Ian Chadwick
IanJC
U90596416
ME 570 Robot Motion Planning
Fall 2021
9/12/2021

Homework 1 Report

Question 1.1 Drawing Polygons

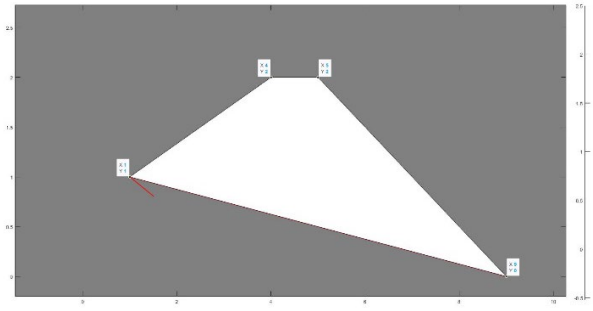


Figure 1: Hollow Polygon

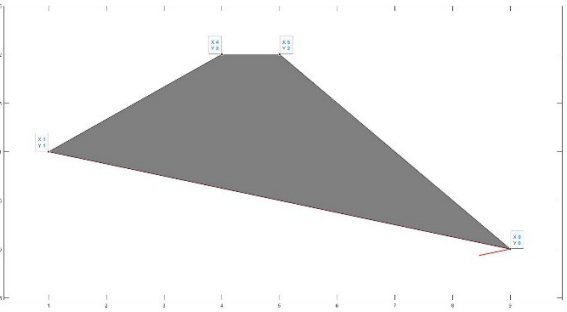


Figure 2: Filled Polygon

The method I used was adapted from *https://www.element84.com/blog/determining-the-winding-of-a-polygon-given-as-a-set-of-ordered-points*, which breaks the problem into a simple geometry problem where you sum the areas of all the trapezoids formed by each edge sequentially. If the sign of the resulting sum is negative, the polygon is oriented counterclockwise, and if the sum is positive it is oriented clockwise.

Question 1.2

In the function **edge_angle**, the variable, **cAngle** is the dot product of the the two unit vectors forming the angle and the variable **sAngle** is the cross product of the unit vectors forming the angle. Theta is given in radians as the output variable **edgeAngle,** by using arctan of **cAngle** and **sAngle** to give an angle from $-\pi \leq \Theta \leq \pi$. If the angle is unsigned it is converted to be in the range $0 \leq \Theta \leq 2\pi$.

Question 1.3

I ran into quite a few issues with the implementation of *polygon_isVisible()*. I spent close to 10 hours on this question alone, and for some reason, even though the functions proved to be correct when implemented manually, when I put them together in this function, the lines drawn were sometimes incorrect. I suspect it had something to do with the way that the point is checked against the edges of the polygon, and I must have made an error in making the comparison, but I have not been able to fix the bug completely.

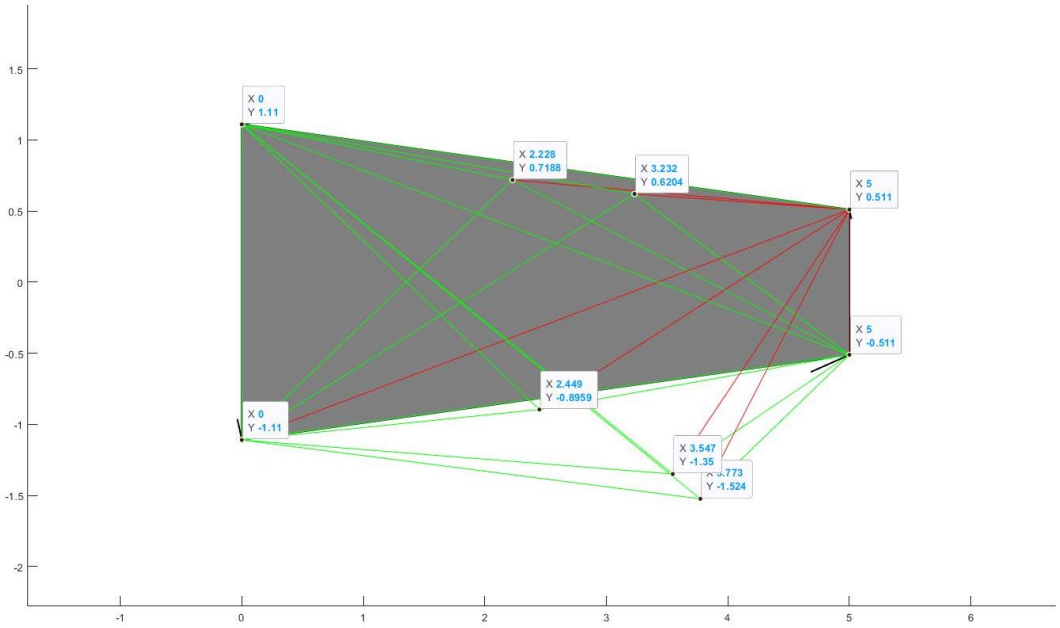


Figure 3 Vertices1 plot

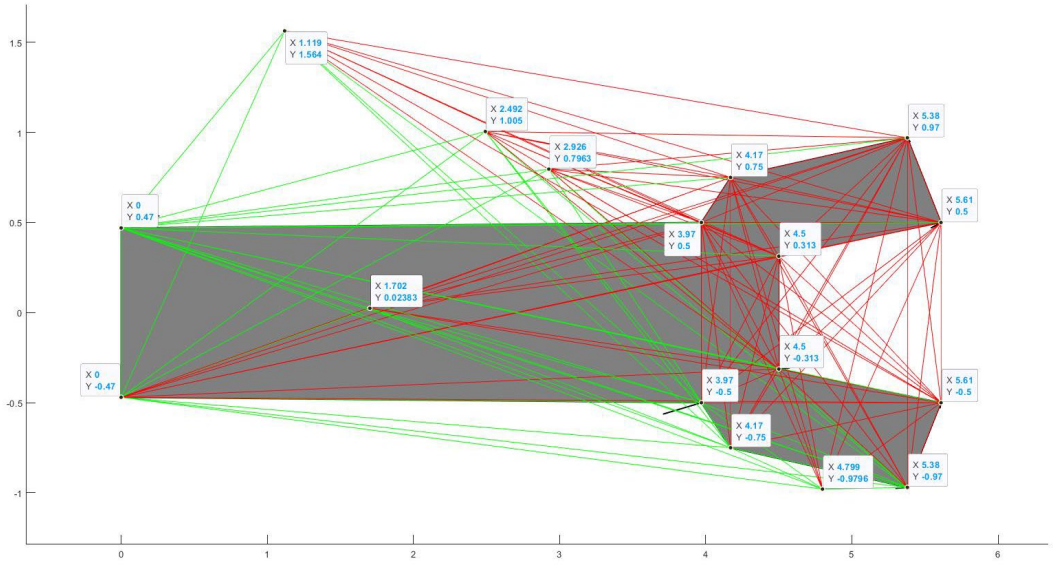


Figure 4 Vertices2 plot

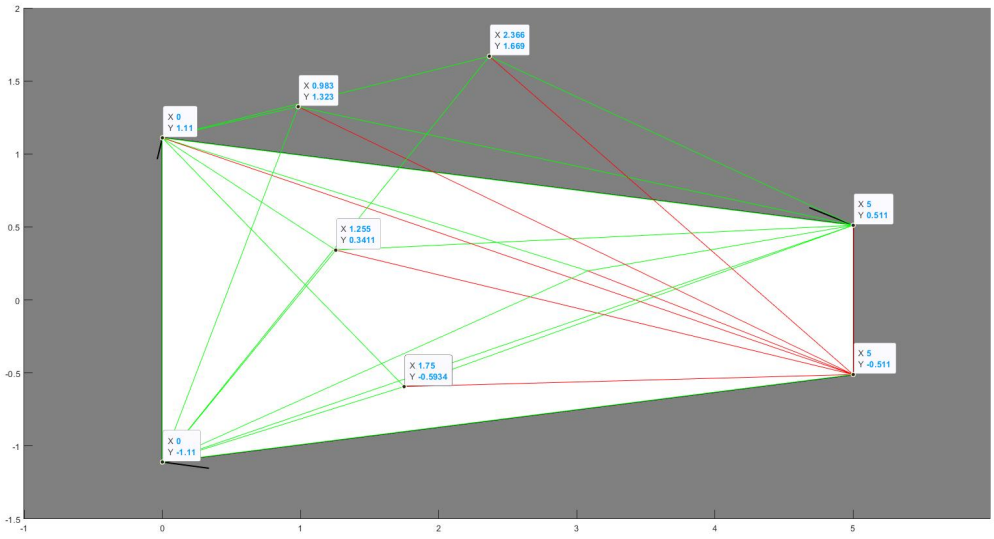


Figure 5 Flipped Vertices1 Plot

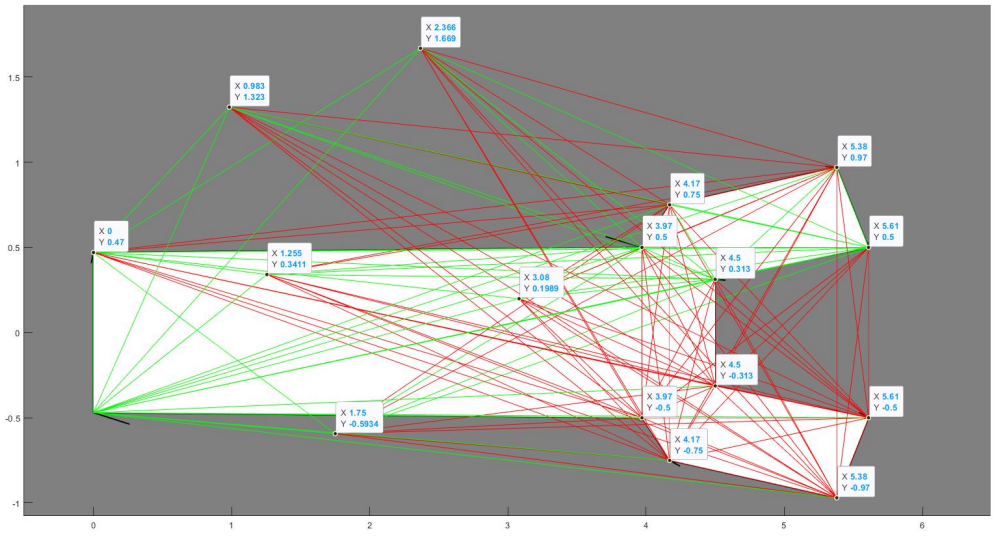
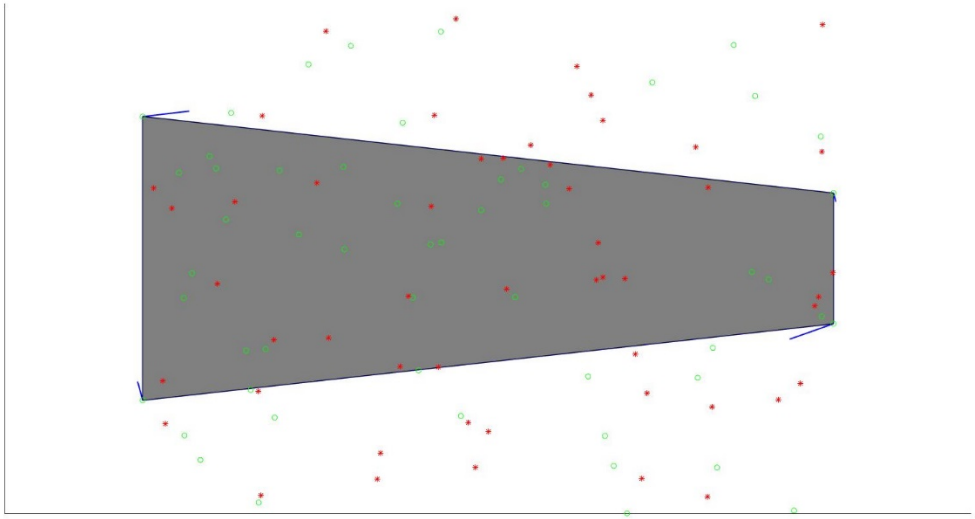


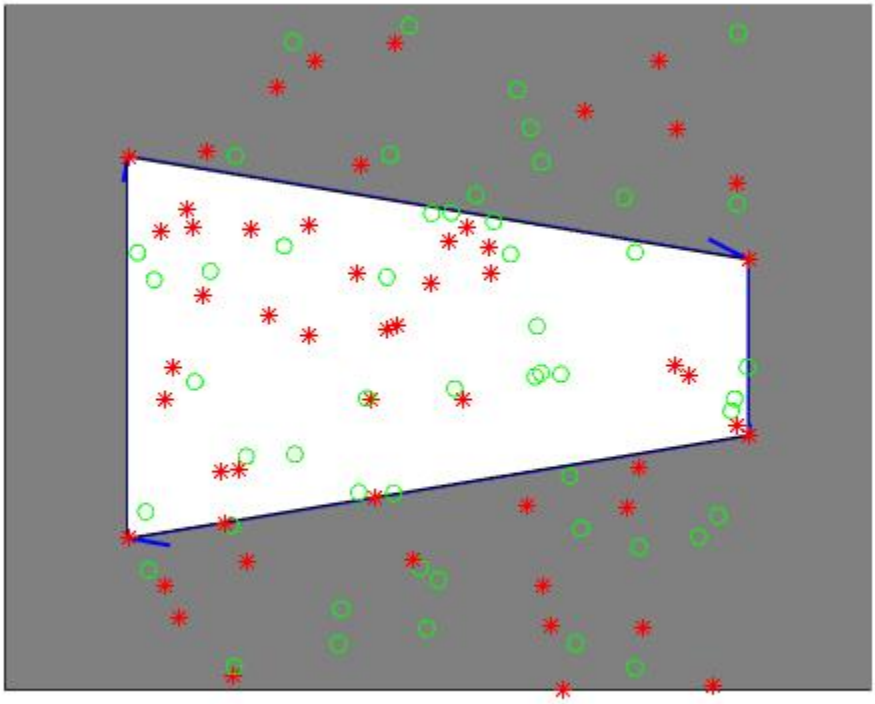
Figure 6 Flipped Vertices2 Plot

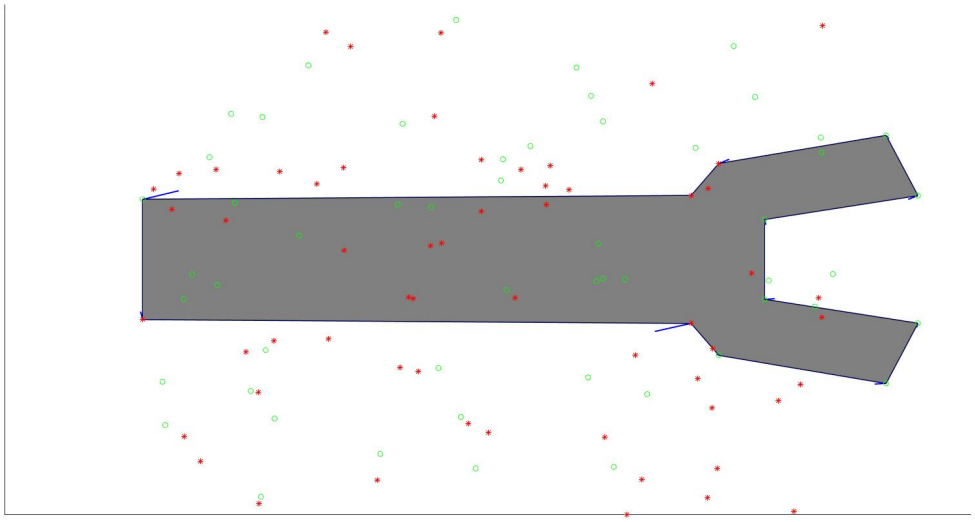
Question 1.4

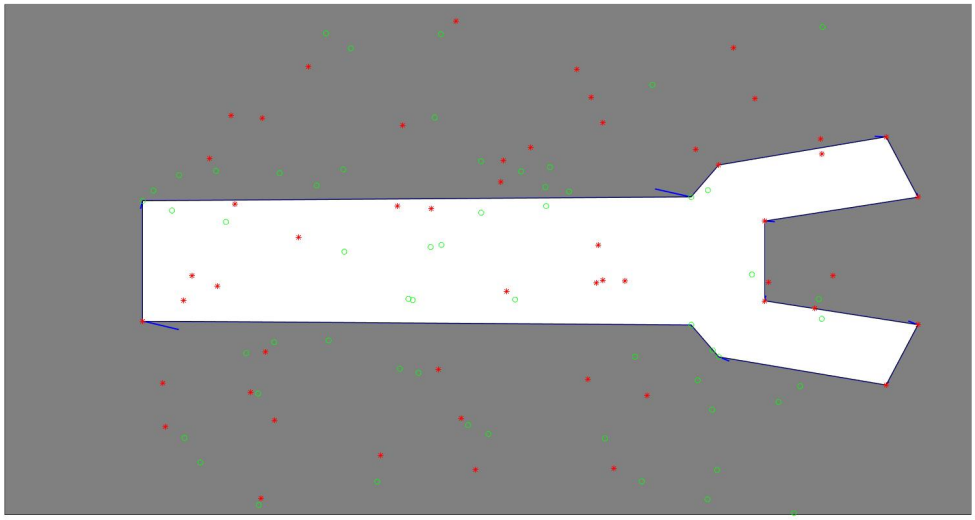
I ran into another issue with not having enough time to debug this particular problem. I used the even-odd algorithm to determine how many times a ray drawn from the point crossed the edges of the polygon, from left to right. If the X or Y value of the point is greater or less than the maximum or minimum X or Y value of the polygon respectively, then it cannot be inside the polygon. Otherwise, if the ray crosses the polygon's edges an odd number of times it is inside, and if it crosses an even number of times it is outside. Then I determined when the polygon was filled or not and adjusted the point values

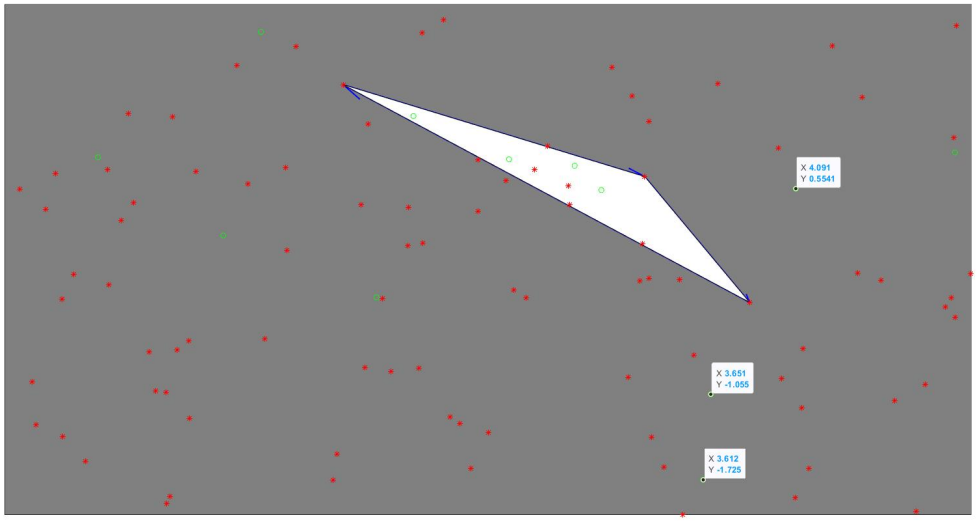
appropriately. However, when I ran the automated test, I get very odd results that look almost random (see figures below).

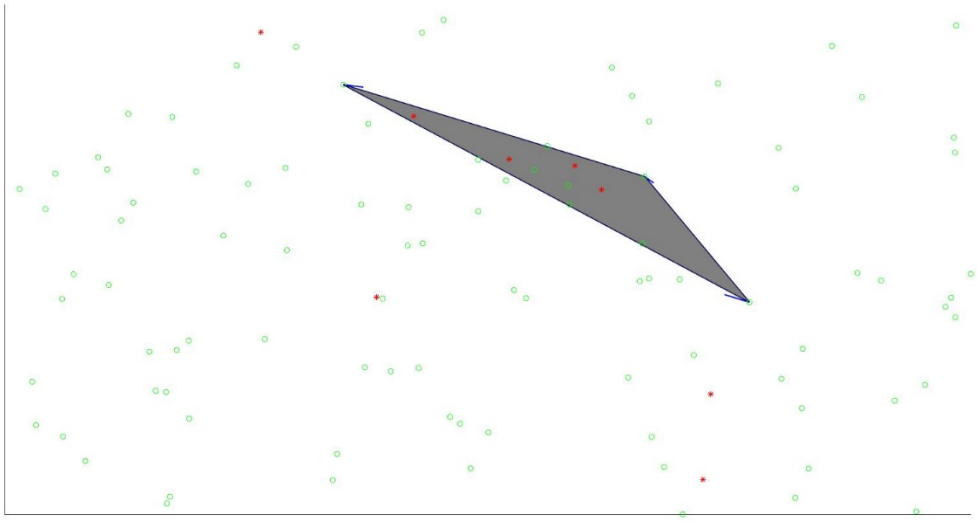












Question 2.1
Sample Run of Priority Test

priority_test()

'Queue Contents:'
pQueue = 0×1 empty struct array with fields:
key
cost

'Insert Oranges, 4.5'

‘Queue Contents:’
struct with fields:
key: 'Oranges'
cost: 4.5000

‘Insert Apples, 1'

‘Queue Contents:’

struct with fields:
key: 'Oranges'
cost: 4.5000

struct with fields:
key: 'Apples'
cost: 1
‘Insert Bananas, 2.7'
‘Queue Contents:’
struct with fields:
key: 'Oranges'
cost: 4.5000
struct with fields:
key: 'Apples'
cost: 1

struct with fields:
key: 'Bananas'
cost: 2.7000

'Extract Min'
priorityQ = 2×1 struct array with fields:
key
cost
extractKey = 'Apples'
extractCost = 1
‘Queue Contents:’
struct with fields:
key: 'Oranges'
cost: 4.5000

struct with fields:
key: 'Bananas'
cost: 2.7000
‘Insert Canaloupe, 3'

‘Queue Contents:’
struct with fields:
key: 'Oranges'
cost: 4.5000
struct with fields:
key: 'Bananas'
cost: 2.7000
struct with fields:
key: 'Cantaloupe'
cost: 3
'Is Oranges a member?'
member = logical   1
'Extracting remaining contents'
priorityQ = 2×1 struct array with fields:
key
cost

extractKey =   'Bananas'
extractCost =   2.7000
‘Queue Contents:’
struct with fields:
key: 'Oranges'
cost: 4.5000
struct with fields:
key: 'Cantaloupe'
cost: 3
priorityQ = struct with fields:
key: 'Oranges'
cost: 4.5000
extractKey =   'Cantaloupe'
extractCost =   3
‘Queue Contents:’
struct with fields:
key: 'Oranges'
cost: 4.5000
priorityQ = 1×0 empty struct array with fields:
key
cost
extractKey = 'Oranges'
extractCost = 4.5000
‘Queue Contents:’
pQueue = 1×0 empty struct array with fields:
key

cost
diary off

Sample Run with non-string Keys:
Q = priority_prepare()

Q = 0×1 empty struct array with fields:
key
cost

Q = priority_insert(Q, 12312, 34234)

Q = struct with fields:
key: 12312
cost: 34234

Q = priority_insert(Q, [1 2], 34234)

Q = array with fields:
key
cost

priority_isMember(Q, [1 2])

ans = logical 1
diary off

I created a queue with some non-string keys. Specifically a number (12312) and a vector ([1,2]) and I was able to test membership by the key, using *priority_isMember* because I utilized the *isEqual* function to determine if the key values were equal, which allows for comparison between non-numeric and non-string datatypes.

Question 2.2
You would first create a priority queue using priority_prepare. Then you could use the *priority_insert* function to insert all the elements, regardless of their order. After that you could simply use *priority_minExtract* until the queue is empty. This would sequentially list all the elements in descending order according to their cost.