Algorithmic motion planning – 02360767
Homework 1

Date: 19/12/24

Ameer Daood – ameer.daood@campus.technion.ac.il – 214039794

Henrique Santos – Henrique.s@campus.technion.ac.il - 800006355

2. Properties of Minkowski Sums and Euler's Theorem

2.1 Proof of $A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$:

1. **Definition of the Minkowski Sum:**

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$

2. **Left-hand side:** Expanding $A \oplus (B \cup C)$:

$$A \oplus (B \cup C) = \{a + b \mid a \in A, b \in (B \cup C)\}$$

$$A \oplus (B \cup C) = \{a + b \mid a \in A, b \in B\} \cup \{a + c \mid a \in A, c \in C\}.$$

3. **Right-hand side:** Expanding $(A \oplus B) \cup (A \oplus C)$:

$$(A \oplus B) \cup (A \oplus C) = \{a + b \mid a \in A, b \in B\} \cup \{a + c \mid a \in A, c \in C\}.$$

4. **Since both sides are equal:**

$$A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$$

2.2 Minkowski Sums of Different Geometric Objects

1. **Two Points A and B:**

$$A = (x_1, y_1), \quad B = (x_2, y_2) \implies A \oplus B = (x_1 + x_2, y_1 + y_2)$$

2. **Point and Line:**

$$A = (x_a, y_a), \quad L = \{(x, y) \mid ax + by = c\} \implies A \oplus L = \{(x + x_a, y + y_a) \mid ax + by = c\}$$

3. **Two Line Segments:**

- **Parallel case:**

$$A \oplus B = \text{A segment whose length is the sum of both lengths.}$$

- **Non-parallel case:**

$$A \oplus B = \text{A parallelogram formed by the endpoints.}$$

4. **Two Disks with Radii r_1 and r_2:**

$$r = r_1 + r_2$$

2.3 Proof of $E \leq 3V - 6$:

Using Euler's formula for planar graphs:

$$V - E + F = 2$$

Where V, E, and F are the number of vertices, edges, and faces, respectively. For a simple planar graph:

1. Each face is bounded by at least 3 edges:

$$F \leq \frac{2E}{3}$$

2. Substituting F into Euler's formula:

$$V - E + \frac{2E}{3} = 2$$

3. Rearranging:

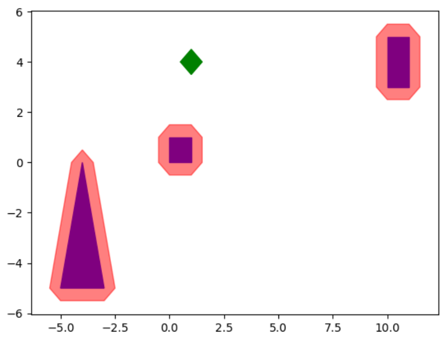
$$E \leq 3V - 6$$

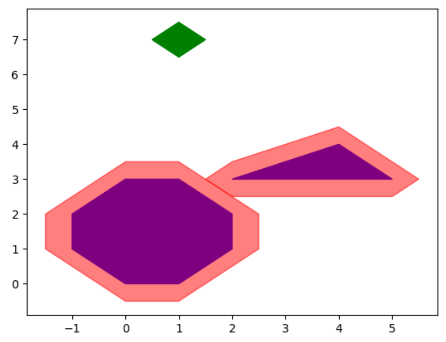
3. Exact Motion Planning for a Diamond-Shaped Robot

<u>3.2 Preprocessing phase (1)—constructing the C-space</u>

<u>Your Instance</u>

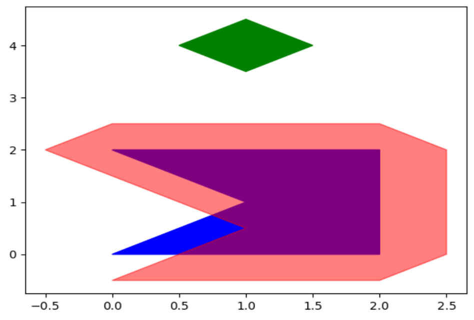
<u>Our Instance</u>





The computational complexity of the minkowsky sum function is $\boldsymbol{O(n)}$, where n is the number of vertices in the input polygon (original_shape). This is because the function processes each vertex of the polygon in a single pass, combining it with the constant number of vertices (4) from the robot because its shaped like a rhombus. Within each iteration, computing the angles and comparing them are done in constant time, making the overall complexity scale linearly with the size of the input polygon. Generically, if the robot has m angles, then the complexity would be $\boldsymbol{O(n+m)}$.

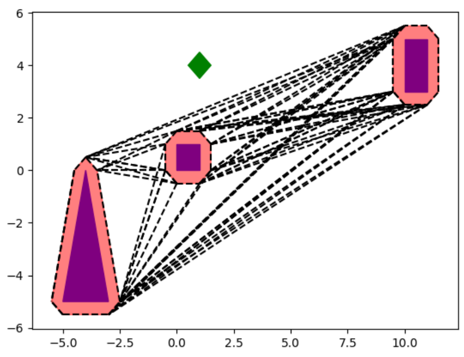
Non-convex obstacles can lead to results that are more complex and potentially less intuitive because the sum will combine all edges of the obstacles, including concave regions. This can result in "bulges" or additional areas in the resulting shape that may not accurately represent the reachable or obstructed regions. Attached bellow a result of minkowsky sum of a non-convix obstacle.



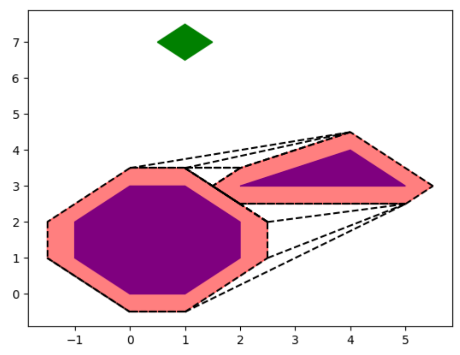
We can see clearly that the result isn't correct.

3.3 Preprocessing phase (2)—building the visibility diagram

Your Instance



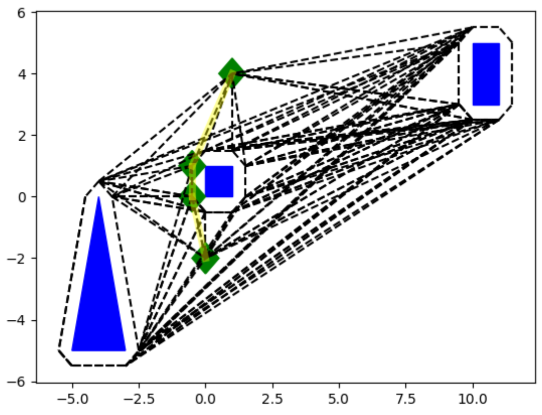
Our Instance



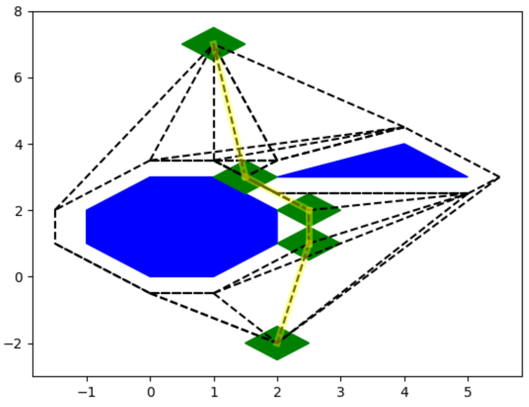
Complexity: Lets assume we a total of n vertices across all obstacles. First we iterate through all obstacles and add the vertices into a list, which is $O(n)$**.** Then, we take every possible pair of vertices from the list, check if it intersects with an obstcale, and if not add it to the result. There are n^2 possible pairs, and every check we did it naivly with all of the obstacles, and the number of obstacles is linear with n, so the overall complexity is $\boldsymbol{O(n^3)}$.

3.4 Query phase—computing shortest paths

Your Instance



Our Instance



Complexity: Lets assume we a total of n vertices across all obstacles. Then at most we have n^2 lines or "edges" of the graph. First, we build a hash map of the lines we receive as an input with $O(n^2)$. Then, we implemented Dijkstra with a priority queue. Inserting into the priority queue takes $O(\log{(n^2)}) = O(\log(n))$. At most we do this for every vertice, so n^2 times. We keep a "visited" list to not go over the same vertice twice, which is valid because the "cost" is positive in all of the edges (length of path is positive). Finally, we get $\boldsymbol{O(n^2 \log{(n)})}$

4. Forward and Inverse Kinematics of the Manipulator

4.1 Forward Kinematics

1. **General Equations for the End-Effector:**

$$x_e = \ell_1 \cos(\theta_1) + \ell_2 \cos(\theta_1 + \theta_2) + \ell_3 \cos(\theta_1 + \theta_2 + \theta_3)$$

$$y_e = \ell_1 \sin(\theta_1) + \ell_2 \sin(\theta_1 + \theta_2) + \ell_3 \sin(\theta_1 + \theta_2 + \theta_3)$$

2. **Computation for $\theta_1 = 30°, \theta_2 = 45°, \theta_3 = -15°$:** Converting angles to radians:

$$\theta_1 = \frac{\pi}{6}, \quad \theta_2 = \frac{\pi}{4}, \quad \theta_3 = -\frac{\pi}{12}.$$

$$x_e = 3\cos\left(\frac{\pi}{6}\right) + 2\cos\left(\frac{\pi}{6} + \frac{\pi}{4}\right) + 1\cos\left(\frac{\pi}{6} + \frac{\pi}{4} - \frac{\pi}{12}\right)$$
$$y_e = 3\sin\left(\frac{\pi}{6}\right) + 2\sin\left(\frac{\pi}{6} + \frac{\pi}{4}\right) + 1\sin\left(\frac{\pi}{6} + \frac{\pi}{4} - \frac{\pi}{12}\right)$$

3. **Numerical Results:**
$$x_e \approx 4.13, \quad y_e \approx 3.03$$

4. **Final Orientation:** The sum of the angles gives the orientation:

$$\theta_{\text{final}} = \theta_1 + \theta_2 + \theta_3 = 60°.$$

4.2 Inverse Kinematics

1. **Reachability Check:** For the desired position $(x_e = 4, y_e = 2)$:

$$r = \sqrt{x_e^2 + y_e^2} = \sqrt{4^2 + 2^2} = \sqrt{20} \approx 4.47\,\text{units}.$$

Since $r \leq \ell_1 + \ell_2 + \ell_3 = 6$, the position is reachable.

2. **Possible Joint Angles:** One possible solution:

$$\theta_1 = 20°, \quad \theta_2 = 40°, \quad \theta_3 = -30°$$

3. **Multiple Solutions:** Geometric redundancy allows for multiple angle configurations. For instance:
$$\theta_1 = 25°, \quad \theta_2 = 35°, \quad \theta_3 = -30°$$