**TecAlliance**

**Prerequisite**

* Java 8
* IDE (IntelliJ) is recommended.
* Using GIT then clone source-code from

<https://github.com/nguyendaonq/tecalliance>

**Explanations**

Given two pipe-box including starting angle, end angle, starting distance, end distance

Input:

Box1(starting angle, starting distance, end angle, end distance)

Box2(starting angle, starting distance, end angle, end distance)

Note that a pipe-box can be represented by two coordinates, top left and bottom right. So mainly we are given following four coordinates.

Step 1: Calculate Point based on angle and distance input following formula:

This is an angle and distance from the origin (0,0)

**X=starting distance\*cos(starting angle)**

**Y=end distance\*sin(end angle)**

Then, we have Point pairs for each of pipe-box

PointTopLeft(X, Y);

PointBottomRight(X, Y);

**Box1(PointTopLeft, PointBottomRight)**

**Box2(PointTopLeft, PointBottomRight)**

Step 2: First, consider one way to define a two dimensional box is with:

A picture containing shape

Description automatically generated

I indicate the top left with a triangle and the bottom right with a circle

Two overlapping box might look like this:

A picture containing shape

Description automatically generated

To find the overlap we're looking for the place where the orange and the blue collide

Once we recognize this, it becomes obvious that overlap is the result of finding and multiplying these two darkened lines:

A picture containing shape

Description automatically generated

The length of each line is the minimum value of the two circle points, minus the maximum value of the two triangle points.

Here, I'm using a two-toned triangle (and circle) to show that the orange and the blue points are compared with each other. The small letter 'y' after the two-toned triangle indicates that the triangles are compared along the y axis, the small 'x' means they are compared along the x axis.

Application

Description automatically generated with low confidence

Finding those lines gives complete information about the overlapping areas.

Shape

Description automatically generated with medium confidence

Once we have this, finding the percentage of overlap is trivial:

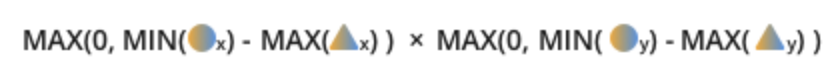
A picture containing box and whisker chart

Description automatically generated

But, if the orange rectangle does not overlap with the blue one then we're going to have a problem:

We get a -value for our overlapping area, that can't be right. Even worse, if a detection doesn't overlap with either dimension (neither on the x or y axis) then we will still get a positive number because both dimensions are negative. This is why you see the Max(0, ...) \* Max(0, ...) as part of the solution; it ensures that if any of the overlaps are negative we'll get a 0 back from our function.

The final formula in keeping with our symbology:



To summarize:

***intersecting\_area =***

***max(0, min(orange.circle.x, blue.circle.x) - max(orange.triangle.x, blue.triangle.x)) \* max(0, min(orange.circle.y, blue.circle.y) - max(orange.triangle.y, blue.triangle.y))***

***percent\_coverage = intersecting\_area / (orange\_area + blue\_area - intersecting\_area)***

Step 3: Please take a look codebase implementation for more details as above descriptions

Step 4: Common test use-case

1/ Two pipe boxes overlap

Text

Description automatically generated with medium confidence

2/ Two pipe boxes don’t overlap

Text

Description automatically generated

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Thank you for reading\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*