RBE550: Motion Planning

Project 1 Fundamentals

Student 1: Udaygirish, Maradana
Student 2: -

1. Proof to show that a single algebraic primitive centered at origin when rotated about the origin, then the transformed primitive is unchanged.

   Algebraic primitive:
$$H = \left\{(x, y)\,|x^2 + y^2 \leq 1\right\}$$

   Let us take (x1, y1) as initial points on the primitive. This is a equation for region bounded by a Unit circle centered at origin.

   If the primitive is rotated about the origin by angle θ.

   By multiplying with the rotation matrix as mentioned below
$$\begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} xcos(\theta) - ysin(\theta) \\ xsin(\theta) + ycos(\theta) \end{bmatrix}$$

   We get the below new coordinate which is on the transformed primitive. Here x1 , y1 are new coordinates in new primitives.
$$\begin{bmatrix} x1 \\ y1 \end{bmatrix} = \begin{bmatrix} xcos(\theta) - ysin(\theta) \\ xsin(\theta) + ycos(\theta) \end{bmatrix}$$

   To construct new primitive placing the transformed coordinate and expanding:
$$x^2cos^2(\theta) + y^2sin^2(\theta) - 2xycos(\theta)sin(\theta) + x^2sin^2(\theta) + y^2cos^2(\theta) + 2xycos(\theta)sin(\theta) \leq 1$$

   Which will leave us with:
$$x^2(cos^2(\theta) + sin^2(\theta)) + y^2(cos^2(\theta) + sin^2(\theta)) \leq 1$$
$$\because sin^2(\theta) + cos^2(\theta) = 1$$

   We get new primitive
$$x1^2 + y1^2 \leq 1$$

   So this proves that if the rotational transformation is centered at the origin of the primitive, the primitive remains unchanged. As the transformation itself becomes a trigonometric identity.

2. Pseudocode to compute the intersection points of these two line segments, if one exists:

Algorithm 1 Point of Intersection Algorithm

```
    Input Two line segments where each one of them contain the start and end points of the line segment
    Output Intersection point if the lines are intersecting.

  function POINT OF INTERSECTION ALGORITHM($L1$, $L2$)

      $A1, B1 = L1$
      $A2, B2 = L2$
      interpoint = (x,y)                          ▷ Assume $A1$ where $A1.x$ = x, $A1.y$ = y

      $a1 = A1.y - B1.y$                          ▷ First line $\overline{\text{A1B1}}$ definition $a1x + b1y = c1$
      $b1 = B1.x - A1.x$
      $c1 = A1.x * a1 + A1.y * b1$

      $a2 = A2.y - B2.y$                          ▷ Second line $\overline{\text{A2B2}}$ definition $a2x + b2y = c2$
      $b2 = B2.x - A2.x$
      $c2 = A2.x * a2 + A2.y * b2$
      $DetA = a1b2 - a2b1$
      if $DetA! = 0$ then
          intersect = True
      else
          intersect = False
      if intersect = True then
          $interpoint = ((b2c1 - b1c2)/DetA, (c2a1 - c1a2)/DetA)$
          $listx1 = [A1.x, B1.x]$
          $listx2 = [A2.x, B2.x]$
          $listy1 = [A1.y, B1.y]$
          $listy2 = [A2.y, B2.y]$

          $ep = 5e^{-10}$     ▷ This is to avoid losing points due to high precision computation which can be missed
  when the detA value is close to zero

          $condx1 = min(listx1) - ep \leq inter_x \leq max(listx1) + ep$
          $condx2 = min(listx2) - ep \leq inter_x \leq max(listx2) + ep$
          $condy1 = min(listy1) - ep \leq inter_y \leq max(listy1) + ep$
          $condy2 = min(listy2) - ep \leq inter_y \leq max(listy2) + ep$

          if $condx1 \wedge condx2 \wedge condy1 \wedge condy2$ then return $interpoint$
          else
              return None
          return None
      else
        return None                               ▷ Print - Lines are parallel , non intersecting
```

The time complexity of the algorithm is $O(1)$. But for the pairwise intersection of N segments the time complexity becomes $O(N^2)$.

Math behind the above algorithm

Let us consider $A1 = (x_1, y_1), B1 = (x_2, y_2), A2 = (x_3, y_3), B2 = (x_4, y_4)$
Then we get two line segments
Let us try simplifying the form of one line segment equation to the form $ax + by = c$

$$L(AB) == (y - y_1) = ((y_2 - y_1)/(x_2 - x_1)) * (x - x_1)$$

Simplifying we get , $x(y_1 - y_2) + y(x_2 - x_1) = x_1(y_1 - y_2) + y_1(x_2 - x_1)$
Here comparing with $ax + by = c$, we get

$$a = (y_1 - y_2), b = (x_2 - x_1), c = x_1 * a + y_1 * b$$

Based on this our two line segments are
First line:

$$a_1x + b_1y = c_1$$

Where, $a_1 = A_1(1) - B_1(1), b_1 = B_1(0) - A_1(0), c_1 = A_1(0)a_1 + A_1(1)b_1$

Second line:

$$a_2x + b_2y = c_2$$

Where, $a_2 = A_2(1) - B_2(1), b_2 = B_2(0) - A_2(0), c_2 = A_2(0)a_2 + A_2(1)b_2$
Now solving the linear equations to represent in the matrix form, we get

$$\begin{bmatrix} a1 & b1 \\ a2 & b2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

$$AX = B$$

$$X = A^{-1}B$$

From Matrix algebra we get

$$X = (1/\det(A))\lfloor A \rfloor\ B$$

Where $\det(A) = a_1b_2 - a_2b_1$

$$\begin{bmatrix} x \\ y \end{bmatrix} = (1/\det(A)) \begin{bmatrix} b2 & -b1 \\ -a2 & a1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = (1/(a_1b_2 - a_2b_1)) \begin{bmatrix} b_2c_1 - b_1c_2 \\ c_2a_1 - c_1a_2 \end{bmatrix}$$

So our intersection point would be

$$(x, y) = (\frac{b_2c_1 - b_1c_2}{a_1b_2 - a_2b_1}, \frac{c_2a_1 - c_1a_2}{a_1b_2 - a_2b_1})$$

3. Now imagine that you are given a set of line segments $S = S_1, S_2, S_3$ and you want to compute all the intersection points of all of these segments.

   a. Segment pair intersection- Question

Output Plots after implementing Pseudo code in Python

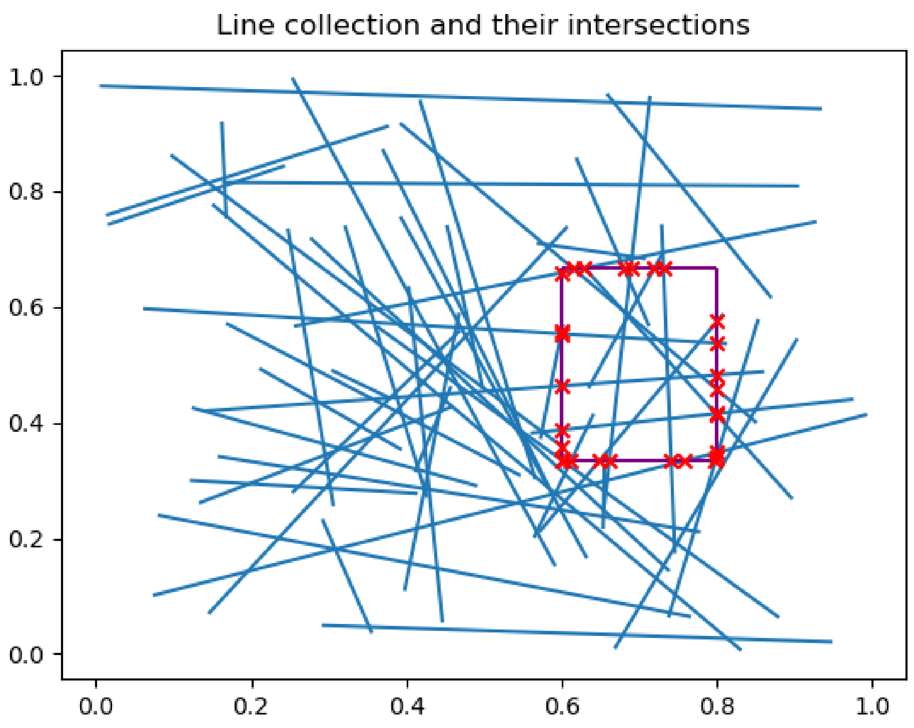

Figure 1: Line Intersection Plot for Test case 1 - Rectangle and Set of Lines

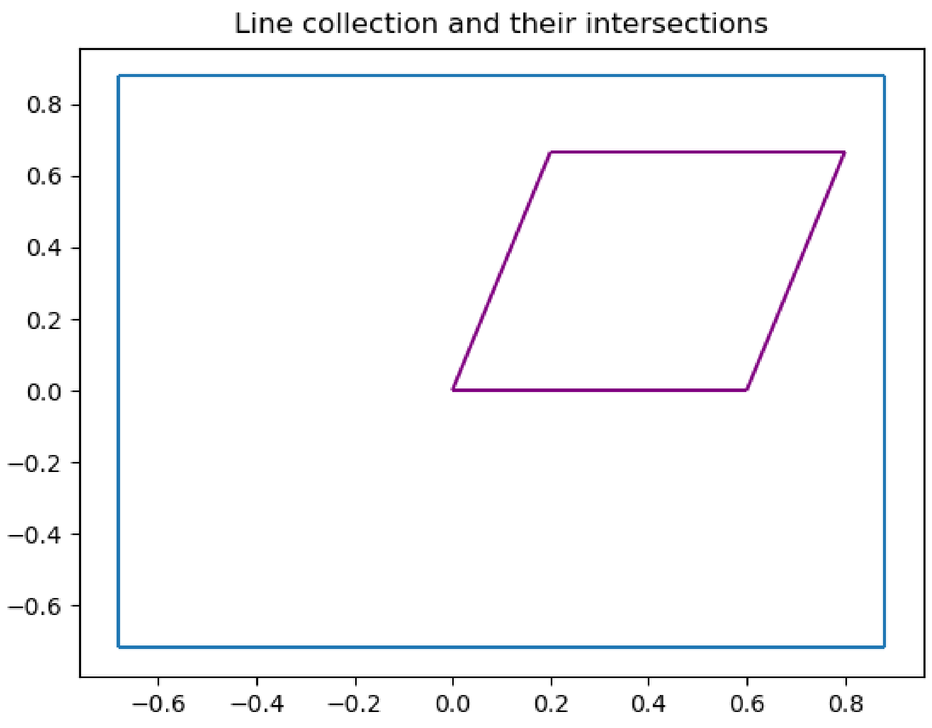

Figure 2: Line Intersection Plot for Test case 2 - Rectangle and Parallelogram

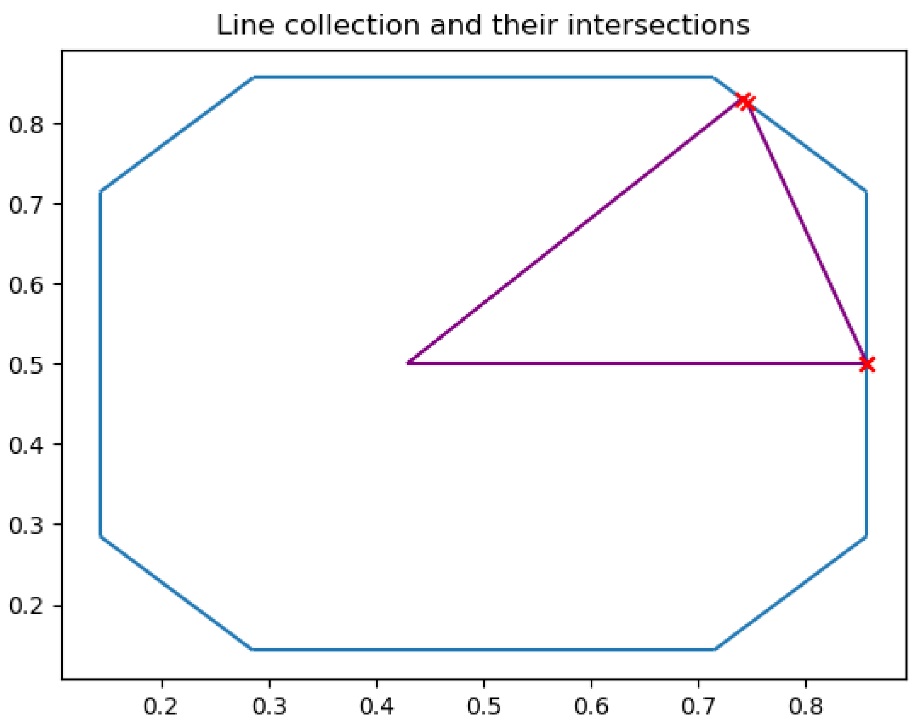

Figure 3: Line Intersection Plot for Test case3 - Hexagon and a Triangle

b. Implement segment_pair_intersection and run main.py - Complexity in BigO Notation and Graph

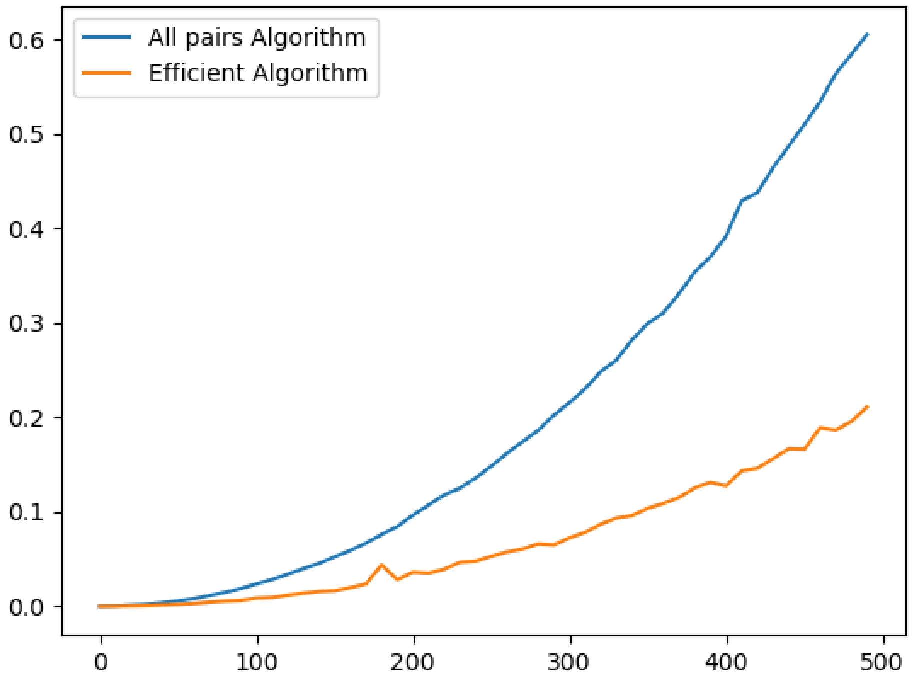

Figure 4: Plot showing the Time taken for running the pseudo code based algorithm and a different efficient algorithm partially based on Sweep line algorithm

- The time complexity of the above all pairs intersection algorithm is $O(NM)$ as there are two for loops which iterate over the N first line segments and M second line segments.
- In case we try to find the intersection of all pairs then it is $O(N^2)$ as here we iterate over the same list of segments to create pairs.
- There are some algorithms such as Bentley-Ottmann algorithm which is based on Sweep line algorithm which can solve the all pairs intersection in $O(NlogN)$ which I tried to solve for the next subsection - Efficient algorithm. The plot of the time taken per iterations is shown in Figure4.

c. Efficient algorithm Output Plots after implementing Efficient algorithm in Python - Same pair of Line segments

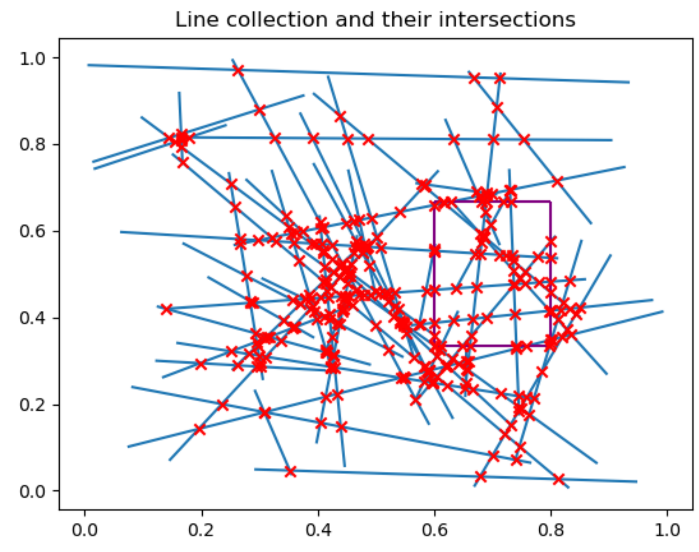

Figure 5: Line Intersection Plot for Test case 1 with Efficient Algorithm

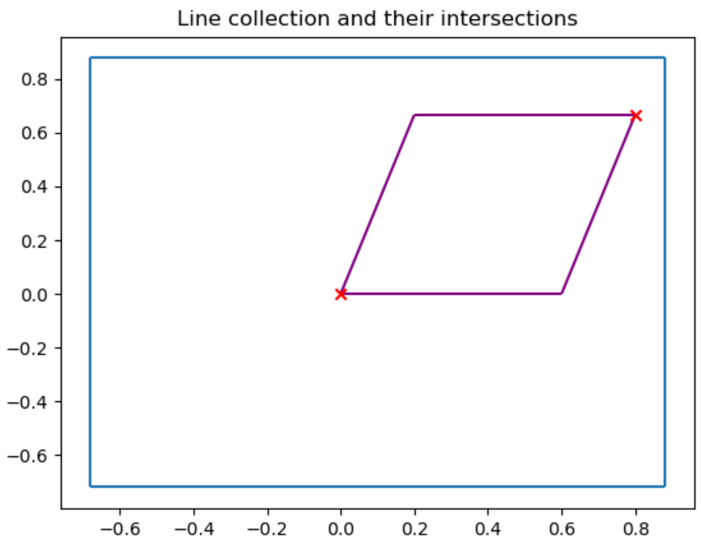

Figure 6: Line Intersection Plot for Test case 2 with Efficient Algorithm

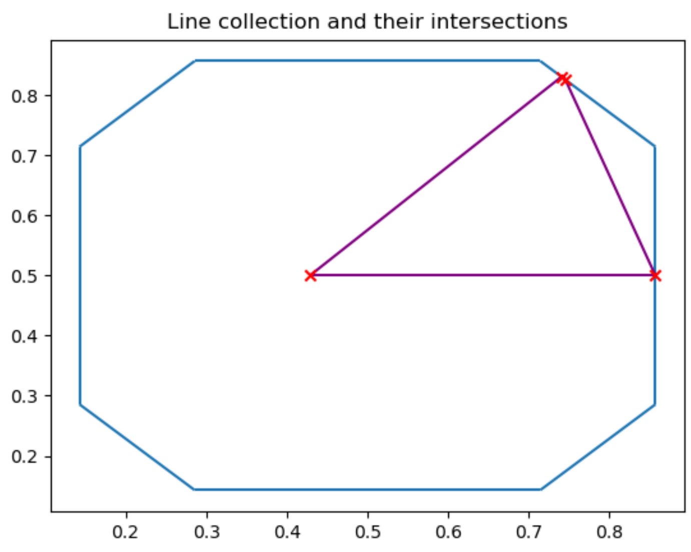

Figure 7: Line Intersection Plot for Test case 3 with Efficient Algorithm

In the above case all the plots above are true as we are trying to find all intersections of all segments in the algorithm , I have implemented which generalizes for both the cases. In case we need to find separately we need to implement a two line segment set algorithm. To have the intersection points on the recentangle or combined intersection points of figures we can add condition such that to check whether the intersection points are on the required line segment or not.

- After implementing the algorithm the algorithm time per iteration plot is attached as Figure 4.
- Even though I'm getting good result but I'm a bit doubtful about my implementation as the code is telling my implementation is not matching with all pairs intersection.
- But what I observed is in all pairs intersection , there can be duplicates as we are iterating over L1 and then again L1 so the result I'm getting in efficient algorithm is exactly half of the all pairs intersection. So my assumption is if we eliminate duplicates my result should match. Not sure, whether I'm wrong or not. I have implemented a custom version of the all pairs intersection which eliminates duplicates which should work. So with that I'm able to satisfy the length condition. This needs to be checked.
- I'm actually new to the algorithm, and for this first time implementation I took help of couple of youtube videos, geeks for geeks and stackoverflow.
- **The algorithm is pretty interesting when it can leverage multiple data structures to solve this especially, the implementation I have written can further be optimized through Balanced Binary search tree based addition/insertion to solve sweep segment insertion and deletion of event points.**
- The algorithm is efficient because it minimizes the number of segments it analyzes by using three concepts - sorting, elimi†nating, maintaining track of seen events.And even though my implementation can be wrong it might be the case once if implemented with Balanced BST it definitely reduces the time complexity as most of the Balanced BST operations are O(logN).
- Overall the implementation I did is a simple sorting , check and eliminating endpoints based one on sweep line but without any datastructures or external libraries.
- **P.S: I have added custom code for checking efficient algorithm and also to handle duplicates in all pairs intersection in the code.**