Problem 1A

Ryan St.Pierre rs520

First I will outline the procedure to calculate the intersection point. Then I will define more formally pseudo-code to get this intersection point.

Let $a = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$, $b = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$, $c = \begin{bmatrix} c_x \\ c_y \end{bmatrix}$, $d = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$ since $a,b,c,d \in \mathbb{R}^2$.

$$\left.\begin{aligned} P_x &= a_x + u(b_x - a_x) \\ P_x &= c_x + v(d_x - c_x) \end{aligned}\right\} \text{Need } u, v \text{ such that } P_x = P_x$$

u, v = scalars

① $a_x + u(b_x - a_x) = c_x + v(d_x - c_x)$

Likewise for y

② $a_y + u(b_y - a_y) = c_y + v(d_y - c_y)$

Thus, a system of 2 linear equations has been created.

① $u(b_x - a_x) - v(d_x - c_x) = c_x - a_x$

② $u(b_y - a_y) - v(d_y - c_y) = c_y - a_y$

$$\underbrace{\begin{bmatrix} b_x - a_x & c_x - d_x \\ b_y - a_y & c_y - d_y \end{bmatrix}}_{W} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} c_x - a_x \\ c_y - a_y \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\Delta} \underbrace{\begin{bmatrix} c_y - d_y & d_x - c_x \\ a_y - b_y & b_x - a_x \end{bmatrix}}_{W^{-1}} \begin{bmatrix} c_x - a_x \\ c_y - a_y \end{bmatrix}$$

where $\Delta = |W|$
$= (b_x - a_x)(c_y - d_y) - (c_x - d_x)(b_y - a_y)$

$$= \frac{1}{\Delta} \begin{bmatrix} (c_y - d_y)(c_x - a_x) + (d_x - c_x)(c_y - a_y) \\ (a_y - b_y)(c_x - a_x) + (b_x - a_x)(c_y - a_y) \end{bmatrix}$$

→

Problem 1A cont

2 cases:

$\Delta = 0$: If $\begin{vmatrix} b_x - a_x & c_x - d_x \\ b_y - a_y & c_y - d_y \end{vmatrix} = 0$ then $\frac{b_x - a_x}{b_y - a_y} = \frac{c_x - d_x}{c_y - d_y}$

Therefore if $\Delta = 0$ the 2 lines have the same slope

Same line (∞ intersections)

or different parallel lines (0 intersections)

$\Delta \neq 0 \rightarrow$ Intersection

To test if the lines are colinear or parallel we can check if 3 of the 4 points are colinear. Since the slopes are the same, 3 points colinear implies all 4 lie on the same line. a, b, & c are colinear if

$$\frac{b_x - a_x}{c_x - a_x} = \frac{b_y - a_y}{c_y - a_y} \Rightarrow (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x) = 0$$

We can now define the procedure as follows:

```
Line Intersection (a, b, c, d) {
  ax = a[0], ay = a[1], bx = b[0], by = b[1], cx = c[0], cy = c[1],
    dx = d[0], dy = d[1];
  det = (bx - ax)(cy - dy) - (cx - dx)(by - ay)
  if det == 0
    colinearCheck = (bx - ax)(cy - ay) - (by - ay)(cx - ax);
```

→

Problem 1A cont   # if det != 0

if colinearCheck = 0
    return "2 lines are colinear"
else
    return "no intersection"

else #

$$u = \frac{1}{det}\left[(c_y - d_y)(c_x - a_x) + (d_x - c_x)(c_y - a_y)\right]$$

$$v = \frac{1}{det}\left[(a_y - b_y)(c_x - a_x) + (b_x - a_x)(c_y - a_y)\right]$$

$$p_x = a_x + u(b_x - a_x)$$

$$p_y = a_y + u(b_y - a_y)$$

Return (P_x, P_y)

}

As problem 1B has shown it is also helpful to return u & v.

In following problems assume (u,v) returned as well as intersection point

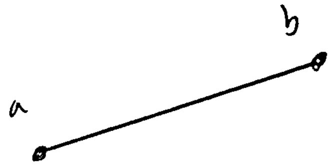
Problem 1B

ms70

Segment - Segment Collision

A line segment is defined as $x = a + u(b-a)$ for $u \in [0,1]$

Thus, assuming a, b, c, d are endpoints of their line segments we can check if u, v fall between 0 & 1.



$\underline{U=0}$

$x = a$

$\underline{U=1}$

$x = a + (b-a) = b$

```
SegmentIntersection(a,b,c,d) {
    (u,v) = LineSegment(a,b,c,d) *
    return (u ≥ 0) && (u ≤ 1)
        && (v ≥ 0) && (v ≤ 1);
}
```

intersection &&

* u, v calculated so let's assume they are returned in 1A.

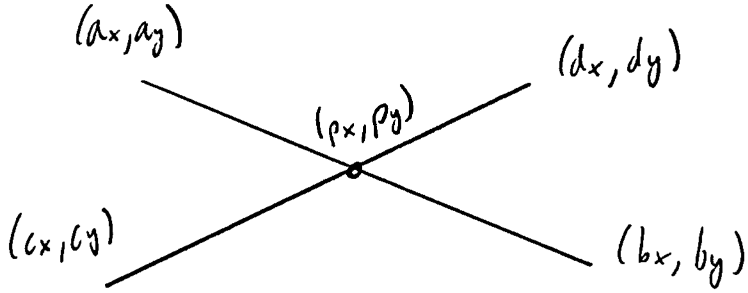
For line segments to intersect we must ensure there is an intersection need,

Line Intersection & $0 \le u \le 1$, $0 \le v \le 1$

Problem 1B - Another Approach.

Came up with this first. Then realized the (u,v) approach through the hint. Please feel free to ignore this page

The solution to Problem 1A returns a point of collison (p_x, p_y). This must be on the segment of both lines



☆ Min & max needed because no guarentee where a, b and c, d are w.r.t each other

$$\min(a_x, b_x) \le P_x \le \max^{☆}(a_x, b_x)$$
$$\min(c_x, d_x) \le P_x \le \max(c_x, d_x)$$
$$\min(a_y, b_y) \le P_y \le \max(a_y, b_y)$$
$$\min(c_y, d_y) \le P_y \le \max(c_y, d_y)$$

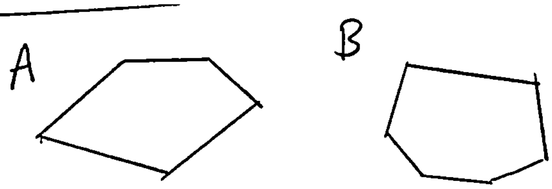
☆☆ if (no intersection) return false

```
SegmentIntersection (a, b, c, d) {
    (px, py) = LineIntersection (a, b, c, d)  ☆
    check1 = min(ax, bx) ≤ Px ≤ max(ax, bx)
    check2 = min(cx, dx) ≤ Px ≤ max(cx, dx)
    check3 = min(ay, by) ≤ Py ≤ max(ay, by)
    check4 = min(cy, dy) ≤ Py ≤ max(cy, dy)
    return check1 && check2 && check3 && check4
}
```

Note: I don't have to try values of U & V because my method from 1A returns the point of intersection not the parameters.

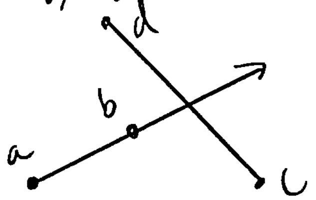
ns70

Problem 1C



A, B collide if any segment of A intersects any segment of B <u>or</u> A contains B, or B contains A.

To check whether A contains B or B contains A we will need to define a ray, segment collision method.



Ray: $x = a + u(b-a) \quad u \in [0, \infty)$

\\ a = endpoint of ray, b = point on ray, c = one endpoint of segment, d = other endpoint of segment

RaySegmentIntersection (a, b, c, d) {

(u, v, intersection) = LineIntersection (a, b, c, d)

return intersection ~~if~~ (u ≥ 0) && (v ≥ 0) && (v ≤ 1);

}

Now pseudo code:

1. segment detection
2. B contains A
3. A contains B

Problem 1C cont

rus26

Method from 1B actually takes in points not segments. However, this is an easy fix - just extract the points from the segments. I felt this pseudocode was more clear

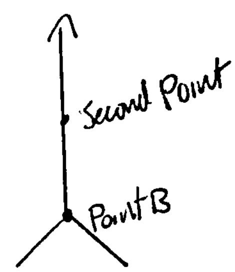
```
polygonCollision (A,B) {
    for segmentA in A
        for segmentB in B
            if (segmentCollision(segmentA, segmentB))
                return true
    if containmentCheck(A,B) || containmentCheck(B,A)
        return true

    return false;
}

// returns true if A contains B
containmentCheck (A,B)
    pointB = any point in B
    count = 0
    secondPoint = pointB + (0 1)
    for segmentA in A
        if (raySegmentCollision(pointB, secondPoint, segmentA(0), segmentA(1)))
            count++;
    // if count is even pointB is outside A
    if (count%2 == 0)
        return false

    return true;
}
```

Point B & second Point define a ray in the upward direction



return true if all points have been checked & are inside A.

Problem 1C cont

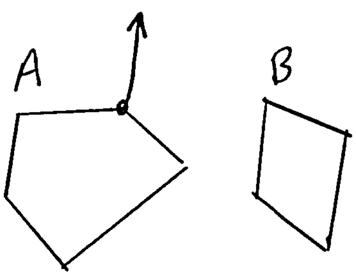
ns70

Why does only 1 point need to be checked for containment checks?

It A doesn't intersect B (by pair-wise segment collision check) the A is either within B (or B completely within A) or B & A are not within each other.

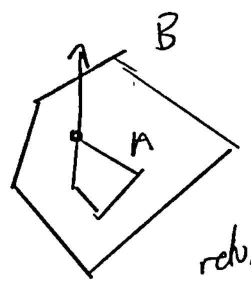
Therefore, to see if A is within B only one point has to be checked. If one point is within B all must be within B

Case 1



return false

Case 2



return true

Not a valid case if segment collision fails:

↳ some points of A within B some outside B is ~~not~~ valid

Problem 1D

Seg Collision Check

Each segment in A must check collision w/ each segment in B

$\underbrace{m+m+m+\cdots}_{n} = \underline{nm}$

Containment Check

A contains B: a point* in A must check if it is in B, which requires a collision check w/ each segment in B

$\underline{m \text{ checks}}$

*only one point has to be checked for reasons listed in Problem 1C

Likewise for B in A: $\underline{n \text{ checks}}$

Total checks = $nm + m + n$

$= O(nm)$

Problem 2A

If there are N vehicles there are $\binom{N}{2}$ or $\frac{N!}{2!(N-2)!}$ checks between cars

between cars

Car 1: $N-1$ checks

Car 2: $N-2$ checks (doesn't have to recheck with car 1)

$N-3$

$\vdots$

$$\sum_{i=1}^{N}(N-i)$$

car N: 0 checks

$$\frac{N!}{2!(N-2)!} = \frac{N(N-1)\cancel{(N-2)}\cdots}{2\cancel{(N-2)}\cancel{(N-3)}\cdots} = \frac{N(N-1)}{2}$$

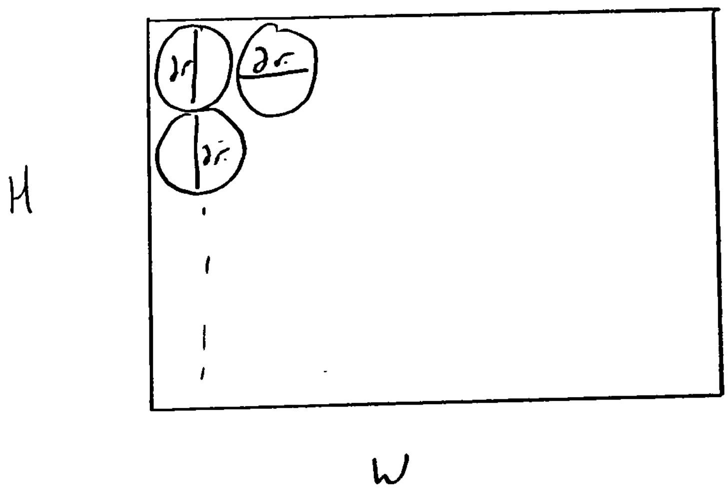
Each check between cars has up to 100 segment checks. Each segment of car 1 has to check collision with every segment in car 2. $10 \cdot 10 = 100$

Total segment-segment collision checks is at most $100\frac{n(n-1)}{2}$

$= 50n(n-1)$

ns 20

Problem 2B



$\# \text{rows} = \dfrac{H}{2r}$ ← assume division floors (chops)

$\begin{array}{l}\# \text{cars} \\ \text{per row}\end{array} = \dfrac{W}{2r}$

$\# \text{cars} = \# \text{rows} \cdot \# \text{cars}/\text{row} = \dfrac{H}{2r}\left(\dfrac{W}{2r}\right)$

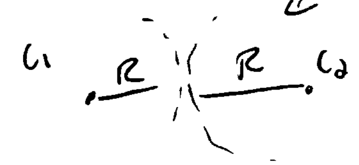
$$= \frac{HW}{4r^2}$$

Problem 2C

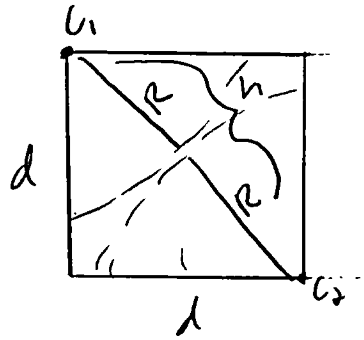
ns70

If two cars do not intersect their radii must be at least 2R apart.

Using R to upper bound Pv



Therefore, to ensure only one car is in a cell we need to ensure the largest distance between any two points in the cell < 2R. The largest distance between 2 points in the cell is along the diagonal.



If $h = 2R$ then exactly 2 cars can fit in a cell.
If $h < 2R$ then only 1 car can fit in a cell

$$\frac{2}{\sqrt{2}} = \frac{2\sqrt{2}}{\sqrt{2}\sqrt{2}} = \frac{2\sqrt{2}}{2} = \sqrt{2}$$

$$h^2 = d^2 + d^2 = 2d^2$$
$$h = \sqrt{2}\, d$$

Want $h < 2R$

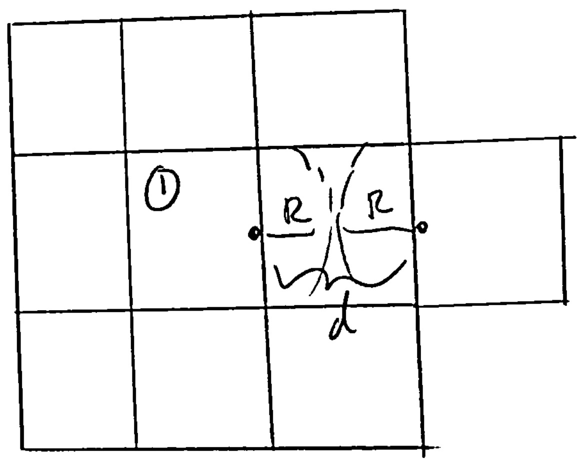
$$\sqrt{2}\, d < 2R$$
$$d < \frac{2R}{\sqrt{2}}$$

$$d < \sqrt{2}\, R$$

Problem 2D

as520

We must find the smallest d such that a car can not collide with a car in a grid outside its 8 closest neighbors.

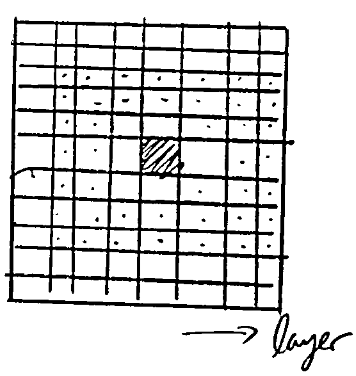


The shortest possible distance a car in ① can be to a car in a cell outside its eight neighbors (by origin) is d. This needs to be greater than $2R$ such that they don't collide.

$$\boxed{d > 2R}$$

In general, for a given d, at most how many cells would you have to check in step 2?

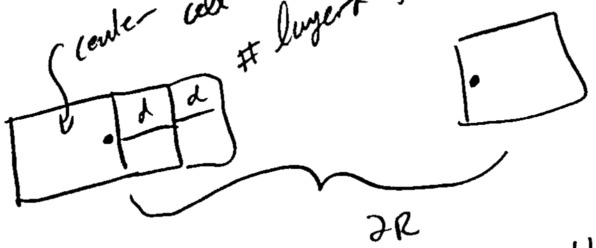
Let's first observe that at each layer there are $8i$ cells to check where i is the layer #



Layer 1 : 8
Layer 2 : 16
Layer 3 : 24
⋮

Problem 2D cont

Checks have to be made on all cells $< 2R$ from the center cell, as explained previously. Let's examine the horizontal distance - $\frac{2R}{d}$ layers need to be checked. The diagonal is more complicated. However, we can give a rough upper bound on the total # cells by saying we must include whole layers.



Total cell checks = checks to right + checks to left + checks above + checks below + middle check $= \left(2\left(\frac{2R}{d}\right)+1\right)^2$

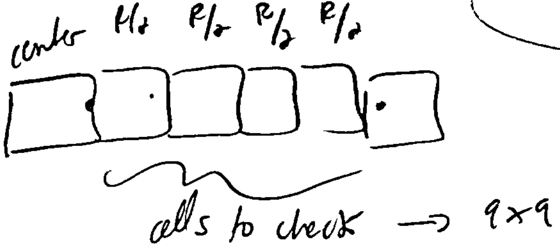
This should actually be a ceiling. If last cell does not perfectly fit we have to check it

d	$\text{ceil}\left(\frac{2R}{d}\right)$	checks
	1	3×3
$2R$	2	5×5
R	3	7×7
$\frac{2R}{3}$	4	9×9
$\frac{R}{2}$		

$\llcorner$ only right → symmetric to left, up & down

"upper bounded by" (roughly)

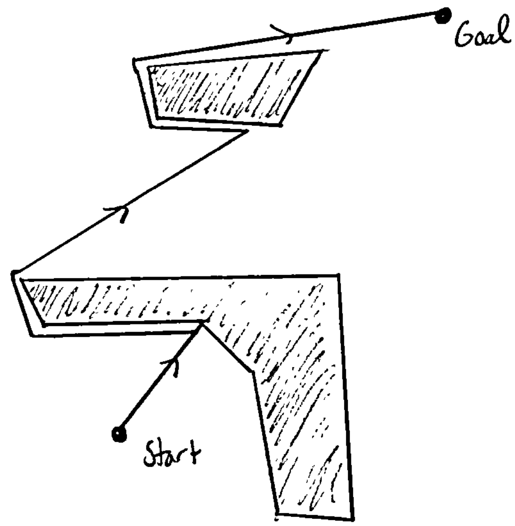
$$\#\text{ cell checks} \leq \left(2\,\text{ceil}\left(\frac{2R}{d}\right)+1\right)^2$$



Problem 3A

RS 20

Bug A's path is given to the right

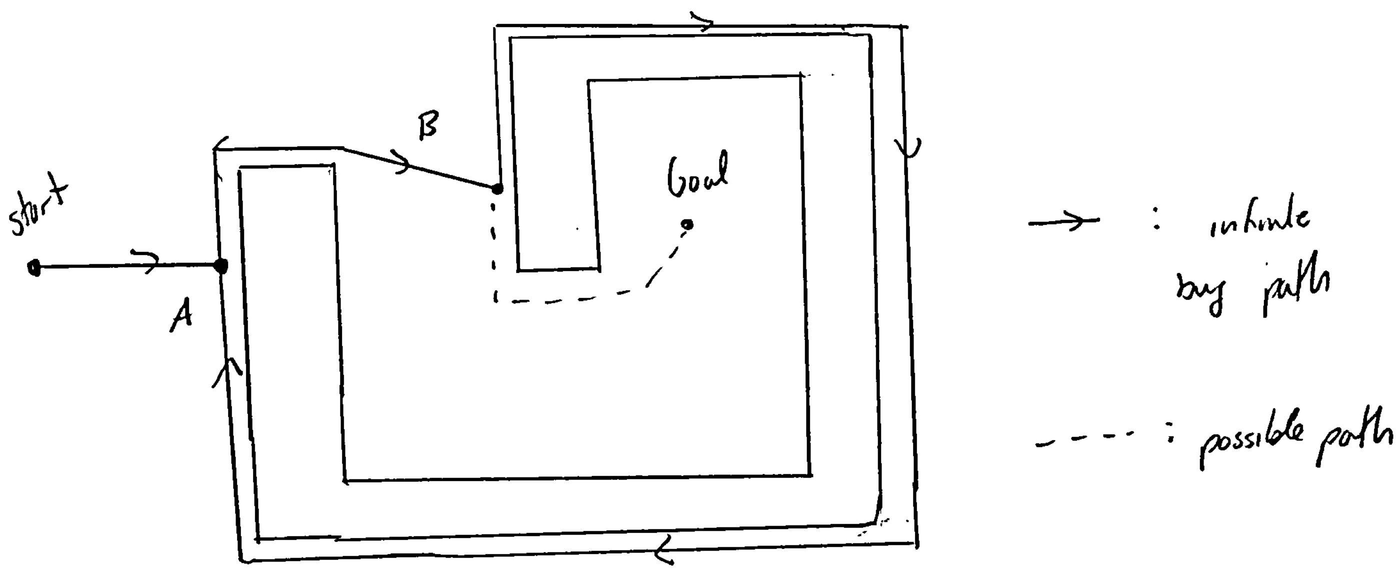


Problem 3B

rus70

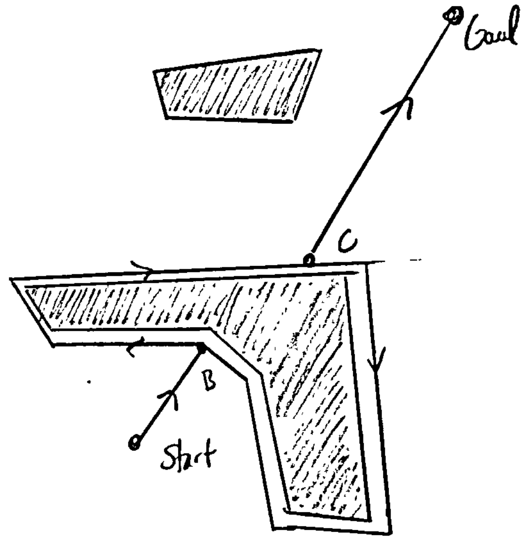
Bug A will not always find a path, even if one exists.

Loops can form



Bug gets stuck going from $A \to B \to A \to B$ .....

Problem 3C



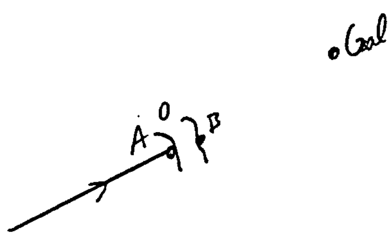
C = closest point

Path: start → B → C → B → C → Goal.

Problem 3D

ms70

Bug B will always find a path in a finite number of steps if one exists. Let's first establish that if Bug B hits obstacle O it must leave O at a point closer to the goal than it arrived.



If Bug B arrives at A there must be a point B on O closer to to goal as long as O has some finite thickness

Also, given obstacles are finite. Bug B will always be able to circumnavigate an obstacle in finite time & therefore always leave an obstacle.

Let Bug B encounter obstacles $O_1, O_2, O_3 \dots O_K$,* let $d_1, d_2, d_3 \dots d_K$ be the closest point on each obstacle to the goal. By design of the algorithm $d_1 > d_2 > d_3 > \dots > d_K$

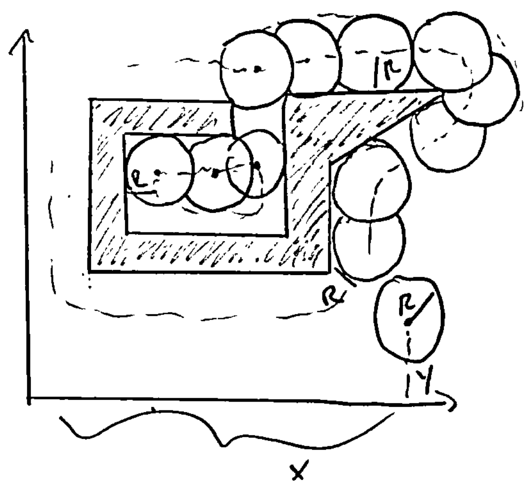
* in order $1 \to K$ before reaching the goal

Therefore, since
1. Bug B will always leave an obstacle
2. The distance to the goal decreases everytime an obstacle is encountered

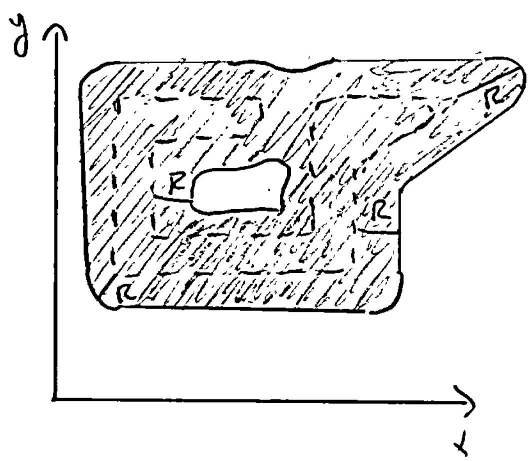
eventually Bug B will converge towards the goal in a finite number of steps.

Problem 4A

We can find the configuration space by tracing the robot around the edge of the obstacle.



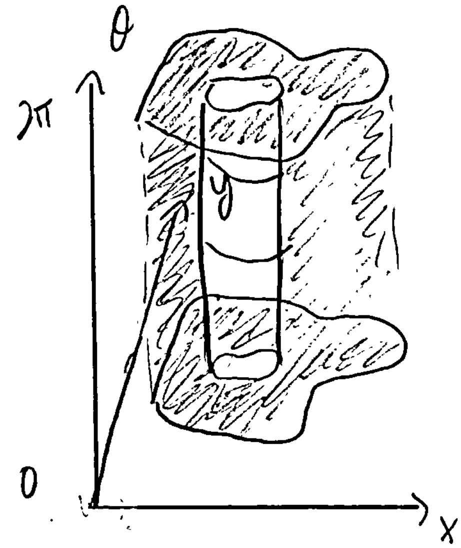
C-space



dotted - obstacle
shaded - obstacle in C-space
(not valid configuration of center)

Problem 4B

Since the robot is a circle, rotating about its center has no effect on the configuration space it can reach in x-y space. Therefore, the C-space looks the same in x-y space for all values of θ. If θ is the z-dimension the the c-space simply looks like the shape given in Problem 4A in the x-y plane stacked one on top of each other from $\theta = 0$ to 2π. In other words any slice of the c-space in the x-y plane will look exactly like that given in Problem 4A.

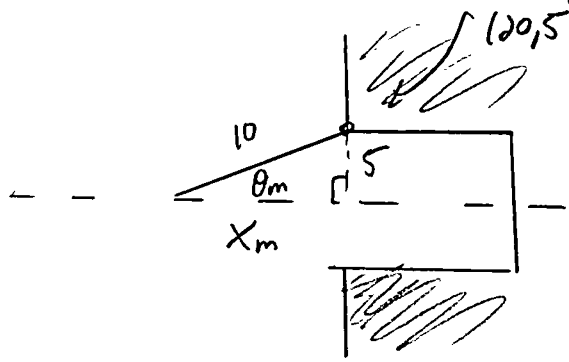


Problem 4c

as20

For $\theta = 0 \quad x \in [0, 25]$

For some θ the arm can just fit in the C-cutout, allowing x to range from 0 to 25. Let's find that θ_m



$$\sin\theta_m = \frac{5}{10}$$

$$\theta_m = \sin^{-1}\left(\tfrac{1}{2}\right) = 0.524$$

also $\theta_m = 2\pi - 0.524$ by symmetry

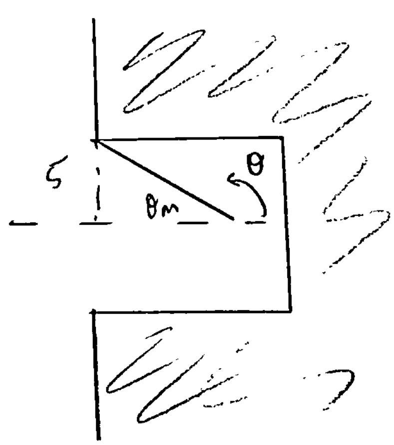
If $2\pi - \theta_m \le \theta \le 2\pi$ or $0 \le \theta \le \theta_m \quad x \in [0, 25]$

Also if $x \le 20 - x_m$ then θ has the full range of motion.

$$x_m = \sqrt{10^2 - 5^2} = \sqrt{75}$$

As $\theta = 90°$ or $270°$ $\left(\pi/2 \text{ \& } \frac{3\pi}{2}\right) \quad x \in [0, 20]$

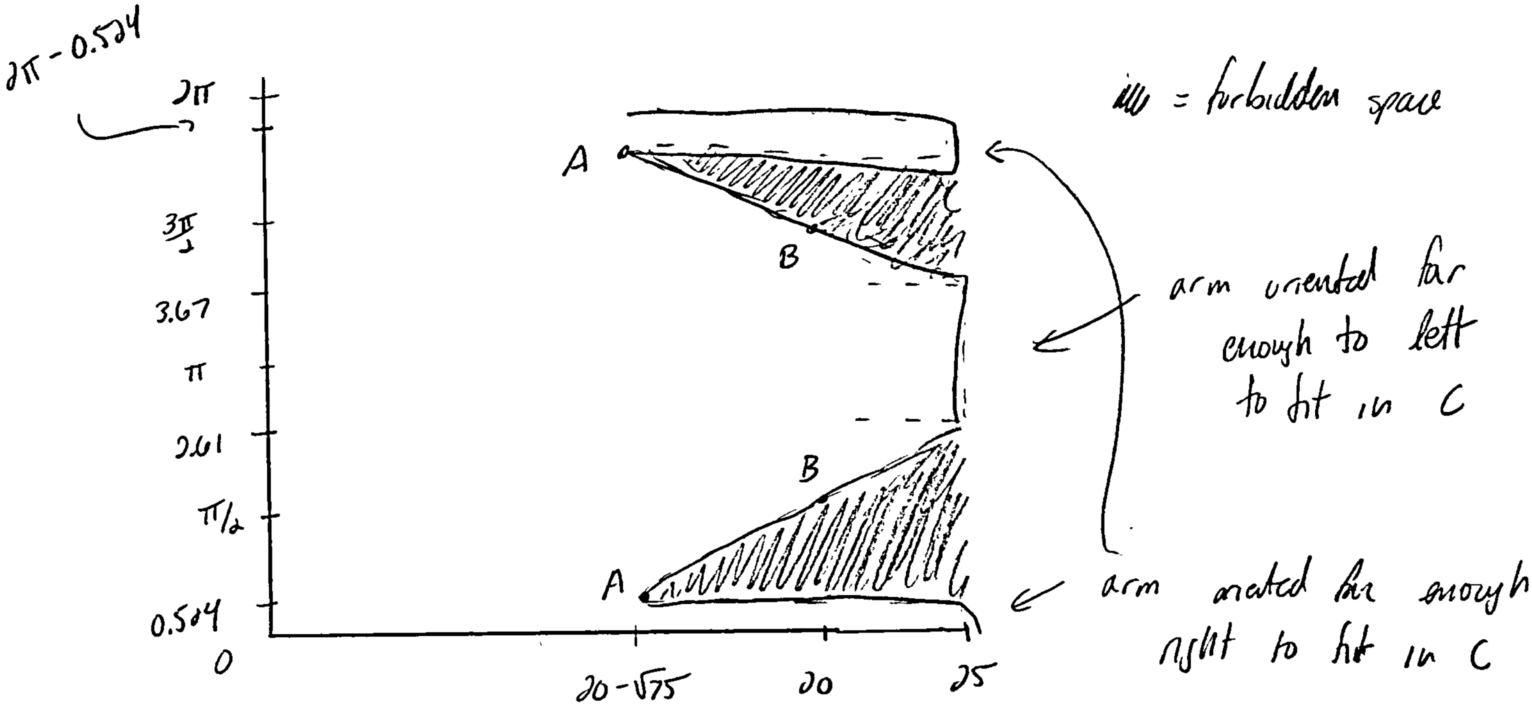
As θ ranges from $\pi/2 \to \frac{3\pi}{2}$ the joint can make it further in x



Also $\pi - \theta_m \le \theta \le \pi + \theta_m$ produces an orientation where $x \in [0, 25]$

$$2.61 \le \theta \le 3.67$$

Problem 4c cont



A: largest angle up & down where you can still fit in C

B: Arm straight up/down... can only make it to $x = 20$.