Algorithmic Motion Planning HW1

Shai Kimhi, Dor Noti

November 24, 2021

1 Properties of Mikowski's sum and Euler's theorom

1.1 First Question

For $X \oplus Y$, if $X = \emptyset$ or $Y = \emptyset$ then $X \oplus Y = \emptyset$. So if $A = \emptyset$ or $B = C = \emptyset = B \cup C$, then $A \oplus (B \cup C) = \emptyset$.

If $B = \emptyset$ or $C = \emptyset$ then $B \cup C = B$ or $B \cup C = C$.

Assume that $B = \emptyset$ without loss of generality, then $A \oplus (B \cup C) = A \oplus C$. Since $A \oplus B = \emptyset$ we can write $A \oplus C = (A \oplus B) \cup (A \oplus C)$.

We're therefore getting: $A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$.
Let $a \in A$, $b \in B$, $c \in C$ and $\alpha \in A \cup B$ (meaning, $\alpha \in A$ or $\alpha \in B$). Since $b \in B$ then $b \in B \cup C$ and there, according to Minkowski sum definition: $a + b \in A \oplus (B \cup C)$.

Also, according to Minkowski sum: $a + b \in A \oplus B$ then according to the union definiotion: $a + b \in (A \oplus B) \cup (A \oplus C)$. This is, of course also right for $c \in C$.

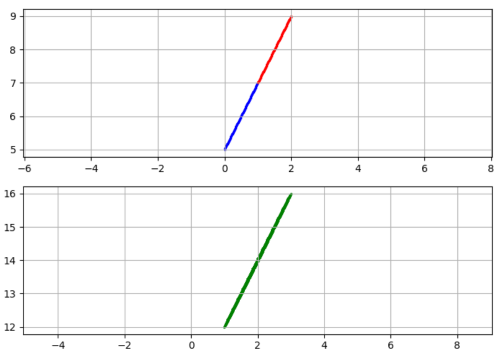
Therefore, $a + \alpha \in A \oplus (B + C)$ iff $a + \alpha \in (A \oplus B) \cup (A \oplus C)$ (α gives us the WLOG). Threfore $A \oplus C = (A \oplus B) \cup (A \oplus C)$.

1.2 Second Qusetion

1. For poins p, q the Minkowski sum is defined as $\{p\} \oplus \{q\} = \{p + q\}$. The Minkowski sum is just the sum of the two points.

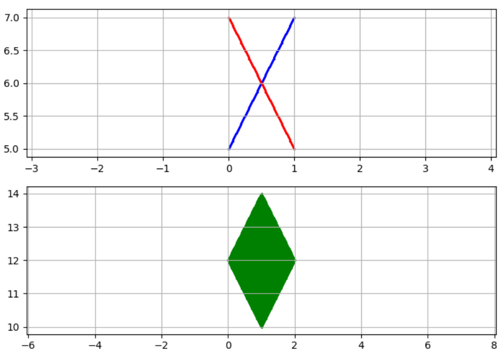
2. For a point p and a line $l = \{l_1, l_2 ...\}$ we get a translation of the line by p since we're adding a constant (p) to all the points of the line.

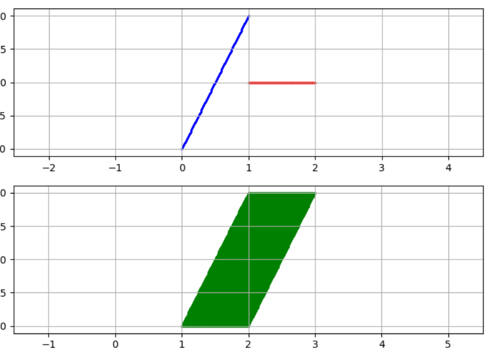
Figure 1: Minkowski sum of two lines segments in the same direction



3. For two line segments, we get an extention of the line in the direction of the other line and translated. For example, if we have a two lines, not in the same direction (not parallel), we get a wide line. If the lines are parallel, we get an extention of the line. The length is the sum of the two line length minus the overlaping (of the projection) since we add it twice. Of course this is also upsampled. The translation is according to the sum of the two lines.

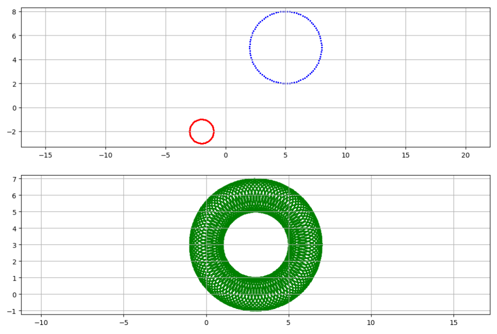
Figure 2: Minkowski sum of segments of crossing lines





4. In a case of a disk each point of one disk induces a disk around it. Therefore, we get a "donut". The center of the donut is located at the sum of the center points of the two disks, the outer radius is the sum of radii (the maximum point in each direction) and the inner radius is the difference of the radii. We can also look at it as a small circle aggragated along the bigger circle's pointswith a translated center).

Figure 3: Minkowski sum of two 2-dimensional disks



1.3 Third Question

The maximal number of faces in a graph is obtainde if every face has no more than three edges. In additin, every edge is between two faces. Threrefore we get for a graph $G = (V, E)$, $e = \|E\|$, $f = \|F\|$ (number of faces) $v = \|V\|$:

$$2e \geq 3f \rightarrow f \leq \frac{2}{3}e$$

From Euler equation we know that $2 = v - e + f$ and therefore:

$$2 = v - e + f \leq v - e + \frac{2}{3}e = v - \frac{1}{3}e$$

$$6 \leq 3v - e$$

Threfore, we're getting:

$$\boldsymbol{e \leq 3v - 6}$$

2 Code Review- Exact Motion Planning for a Diamond-Shaped Robot

2.1 Preprocessing (1)- Construction of C-Space obstacles

2.1.1 Complexity

Let n be the number of vertices in the input polygon and let m be the number of vertices of the robot.

First, we make sure that the input correspond with the criterias of the C-Space sonstruction algorithm.

(a) Forming the arrays is $O(m+n)$ (adding one point at a time)

(b) Checking that the vertices are organized are ccw is $O(n)$ (multiplying consecutive pair of edges)

(c) reversing the order is $O(n)$ (adding one point for each point)

(d) Finding the minimum y index as the first one and reorganizing the polygon accordingly is $O(n)$ (checking each point and adding points at the correct order)

Next, the loop over the points. In each iteration we promote at least one index for the polygon vertices (of $n+2$ vertices) and one for the robot vertices (of $m+2$ vertices). We have maximum of $n+m+4 = O(n+m)$ iteration. In each iteration:

(a) Updating values - $O(1)$

(b) Adding vertex to the output - $O(1)$

(c) Calculating angles - $O(1)$

(d) Updating indices - $O(1)$

In total - $\boldsymbol{O(m+n)}$.

2.1.2 Why is the obstacle has to be convex?

For a convex polygon, in each iteration, each angle's value doesn't decrese (the angle increases or remains the same). If we have a non-convex polygon, for some polygon index k, the angle of the obstacle decreses and therefore, the index for the robot might increase. Once the index increases, there's no way back, so the direction vector is set to be the one of the next robot's vertex even though the angle for the $k+1$ polygon index is lower.

For example, for a non-convex polygon and a squared robot, for $i=2$ and $j=3$ the C-Space obstacle vertex is placed inside the workspace obstacle and therefore doesn't even contain the workspace obstacle.

2.2 Preprocessing (2)- Building the Visibility diagram

In this phase, we collect all the points of the C-space polygons together with source and destination if they are given as parameters as vertices of the visibility graph. After that we check each possible selection of two different vertices whether they intersect with any C-space obstacle, and so we have $O(n^2)$ selections of two vertices, and for each one the number of tests to check intersection with each existing C-space obstacle will be with $O(n)$ time complexity depending on the specific implementation of the intersection functions in *shapely*.
In total, the complexity is $O(n^3)$.

2.3 Query Phase- Computing shortest paths

We are running Uniform-Cost-Search algorithm on the Visibility graph which is equivalent to dijkstra but calculates the shortest path and its distance to a specific destination.
Claim: Each vertex will be added to the open queue at most once
Proof: Lets assume: $\exists e = u \rightarrow v$ such that e is entered twice in to the open queue.
Thus, is means u was expanded twice since the only way an edge enters the open queue is through expanding of its source vertex.
At the end of the first expansion, u is added to *Close* and at any possible later expansion, if u is already at close the expansion will not occur.
Thus, there is no such edge e and there are at most |E|=O(|V|^2) iterations over edges.
Each edge iteration causes addition of its target vertex to the open queue and at most one check of *Close* in case that vertex is sent to expansion, only at most once each vertex is added to *Close* and added to *Prev* dictionary.
Each Operation on *Close* and a minimum removal from open queue, as well as operation of *Prev* takes O(log |E|) time (since open queue is a minimum heap and *Close* and *Prev* are AVL trees).
In Conclusion, the time complexity is O(|E|log|V|)=O(|V|$^2 log|V|) = O(n^2 logn)$ as n is the number of points in the original polygons and so the number of vertices in the graph is at most the number of points in the C-space polygon obstacles which is O(n).

3 Results

3.0.1 Provided Setting

Now we will show the results for the provided and custom settings

Figure 4: C-space obstacles for the provided setting

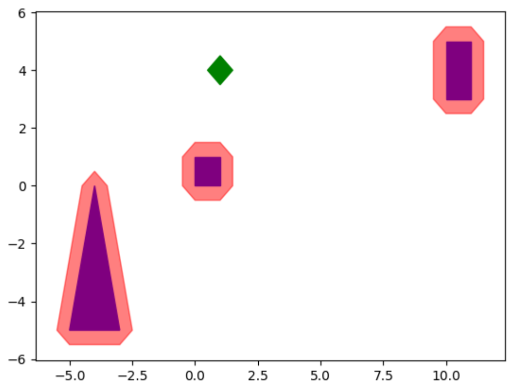


Figure 5: Visibility Graph for the provided setting

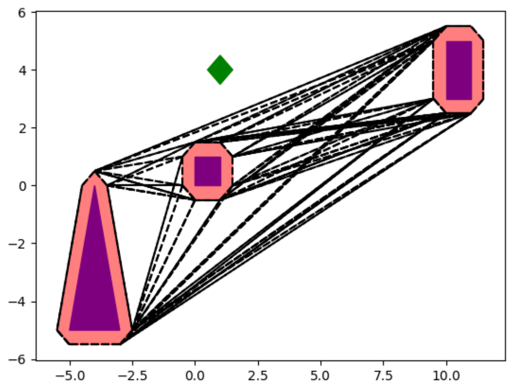


Figure 6: Shortest route to destination for the provided setting

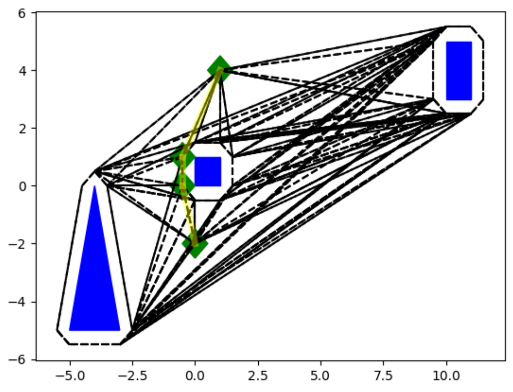


Figure 7: C-space obstacles for our setting

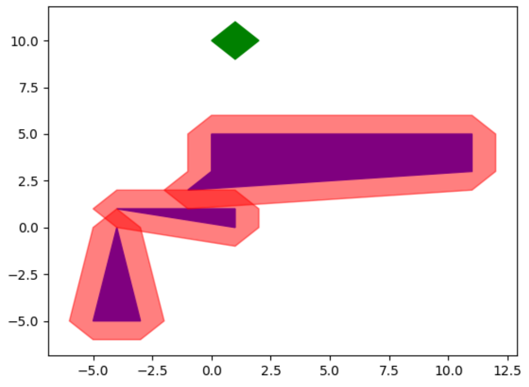


Figure 8: Visibility Graph for our setting

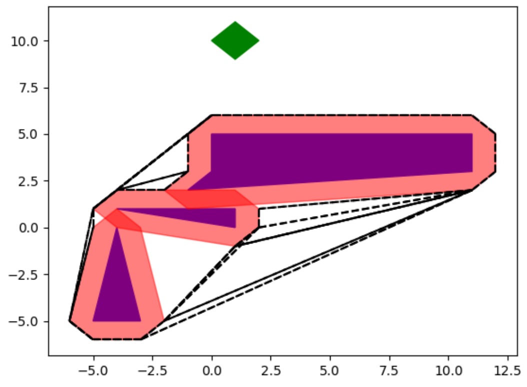
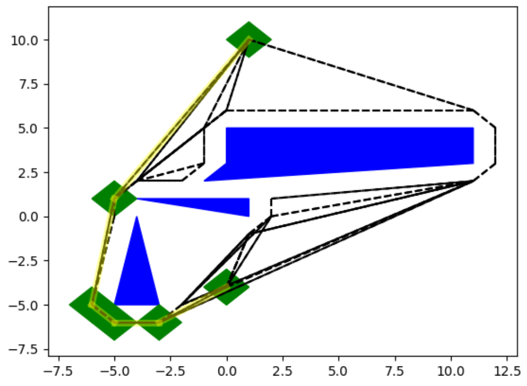


Figure 9: Shortest route to destination for our setting


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5