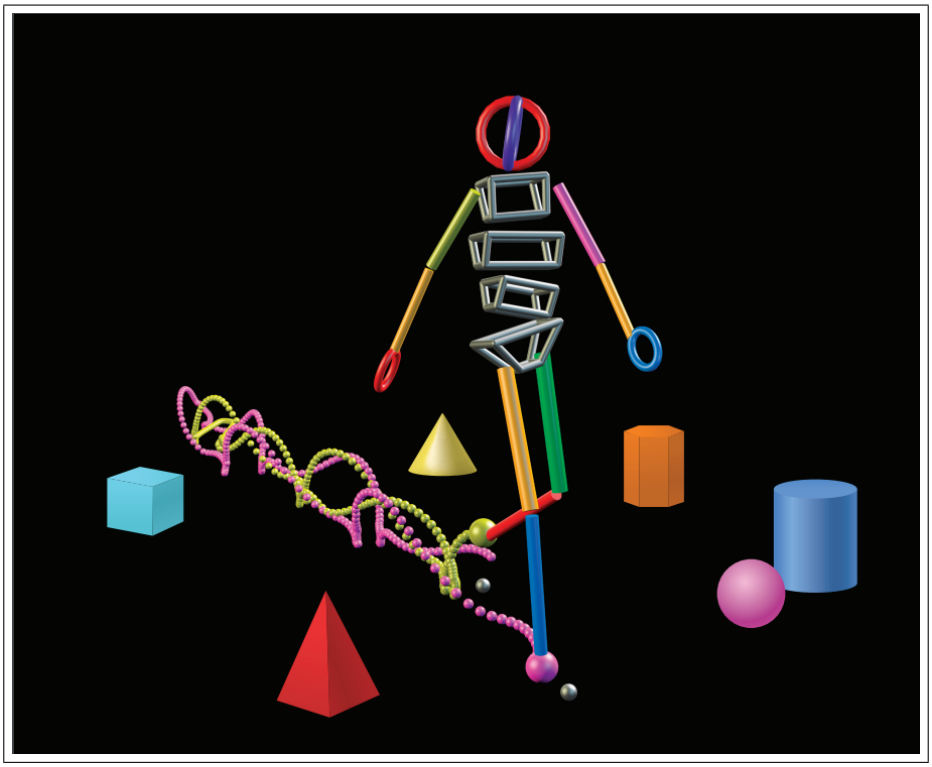
Algorithmic Robot Motion Planning - 236901

Homework #1: Exact Motion Planning



Name	ID	Email
Niv Ostroff	212732101	nivostroff@campus.technion.ac.il
Yotam Granov	211870241	yotam.g@campus.technion.ac.il

Faculty of Computer Science
Technion - Israel Institute of Technology

Winter 2022/23

(1) Properties of Minkowski Sums & Euler's Theorem

Part 1.1

Given sets A, B and C formally prove that $A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$.

Solution:

The Minkowski sum of two sets $S_1, S_2 \subset \mathbb{R}^2$ is defined as $S_1 \oplus S_2 = \{p+q \ : \ p \in S_1, q \in S_2\}$, where $p + q$ is the vector sum of p and q. We will prove that, given sets A, B, C, it holds that $A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$ by showing that the two sides of the equality are subsets of each other:

(1) $(A \oplus B) \cup (A \oplus C) \subseteq A \oplus (B \cup C)$

Let there exist a_1, a_2 such that:

$$a_1 + a_2 \subseteq (A \oplus B) \cup (A \oplus C) \rightarrow a_1 + a_2 \subseteq (A \oplus B) \ or \ a_1 + a_2 \subseteq (A \oplus C)$$

If $a_1 + a_2 \subseteq (A \oplus B)$, then:

$$a_1 \in A \ and \ a_2 \in B \rightarrow a_2 \in B \cup C \rightarrow a_1 + a_2 \in A \oplus (B \cup C)$$

If $a_1 + a_2 \subseteq (A \oplus C)$, then:

$$a_1 \in A \ and \ a_2 \in C \rightarrow a_2 \in B \cup C \rightarrow a_1 + a_2 \in A \oplus (B \cup C)$$

Thus $a_1 + a_2 \subseteq (A \oplus B) \cup (A \oplus C)$ implies that $a_1 + a_2 \in A \oplus (B \cup C)$, and so it must be the case that $(A \oplus B) \cup (A \oplus C) \subseteq A \oplus (B \cup C)$.

(2) $A \oplus (B \cup C) \subseteq (A \oplus B) \cup (A \oplus C)$

Let there exist a_1, a_2 such that:

$$a_1 \in A \ and \ a_2 \in B \cup C$$

and so either $a_2 \in B$ or $a_2 \in C$. If $a_2 \in B$, then:

$$a_1 + a_2 \in A \oplus B$$

If $a_2 \in C$, then:

$$a_1 + a_2 \in A \oplus C$$

Thus, for $a_1 + a_2 \in A \oplus (B \cup C)$ implies that $a_1 + a_2 \in (A \oplus B) \cup (A \oplus C)$, and so it must be the case that $A \oplus (B \cup C) \subseteq (A \oplus B) \cup (A \oplus C)$.

Finally, since $(A \oplus B) \cup (A \oplus C) \subseteq A \oplus (B \cup C)$ and $A \oplus (B \cup C) \subseteq (A \oplus B) \cup (A \oplus C)$, it is clear that $A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$. ■

Part 1.2

What is the Minkowski sum (what geometric object and what can you say about it) of (i) two points, (ii) a point and a line, (iii) two lines segments (think of all possible cases), and (iv) two disks?

Solution:

(i) Two Points: The Minkowski sum is defined as $A \oplus B = \{a + b \mid a \in A, b \in B\}$. In the case of points, the sets A and B include only one element a and b respectively, and thus their Minkowski sum will just be the vector addition of their lone elements, i.e. $A \oplus B = a + b$. Thus, if we define those elements in 2D:

$$\mathbf{a} = (a_1, a_2),\ \mathbf{b} = (b_1, b_2)$$

then their Minkowski sum will just be a **new point**:

$$A \oplus B = \mathbf{a} + \mathbf{b} = (a_1 + b_1, a_2 + b_2) \triangleq (c_1, c_2) = \mathbf{c}$$

(ii) A Point & A Line: In this case, we have one point along with a line (which can be represented as a set of points using a vector). We let the point be $\mathbf{a} = (a_1, a_2)$ and the line be $\mathbf{b} = (b_{1,0}, b_{2,0}) + x \cdot (b_1, b_2) = (b_{1,0} + x \cdot b_1, b_{2,0} + x \cdot b_2)$. The Minkowski sum of these two entities will once again just be their vector sum, which can be conducted as follows:

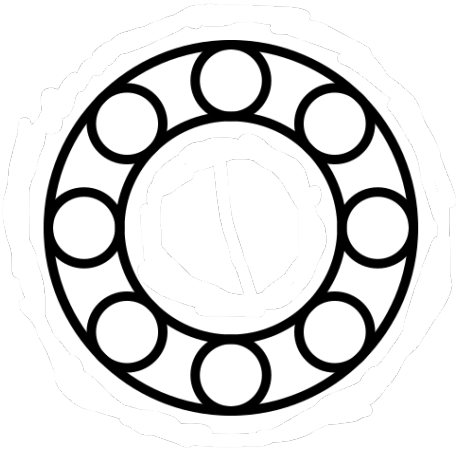
$$A \oplus B = \mathbf{a} + \mathbf{b} = (a_1 + b_{1,0} + x \cdot b_1, a_2 + b_{2,0} + x \cdot b_2) = (a_1 + b_{1,0}, a_2 + b_{2,0}) + x \cdot (b_1, b_2)$$

We can see that this is also **a line**, and in fact it is the same line we had before but now translated in space by (a_1, a_2).

(iii) Two Line Segments: We would expect three results in this case, depending on whether or not the line segments are collinear and/or parallel.

- If they are collinear, then their Minkowski sum will return another line segment which is also collinear with the two original lines, and whose length is equal to the sum of the lengths of the original lines.
- If they are parallel, then their Minkowski sum will return another line segment which is also parallel to the two original lines, and whose length is equal to the sum of the lengths of the original lines.
- If two line segments are not parallel, then their Minkowski sum will return a polygon (specifically, a parallelogram).

(iv) Two Disks: We can easily visualize the Minkowski sum of two disks with the help of ball bearings, which might look as follows:



We see that if we roll one disk along the outer boundary of the other disk, we obtain a larger disk whose diameter is exactly the sum of the diameters of the two original disks (and the center of the new disk will be displaced according to the location of the centers of the original disks). Thus, the Minkowski sum of two disks is just another **disk**.

Part 1.3

Recall that for the proof of Lemma 6.2 (complexity of a trapezoidal map) we used the property that in a planar graph we have that

$$E \leq 3V - 6 \tag{1}$$

Here E and V are the number of edges and vertices in a planar graph, respectively. Prove Eq. 1 (you can assume that $V \geq 3$).

Solution:

The smallest polygon attainable in a trapezoidal map is a triangle, for which there are 2 faces and 3 edges. Every other polygon (where $E > 3$) will still have 2 faces, and thus we can establish the following relation for any polygon (which will also hold true for any planar graph):

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

The Euler polyhedra formula tells us that $V - E + F = 2$. Thus, we can see that:

$$V = E - F + 2 \geq E - \frac{2}{3}E + 2 = \frac{1}{3}E + 2$$

$$\rightarrow V \geq \frac{1}{3}E + 2 \rightarrow \mathbf{E \leq 3V - 6}$$

The equality holds if the planar graph is made up of only triangles (whose edge-to-face ratio is maximal among polygons).

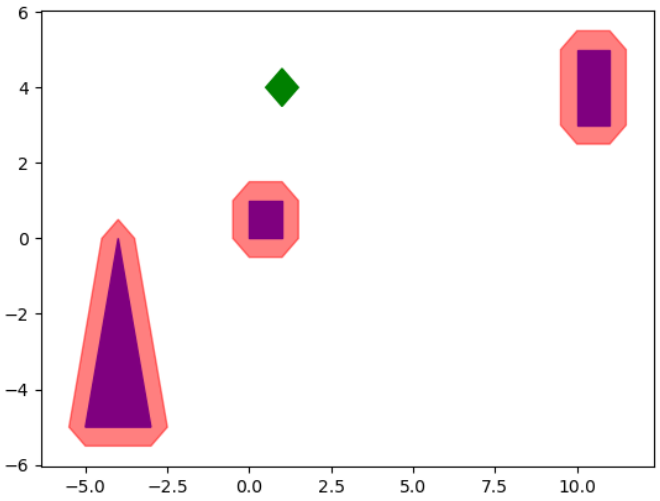
(2) Exact Motion Planning for a Diamond-Shaped Robot

Part 2.1 - Constructing the C-space

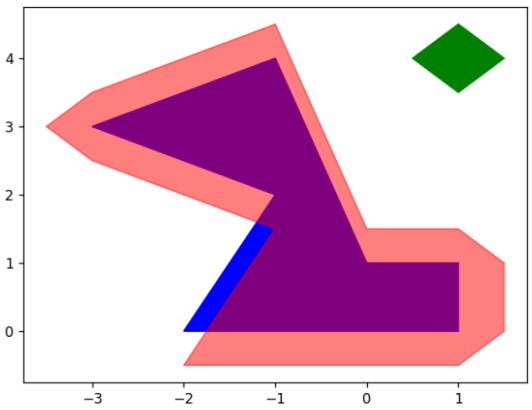
In the first step, we will construct the C-space of our rotated square-shaped robot R. This will be done by computing the Minkowski sum of R with each obstacle O. **Visualize** the C-space and **describe** the computational complexity of your implementation.

Solution:

The Minkowski Sum algorithm we wrote assumes the polygon it receives is convex. Given convex obstacles, the C-space of our robot will look as follows:



If the polygon is not convex, then we can see in the following example that the computed C-space polygon is partially inside the workspace polygon:



The complexity for our Minkowski sum algorithm is $O(m+n)$ (where m, n are the number of vertices in each respective polygon of interest), given that both of the polygons are convex.

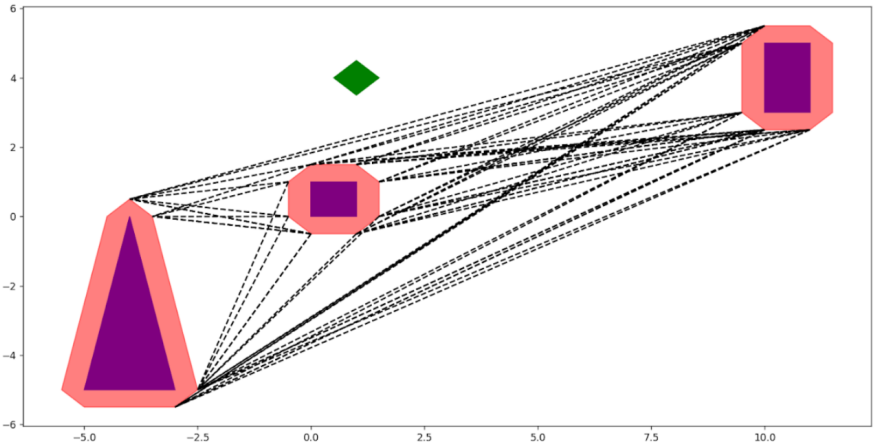
Part 2.2 - Building the Visibility Diagram

In the second step, we will build a visibility graph over the C-space we constructed in the first part of the preprocessing phase. **Visualize** the visibility graph and **describe** the computational complexity of your implementation.

Solution:
The visibility graph algorithm has two main steps:

1. Creates a list of all vertices in the graph - $O(n)$, where n is the number of vertices in the graph
2. Iterates over every pair of vertices in the list - $O(n^2)$
    - For every pair, check if the line created by the pair intersects any polygon: $O(n)$
    - If it doesn't intersect any polygon, add to the list of visibility graph lines: $O(1)$

Overall, the time complexity of our algorithm is $O(n + n^2 \cdot n) = O(n^3)$. Here we can see the visibility graph obtained for the original example:

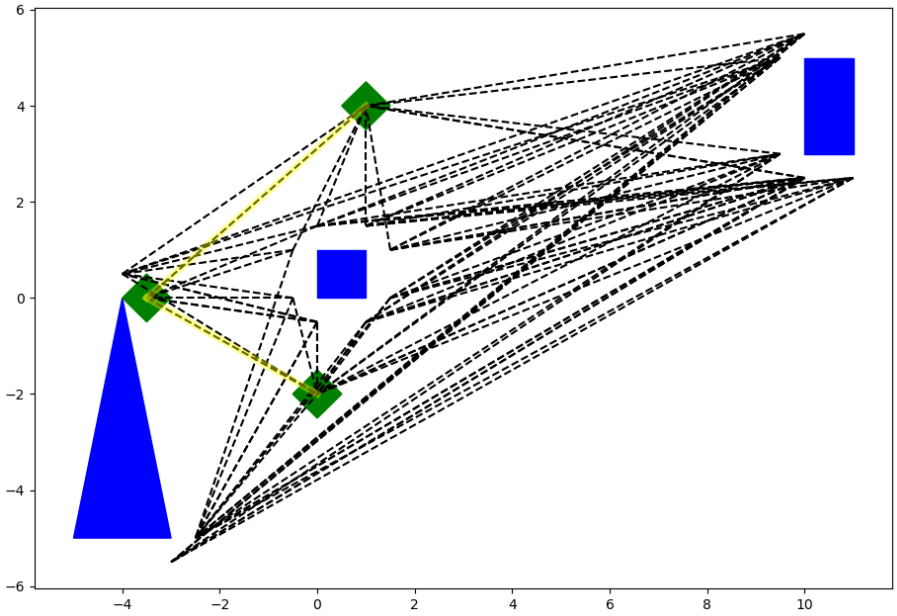


Part 2.3 - Computing Shortest Paths

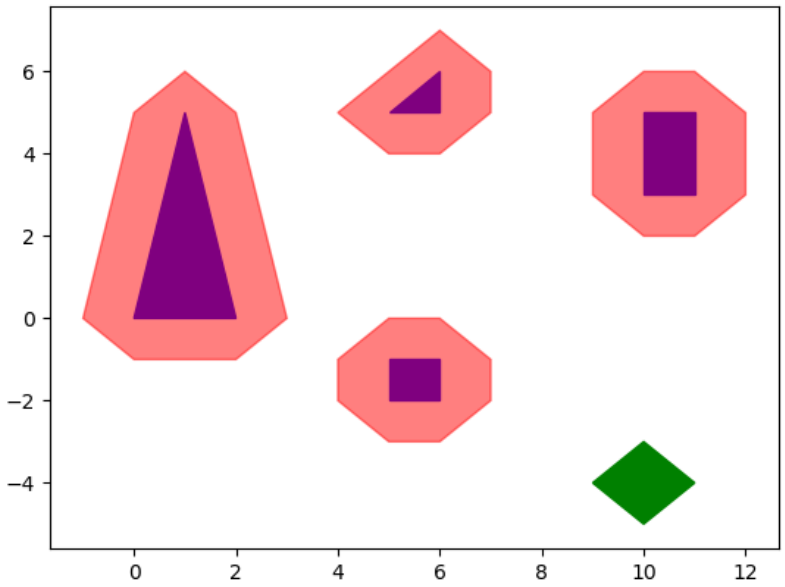
In the query phase we will need to update the visibility graph to include the query and implement Dijkstra's algorithm to compute a shortest path between the start and the goal. For each one of the queries, **visualize** the solution obtained and **describe** the computational complexity of your implementation.

Solution:

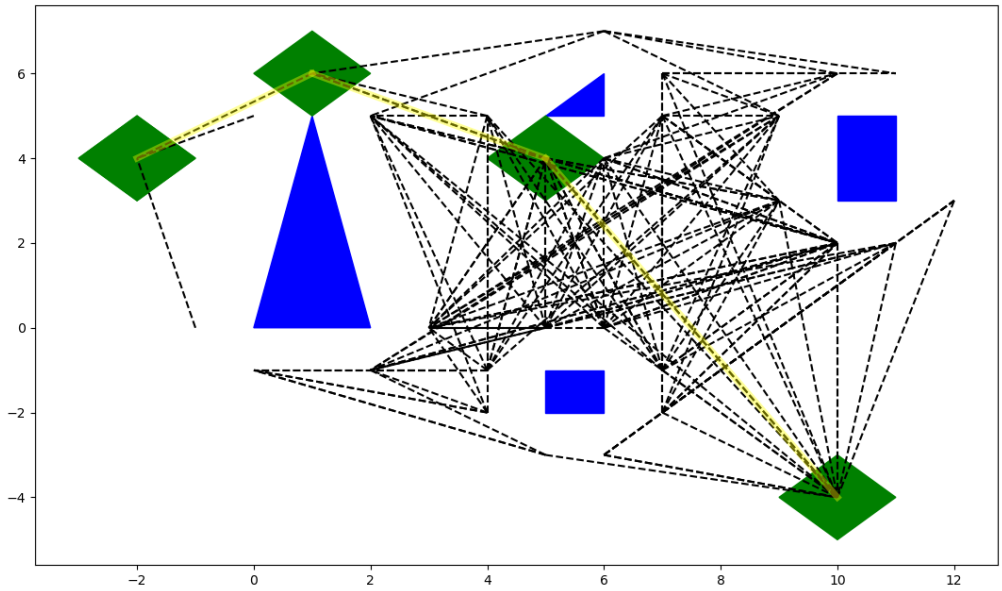
By implementing Dijkstra's graph search algorithm on our visibility graph, we obtain the following shortest path (colored yellow) for the original workspace:



We also created a custom workspace, whose C-space looks as follows:



and whose shortest path appears as follows (in the workspace):



Our implementation of the search component of the Dijkstra algorithm has a complexity of $O(|V|^2)$, due to the fact that we are using arrays to store the frontier, while the best known time complexity in the literature (which relies on Fibonacci heap min-priority queues) is $O(|E| + |V|log|V|)$. We note that we transformed the visibility graph into a more easily searchable graph for the Dijkstra graph using an algorithm of complexity $O(|V|^2|E|)$, which is quite inefficient. Thus, our total Dijkstra search process on a visibility graph is really of complexity $O(|V|^2|E|)$, which could've been reduced had we built our searchable graphs more intelligently.