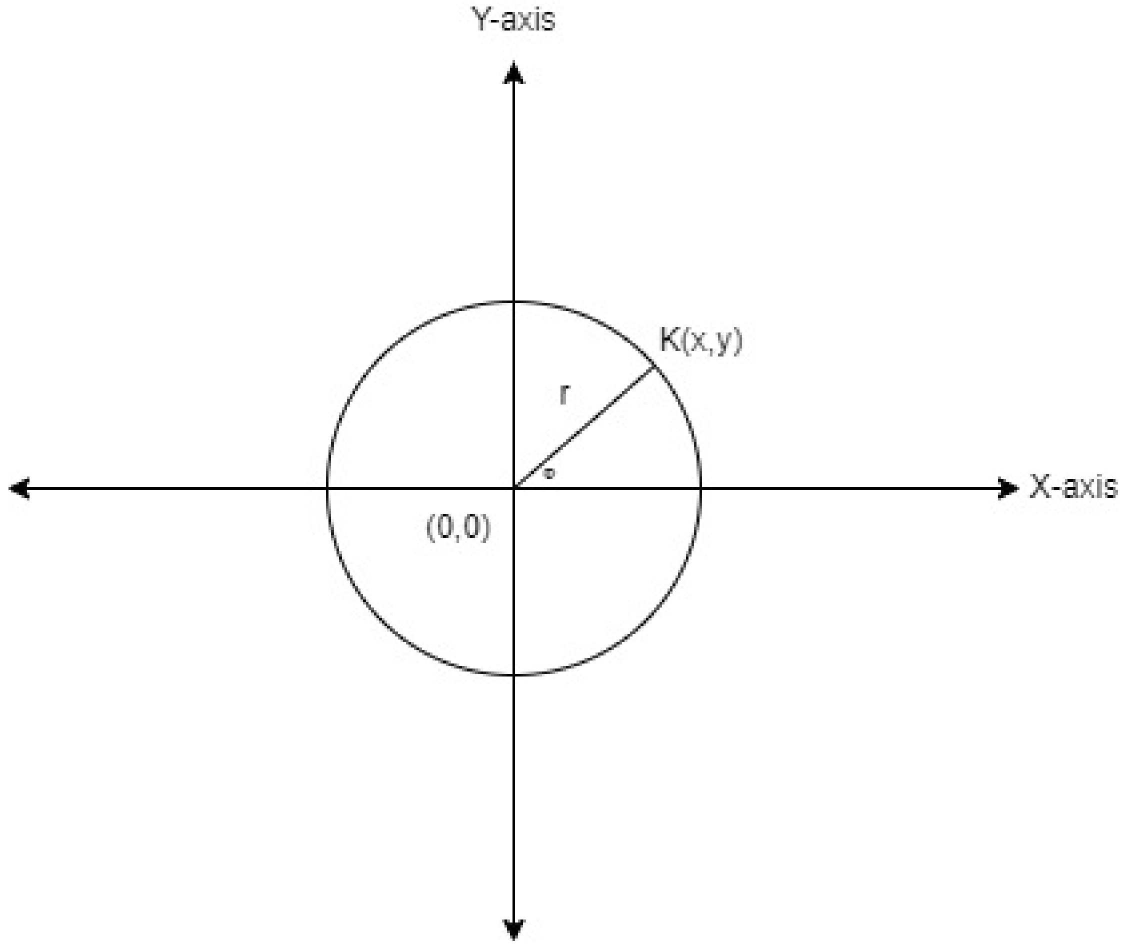
RBE550: Motion Planning

Project 1

Student 1: Krunal M. Bhatt

1. Let A be a unit disc centered at the origin in a workspace $W = \mathbb{R}^2$. Assume that A is represented by a single algebraic primitive H = $\{(x, y)|x^2 + y^2 \leq 1\}$. Show that if this primitive is rotated about the origin that the transformed primitive is unchanged. This can be shown by showing that any point within the transformed primitive H_0 must be within H, and vice versa.

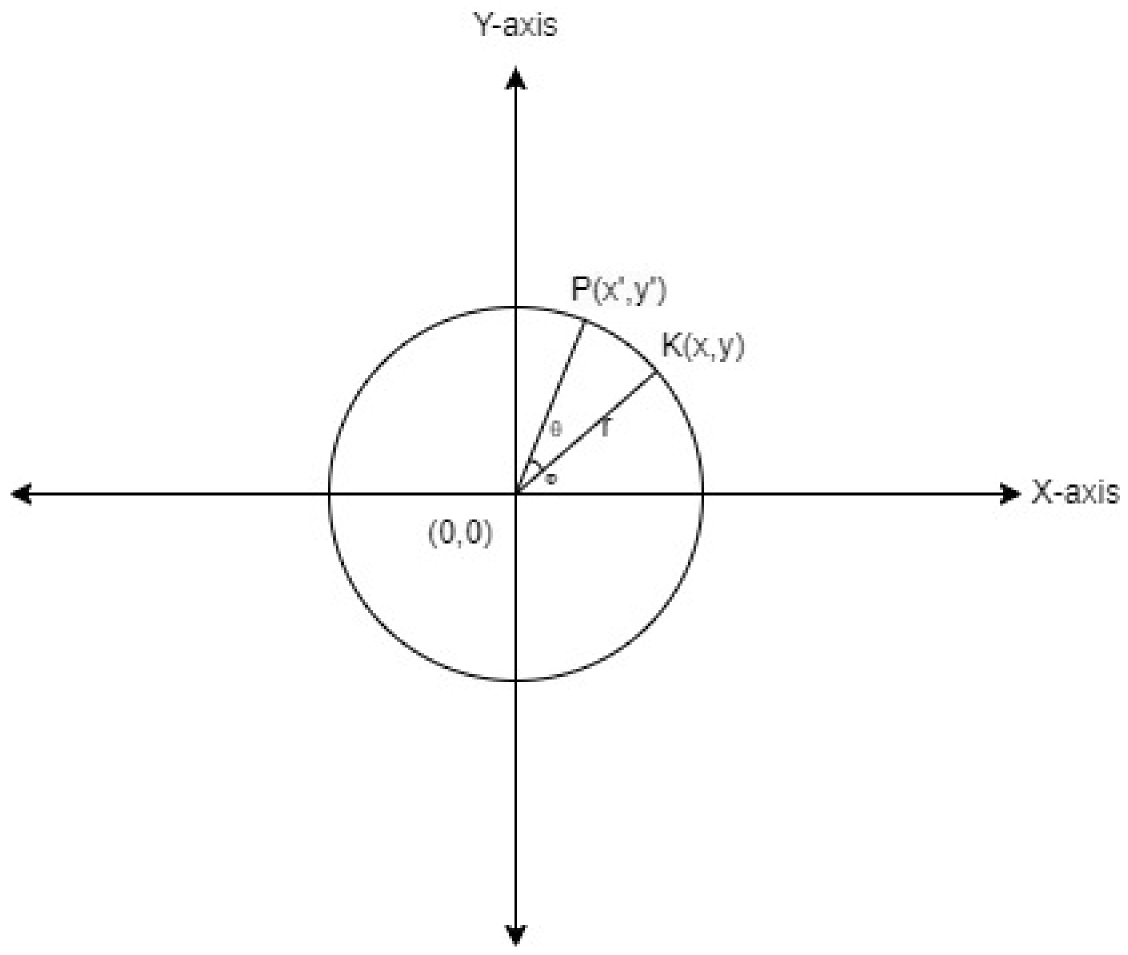
   Sol^n :
   Given that a unit disc A is centered at the origin in a workspace $W = \mathbb{R}^2$ as shown below. We assume a point in the Cartesian plane K(x,y) that is a part of the primitive H given in the question. The angle that the point makes with the X-axis and origin is ϕ.



   To show a rotation of a point around the origin, we will use the polar coordinates of point K. The polar coordinates can be represented as follows:

$$\begin{aligned} x &= r\cos(\phi) \\ y &= r\sin(\phi) \end{aligned} \tag{1}$$

   CASE 1: Counter-clockwise rotation
   After the rotation, it will have some different coordinates. The image shows a point P(x', y') with the rotated primitive about the origin. The coordinates would be:



$$\begin{aligned} x' &= r\cos(\phi+\theta) \\ y' &= r\sin(\phi+\theta) \end{aligned} \tag{2}$$

Therefore, using the sine and cosine addition formula we can express the coordinates as:

$$\begin{aligned} x' &= r(\cos(\phi)*\cos(\theta) - \sin(\phi)*\sin(\theta)) \\ y' &= r(\sin(\phi)*\cos(\theta) + \cos(\phi)*\sin(\theta)) \end{aligned} \tag{3}$$

From equation 1, we can use the values of x and y and substitute them in equation 3 as follows:

$$\begin{aligned} x &= r\cos\phi \\ y &= r\sin\phi \end{aligned}$$

$$\begin{aligned} \implies x' &= x\cos(\theta) - y\sin(\theta) \\ y' &= y\cos(\theta) + x\sin(\theta) \end{aligned} \tag{4}$$

Given for point (x,y), the primitive H is given as $x^2+y^2 \leq 1$. To prove that this primitive is rotated by θ around the origin, the transformed primitive should remain unchanged, i.e.:

$$H_0 = \{(x',y')|x^2+y^2 \leq 1\}$$

To prove that after rotation around the origin, H_0 remains unchanged, we show:

$$(x')^2 + (y')^2 \leq 1$$

$$\implies (x\cos(\theta) - y\sin(\theta))^2 + (y\cos(\theta) + x\sin(\theta))^2 \leq 1$$

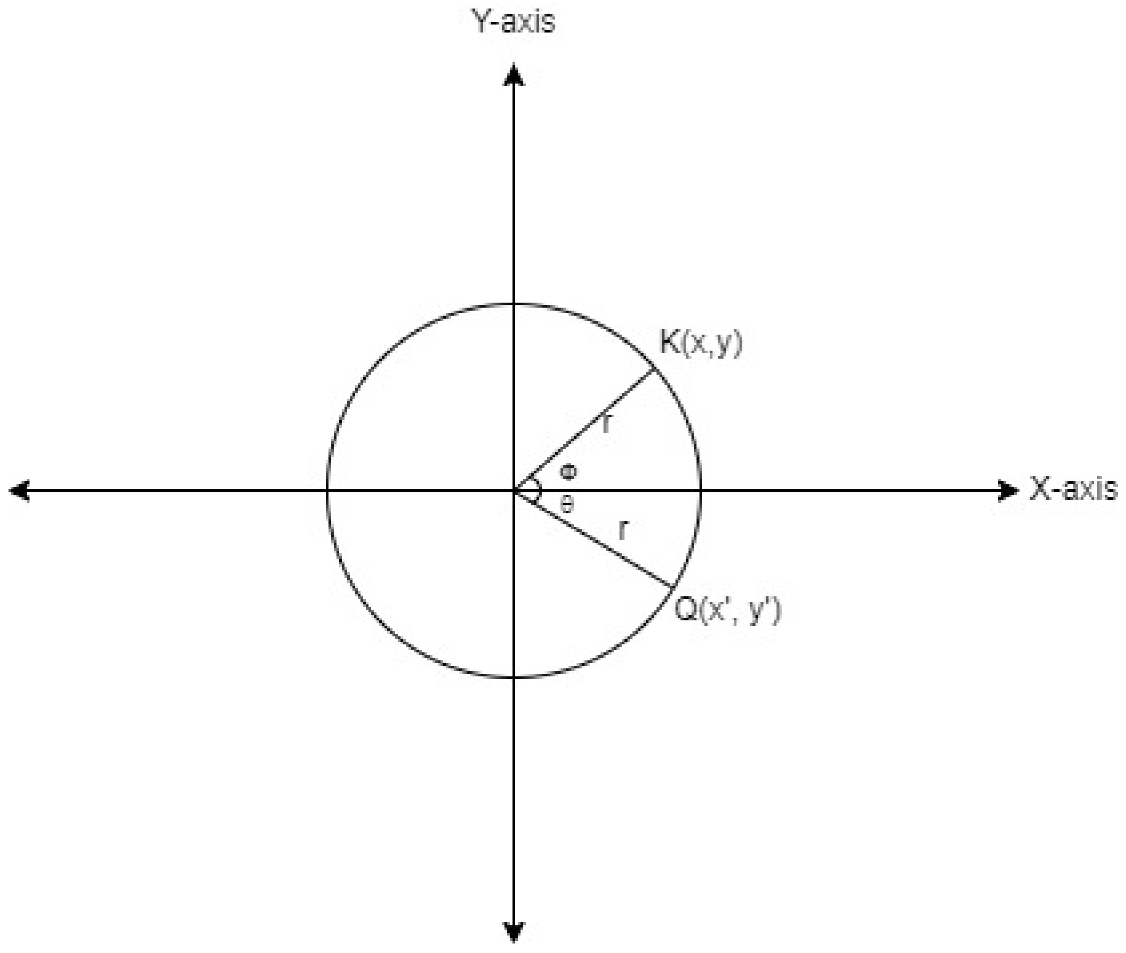
$$\implies (x^2cos^2(\theta) - 2xy\cos(\theta)\sin(\theta) + y^2sin^2(\theta)) + (y^2cos^2(\theta) + 2xy\cos(\theta)\sin(\theta) + x^2\sin^2(\theta)) \leq 1$$

$$\implies x^2\cos^2(\theta) + y^2\sin^2(\theta) + y^2cos^2(\theta) + x^2\sin^2(\theta) \leq 1$$

$$\implies x^2(\cos^2(\theta) + \sin^2(\theta)) + y^2(\sin^2(\theta) + \cos^2(\theta)) \leq 1$$
$$\therefore x^2 + y^2 \leq 1 \tag{5}$$

Equation 5 above shows that when a primitive H_0 is rotated around the origin, the transformed primitive remains unchanged.

CASE 2: Clockwise rotation
So, we proved that the primitive H_0 rotated about the origin remains unchanged. Previously, we showed the counter-clockwise rotation (where θ is positive). In this case, a new point Q(x', y') is taken with clockwise rotation and then it is checked if the primitive remains unchanged after rotation or not. The figure shows the point and the rotation around the origin.



Equation 2 can be re-written as,

$$\begin{aligned} x' &= r\cos(\phi - \theta) \\ y' &= r\sin(\phi - \theta) \end{aligned} \tag{6}$$

We can use the trigonometric identity and write x' and y' as

$$\begin{aligned} x' &= r(\cos(\phi) * \cos(\theta) + \sin(\phi) * \sin(\theta)) \\ y' &= r(\sin(\phi) * \cos(\theta) - \cos(\phi) * \sin(\theta)) \end{aligned}$$

Which gives us (form equation 1),

$$\begin{aligned} x' &= x\cos(\theta) + y\sin(\theta) \\ y' &= y\cos(\theta) - x\sin(\theta) \end{aligned} \tag{7}$$

Now, we check the inequality after the rotation, i.e.:

$$(x')^2 + (y')^2 \leq 1 \implies (x^2\cos^2(\theta) + 2xy\cos(\theta)\sin(\theta) +$$
$$y^2\sin^2(\theta) + y^2\cos^2(\theta) - 2xy\cos(\theta)\sin(\theta) + x^2\sin^2(\theta)) \leq 1$$

Equation above gives the same result

$$x^2 + y^2 \leq 1$$

Now, through CASE 1 and CASE 2, we know that after rotating the primitive, there is no change in the transformed primitive.

2. You are given the endpoints to two line segments, A_1B_1 and A_2B_2, in a 2D workspace. The line segments include their endpoints. Provide an algorithm in pseudocode to compute the intersection points of these two line segments, if one exists. Be careful and consider all corner cases. Your algorithm must provide the correct output with every input. Besides providing the pseudocode you must also clearly explain it. Hint: There are many ways to represent line segments. Choosing wisely will allow for a shorter and more efficient implementation.

   Sol^n :
   Given the endpoints of two line segments A_1B_1 and A_2B_2 we can define cases and accordingly we will define the intersection point for the same.

   (a) Line segments are parallel
   (b) Line segments intersect at 1 point
   (c) Line segments are not parallel and do not intersect

   For these cases, we will be using simple co-ordinate geometry concepts like slope using two pairs of coordinates, two-point form, slope intercept form. We will be defining them in the pseudo-code below:

Algorithm 1 Line Segment Intersection

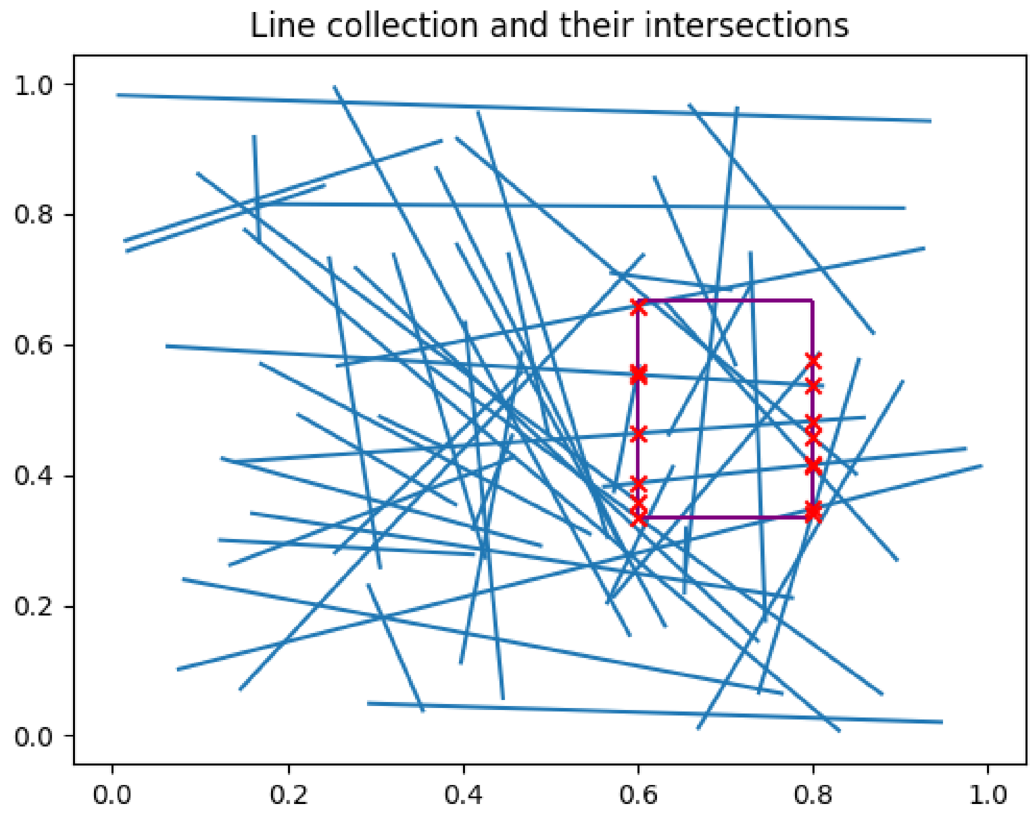
```
function SEGMENT_PAIR_INTERSECTION($A_1, B_1, A_2, B_2$)
    $m_1 = \frac{B_1y - A_1y}{B_1x - A_1x}$          ▷ Calculate the slope of line segment using formula (y2 -y1)/(x2 - x1)
    $m_2 = \frac{B_2y - A_2y}{B_2x - A_2x}$

    if $m_1 == m_2$ then return Segments are parallel!
    $y\_intercept_1$ = $A_1$ y - $m_1$ * $A_1$x
    $y\_intercept_2$ = $A_2$ y - $m_2$ * $A_2$x          ▷ Calculate y intercepts if segment not parallel using y-mx
    intersect_x = $(y\_intercept_2 - y\_intercept_1)/(m_1 - m_2)$          ▷ X coordinate of intersection point
    intersect_y = $m_1$ * intersect_x + $y\_intercept_1$          ▷ Y coordinate of intersection point

    if (intersect_x < min($A_1$x, $B_1$x) || intersect_x > max($A_1$x, $B_1$x) || intersect_x < min($A_2$x, $B_2$x) ||
——  intersect_x > max($A_2$x, $B_2$x) || intersect_y < min($A_1$x, $B_1$x) || intersect_y > max($A_1$x, $B_1$x) ||
——  intersect_y < min($A_2$x, $B_2$x) || intersect_y > max($A_2$x, $B_2$x)) then
——  return "No Intersection"
                                        ▷ We defined not parallel and not intersecting cases
    return $intersect\_x$, $intersect\_y$
```

The above algorithm works for all the cases. It considers the corner cases and also the three cases which are mentioned above. I will explain the algorithm in brief. Firstly, we defined a function named *segment_pair_intersection* which takes the endpoints of line-segment as input parameters. Then in the next step, we calculated the slopes m_1 and m_2.
—— If the slope values are equal, the line segments are parallel. Furthermore, if slopes are not equal, we calculate the Y-intercepts for both the lines. Then, we calculate the X-coordinate of the intersection point with the help of Y-intercepts calculated in the previous step. With the help of line equation y = mx + c we calculate the Y-coordinate of the intersection point.
—— The equations in some cases might result in in points that are out of the segments, meaning that the segments are neither parallel nor intersecting. For such a case, we use if conditions and find if the intersection point lies on the segment or outside of it.

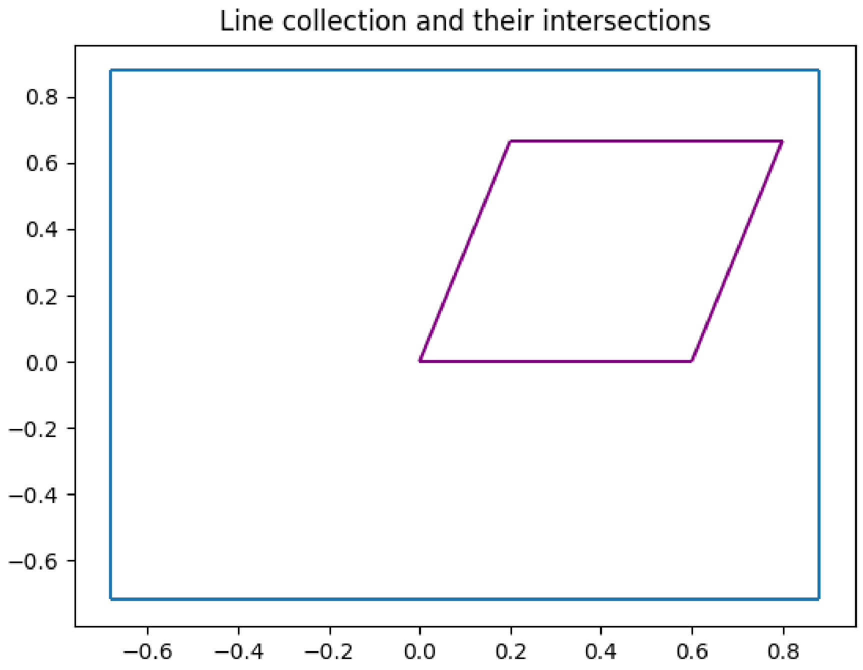
```
##File made by Krunal Bhatt
## This is the interseciton.py file used in the Quesiton 3 part (A)
## The code below is the implementation of the pseudocode algorithm shown in
## the previous question 2.

import time

def line_eq(p1,p2):
    A = p2[1] - p1[1]
    B = p1[0] - p2[0]
    C = A * p1[0] + B*p1[1]
    return A,B,C


def segment_pair_intersection(line1, line2):
    intersect = True
    x1,y1 = line1[0]
    x2,y2 = line1[1]
    x3,y3 = line2[0]
    x4,y4 = line2[1]

    A1,B1,C1= line_eq([x1,y1],[x2,y2])
    A2,B2,C2 = line_eq([x3,y3],[x4,y4])

    det = A1*B2 - B1*A2
    if det==0:
        intersect = False

    if intersect:
        x = round((C1*B2 - C2*B1)/det,12)
        y = round((A1*C2 - C1*A2)/det,12)
        # intersection point between line1 and line2
        if (
            min(x1,x2) <= x <= max(x1,x2) and
            min(y1,y2) <= y <= max(y1,y2) and
            min(x3,x4) <= x <= max(x3,x4) and
            min(y3,y4) <= y <= max(y3,y4)
        ) :
            return (x,y)
        else:
            return None
    else:
        return None


def efficient_intersections(L1, L2):
    #This is the Bonus part of the assignment

    ### YOUR CODE STARTS HERE ###
    return ([], [])
    ### YOUR CODE ENDS HERE ###

def all_pairs_intersections(L1, L2):
    x = []
    y = []
    for l1 in L1:
        for l2 in L2:
          point =  segment_pair_intersection(l1, l2)
          if point:
            x.append(point[0])
            y.append(point[1])
    return(x,y)
```

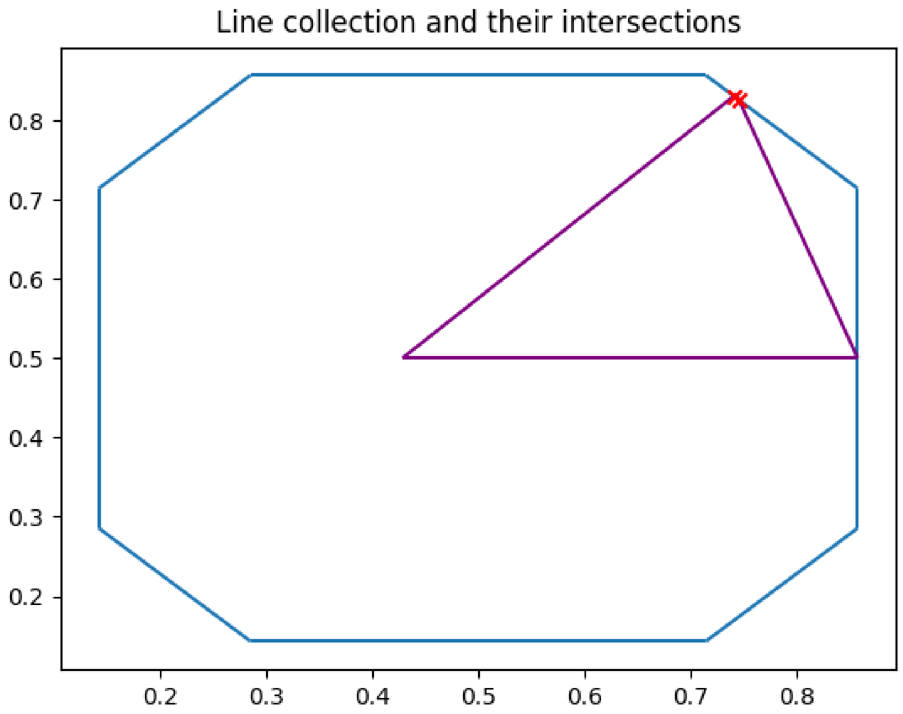
```
## This is the main.py file.

import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection
import pickle
from pathlib import Path
import time

from intersections import all_pairs_intersections, efficient_intersections


def read_pickle(filename):
    # Can sometimes read pickle3 from python2 by calling twice
    with Path.open(Path(filename), 'rb') as f:
        try:
            return pickle.load(f)
        except UnicodeDecodeError:
            return pickle.load(f, encoding='latin1')

def write_pickle(data, filename):  # NOTE - cannot pickle lambda or nested functions
    with Path.open(Path(filename), 'wb') as f:
        pickle.dump(data, f)

def read_segments(type, index):
    if type =="random":
        segments = read_pickle(f"./data/Random/random_{index}.pkl")
    elif type =="convex":
        segments = read_pickle(f"./data/Convex/convex{index}.pkl")
    elif type =="nonconvex":
        segments = read_pickle(f"./data/NonConvex/non_convex{index}.pkl")
    else:
        ValueError("No such type exist")

    return segments

def visualize_lines_and_intersections(L1,L2, points):
    fig, ax = plt.subplots()
    ax.add_collection(LineCollection(L1,  linestyle='solid', zorder=0))
    ax.add_collection(LineCollection(L2,  linestyle='solid', color="purple",
zorder=1))
    ax.scatter(points[0], points[1], color="red", marker="x", zorder=2)
    ax.set_title('Line collection and their intersections')
    plt.show()

#Create random line segments
def main():
    L1 = read_segments("random", 50)
    L2 = read_segments("convex", 1)
    points =  all_pairs_intersections(L1, L2)
    visualize_lines_and_intersections( L1, L2, points)
    L1 = read_segments("nonconvex", 11)
    L2 = read_segments("convex", 3)
    points =  all_pairs_intersections(L1, L2)
    visualize_lines_and_intersections( L1, L2, points)

    L1 = read_segments("nonconvex", 2)
    L2 = read_segments("convex", 10)
    points =  all_pairs_intersections(L1, L2)
    visualize_lines_and_intersections( L1, L2, points)

    times = {}
    times_efficient = {}
    for N in range(0, 500, 10):
        L1= read_segments("random", N)
        start = time.time()
        points1 = all_pairs_intersections(L1, L1)
        end = time.time()
        times[N]  = end-start

        start = time.time()
        points2 =  efficient_intersections(L1, L1)
        end = time.time()
        times_efficient[N]  = end-start
        if len(points1[0])!=len(points2[0]):
            print(f"Efficient intersections for {N} is not implemented correctly!")

    plt.plot(times.keys(), times.values())
    plt.plot(times_efficient.keys(), times_efficient.values())
    plt.legend(labels = ["All pairs Algorithm", "Efficient Algorithm"])
    plt.show()

if __name__== "__main__":
    main()
```

- This is the output generated by the main.py function after writing the segment_pair_intersection() is written.

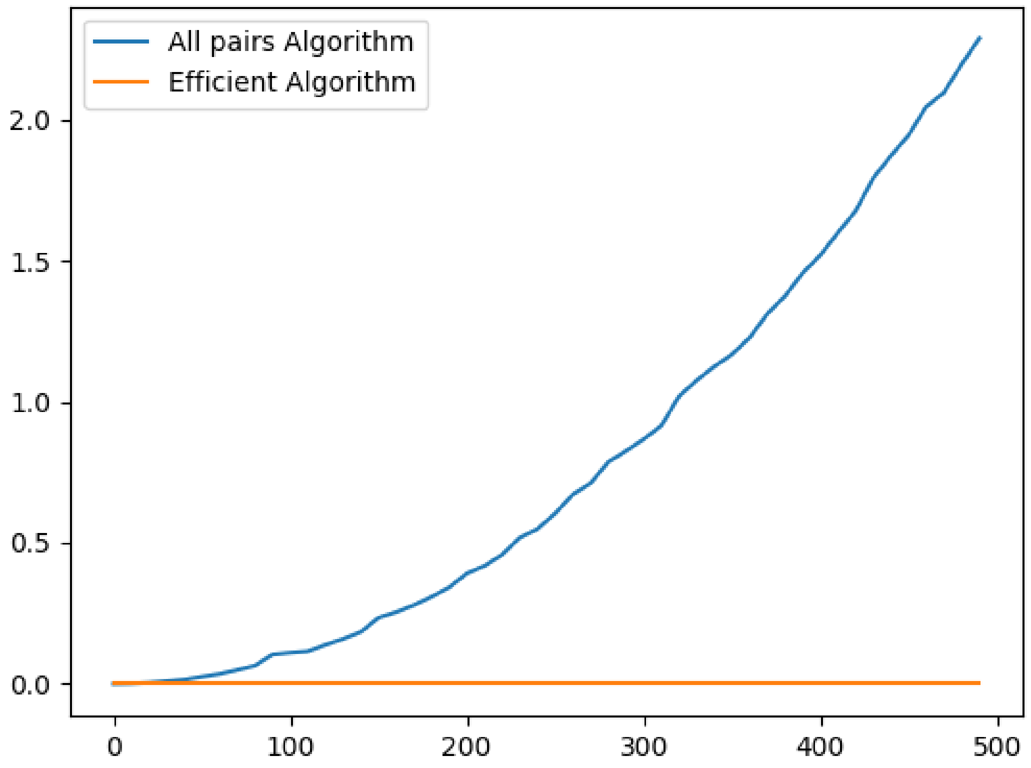


KRUNAL BHATT





- After this the main.py generates an efficiency graph compared to the original algorithm which is shown below:



Time complexity

Let's examine each section of the above code separately in order to determine its temporal complexity:

- Since the line_eq function only does a given number of arithmetic operations, its constant time complexity is O(1).
- Additionally, the segment_pair_intersection function has an O(1) constant time complexity. It calculates the intersection location between two line segments using a given number of operations.
- The empty list that the efficient_intersections function currently returns has an O(1) constant time complexity.
- The time complexity of the all_pairs_intersections function, where "n" is the total number of line segments, is O(n^2). It calls the segment_pair_intersection function, which has a constant time complexity, after iterating through each pair of line segments. As a result, O(n^2) is the general time complexity.