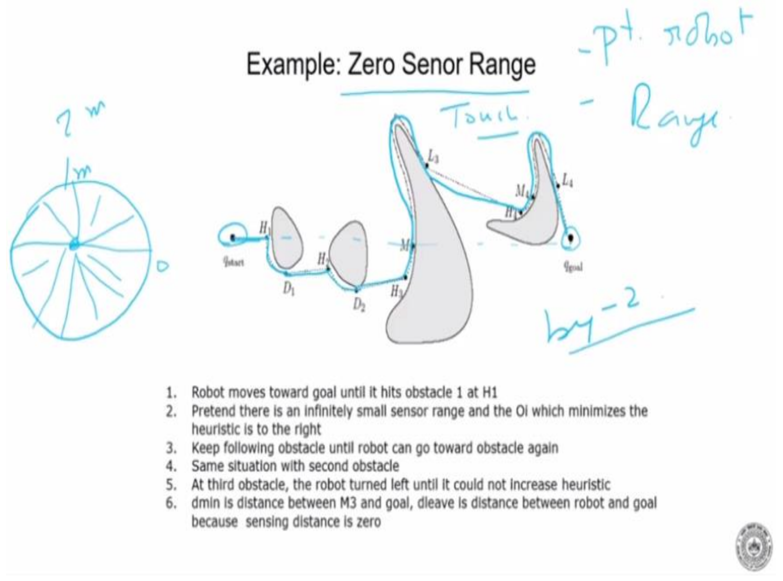
Blocking obstacle: the obstacle that intersects the segment

Maintain d_{min} and d_{leave}

- d_{min} is the shortest distance, observed thus far, between the sensed boundary of the obstacle and the goal
- d_{leave} is the shortest distance between any point in the currently sensed environment and the goal
- Terminate boundary following behavior when $d_{leave} < d_{min}$

Now, that is basically the boundary following behaviour. So, you can transition between this 2 behaviors and then try and go to the goal. So, this is an example also where the robot was moving towards the obstacle. It finds the point M, goes towards M. But, then, after M, again, this is concave. And, this happens very often for concave obstacles. Now, there is no other way. So, it has to use the bug algorithm. Do or the Tangent Bug follow the profile. And then, go like this and then come to the goal.

(Refer Slide Time: 15:04)



Example: Zero Senor Range

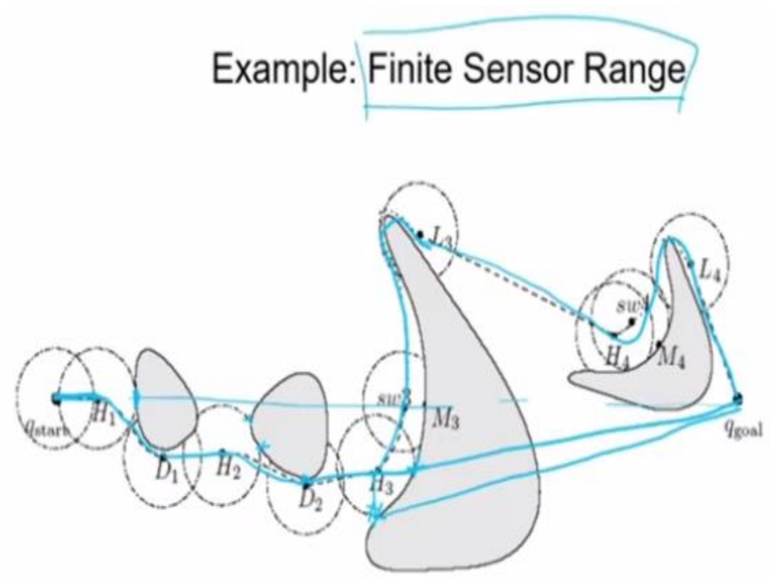
1. Robot moves toward goal until it hits obstacle 1 at H1
2. Pretend there is an infinitely small sensor range and the Oi which minimizes the heuristic is to the right
3. Keep following obstacle until robot can go toward obstacle again
4. Same situation with second obstacle
5. At third obstacle, the robot turned left until it could not increase heuristic
6. dmin is distance between M3 and goal, dleave is distance between robot and goal because sensing distance is zero

Now, so far, we talked about the range of the sensor. We talked about a point robot or robot is still a point robot. And, we talked about the range of the sensor. Now, the sensor has a range. For example, it can see maybe an ultrasonic sensor. It can see maybe 1 meter. It can see 2 meters. And, it is circular like this. So, this is my zero. And, this is my scan angle. So, where if an obstacle comes in that circle of radius whatever is the range of the sensor then it can see.

Otherwise, it cannot see. So, if it has zero range basically it means that it cannot see it is like a touch sensor. Zero sensor is equivalent to a touch sensor. So, it is equivalent. So, in this particular case, you can see, we have the start point here and we have a goal point here. Now, we know which is the direction of the goal. So, basically it always has to move towards the goal. So, if you have a straight line there then it starts off in that direction.

It hits the first obstacle. Now, once it hits the first obstacle, what it does is it minimizes the heuristic and goes towards the right. Whatever we have seen, it goes towards the right is following this from here. It again goes towards the goal and it comes here. Then, again, it goes around the obstacle, comes here, goes like this, goes here, goes here, goes like this and goes down. So, this is almost similar to the Bug 2 algorithm because it is behaving like a touch sensor.

(Refer Slide Time: 16:43)


Example: Finite Sensor Range

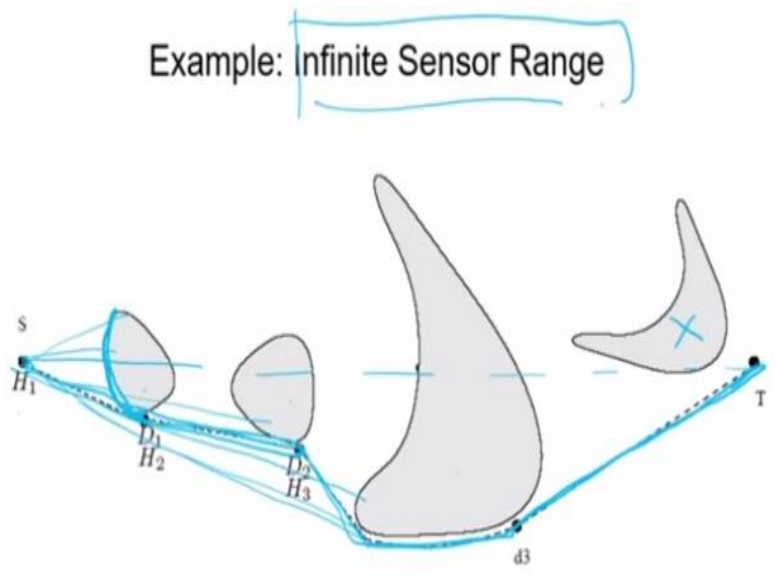
Now, suppose, there is a finite range on this sensor. That means it does not have to go and touch the obstacle. It can decide whether to go left or right much earlier the moment it sees the obstacle. So, now, you see what is happening to the path. So, it started off from here and we are connecting with a straight line with the goal. It tries to go in a straight line but the moment it has come here, the circular range of the sensor indicates that there is an obstacle there.

So, now, it has to immediately decide whether to go left or to go right. So, by checking the heuristic distance function basically, what it does is it goes right like this now. Then, from here, it again goes towards the goal. So, it has come here. Again, it sees an obstacle. So, it has to decide whether to go here or to go here. So, again, it uses the distance formula which the distance formula this one.

So, it minimizes this distance function where which is the sum of the distance between the robot and the endpoint of the curve and the point from the end of the curve on the obstacle and the goal point. So, when minimizing that what it does is it finds the next point. So, from here, it has found this point. Then, from here, it starts going towards the goal again. Now, it has hit the obstacle. Now, it is finding these 2, this is one point.

This is another point. Now, it is basically seeing the sum of the distance from here to here and from here to the goal. That is much less than the distance from here to here to here and that. So, it will go towards the left now. So, now, it is going towards the left following this curve coming here again the same thing going here and then going to the goal. Now, this is the case where it has a finite range sensor.

(Refer Slide Time: 18:28)



Example: Infinite Sensor Range

Now, suppose, this has infinite range sensor, it can see really far. Then, from here, it starts off. This is a straight line. But, from here itself, it has infinite range. Please remember. So, it automatically knows what is where. So, it can see this obstacle from there because it has very large range. Now, it knows that this is my obstacle. So, it does not go towards that obstacle or it gets the last point the end of the curve.

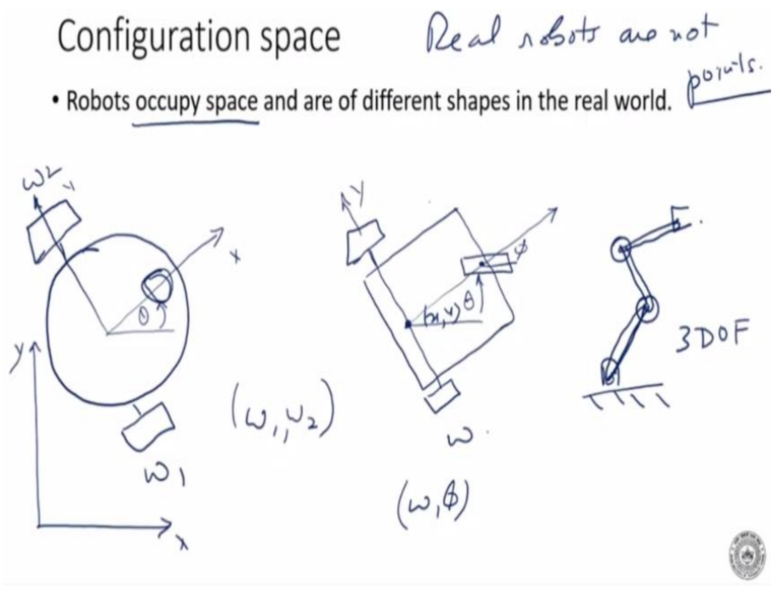
From here, it again sees. And, it basically sees this one. So, once it sees this one it goes there, from here, it goes here. It goes there, there and goes straight there. So, this is basically showing a very large range sensor. And, using the very large range sensor, it can find which points are hitting the obstacle. And then, it is going from the start point to the first obstacle, second, third. And then, it is not touching the fourth obstacle and going off straight to the wall.

So, if you compare these 3 paths, this is the path 1 which has zero range sensor. You can look at the length of the path. This is the one that has a finite range sensor which is a little better than the previous one. But, the direction was more or less the same. But the third fellow which had a very large infinite range sensor, the path was much shorter. So, it simply avoided, it went to the corners of the obstacles and then simply went towards the core.

So, this is basically to show that a lot of it depends on the range of the range sensor. So, if your range sensor has very small range or large range, so, the path is going to depend on that now. Now, so far about the bug algorithm and the Tangent Bug and the heuristics that is used to decide whether the robot should go left or the robot should go right using a range sensor. Now, let us come to the, probably the, one of the most important topics in this course.

That is configuration space. Now, we have seen that robots occupy space and are of different sizes and shapes.

(Refer Slide Time: 20:22)



Configuration space

- Robots occupy space and are of different shapes in the real world.

Real robots are not points.

That is something we have seen. For example, we can have a mobile robot which is circular like this. And, it has 2 wheels. It has ω_1 velocity there and ω_2 velocity on the other side. And, we have a caster wheel in front. So, this is one robot and we can assign an axis in this direction. So, this is my x and that is my y. And, this makes an angle of θ with the global axis. Let me draw the global axis here x and y.

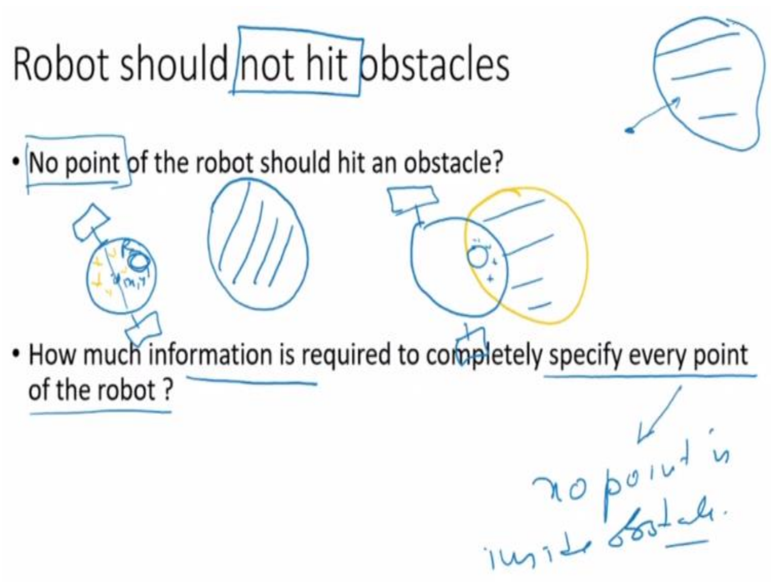
So, now, we have seen that this robot is a circular robot. And, we derive it by ω_1 and ω_2. And, this is the differential right robot. So, it goes forward. It can turn simply by changing the ratio of ω_1 by ω_2. Now, we have another kind of robot which is let us say a car like robot. So, we have a wheel which is connected here. And, we have a steering wheel here. Let us say, we have a steering wheel.

Now, let me, just have one steering wheel here like an auto or a tempo you must have seen that. There is a steering wheel here. So, this is ω and that has the steering angle which is Φ. Now, this also has the point x y which is there. This is my axis. This is my y axis. And, there is a Φ there which is the turning angle.

This robot also makes an angle of θ with respect to the x-axis of the global. So, this is another kind of robot where the input is ω and Φ to make it go. Now, you can also have a serial arm where we have a serial arm robot like this. So, we have one. You can imagine. This is multiple degrees of freedom 1, 2, 3 DOF arm. Now, these arms are going to have physical lengths. They will have physical dimensions. That is my arm.

So, now, what we have seen that that these robots mobile robots and serial arms you can even talk about a flying robot quadcopter these robots occupy space in the real world. They are not points. Basically, robots are not real robots are not points as we have done in the bug algorithm case. So, there are different kinds of robots and they occupy space and other different shapes in the real world.

(Refer Slide Time: 22:56)



Robot should not hit obstacles

- No point of the robot should hit an obstacle?
- How much information is required to completely specify every point of the robot ?

no point in
inside obstacle.

So, now, what we need to do is when we are doing the path planning in the case of the bug algorithm we had a mobile robot which had a sensor in front. Now, what is the objective of this path planning? The objective of the path planning is that no point, the robot should not hit obstacles. That is English language. Robot should not hit obstacles. What is the meaning of this?

It basically means in terms of geometry that no point of the robot should be inside an obstacle. So, for example, I have a mobile robot like this. This is a mobile robot, differential drive with a caster in front. And, I have an obstacle. That is my obstacle. So, this fellow has a a point x, y and it has a radius R. So, when we are saying the robot should not hit the obstacle it basically means that no point inside the robot, geometrical.

These are points inside the robot. No point inside the robot should be inside the obstacle. So, this is my obstacle. The robot should not go and no point of the robot should encroach upon the obstacle and go inside the obstacle. That is what it means. So, this, my, so, this points should not go inside the obstacle. Geometrically, this is the meaning of do not hit the obstacle.

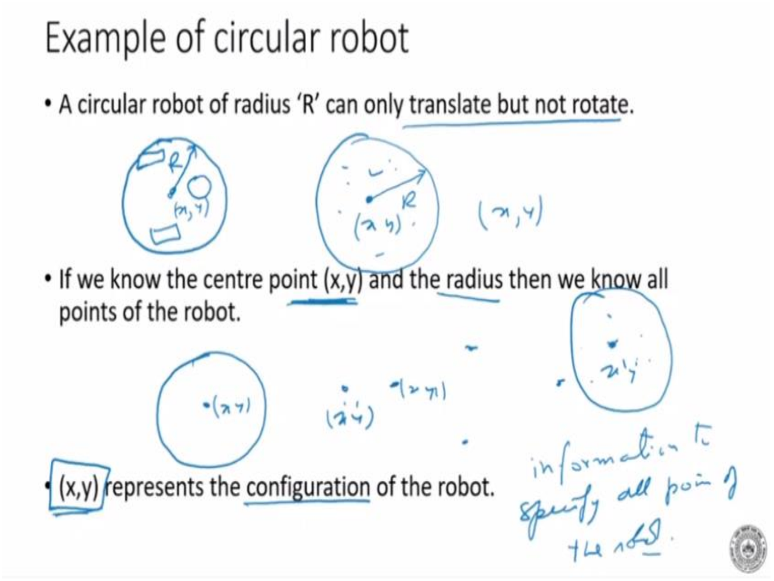
So, no point of the robot should be inside the obstacle that inside the obstacle boundary. So, the question that comes up here is if you are talking about a point and that is not very difficult. Like, in the case of bug algorithm, we had an obstacle and this is a point. You can check if the point is inside that boundary or not. Geometrically, you can do that. But, in the case of a real robot of different shapes, for example, here, we are having a robot which is circular.

Again, it is easy to understand that there are different points. And, this point should not be inside the obstacle here. What about in the case of a serial arm? There is an obstacle here. Now, any point on the link should not be inside this obstacle. That is not very easy to visualize. In the case of mobile robot, it is little easier. In the case of serial robot, it is more difficult.

So, what is the, or, how much information is required to completely specify every point of the robot? Because we are saying that no point of the robot should be inside the obstacle. So, how much information or what information is required to completely specify every point of the robot? If we can do that then we can ensure that no point is inside the obstacle because we are ultimately going to check geometrically.

So, we will see if there is any point of the robot inside the obstacle and how do we do that we check it geometrically.

(Refer Slide Time: 25:41)



Example of circular robot

- A circular robot of radius 'R' can only translate but not rotate.
- If we know the centre point (x,y) and the radius then we know all points of the robot.
- (x,y) represents the configuration of the robot.

R
(x,y)
(x,y)
R
(x,y)
(x,y)
(x,y)
(x,y)
x',y'
information to specify all points of the robot.

So, let us take the example of a circular robot. We will come to arm manipulators (()) (25:45) little bit later because they are little bit more tricky. Now, a circular robot of radius R can only translate but not rotate let us make this assumption here. So, we have a circular robot. This is my circular robot. These are 2 wheels of the robot. There is a caster wheel and it has radius R. Now, we are saying that this robot can translate.

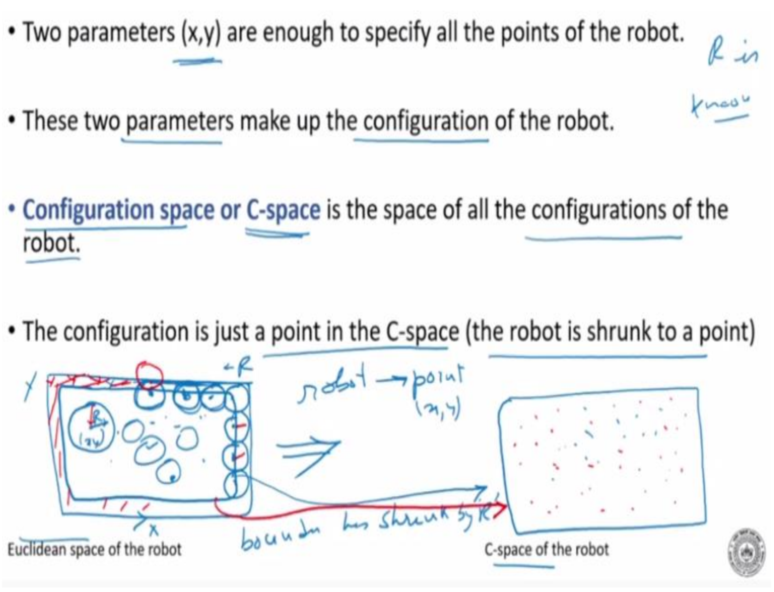
And, it cannot rotate. This is the x y. Now, if you know the center point x y and the radius, then we know all points of the robot because we know the radius of the robot. So, if I have the center point of the robot here, this is my x y and I know the radius of the robot which is R. Then, if I draw a circle here, then I know all the points inside. So, what is enough here? So, what is enough to know is only x y.

So, if I know the center of this robot because it is a circle. So, if I know the radius of the circle, then I know all the points inside this robot. So, if I know the center point x y and the radius, then we know all the points of the robot. So, here, again, I am showing x y is here. If I simply say x y is here or here or here or here or here, then I know where the robot is also because I know the radius.

So, the information that is required here is the x y of the robot is required. And, this x y represents the configuration of the robot. So, here, we are using a new term. That is the configuration of the robot. So, the information required to specify all points of the robot is called the, that is the basically the configuration of the robot. And, a collection of the configurations makes up the configuration space.

So, this is the information to specify all points of the robot. So, what is this information to this x y?

(Refer Slide Time: 27:52)



- Two parameters (x,y) are enough to specify all the points of the robot.
- These two parameters make up the configuration of the robot.
- **Configuration space or C-space** is the space of all the configurations of the robot.
- The configuration is just a point in the C-space (the robot is shrunk to a point)

And so, what we say is that 2 parameters x and y are enough to specify all the points of the robot. Why? Because R is known, so, the radius of the robot is known. That means if I tell you x y now you know where the other points of the robot are. Now, these 2 configurations these 2 parameters make up the configuration of the robot. Now, configuration space in short is called C space. So, it is the space of all the configurations of the robot.

Now, I hope you can make the connection that all the (x, y)'s of the robot make up the configuration space. Collection of all the (x, y)'s of the robot make up the configuration of the robot. Now, the configuration is just a point in C space. That basically means it is a point because the robot has shrunk to a point now. As a example I gave here. I was talking only about x y. That is (x_1, y_1) (x_2, y_2).

These are just (x, y)'s collection of the configuration of the robots. So, the configuration is just a point in C space and the robot has shrunk to a point. So, please note that we have Cartesian space and we have C space. Now, let us say that this is a work area of a mobile robot. We have a mobile robot here which has radius R and it center is (x, y). This is my workspace. Now, the configuration of the space is a collection of all the configurations of the robot all the possible configurations.

Now, obviously, this robot cannot go out of this boundary. It cannot go like that. So, this point is out of the configuration space. It cannot go out. So, what is the maximum this robot can go? The maximum the robot can go is on the boundary. Remember, it can translate only like this, like this. So, it can go and sit on the boundary like this. That is the maximum. Inside is okay, it can go anywhere.

So, when it goes and sits on the boundary, this is my (x, y) (x, y) (x, y). So, now, if I am saying that the robot has shrunk to a point I am only interested in the x y of the robot then my, this space has shrunk to this distance now. I hope you remember you understand that. So, what has happened is the robot has become a point. And, the work space the boundary has shrunk by how much by R. So, it has come down by R.

That means all the points here are actually possible. So, now, here, I am converting this into configuration space. And, this is my configuration space. So, the inside fellow is now my configuration space. This is smaller. Let me draw it here. It does not look smaller just longer and smaller here and then let us say that now. Now, what has happened is this has become my configuration space now.

So, this is all the points where the robot can go. Now, the robot has become a point because it only has (x, y). It does not have anything else. And, the boundary has shrunk by R. So, this is my Euclidean space. It is also called Cartesian space which you are familiar with. If I say this is my (x, y) directions, then this is my Cartesian space the one shown in the left. The one on the right is my C space now. Now, the C space and the Cartesian space they look same. Why?

Because we are talking of (x, y) only but they are not actually the same spaces. As we go along, we will see how. So, please, I will just repeat again whatever I have done because this is very important. We are saying that the robot should not hit any obstacle. In geometrically, what does this mean? This means that no point on the robot should be inside the obstacle. This is what it means. And geometrically, you can check that.

Now, to do that, we need to know how much information is required to completely specify every point of the robot. Now, and in order to do that, what we see is that if you know where is the center of the robot (x, y) and you know the radius, then if you know where is (x, y) then you know where the other points of the robot is. Like, if I give you (x, y) here, $(x'y')$ robot is there, then I immediately know, where are the other points?

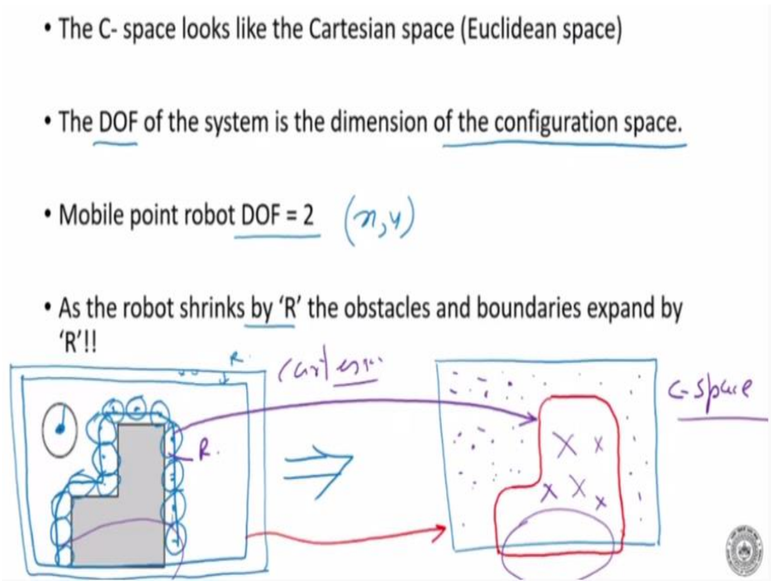
The other points are all here. So, (x, y) is the information required to find out how to specify all the other points of the robot. And, this (x, y) is called the configuration of the robot. It is like saying the robot has shrunk to this point which is (x, y) now. Now, these 2 parameters make up the configuration space which we call the C space. Now, if you have a workspace like this as shown on the left side and we have the robot then what are the possible configurations that the robot can go to without going out of the boundary?

At most, it can go and sit on the wall. Inside is fine. There is no problem. At most, it can go and sit on the wall. And, if it goes and sits on the wall or touches the wall then this is my R distance the center of the robot. That means the (x, y) can only go till here. It cannot go into that space there. So, it cannot go inside in this space. The (x, y) cannot go into that space. That is very clear from here. Why?

Because the center cannot go inside there, that means, no point the robot it will not be possible for the robot to go here and sit like this. In that case, it has gone through the side which means that the boundary has shrunk by R and the robot has become a point. And, the collection of all those points is basically what makes up the configuration space. So, this inside boundary is what is drawn here. So, the robot has become a point.

Now, all these points inside here are possible positions for the robot to go. There is no problem. It can go anywhere. It cannot go outside the boundary only. So, this collection of points is basically called the configuration space. Now, please note, that the Cartesian space which is shown on the left or Euclidean space is different from C space. They look the same because this also looks like there is x here and y there. So, they look same but they are not.

(Refer Slide Time: 34:17)



- The C- space looks like the Cartesian space (Euclidean space)
- The DOF of the system is the dimension of the configuration space.
- Mobile point robot DOF = 2 (x,y)
- As the robot shrinks by 'R' the obstacles and boundaries expand by 'R'!!

So, the C space looks like Cartesian space or Euclidean space. Now, the degree of freedom, so, these are terms, let us go slowly. The degree of freedom of the system is the dimension of the configuration space. So, mobile point robot, the degree of freedom is 2 because there is only x and y. And, that is why we said that it is not rotating. It is only translating. Now, the robot shrinks by R. The obstacles and boundaries expand by R.

So, let us look at this case. So, in this case, if we have a workspace like this, in the previous case, I did not have an obstacle I only had the robot. The robot shrinks by R, the radius. So, it becomes a point. And, the outside boundary also becomes smaller. So, it will become smaller like this now because it has shrunk by R because the center of the robot cannot go anywhere inside there. Now, what about the obstacle? Well, the same logic applies.

The robot can go and sit at most on these points on the obstacle on the boundary of the obstacle. It cannot go inside the obstacle because we said that hitting an obstacle basically means points of the robot have gone inside the obstacle boundary. So, now, we see that again the same logic applies if this is the center of the robot the c_g the central point then there will be a straight line like this, like this, curve, like this, straight line, curve like this.

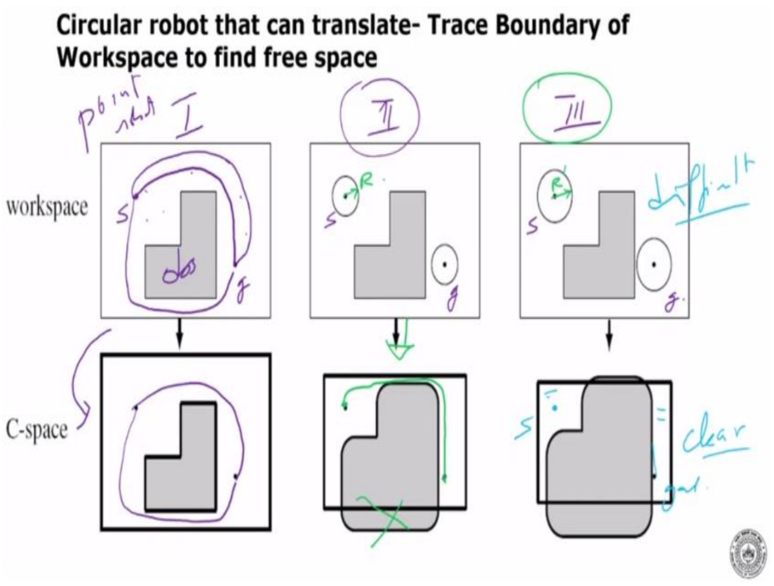
So, what has happened is that this is my Cartesian space. And, I am converting it. First of all, it has become smaller. So, this inner fellow has become here. So, this is my inner fellow. So, the inner one is here now which has become smaller by R. And, what has happened to the obstacle? The obstacle has become larger. So, here, we can see that what is happening to the obstacle. It is becoming larger.

It has gone out this side. So, what we are seeing is that the obstacle space has become larger now. There is no obstacle space. So, the obstacle space has now become bigger. Now, this is interesting. Why? If you look here now it was not clear if the robot would go through that space or not but here it is very clear that there is no space here. The obstacle has taken away all the space. That means the robot cannot go from the bottom now.

So, the robot has become a point. It can go at all points here. It cannot go outside the boundary of course. And, it cannot go inside the obstacle. Now, it is very clear that these are points where the robot can go and these are points where the robot cannot go. So, we have reduced the robot to a point. We have shrunk the workspace the boundaries and we have increased the size of the obstacle, by how much, by R.

So, the obstacle has become larger by how much, by R again. It is like the obstacle has expanded and the robot has shrunk. Now, this is my C space. And, this is my Cartesian space. They look like they are same. But, they are different spaces. In the case of a serial arm, we will see that the spaces even do not look same. They look very different. Then, you look at it.

(Refer Slide Time: 37:43)



Circular robot that can translate- Trace Boundary of Workspace to find free space
point robot
I
II
III
workspace
C-space
s
g
R
difficult
clear
goal

Now, this is, this will also give a clearer idea of what is the use of this. The use of this is that we can very easily ensure, we can very easily find a path from one point to another point. Why? Because it is a point which is moving in space and as long as the point is not inside the C space of the obstacle then the path is fine. So, when we are trying to find a path, for example, in this case, let us look at Case 1, Case 2 and Case 3, 3 cases.

There is an obstacle here. So, these are the obstacles. The workspace is given. In the first case, the robot is a point robot. I have to find a path from the start. Let us say, start to goal, start to goal, start to goal. So, now, in the first case, the robot is a point robot. We have the obstacle. So, there is no point. There is no problem at all. This is our free space. So, the robot can, this point can go anywhere as long as it does not go inside the obstacle.

So, we can very easily find that there is a path like this or there is a path like this. There are multiple parts actually which can take the robot from the start point to the goal point which is very clear. So, we are converting this into C space because it was a point robot. It still remains the same. There is no change. And, we can find a path like this for example. And, it is also clear that it can go from the bottom like this to this point.

What about Example 2? In Example 2, the robot is not a point robot. It is having a radius R. So, now, if I want to find out whether the robot can go like this just by visually looking at it, it is not very easy actually to say plus this is geometry the robot does not have any eyes. You are writing a computer program which is the when you say path planning algorithm. So, algorithm is a program. So, the program is to find out if the robot can go like this or not.

We have eyes. So, we can see. But, even then, it is difficult. But, now, if you see we are converting this into C space, the robot is shrinking and becoming a point and the obstacle is increasing the boundary of the obstacle has increased. And, it is becoming this shape now. Now, it is very clear that there is a path. But, the path is like this. There is no path on this side but it cannot go through the obstacle. So, we do our path planning in the C space.

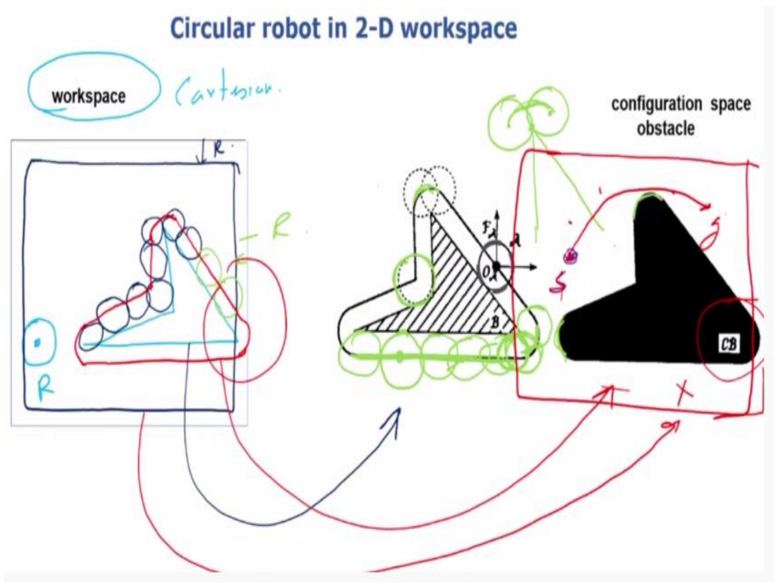
We do not do it in Cartesian space. Now, let us look at the third case. We have a robot which has a larger radius. This R is lets call it R'. This is my Case 3. So, in Case 3, the robot is a little larger. Now, can you say, can the robot go through here? Can you go through here? Here, probably cannot go through. Simply, by looking, we can say. What about the top? It is difficult to say.

But, for a program which does not have eyes, what we basically do is we convert the robot to a point. That means shrink it by R and expand the shape of the obstacle. And, if we expand the shape of the obstacle, what we see is that it covers the works the boundary here. It is also covering the boundary here which basically means there is no path which will take it from here to there.

There is no path which was actually difficult to say just by looking at this figure. This was difficult to say. But, here, it is very clear. It is extremely clear that there is no path because the 2 sides the start and the goal are disjointed. There is no connection between this side and this side. So, how can it go? So, this gives us an idea of how to go from the Cartesian space to the C space.

And now, if you are wondering about the shape of the obstacle like in the previous case also when I drew what you are seeing is that what is happening here on this side is that there is a curve which is coming. When there is a sharp corner, what happens? The curve comes. Now, how does that happen? Let us just look at a little bit more.

(Refer Slide Time: 41:43)



Now, let us say that we have a robot. This is my workspace on the left side. And, we have a robot which is having this is my robot which has radius R. And, this is my obstacle. Now, I want to draw the C space. This workspace or we can call this the Cartesian space. So, I want to make the C space now. So, what would happen is I am going to reduce this. I am going to shrink it. Let me say I am shrinking it now.

I am shrinking this by R. So, this is become smaller plus I am going to see all the possible configurations where the robot can go without going inside the obstacle. Now, how do we do that? If you can imagine, let us say, let us look at the obstacle here now. Now, if you can imagine the robot is a circular ball and you are moving this circle along the boundary of the obstacle, so, this point is you are shifting the radius. You are moving the c_g like this.

So, the robot has come here. In this case, the robot is here. So, I am simply sliding the robot. So, the robot is being slid. Now, it is being slide it is sliding. And, this is the locus of the center points. So, this side has increased there. What is happening at the corner? Please look at the corner very carefully. The robot is here. So, it is making point contact here between that vertex and the ball on the circle.

Now, on this point, the robot can actually it is like saying this circle can actually go like this because it is making a point contact. So, this is a, if you can imagine that this is the point. And, this is the circle at that point. So, the contact it is making is a point contact there. So, this is the circle making that point. Now, when the circle is on this side also, it is the same point. It is like saying the circle is rotated like this.

The c_g has moved like this. That is where this arc comes from. So, you can see that there is an arc here this arc. Similarly, in the case, wherever there is a sharp corner here. So, we can have the robot sitting here. It is making a contact at that point. And, when it is sitting a little bit on this side on the right side that is here, that contact point is still the same. So, in the process, what happens is it is like the robot has rotated a little bit. So, an arc comes in there.

So, you see the shape of the obstacle of the C space obstacle is that there are curves wherever there are sharp corners. So, you can see that there is a sharp corner here. There are curves on this side. So, now, we understand how to make expand the shape of the obstacle. Exactly, the same thing, I did here. When you look here, there is a sharp corner. So, what happens here? You can go and sit on that point actually on this point or here the c_g.

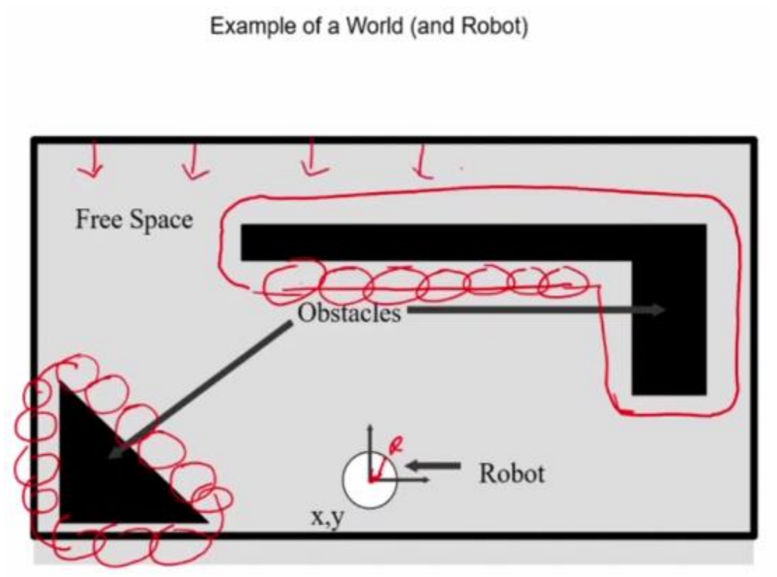
The contact point is still the same. But, the c_g's are different because it is an arc there. So, in both positions, Position 1 and Position 2, the 2 extreme cases, the contact point is still the same point. It is like the circle is rotating about that point. Now, this is what is, this is how we make the C space obstacle. So, what has happened is the workspace has become smaller by a small amount by a amount R and the, of the obstacle has expanded.

How it has expanded. Let me just draw the shape. So, there will be a curve here. There will be a curve here. And, there will be a curve there. It will become like this. It is going like this. So, that is how we have come from here to there. Now, if I were to draw the workspace here and let me just complete the workspace. So, this is my workspace. So, this is my workspace. So, workspace is this one now. So, now, the robot has become a point.

So, it can go anywhere inside here. So, if I want to go from one point to another point say I want to go from the left side is coming out of the boundary areas we are seeing here. So, this has come out. So, the left side boundary is hitting this. It will go like that. So, it cannot go there. So, now, to find a path from start point to goal point, it can go like this. It cannot go on this bottom side because there is an, the obstacle is moving out of the boundary now.

So, there is no way. So, this gives you an idea of how to convert the robot to a point robot and then how to go from the Cartesian space to the configuration space.

(Refer Slide Time: 46:42)

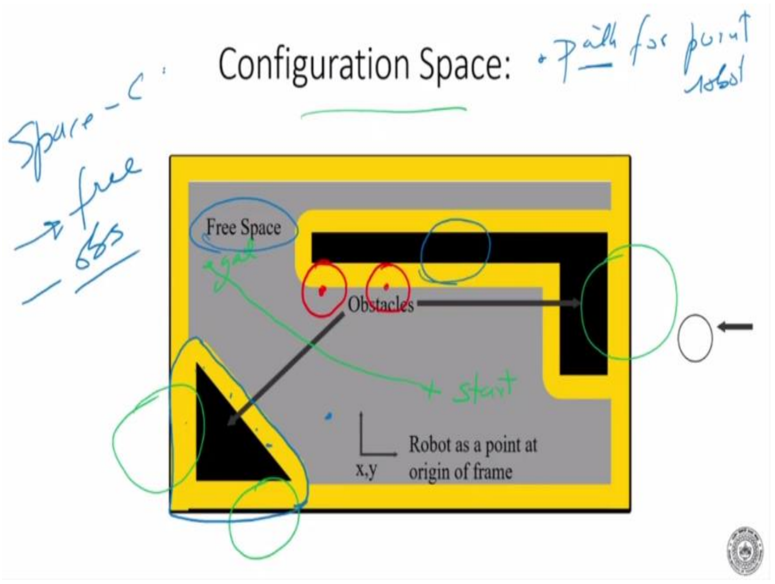


Now, let us see a little further and see that we have 4 other kind of shapes, example of real world configuration space. So, we have an obstacle. And, we have a mobile robot which is a circular robot. Now, the obstacle is marked in black and we want to convert this into the configuration space. So, what do I do? I shrink my robot by R. It becomes a point. And, I increase these obstacles by the corresponding amount that is by R.

So, here, it will become a curve. Wherever there are sharp corners, there will be a curve, similarly, here. So, this will I can simply wherever it is possible for the robot to go and sit. So, I will take the locus of the c_g at the center of the robot and I am going to complete it. So, it will be like this. Like this. Here, it will curve to be curved. It will go like this curve. Again, like this curve, again straight, curved down, curve and back here.

So, now, what has happened is this outer space has come down by some amount by R not by some amount by R and the corresponding obstacle has expanded. Let us look at a little bit more here.

(Refer Slide Time: 47:56)



So, the yellow one is showing where the robot is going and sitting. So, it is something like this my robot now. That is the c_g. So, you can see very clearly that in the previous case there appeared to be gaps here. There appear to be gaps there but and gap here. So, it was not very clear. The robot could go that side. But, the moment we have come to configuration space, what has happened is you see there is no space here, no space here.

Similarly, there is no space there. So, now, if I want to go from one point to another point, I have a start point here. Let us say this is my start and this is my goal, then the only possibility is that I have to go from this side. I cannot go from any other side because it is very clear now. So, what it basically did is and what you can imagine and guess by now is that before starting our planning algorithms we can come to the configuration space.

And then, convert the robot to a point. Then, plan the path for point robot. So, we are finding the path for a point robot. And, the other thing that also becomes very clear from here is that the configuration space looks like the Cartesian space. But, it is not the Cartesian space. Now, so, what you can guess from here is that the whole of path planning algorithm actually has to do with free space in obstacle space.

So, that, you would have guessed by now. So, if you want to find your path you basically divide the space divide the C space into free space and obstacle space. So, this is my free space. And, this is my obstacle space. This is my obstacle. So, basically the path has to go from start point to goal point without entering into the obstacle space. That means it has to move on in the free space.

That is not very difficult because geometrically you are moving a point. So, if you know what is the coordinates of this points or you know the vertices of the point you know the coordinates here, so, you can just check that if, the point should not go inside anywhere there. That is the meaning of the robot going inside an obstacle. So, the paths are basically planned in Cartesian space. And, this is how we come to the Cartesian space.

So, today, we will stop here and then proceed. Now, what I would suggest is that please draw a few sample spaces Cartesian spaces and C spaces. And then, for a mobile robot, please understand how we move from the Cartesian space to the C space and also understand that why we are doing this. It is very important to understand that path planning will be basically done for a point robot.

Otherwise, if you are trying to plan for a circular robot which is moving in different directions, you can imagine that. And, you can of course imagine in the case of the arm robot the serial arm, now, where the serial arm is going in which direction whether it is interfering with an obstacle that is extremely difficult to find out. But, if you convert it into a point then it is pretty easy.

You know this is obstacle space. That is free space. Plan your path in free space. And, that will ensure that you are never hitting the obstacle. So, we will stop here today and we will continue in the next class. So, thank you.