6.270 2003 Team 39

"Radar"

This notebook describes the mathematics involved with designing a primitive "radar" system from a servo and a distance sensor. The system is able to detect game objects at a useful range of approximately two feet.

It is also possible to extend the system to classify obstructions, for example to distinguish a ball from a wall, with a reasonable degree of accuracy. These latter capabilities, however, can take a significant amount of difficult programming, and if you combine them with an actual contest program then you stand a fair chance to reach the memory limitations of the HandyBoard, and a fair chance to invest a lot of time that would have been more useful elsewhere.

In the interests of legitimacy, I will describe the system only in these abstract mathematical terms, rather than copy-and-paste code. There are many rather complicated details of the system that are specific to the 2003 Montezuma's Revenge contest; because this document is primarily intended for future contestants, I will omit these.

■ Step 1. Preparation

During the implementation of this system, it may be useful to make use of certain trigonometric functions that are not immediately available in Interactive C, but which can be implemented with some basic identities. In addition to the definitions of Csc and Sec in terms of Sin and Cos, respectively, these are:

$$\mathrm{Arcsin}(x) = \mathrm{Arctan}\left(\frac{x}{\sqrt{1-x^2}}\right)$$

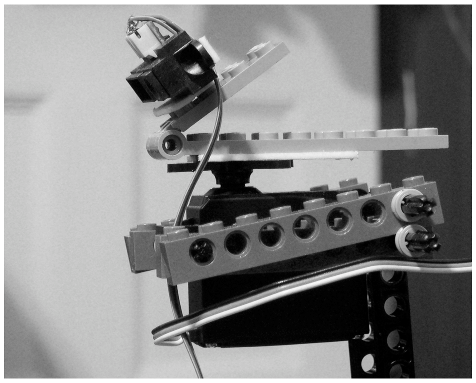
$$\mathrm{Arccos}(x) = \mathrm{Arctan}\left(\frac{\sqrt{1-x^2}}{x}\right)$$

$$\mathrm{Arcsec}(x) = \mathrm{Arccos}\left(\frac{1}{x}\right)$$

$$\mathrm{Arccsc}(x) = \mathrm{Arcsin}\left(\frac{1}{x}\right)$$

■ Step 2. Construction

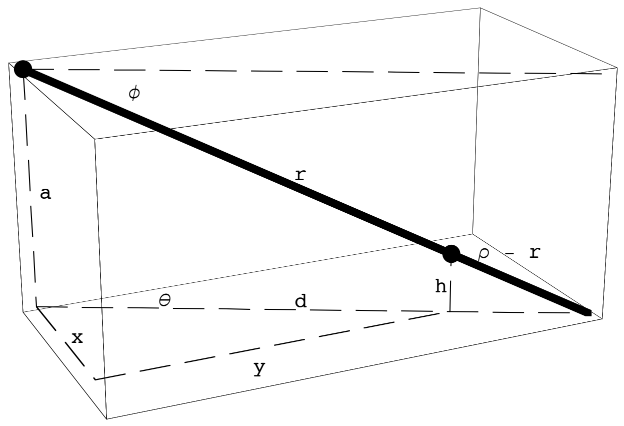
Here is how we constructed our radar. As you can see, it is a distance sensor taped to a pivoting bracket, taped to a servo. The sensor should be mounted such that the distance sensor is facing "forward" when the servo is in its center (pulse=2000) position. The servo is mounted sufficiently high above the robot such that the "view" of the distance sensor will not be obstructed by any part of the robot.



The distance sensor should be pointed downward at some angle ϕ. The inclination determines the range of the sensor; a sharper inclination means a wider detection range but lower maximum range, while a gentler inclination means a narrower detection range but higher maximum range. More on this later.

Step 3. Coordinate System

I will now describe a coordinate system that we will use to operate the radar.This system is closest in principle to spherical coordinates, but is especially designed for our radar. (Most notably, the radar is always pointed "down", and there is a fixed "ground".)



This diagram does a mediocre job of depicting the following:

- **The sensor is positioned at the dot in the upper left corner of the box.**
- **The sensor is at elevation a above the ground.**
- **The sensor is pointed downward at angle ϕ from the horizontal.**
- **The servo is pointed at angle θ from the "forward" position.**
- **Given a fixed ϕ and a, there is a fixed distance ρ from the sensor to the ground.**
- **There is an obstruction on the game table, x to the right of the robot, y in front of the robot, and at least h tall. The distance from the robot to the obstruction is $d = \sqrt{x^2 + y^2}$.**
- **The distance from the sensor to the obstruction is r.**

By thinking about it a little, we can work out the following relations:

$$\rho = a \csc(\phi)$$
$$h = a - r \sin(\phi)$$
$$d = r \cos(\phi)$$
$$x = d \sin(\theta) = r \cos(\phi) \sin(\theta)$$
$$y = d \cos(\theta) = r \cos(\phi) \cos(\theta)$$

Step 4. Basic Operation

The radar operates by measuring r at various values of θ: you set the servo to some angle, then take a distance measurement at that angle. You can convert a desired value of θ (in radians) to a servo pulse value with the relation

$$\text{pulse} = 2000.0 - 1209.58\,\theta$$

This can sweep the radar approximately over the range $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$. You may need to change the sign of the θ term depending on how you mount the servo. Note that the Interactive C servo() function is asynchronous: it returns control to your program before the servo actually reaches the designated position. Therefore, you will need to pause for a brief amount of time before taking a distance measurement in order to allow the servo to synchronize to the new position.

Given a value of r at angle θ, you can conclude:

● If $r \approx \rho$, then there is no obstruction bearing θ from the robot. Of course, it will almost surely never be the case that the r measured by the distance sensor will exactly equal ρ, so you will need to make some thresholding approximations.
● If $r < \rho$ by some significant amount, then there is an obstruction bearing θ from the robot, x to the right, y in front, and which is at least h tall, where x, y, and h are given by the relations above. Again, it is up to you to determine what a "significant amount" means.
● If $r \gg \rho$, then the edge of the table is less than d away at angle θ.

■ Step 5. Calibration

This should actually have come before, but it makes more sense having already described how the radar works.

The assumed constants in our system are a, the elevation of the sensor above the table, and ϕ, the angle at which the sensor is pointed down. You can determine a with a ruler once your robot is constructed. It is best to measure ϕ during calibration with the following procedure:

● Set the servo to $\theta = 0$, straight ahead.
● Ensure that there are no obstructions ahead of the robot, and measure r.
● Since there are no obstructions, $\rho = r$; the distance sensor reads the distance to the ground.
● Finally, $\phi = \text{Arccsc}(\frac{\rho}{a})$

■ Step 6. Less Basic Operation

The knowledge that an obstruction lies at some bearing from the robot is helpful, but obviously there is a caveat: it might be a ball, in which case we'd want to pick it up, but it could be a wall, in which case we'd want to avoid it! Later, I'll describe some advanced techniques for distinguishing the two. It is much easier and practically as good, however, to use a few tricks:

First and foremost, it is often the case that you can drive your robot accurately enough such that you're pretty sure a ball is in front of the robot somewhere, but you're not precisely sure where. In this case, it is usually sufficient to sweep the radar across, say, $-\frac{\pi}{3} < \theta < \frac{\pi}{3}$, and find the angle at which you observe the closest obstruction. This is all we need in order to do the demo where we place a ball in front of the robot and the robot turns to face it, which is usually enough to impress people.

When we perform such a "sweep", we usually rotate the servo by $\frac{\pi}{16}$ between each measurement, which gives more accuracy than we can actually match by making a turn. It would be cool if you could take measurements continuously as the servo rotates smoothly, but I wouldn't trust this to be accurate. Therefore, unfortunately, the radar has to sort of jerk from position to position.

You might consider also taking note of where you get "off the table" measurements. If you find an obstruction very close to an "off the table" measurement, the obstruction is probably a wall.

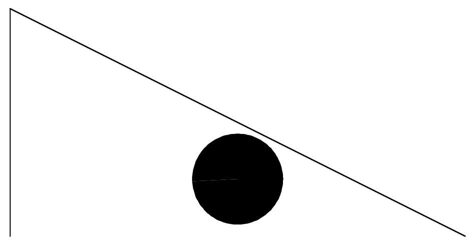
Finally, you could just drive at the obstruction as if it were a ball and try to capture it. If it doesn't budge then it's probably a wall; you could then turn around and look elsewhere.

■ Step 7. Accounting for Limitations

Or: On the Selection of ϕ

Obviously, the radar cannot detect an object that is farther than $\rho\cos(\phi)$ from the robot. Note that the distance sensor is accurate to about a meter, so you'll have to make sure your ϕ is steep enough so that ρ is less than this limitation.

More subtly, it is possible for the radar to miss an object that is "in range", as brilliantly depicted by the following diagram.



In general, this happens when the height of the object $\eta < a - d \tan(\phi)$, or when the distance to the object $d < (a - \eta) \cot(\phi)$. Taller and taller objects, or closer and closer objects, will be missed as ϕ becomes less steep; on the other hand, the maximum range decreases as ϕ becomes steeper. Because game objects are probably not as tall as your robot, there will probably always exist some minimum range below which you'll miss. You'll have to figure out where the happy medium is.

Also note that with lower values of ϕ (greater maximum range), the difference between obstructed and unobstructed readings become smaller and thus may be more difficult to distinguish.

■ Appendix 1. Advanced Operation

The following ideas are sound in principle, but they are difficult to implement and we've only verified their practicality in some specific cases. You will probably not want to attempt them because you should spend more time on other parts of your robot. This is really moving beyond a "radar" and into the realm of a primitive vision system. Nonetheless, this is interesting stuff to muse about.

The most accurate way to classify an obstruction is probably to determine its size. You can make some progress towards determining the breadth of an object by sweeping the radar and figuring out at what angular coordinates the object "begins" and "ends". A wall will be much wider than a ball. There are a lot of subtleties here related to how close you are to the object that you'll have to figure out.

Note that if you turn to face a wall and you sweep some measurements across it you will observe a constant y across the wall, reflecting the fact that the wall is flat. On the other hand, if you sweep across a ball, then you should detect some of the curvature.

You can figure out the height of an object by turning to face it, and then moving forwards until it is so close that the radar misses it. The last height measurement will be about the height of the object.

Last and least,one could amuse oneself by thinking about using an extremely fine-grained sweep as input to a hidden Markov model or neural network which could try to interpret the measurements. One should slap oneself if one were to actually seriously consider doing this.

■ Appendix 2. Final Tips and Future Thoughts

It's a good idea for debugging purposes to make the radar look at whatever obstruction you are interested in once you have detected it in a sweep.

If you mount a couple bright red LEDs to the sides of the distance sensor, it looks really badass as it looks around and locks. Think "Terminator".

It might be worth considering a simplified radar to detect objects in front of the robot, which is mounted on the bottom front of the robot. The radar would look straight out (instead of angled downwards) at what is on the table in front of the robot. The simplicity advantage here is that you get to think two-dimensionally, which is a lot easier. You have greater range, you practically eliminate the problem where you can't detect an object because it's too short, and it's easier to figure out whether you're looking at a wall. The disadvantages are that you lose any information about the height of the object, and (this is the real killer) you almost always want to put something more important on the front of your robot.