

Robot Motion Planning
Prof. Ashish Dutta
Department of Mechanical Engineering
Indian Institute of Technology - Kanpur

Lecture – 6
Bug Algorithms

Hello and welcome to this lecture number 6 of the course robot motion planning. In the last couple of classes, we have looked at the basic background review of mobile robots and serial arms that is forward kinematics, inverse kinematics, transformations to give you an idea of how robots move and how do we analyse robotic systems. Now using the knowledge, we will move on from today to the formal study of the course robot motion planning.

So today, we will start off with the first and the earliest algorithms, which are called the bug algorithms. Now, as the name suggests bug algorithms means the way bugs move around, bugs mean like ladybird, cockroach, we must have seen them moving around. So that is where the idea of bug algorithms came from. And these are the earliest algorithms.

(Refer Slide Time: 00:59)

The earliest **BUG** algorithms



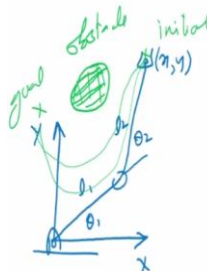
Dr. Ashish Dutta
Professor, Dept. of Mechanical Engineering
Indian Institute of Technology Kanpur, INDIA

So we start the study of our formal course today after the revision. Today also I will briefly quickly review what we have done in the last couple of classes regarding forward kinematics, inverse kinematics, and then we will move on to the earliest bug algorithms. Now as the name indicates bug algorithm, you can already see a bug on the screen and it is showing that this is a ladybug and how the bug is moving from one point to another point.

It has legs and it is moving forward in a straight line. Now if there is an obstacle, the bug will avoid the obstacle and will simply go to the other side avoiding the obstacle, right. So the earliest bug algorithms were designed after looking at the way insects move in nature. Now, before we move on, let me very quickly go into forward and inverse kinematics.

(Refer Slide Time: 01:44)

Forward and Inverse Kinematics $(\cos(\theta_1 + \theta_2))$



$$x = l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2) \quad \text{--- ①}$$

$$y = l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2) \quad \text{--- ②}$$

FK = given $\theta_1, \theta_2 = \text{numerical value}$
 $x, y?$

IK = given (x, y) find $\theta_1, \theta_2?$

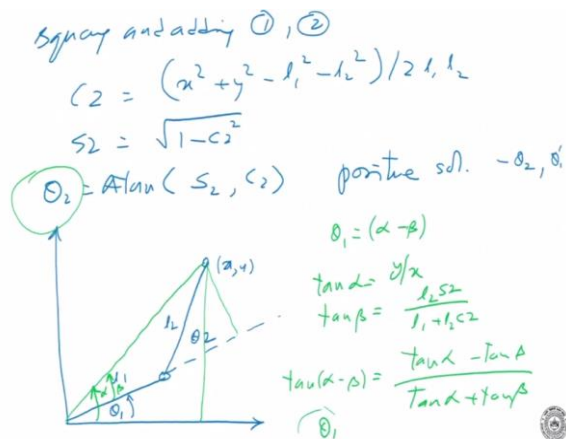
multiple Sols.

So in the last class, we looked at forward and inverse kinematics and we said that if you have two-link manipulate a system like this, this is two links for simplicity, this my X axis. Y axis and this my point x, y. So in terms of the link length l_1, l_2 and the link variables θ_1 and θ_2 , I can write this as $x = l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2)$ and $y = l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2)$.

Now in forward kinematics, I am writing in short by FK where we are given θ_1, θ_2 , it is a numerical value, some numerical value, and we are asked to find what is x, y. This is my forward kinematics problem. Now, it is very simple because in these two equations, equation 1 and equation 2 we can simply put the values of $\cos \theta$ here $\cos \theta_1$ and θ_2 and we can get what is the values of x and y, so this is a very simple.

Whereas inverse kinematics that is IK what we are given is given (x, y) of the end effector, find θ_1 and θ_2 . This is a little bit more involved why because you can see that the way these terms appear, it is $\cos(\theta_1 + \theta_2)$, so we cannot directly solve and also there is the question of multiple solutions. So, there are multiple solutions which are possible. So, in the last class we looked at the inverse kinematics solution of this two-link arm and we found θ_1 and θ_2 .

(Refer Slide Time: 03:31)



So, squaring and adding equation 1 and 2, we basically got $\cos \theta_2 = \frac{(x^2 + y^2 - l_1^2 - l_2^2)}{2l_1l_2}$. And

from $\cos \theta_2$, we got $\sin \theta_2 = \sqrt{1 - \cos^2 \theta_2}$ and then we used the A tan function to get, $\theta_2 = A \tan(\sin \theta_2, \cos \theta_2)$. This gave us the positive solution. And once you get the positive solution, what is the other solution.

So it become $-\theta_2$ this is the other solution and then you have to find what is θ_1 . And that we found geometrically by using geometry. And you can also find it by using algebraic method, but this is an easier method, geometrical methods are easier. So, this is my x, y and this is my l_1, l_2 . So, we have got θ_2 , two that means I have got this angle θ_2 and what I want to find is θ_1 now.

So, geometrically we completed this triangle, so this triangle here like this and this triangle here. So, what we did here is we defined an angle called α and we defined another small angle called β . So, what we found is that $\theta_1 = (\alpha - \beta)$ as shown in this figure right and then we found expressions for a $\tan \alpha$, so $\tan \alpha = \frac{y}{x}$ and $\tan \beta = \frac{l_2 \sin \theta_2}{l_1 + l_2 \cos \theta_2}$.

We then used the formula $\tan(\alpha - \beta) = \frac{\tan \alpha - \tan \beta}{\tan \alpha + \tan \beta}$. So, this is what we did in the last class and from here taking tan inverse you can find the what is θ_1 . So, once you find θ_2 and you

find θ_1 , then the problem is solved. Now, we will be dealing only with two-link systems here in our course because it is simple. If you go higher than two links it becomes more complicated with the kinematics.

So, this is just to understand the forward kinematics and inverse kinematics of this two-link manipulator system. So, what is the problem here? What is the problem statement? The problem statement of motion planning would be that if you have an obstacle here like this, let us say there is an obstacle green colour obstacle and you want to go from this point to this point. So, this is my goal and that is my initial point. How many ways are there?

And if there are a large number of ways of going from this initial point to the goal point? Then which path is going to give you the least energy path, least energy or least time all these distances whichever way? So, this is basically the path planning problem, where you are given an obstacle and you have a robotic mechanism either a serial link or a mobile robot and we are asked to find what is the shortest path.

(Refer Slide Time: 06:43)

Jacobian and singularity

$(x, y) = (l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2), l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2))$

velocity at end effector

joint velocity

end effector vel

joint vel

end effector vel

Jacobian (2x2)

$$\begin{pmatrix} dx \\ dy \end{pmatrix} = \begin{pmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} \end{pmatrix} \begin{pmatrix} d\theta_1 \\ d\theta_2 \end{pmatrix}$$

$\dot{x} = J \dot{\theta}$

Now, so in order to find the energy, we looked at the concept of Jacobian and singularity. And we said that for this manipulator system, I will draw very quickly here. For this manipulator system that I just drew $x = l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2)$ and $y = l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2)$.

The same figure that I drew some time back l_1, l_2 , this is $(\theta_1 + \theta_2)$. Now, if I take the derivative of this, basically I want to find the relationship between the velocity at end effector and joint velocity.

So, I want to find the relationship between the end effector velocity and joint velocity why because as far as the control system is considered, we are going to control the velocity of the end effector by controlling the joint velocity because there is a motor or an actuator at the joint which we are controlling using the control system, right. So, we are more interested in looking at the velocity relationships plus we are interested in the energy consumed. You know that more the velocity, more will be the energy.

So, somehow this velocity and energy are related that is also the reason why we need to look at the relationship of the velocities. So, in these two equations, we see that x and y is a function of θ_1 and θ_2 . So, it means that I need to find the derivative of this, partial derivative with respect to θ_1 , one then with respect to θ_2 . If I do that I get,

$$\begin{pmatrix} dx \\ dy \end{pmatrix} = \begin{pmatrix} -l_1 \sin \theta_1 & -l_2 \sin(\theta_1 + \theta_2) & -l_2 \sin(\theta_1 + \theta_2) \\ l_1 \cos \theta_1 & l_2 \cos(\theta_1 + \theta_2) & l_2 \cos(\theta_1 + \theta_2) \end{pmatrix} \begin{pmatrix} d\theta_1 \\ d\theta_2 \end{pmatrix}.$$

Now, this is called the Jacobian and in this particular case it is a 2×2 matrix and we can write it in shorter $\dot{X} = J \dot{\theta}$. What is \dot{X} ? \dot{X} is the end effector velocity and this one is the joint velocity. Now, you would understand that this mapping is not one to one mapping. What I mean by that? Suppose I want end effector velocity 1 m/s, it does not mean that I will need a joint velocity of 1 radian /s, it is not one to one.

So, what we are looking at is like if I draw it this way, so this is my joint velocity and this is my end effector velocity, there is some mapping which goes on. So, we are interested in looking at this mapping and this mapping is controlled by the Jacobian. So, things like whether it is linear versus nonlinear, whether the mapping exists at all, for example a singularity the mapping may not be there at all.

(Refer Slide Time: 09:40)

Minimum energy $\dot{X} = J\dot{\theta}$

$\Rightarrow \dot{\theta} = J^{-1}\dot{X}$

$\dot{\theta} = \frac{adj|J|}{\det|J|} \dot{X} \Rightarrow \text{if } \det|J|=0 \text{ then } \dot{\theta} = \infty.$

$\det|J| \rightarrow 0 \Rightarrow \dot{\theta} \rightarrow \infty$ infinite energy.

Singularities:

[Min energy \rightarrow Max $\det|J|$

[Min energy \rightarrow stay far away from Singularity.

- Min energy $\theta_2 = 90^\circ, 270^\circ$

So what we are interested in is trying to find the minimum energy. And we saw that

$\dot{X} = J\dot{\theta}$ and what we are interested in is $\dot{\theta} = J^{-1}\dot{X}$, why because we are interested in controlling the velocity of the joint that is how we control the robot, right. So what we see is

$\dot{\theta} = \frac{adj|J|}{\det|J|} \dot{X}$. Now, this relationship shows us that if $\det|J|=0$ then $\dot{\theta} = \infty$.

So, as the $\det|J| \rightarrow 0$ implies that $\dot{\theta} \rightarrow \infty$, now going towards ∞ means infinite energy which is obviously not possible. So, these points are called singularities. So, it serves two purposes. Number one is if you want to find the minimum energy requirement, so minimum energy would mean maximize $\det|J|$ directly so that gives us one way. The other is minimum energy means stay far away from singularity.

As nearer you come to a singularity, the more you will have, your energy requirement will go up and finally it will go to ∞ . So it serves the two answers of minimum energy that first you have to maximize determinant and then you can minimize the energy also by staying away from singularity. So, for example, I am just giving an example if there is a singularity, let me take the example of this two-link manipulator.

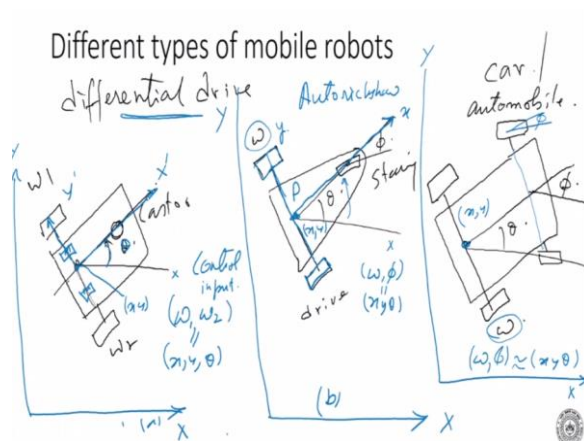
So, this is small and so the work volume of this will be something like this, this will fold for this much and here. So, this is a boundary singularity, this also boundary singularity. So, if you are trying to go in a path like this which is coming very near the singularity that will

consume more energy. Whereas if you take a path which is going far from the singularity, away from the singularity somewhere here, then that is going to take minimum energy, it will take less energy.

For a two-link manipulator, we also saw that minimum energy corresponds to $\theta_2 = 90^\circ$ and 270° . So when you are planning the motion, we can make use of this business of minimum energy to find a path for which the energy consumption is going to be minimum. So now let us proceed with this background what we have done in the last class. Now, as I said in the last class, please I am doing it in very brief.

If you are interested in looking at more details of this, please refer to the Introduction to Robotics course in NPTEL or look at the textbook where we discuss kinematics and forward kinematics, inverse kinematics, then manipulation ability, $\det|J|$ in more detail. Here we are doing just how much we need.

(Refer Slide Time: 12:51)



So, we are going to study about serial arm robots and it will be limited only mainly to two degrees of freedom system, at most three. Now, the other kind of robots that we are going to look at are mobile robots. So mobile robots can be of differential drive type. Now differential drive robot is shown here in figure a where this is a robot, which has two wheels and there is no connection the wheels here, so for example, this connection is not there.

And each of the wheels is driven by a motor. So, motor here and motor here. They are independently driven. Now we have a frame. This is my frame. This is a fixed frame or the reference frame, this is my X, this is my Y. Now what we do is from the centre point of the two wheels, we have an X axis. So let us call it the axis of the car, the axis of the mobile robot, and this is making an angle of θ to the X axis, this is my x.

So what we see here is that we can control this robot by looking at ω_1 , ω_2 . By controlling ω_1 and ω_2 , we basically control the position of this one that is my x and y. So x, y and θ , I need to control, right. So in order to control this basically, I have to control, so my control input is my ω_1 , ω_2 , I have no other control input. So with two control inputs, I am controlling three outputs now x, y and θ .

So this is one type, so let us call it type 1. This is type b. The same thing. I am going to put a reference frame. This is my X and that is my Y. Now, in this particular case there is no differential drive. But what you can see is that the rear wheels are joined and they are driven by drive. It can be a motor, it can be an engine. So it has an angular velocity ω and there is a steering angle in front.

So, this wheel in front can be steered, so it is something like the autorickshaws that you see on the roads. So they have a back wheel and then there is a steering in the front. So, in this case also we have to fix our axis. So this is my x axis, this is my y axis and this is the coordinates x and y of the centre of that point, let us call it P. And again, we have θ is the angle which you are making with the X axis.

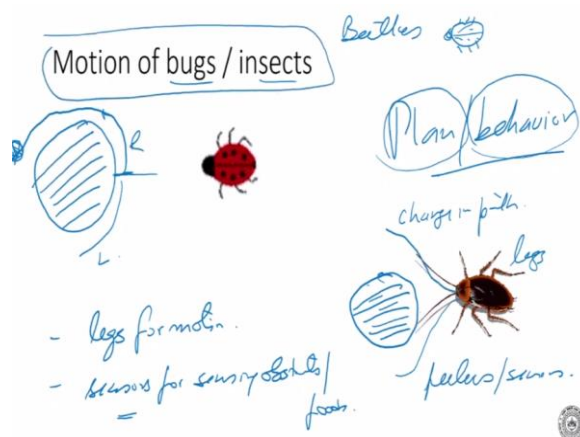
So x, y and θ are the outputs. And the input is what? There is a drive and there is a steering.

Let us call this ω and $\dot{\theta}$. So we have a drive and a steering and both of this is controlling these three output x, y and θ . But the structure of the vehicle is different, you can see that the rear wheels both of them are driven and then there is a steering in front. Now in a car, which is four wheeled, which you can see here again this is a car. So, the rear wheels are driven with a velocity ω and there is a steering in front.

So, this is the steering, steering of the wheels. So this one has a steering angle here that is my Φ . So how do we control the car? We can control the speed of the car by changing ω and we

can control the direction by changing Φ . And again, at this point we have x and y and then we have Φ which is the steering angle. So, my input is ω and Φ . And my output is x , y and θ . So these are the different types of mobile robots that we will look at as we proceed. Now. So let us proceed from here.

(Refer Slide Time: 16:44)



Now the earliest bug algorithms, the idea for them were obtained from basically looking at insects, like motion of bugs and insects. Now, what is a bug? Bug is an insect, you know there are different kinds of bugs, there are beetles. So there are beetles which look like this, I am sure you must have seen them. Their wings also some of them like this. Then there is a ladybird as shown here.

Then there is a cockroach, the good old cockroach that you can see in your houses or in other places where the cockroach has six legs. This bug also shown here has six legs. Now if you observe this, I am sure you must have observed that the insect is going straight. Now suppose there is an obstacle here. Suppose there is an obstacle here, what it will do is basically it after it hits the obstacle or even before it hits the obstacle, it will try and bypass the obstacle like this and go from the other side.

So either it will go on the right side or it will go on the left side. Now that is one. So what we are seeing is that it has legs for motion that is how it is moving. It has some sensors for sensing obstacles. Obstacles, food, it has sensors for that. Now, in the case of the ladybird that we are seeing is moving here, it has 6 legs and it can walk, turn. In the case of the cockroach, it also has 6 legs and it has these feelers.

These feelers are the sensors where it can touch something. For example, if there is an obstacle here, let me draw it like this. So if there is an obstacle here, it can touch that. It can feel that something is there. So, this is basically the legs are for motion and the feelers are for sensing. So, what we are seeing that if you are going to design a mobile robot which can actually move around like an insect, it must have some kind of motion capability.

So, either it has legs or it has wheels. Insects do not have wheels, they all have legs, but mobile robots have wheels. And there must be some kind of sensor for sensing the environment and if there is an obstacle it must sense and then there must be some kind of a plan, plan or behaviour of what is the objective here. So, for example if this lady bird wanted to go from here to here like that, so it can bypass that and go there.

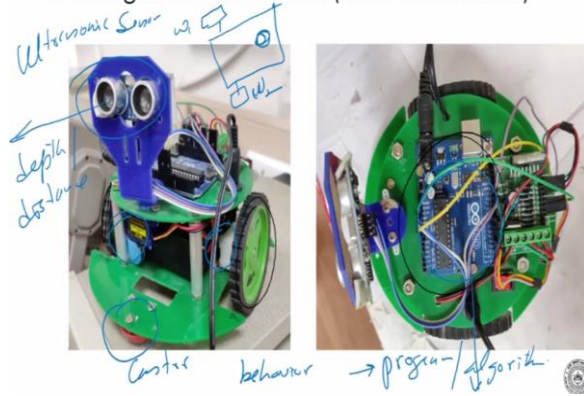
Now, if the cockroach is trying to avoid this obstacle, it will touch the obstacle and it understands the moment it has touched and then it can change its path, either that side or this side. So, what is the behaviour? The behaviour is the change in path. Now, in order to do that, in order to have a particular behaviour and to have a particular kind of plan, then it needs number one is some ways of moving legs or wheels and it needs sensors also to sense the environment, and then it can force a plan.

In this case, the plan is very easy for the cockroach, go left, go right. Go left or go right, it can do both. Similarly, for the lady bug also it can go left or it can go right. So, we are in this path planning and motion planning course, we are interested in finding the path in going from an initial point to a goal point, maybe avoiding obstacles. So, here the mobile robot will need some ways of moving forward and it will need some kind of sensor for sensing the environment.

And then it will have some kind of a plan as to when hits the obstacle what to do that is the behaviour. But in this case it is very simple, go left or go right. So the earliest ideas of bug algorithms came from this motion of bugs and insects.

(Refer Slide Time: 20:38)

Working of mobile robots (differential drive)



Now let us come to mobile robots, which we will be dealing with. So working with mobile robots having differential drives, now this is a differential drive mobile robot. So, you see that there are two wheels and there are motors there. And these motors are independently activating the two wheels and there is a castor wheel in front. So, there is a castor wheel in front which is providing support.

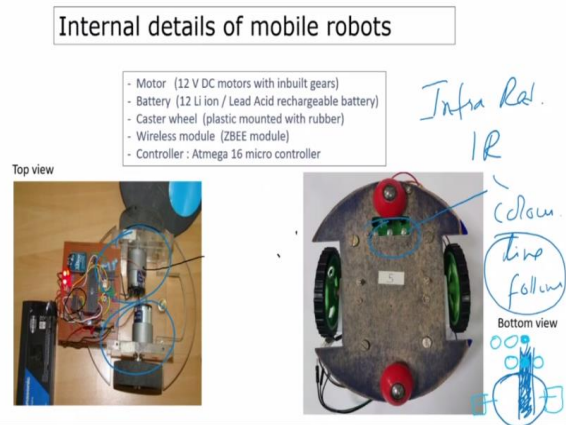
So, this is similar to our figure which is shown some time back that there are two wheels and there is a castor wheel here. So this is ω_1 , ω_2 and it basically moves by controlling ω_1 and ω_2 . In front there is an ultrasonic sensor. Now, this ultrasonic sensor basically can check for depth or distance. So, if there is something in front it can sense that there is something in front and it can enforce some kind of a behaviour, for example go left, go right or if there is something in front just stop.

Now, this is controlled in this particular case by an Arduino microcontroller. So, the behaviour of this will be a program, which will be written in the form of a programme and the behaviour is going to be embedded in a programme. So these are the elements. So the programme is the one that is going to actually determine what the robot is going to do. So the programme is the one that will find the path.

So in pipelining, our essential function is to write a program or an algorithm that will enforce some kind of behaviour or a plan. So, you might have been familiar with this kind of robots, which a lot of students make in high school these days and they are available in

supermarkets. So, we can do motion planning path planning with these types of very simple robots.

(Refer Slide Time: 22:28)



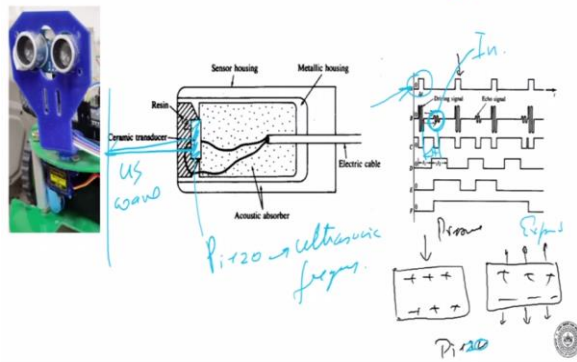
Now, this shows the internal details of the mobile robot. So it has DC motors which are driving it. And in the previous one there was an ultrasonic sensor, which is sensing if there is an obstacle in front. In this one, there is an IR sensor there, so IR stands for infrared, infrared sensors. So this also can check. It can check for colour, so for example, this is using if you are following a line for line following kind of robots.

So, if you draw a line like this, for example I have a black line and I leave the robot on this black line. So, what it can do? These infrared sensors can sense that where is the black line, say for example these are three infrared sensors. So, you can have a logic to ensure that two of the central infrared sensors will always remain on the line. So, the moment two have this comes out it immediately understands that it has gone left.

So it has to actuate the wheels to go right. So that is the behaviour of the robot now. So, this is just to give you an idea of what the path plan actually does or what is our objective here. So, there are different types of sensors. There are actuators for the robot and we can enforce different kinds of behaviour of the robot.

(Refer Slide Time: 23:44)

Range sensor : Ultrasonic sensor *← Range*

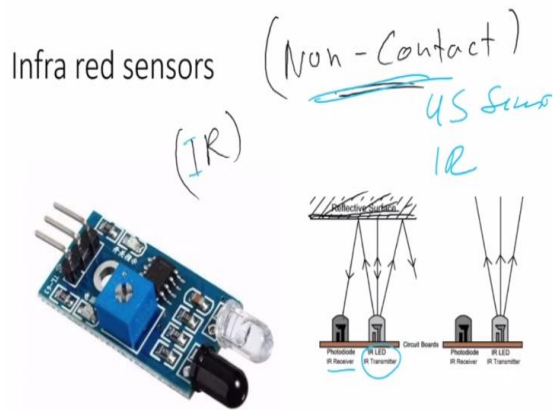


Now, very quickly looking at the ultrasonic sensor, so ultrasonic sensor basically has a ceramic transducer inside here which is basically a piezo. So, piezo crystal as you know that if you give pressure, it will produce charge; and if you give it charge it is going to expand. So, it will expand here. So, what is done is that this piezo crystal is actuated at a particular frequency. Now, this is shown here. So, the piezo crystal is being actuated at a particular frequency and it is vibrating.

So, this is a piezo crystal is getting actuated at a particular frequency and this is the ultrasonic frequency. So because it is vibrating at that frequency, it is going to give out an ultrasonic wave. So, the ultrasonic wave goes out from the front like this. So it is an ultrasonic wave. Now the moment there is something in front, it will hit the object and then it will come back. Now once it comes back, it causes the pressure on the crystal and the pressure produces a charge. So depending on how much time it took to go and come back, you can compute.

You know the velocity of sound in air, so you know exactly how much time, what is the distance of the object from the sensor. So, this is the out signal, this is the in signal. So, this my in signal. So, the distance from here to here is my Δt , the time that it took to go and come back. So, now you know what is the distance of the object. So, these ultrasonic sensors which are very simple and they are extensively used in mobile robots to check if there is some object in front to avoid obstacles.

(Refer Slide Time: 25:18)

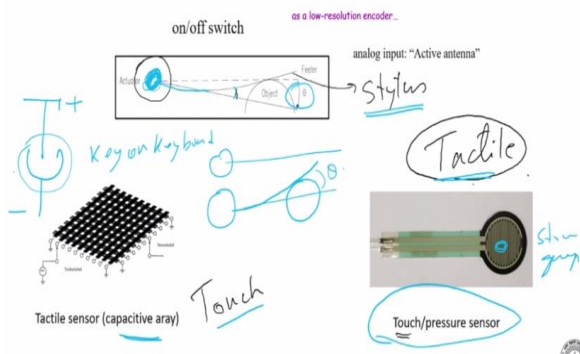


Infrared sensors, again they can also sense reflective surfaces. Now, in infrared sensor there is an emitter, you can see there is an LED which is emitting light and there is a photodiode which is the receiver. So, this LED gives off light and the reflective surface reflects this light, IR infrared rays and it comes back and gets received here. So, exactly here also it can exactly figure out colour of an object or reflective surface in front.

So, IR sensors can also be used in mobile robots extensively. Now you can see ultrasonic sensors and IR non-contact type, they do not touch, from far they can make out. So the robot can enforce a behaviour simply when it gets the signal input. So if the program receives the signal that there is something in front, so it can immediately change its behaviour, it can try and avoid the obstacle.

(Refer Slide Time: 26:14)

Tactile sensors for detecting contact with object



The other kind of sensors are the sensors which are contact sensors like the one that you looked at with the cockroach. So in the case of the cockroach, we saw that there is stylus in front, so the stylus is there, the stylus is there for touching. okay It is very sensitive, it can sense not only obstacles, it can sense if there is food, if there is chemicals it can sense. It is like our tongue basically. And with this, it can sense if there is an object in front and then enforces behaviour.

So, in this particular case, it has to physically touch the object in this robot where you can see that this is a stylus. So, it is a stylus and if there is an object which is touching, what will happen is the stylus will bend. So, if it was not touching it would be straight like this. Now, because it is touching, what is happening is that it is getting bend and there is an angle there. So, if there is a sensor here, we can use a potentiometer or an encoder which will actually sense that the stylus is touching something why because it has bent.

So, basically looking at the angle α , sorry this angle θ . So, basically if it was not touching it will be straight, and if it is touching something it will immediately bend, so if it is touching something it will become like that is touching. So, basically this is being sensed and based on that we know that there is something in front. This is also very simple sensor for a robot to make and robots use this kind of sensors for touch the wall and follow the wall, pull the object, etc.

The other is tactile sensor, which is actually sensing the pressure. So, there are different kinds of touch sensors or tactile sensors. So, this one is a capacitive array. So, if this touches something immediately what happens is there is a plus minus on top, so it is a grating like, on this side there are gratings like this, the bottom side there are gratings like this. So if it touches what will happen is; let me explain it like this if it touches what will happen, there is a plus and a minus.

So there is one on top and there is one at the bottom. So, it is something like this and there is a plus on top. So, this is plus, this is minus. So, if you touch it what will happen? This will get shorted. The moment it gets shorted, it is something like a key on keyboard. As soon as this is pressed that part is shorted and it gets a signal. So, the robot understands there is something in the front. This is another kind pressure sensor where if you touch, this a resistive pressure sensor, so they will change the resistance.

So, it immediately understands if anything touches. This is something like our string gauge. So, these are contacted, the earlier ones we have seen are noncontacted, IR and ultrasonic sensor. Now, depending on the kind of sensor we have the kind of range of the sensor, we will have to enforce different kinds of plans. Now, let us move on further and now come here to the bug algorithm.

(Refer Slide Time: 29:10)

BUG Algorithm (Program) *objective find a path from initial pt. to goal pt.*

- Robot is a point robot (no physical dimensions)
- It has a touch sensor to detect contact with obstacles.
- Algorithm input: start point, goal point, obstacles.

Plan/enforce.

• Behavior: Motion to goal •

Obstacle: Boundary following

Now, what is the objective of the bug algorithm? The objective is to find a path. So objective, find the path from initial point to goal point. So, this is shown in the figure here. So, I am marking my X axis and my Y axis and this is my start point. We call it start and that is my goal and these are obstacles. Now a couple of simplifications we make, assumptions we make. The first is that the robot is a point robot, it has no physical dimension.

So, we are considering this robot to be a point robot. Next is it has a touch sensor to detect the contact with the obstacle. So, the moment it makes contact with any obstacle, it knows that contact has been made. Now what is the objective? The objective is to find a path from the start point to the goal point avoiding the obstacles. Well, not avoiding the obstacle, touching the obstacle not going through the obstacles.

So the objective is to find a path from the initial point to goal point. What is the plan that we will have to enforce? The plan that or the behaviour that we are going to enforce is that it will have a motion to go behaviour and it has a boundary following behaviour. Now, what is the start of the algorithm? Now, something to note here is that if I asked you which is the path,

you can simply tell me that go like this, go like this, or rather do not go like that, go like this, go like this, go like this, go like this.

Now we have eyes, so you can see there is an obstacle there and the goal is there. The poor robot does not have eyes and it is in 2D flat ground. So, for us it is very easy to say what is the path. For a robot for a program to find out, it is not that easy. So, the program has to be written such that it can take the robot from the start point to the goal point and it cannot go through the obstacle of course.

The assumptions we have made the robot is a point robot, it has a touch sensor to detect contact with obstacles so they do not try and go, it cannot go through the obstacle. And the input to the algorithm or the program is the start point, goal point and obstacles. So we have to write a program which will take the robot from the start point to the goal point and not through the obstacles, it cannot go through the obstacle.

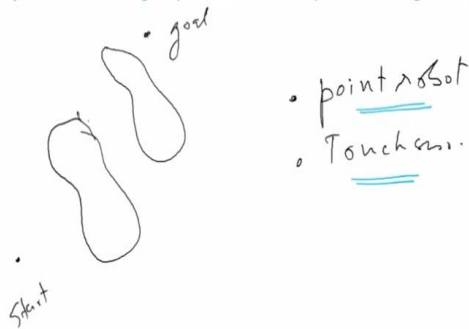
Now, what is the behaviour that we are going to enforce or the program is going to enforce or the plan that is going to be enforced? Now, the first thing we can see is that the robot should move towards the goal, right? Always. So, if the robot starts going this side, then it is going away from the goal, it is not solving anything. So, the first thing is motion to goal behaviour. So, the robot should have a motion towards the goal behaviour.

The second is it should be boundary following, boundary following or rather obstacle boundary following because it cannot go through the obstacles, it cannot do that. So, it should be able to follow the boundary of the contour of the obstacle that is the essential plan for the bug algorithm. So, what have we given? We have given a start point, goal point and the obstacle. Now, the program has to enforce number one motion to goal and boundary following. Let us see how that is done.

(Refer Slide Time: 32:59)

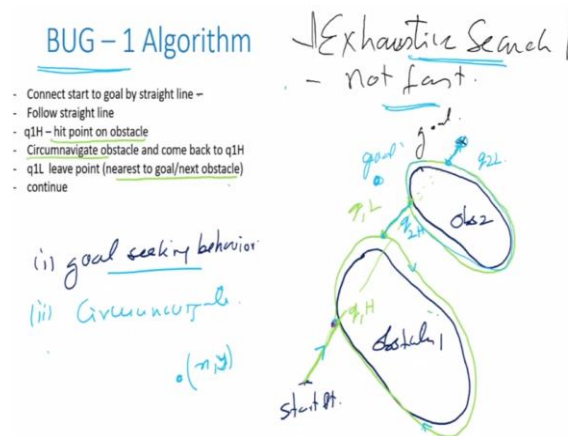
The earliest BUG algorithms for path planning

- Objective : Reach a goal point from a start point avoiding obstacles.



So, in the first algorithm, we have to reach a goal point from a start point avoiding the obstacles, right. So, this is the problem that we are looking at here. Now the robot is a point robot and it has a touch sensor. So now we are supposed to write a program which will actually do perform or execute this motion now.

(Refer Slide Time: 33:25)



Now, the first algorithm that we will look at is the BUG algorithm or the BUG-1 algorithm. So, there are different ways by which you can solve this problem. Let us see how we can think of solving this problem logically. So we have one here and we have another one there obstacles. So, these are obstacle 1 obstacle 2. Let us call this my goal point, this is my start point. Now, the first thing is that it should have goal following behaviour, something we have seen.

So number one is that goal seeking behaviour that means it should always try to go towards the goal because I want to go towards the goal, right, it should not go in some other direction. So, the first thing that we can do is we can connect the goal with a straight line. So, we connect the goal with a straight line like this. Now if there were no obstacles, it will probably follow the straight line and go there.

Now, because there are obstacles, what it does is it follows the straight line until it comes to the hit point. So we call this $q_1 H$, $q_1 H$ is the hit point on obstacle. So q is the hit, $q_1 H$ is the first hit point of the obstacle. So, we are writing a program where the robot is going towards the goal, you know the direction of the goal, it has been given. Now as it is going as we connect it with the start point and the goal point with a straight line, it follows the straight line.

The moment hits an obstacle we know that it has hit, it has touch sensor, so it knows as it has hit. Now what it does? It circumnavigates the obstacle. So, basically it goes round like this, like this all around the obstacle. So, it explores the obstacle and comes back here. So, it explores the obstacle and then comes back there. Now, once it has explored the obstacle now it knows that this is the shape of the obstacle. The mobile robot has actually found out.

I will just change the colour here, so this is next. What it does is it knows this is my $q_1 H$ now and it identifies the $q_1 L$ now, the q_1 leave point. So, the q_1 leave is the next point which is here of this point again. So, it hit that and came back here. So, now it identifies the q_1 leave point which is here. So q , so my initial straight line point was like this. So, now what it does is it identifies the q_1 leave, $q_1 L$ stands for the leave point on this circumference of the obstacle.

Now that q_1 leave point is the nearest to the goal for the next obstacle. So, whichever point is nearest to the goal or it is near to the next obstacle that is the $q_1 L$ point and from the new $q_1 L$ point again, it will continue. So, how it will continue? The same thing it will do, it will go like this and then go here. The moment it has gone there, what it will do? It will circumnavigate once like this and come back here. Now it has seen the full obstacle.

Now after it has seen the full obstacle, what it will do is this is my q_2 hit point, then it will identify the point which is nearest to the goal, this the point which is so, this is the q_2 leave

point. So, this is the q_2 L point. So, now it has come here and now it goes here. So, what it is doing is it is going like this, it is circumnavigating once, coming back to this point, then it is finding the leave point which is nearest to the next obstacle or to the goal.

And then just finding the path which is nearest to the next obstacle. It is circumnavigating again full round like this, coming here, find the point the leave point and then it is going back and getting the goal. This is having goal seeking behaviour that is one. Number two is it can circumnavigate the obstacle. So goal point and circumscribe. Now, as I said before this is a point, robot is a point, it has no eyes, so it cannot see.

It simply is the program which is going to execute this motion. It will always go towards the goal and if it hits an obstacle, circumnavigate, find the leave point which is nearest to the next obstacle or the goal, and then go to the obstacle, circumnavigate and then go to the goal in this case. Now, this search is an exhaustive search and it is not fast why because it is circumnavigating each time.

So if there is a long obstacle, you can imagine it will keep going all around the obstacle, where it actually does not have to do that. So, for example, if the goal was somewhere here, so that was my goal, then for nothing it will be just be going round and round the obstacle and then it will come here, then find the leave point and then go there. But this is an exhaustive search, which means that it has looked at all the possibilities. This is the earliest algorithm called the BUG algorithm.

Now, you can write a very simple program which can do that because a point has coordinates x , y and when it is touching you know that, you know the circumference of this obstacle. So, whenever x , y is hitting or touching that circumference you can find out geometrically. So you know it has touched, you know the shape of the object, so you can circumnavigate very easily. You can find the q_1 L point, then go to the next obstacle. So, this has to be done geometrically. So, path planning is a geometrical problem.

(Refer Slide Time: 39:07)

BUG 1 Algorithm

- Let $q_0^L = q_{start}$; $i = 1$
- repeat
 - repeat
 - from q_{i-1}^L move toward q_{goal}
 - until goal is reached or obstacle encountered at q_i^H
 - if goal is reached, exit
 - repeat
 - follow the obstacle boundary by moving left or right
 - until q_{goal} is reached or q_i^H is re-encountered
 - Determine the point q_i^L on perimeter that has the shortest distance to the goal.
 - Go to q_i^L
 - if move toward q_{goal} moves into obstacle
 - exit with failure
 - else
 - $i = i + 1$
 - continue

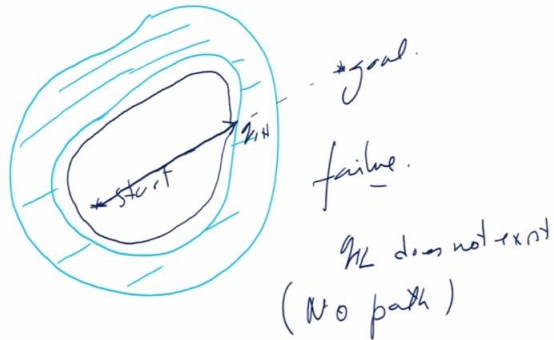
Howie Choset, etc al. - Principles of Robot Motion.

So next what does the algorithm look like? So start with q_{L_0} which is the start point. And then the first thing is from q_{L_1} move towards the goal point until goal is reached or obstacle encountered at q_1^H hit so that is what we did. So, you start off from the start point, go towards the obstacle, if you hit an object that is your hit point q_1^H . Follow the obstacle boundary by moving left or right. Now please note here that it is left or right, you cannot do both.

You cannot go left and right at the same time and then until q_{goal} is reached or q_1^H is reencountered, determine the point q_1^L , leave point on the perimeter that is the shortest distance to the goal. Go to q_{L_1} and then move forward. If you reach the goal, then exit. If move towards q_{goal} , moves into obstacles, then exit with a failure. So, this basically means that we are going to go to this obstacle, circumnavigate, find the leave point, go to the next and then proceed that way.

(Refer Slide Time: 40:21)

BUG 1 – Failure case : enclosed obstacle

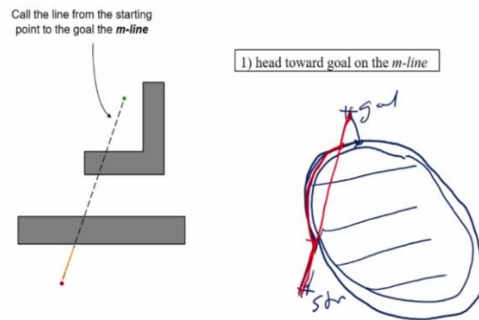


Failure cases of this, will it always work? Well, if then objects which are enclosed objects like this, so we can understand very easily that if you have an obstacle like this and you can see that you have a start point here and have a goal point there. So, what will happen is it will, the algorithm will function because it cannot see, it cannot see that the obstacle is enclosed. So, it will first make a straight line, then follow the straight line get a hit point.

So, this is q_1 hit, circumnavigate like this, come back here, try to find the nearest point to the obstacle, it cannot do that. So, this will end in failure. So in this, the algorithm will start by going towards the goal, it will hit the boundary, it will circumnavigate, come back here, try and find a point a leave point, but the q_1 L does not exist. Because it cannot go out dicing, to the outer boundary there is no connection and there is no way it can go to the goal, so this will be a failure case. So, it should be able to say that there is no path.

(Refer Slide Time: 41:35)

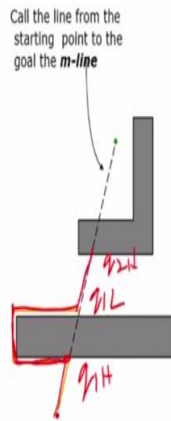
BUG – 2 Algorithm - better



So, when you look at this, whenever we look at path planning algorithms like this, the thing that comes to your mind is, is there a better way of doing this? Is there a faster way? So, the faster way of doing this is the next one which is called BUG-2. So, in BUG-2 what we have seen in the previous cases, there is no need to really go around the obstacle all the time. So for example, if I have an obstacle like this and the goal was here and the start was here, then what it will do is it will come here, then go round like that.

And then go here and then go here and then go there that is a bad way of doing things. That is not a bad way of doing things, but that is one way of doing things. But the other way could be that you connect this to the straight line and follow the straight line. If you hit the obstacle, go around the obstacle, wherever you are getting the straight line, you follow the straight line and then go to the goal. So, I put a straight line like this. So from here, I go here hit, I go like this hit the straight line again and then I go there, I do not circumnavigate.

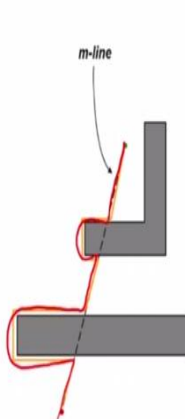
(Refer Slide Time: 42:37)



- 1) head toward goal on the m-line
- 2) if an obstacle is in the way, follow it until you encounter the m-line again.

So, this is showing the path which is the bug algorithm 2. So what we do here is we connect the goal and the start point with the m-line and follow the m-line. So, you can see that it is following the m-line like this, it is gone. The moment it hits so q_1 H, it will go left like this until it finds the line again. Once it finds the line again, this one becomes q_1 leave, it goes there, it becomes q_2 hit.

(Refer Slide Time: 43:09)

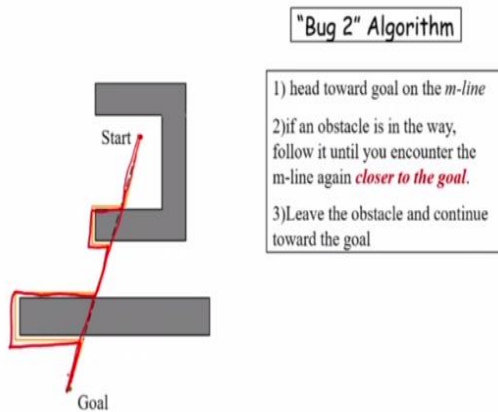


"Bug 2" Algorithm

- 1) head toward goal on the m-line
- 2) if an obstacle is in the way, follow it until you encounter the m-line again.
- 3) Leave the obstacle and continue toward the goal

And from q_2 hit it circumnavigate, it goes on the left side like this and gets there and goes there. This was much faster because it did not circumnavigate. So, this is bug 2 algorithm where we follow a line which is called the m-line. The moment you hit the obstacle you circumnavigate until you hit the line. The moment you get the line, you follow the line, then follow the obstacle and then go to the obstacle. And once hit the obstacle again, then go to find the line again and then you go to the goal point.

(Refer Slide Time: 43:42)



Commented [b1]:

Now this is faster. Now you can see in this particular case also from the start point to the goal point connect with a straight line, follow the straight line, come like this, follow the obstacle boundary, come like this, hit the point and go to the goal.

(Refer Slide Time: 43:58)

BUG 2 - Algorithm

- Let $q_0 = q_{start}; i = 1$
- repeat
 - repeat
 - from q_{i-1} move toward q_{goal} along the m-line
 - until goal is reached or obstacle encountered at q^i
 - if goal is reached, exit
 - repeat
 - follow boundary by moving left or right
 - until q_{goal} is reached or q^i is re-encountered or m-line is re-encountered, x is not q^i , $d(x, q_{goal}) < d(q^i, q_{goal})$ and way to goal is unimpeded
 - if goal is reached, exit
 - if q^i is reached, return failure
 - else
 - $q^i = m$
 - $i = i + 1$
 - continue

Program
Logical plan

slow
left with

Howie Choset, et al. - Principles of Robot Motion.

Now this is the bug algorithm 2 the program, the logic. So when you are writing a program, there is a plan and there is some kind of a logic. Let me call it a logical plan. We start with q_1 . So, q_1 moves towards the goal along the m-line. So, we fix an m-line first and then from the first leave point which is the start point, so this my start point, it goes to the goal point following the m-line.

The moment it hits an obstacle, if goal is reached, exit. If the goal is not reached, then follow the boundary, move left or right. Left or right, you have to choose one, you cannot do both. So go left to right until you hit the m-line again. The moment you hit the m-line, follow the m-line again until you reach the goal or you reach the boundary and then follow the boundary again, hit the m-line and then go to the goal point.

(Refer Slide Time: 44:55)

Search for a path

- Exhaustive search - Bug 1 - more complete *find all possible paths.*
- Opportunistic search - Bug 2 - greedy

• Which is better Bug 1 or Bug 2

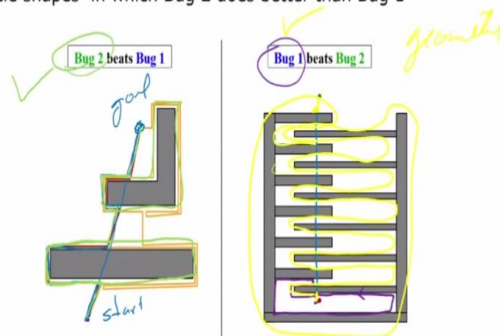
This is basically the bug algorithm 1 and bug algorithm 2. Now bug algorithm 1 is an exhaustive search. It is more complete. It is more complete in the sense that it can find all possible paths. In that sense, it is more complete. So if there is more than one path, it can find multiple paths. But 2 is what we call a greedy search algorithm in the sense that it is an opportunistic search.

It is a greedy search because it tries to find the solution faster and it is faster as you saw in the previous case. So, the question comes which is better, bug 1 or bug 2? When you are path planning, you will always see that this question comes up, is there a better algorithm? Is this better than that? Now, it would depend on the situation or case to case it depends. So, it would just appear to us that bug 2 is better, but you will find that that is not so, it depends on the situation now.

(Refer Slide Time: 45:54)

Comparison of Bug 1 and Bug 2

Obstacle shapes in which Bug 2 does better than Bug 1



No. So let us look at this now. So, in this particular case comparison of bug 1 and bug 2, the obstacle shape in which bug 2 does better than bug 1 in this particular case as shown on the left. So, what we are given is the goal here and we are given a start here. So, it goes straight, gets the m-line. So, let me draw the m-line. It follows the m-line, hits is the obstacle, circumnavigates, hits the m-line again, again then goes like this, like this and then goes to the goal. So, this is faster.

Whereas if I was doing this with the bug 1 algorithm then what it will do is bug 1 will go here, then it will go circumnavigate like this, then it will go back here, it will go there, then circumnavigate everything here, then go here, then go there. So bug 1 is the green line and bug 2 is the blue line. So, we can see in this particular case that bug 2 is faster. What about here? Now, in this kind of situation, we want to go from here to here.

This is my straight line. Let me follow the straight line and see what happens. So, I am following the straight line and let us see what happens here. So, I go straight like this, hit the obstacle. Now I go like this, I go like this, I go like this, I go like this, like this, like this, I hit the obstacle. Now, I can come back like this, like this and come back here again that is one option. The other is I have my straight line like this.

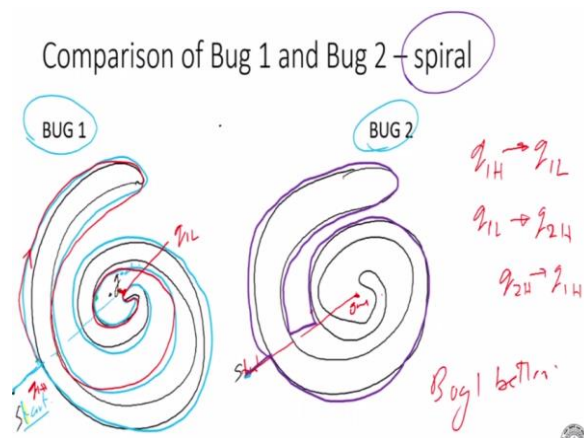
Remember, you can go left or right only, you cannot do both. So here what we do is it follows the line straight, then it goes left like this, it goes here, here, here, it hits the line again comes like this, come like this, come like this, like this, like this is back here again. So, it appears that it got trapped in some way. So in this particular case, bug 1 will beat bug 2. So if

you look at bug 1 what will happen in bug 1 is it is going to circumnavigate all of this and then find the path from here and then find the shortest path.

So in case of bug 1, what it is going to do is it will go from here, it will get stuck, then it starts circumnavigating like this because it is finding the full shape of the obstacle. Then it is coming like this, circumnavigating, now it exactly knows which is the point, where it is the leave point which is closest to the obstacle. So, it will come here and then go there. So, in this particular case bug 1 will beat bug 2.

So, you can see that this path planning problem is very much specific to the problem that is being at or the shape of the obstacles because this is geometry. So, it is very much very related to the geometry of the problem.

(Refer Slide Time: 48:44)



Again, this is another case of comparison of bug 1 and bug 2. So, in this particular case, we have a start point and we have a goal point which is inside. So, this is my goal point. So this is bug 1 and that is bug 2. Now, we can see that this is my start point, you can start here. So, bug 1 what will it do? It will go and first draw the straight line like this, then it will start going straight, then it will start to navigate like this, like this, like this and then come back here.

Now it knows because it has seen the full circumference or the exterior boundary of the spiral. Now it can find the leave point which is nearest to the goal point. So, it will go till here and if this is my goal, then it will go here and then it will reach the goal from there. So in this

particular case, so where was my goal? My goal was, yeah, so after it circumnavigates once, what will happen is it will go like this, like this, like this, like this, like this, come back here.

Next it has to find which point is nearer to the goal. So, it finds that this point is nearest to the goal, so it simply goes there. So, it follows this, goes here, and then simply goes to that goal. So, next shot what it will do is it will come like this, like this, like this, like this, like this, like this, like this and then go to the goal. It knows where is the leave point. So, this might q_1 hit and this is my q_1 leave. But let us see in the case of what happens here?

So, this is my start and that is my goal here. Now, let us see back to what it does? It generates a straight line first. Then what it does is it goes like this, goes from the right side like this here, here, here, here, here, here, here, then finds the line goes this side, now it comes down like this and now what happens is it comes back here again. Now bug 2 has stuck up you can see that. So, in this case also in the case of a spiral, bug 2 loses and bug 1 wins. So, you can see that it is very much dependent on the profile.

(Refer Slide Time: 51:06)

distance.

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

$x = \text{coordinate of the robot center}$
 $\theta = \text{angle of sweep}$

Now, the point robot now has an ultrasonic range sensor. So, we are still with the bug algorithm, but the robot in the previous case had a touch sensor, but in this case, it has an ultrasonic range sensor. So, it can sense the distance to object. So, the range sensor has a finite range of detection and it can sweep 360° . So, if this is my sensor, ultrasonic sensor, it can rotate 360° .

If this is 0^0 , it can rotate like this full 360^0 and it can, so these are the ultrasonic rays which are coming out and can sense distance. So, if there is an obstacle like this, it can sense that there is an obstacle in front. The range sensor will prevent the robot from colliding with the obstacle. Now, you can see here that there are two where one is the position of the robot x and the angle at which it was sweeping, which is a sweep angle at which it was; so we can call it θ , this is my angle θ .

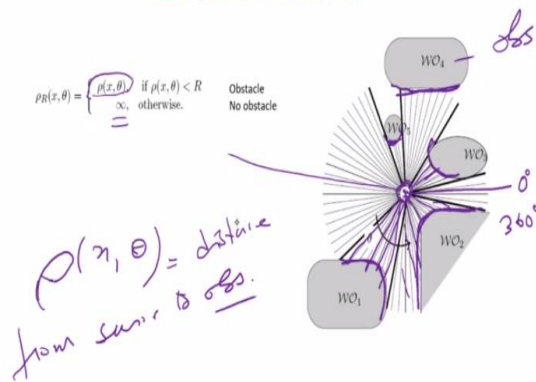
So, what we need to do here is we need to define a distance function which consists of the position where the robot is and what is the sweep angle of the ultrasonic sensor. So, we define a distance function to find the shape of the obstacle because the shape of the obstacle, let me draw the obstacle having some other shape, the ultrasonic sensor has different distances at different angles.

So, the robot x is the same x , but at different angles different θ s you can see that the distance that it is getting of the obstacle is different why because the obstacle has a particular shape. So, basically this ρ is what is going to give us the distance function. So, for the same x for different θ , it will give you a different distance and the distance function is defined as $\rho_{(x,\theta)}$ where x is the coordinate of the robot and the coordinate of the robot centre and θ is the angle of sweep of the sensor.

So, this we are seeing that we are using ultrasonic sensor now, now depending on the range of the sensor and what is the sweep angle of the sensor, you will get a different distance reading now depending on the shape of the object now.

(Refer Slide Time: 53:50)

Raw Distance Function



So, next let us look at raw distance function that we are getting. Now what is the objective here? The objective here is that we need to find out what is the shape of the obstacle. So, we are using this range sensor which is going to give us the shape of the obstacle. Now, so this is my x , let us imagine this my x and θ is going to vary from 0 to 360° one sweep. And this is the shape of the obstacle that we are getting.

Now the shape of the obstacle basically is a function of the distance now. The distance I mean this ray, the length of the ray if I say. So, this is what is giving me the shape of the obstacle now and that is my $\rho_{(x, \theta)}$. There are obstacles here, these are called WO_1 , WO_2 , WO_3 , WO_4 are called obstacles and the moment it hits an obstacle, you will get some value for that. That means that it has some distance, but what is the distance from the sensor to the obstacle that is what is what is my ρ . So, ρ is the distance from sensor to obstacle.

And if there is no obstacle, then it will go infinity of course, this will go in towards infinity that means nothing is there in front. So, basically, we are looking at this distance function and then trying to decide the shape of the obstacles. Today we looked at trying to find out the use of bug algorithms which is based on how bugs move in the real world. And we also saw that the bugs have locomotion and they have some kind of sensors, for example touch sensors or they can have vision.

Correspondingly in the robot, we have a robot which moves around with wheels and the robot has some kind of sensors. So it can have touch sensor, it can have ultrasonic sensors distance

sensors. Now, we stop at the condition where we are using a mobile robot, which is having an ultrasonic sensor and to check the distance it basically looks at the how far the object is and it derives a function, which is the distance, what we call the distance function is a function of the sweep angle and the position of the robot.

So today, we will stop here. In the next class, we will continue from here and look at how to figure out the shape of the object and then how to plan your path from where the robot is to the goal point. And we will see that it depends on the range of the sensor also. So, we will stop here. Thank you.