IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 25, 2021, accepted December 6, 2021, date of publication December 10, 2021,
date of current version December 23, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3134760

Multi-Robot Workspace Division Based on Compact Polygon Decomposition

GEORGY SKOROBOGATOV, CRISTINA BARRADO, AND ESTHER SALAMÍ
Computer Architecture Department, Technical University of Catalonia, 08860 Castelldefels, Spain

Corresponding author: Georgy Skorobogatov (georgy.skorobogatov@upc.edu)

This work was supported by the Ministerio de Economía, Industria y Competitividad, and Gobierno de España under Award BES-2017-079798 and Award TRA2016-77012-R.

ABSTRACT In this work, we tackle the problem of multi-robot convex workspace division. We present an algorithm to split a convex area among several robots into the corresponding number of parts based on the area requirements for each part. The core idea of the algorithm is a sequence of divisions into pairs with the lowest possible perimeters. In this way, the compactness of the partitions obtained is maximized. The performance of the algorithm, as well as the quality of the obtained parts, are analyzed in comparison with two different algorithms. The presented approach yields better results in all metrics compared to other algorithms.

INDEX TERMS Multirobot systems, workspace division, polygon partition.

I. INTRODUCTION

Systems of multiple robots are used in a wide range of applications and can be a better choice compared to single robots. Using several robots can provide such advantages as time efficiency, ability to perform simultaneous actions at different locations, fault tolerance, and others. On the other hand, using several robots makes it more complex to operate them. One of the complexities that can arise in various applications when using several robots is area partition. The area partition problem in multi-robot systems is a problem of splitting a polygon defining an area into several parts, where each part is assigned to one robot.

There are several families of algorithms that perform the area partition. The most common way is where the area is discretized into a grid and then the cells of this grid are divided between the robots. In works like [9], [15], [18], [23], [5], and [36] the area of interest is rectangular with or without a set of obstacles made of rectangles coinciding with the cells of the rectangular grid. Similarly, in [34] an area was covered by footprints of sensors which were then grouped by an improved k-means clustering algorithm into several groups to delimit an initial area into sub-areas of equal sizes. In [8] and several other related works from the same authors, an area is split into a triangular mesh using constrained Delaunay triangulation with Steiner points. The obtained cells are then covered using various approaches such as flood fill algorithm, watershed algorithm, spanning trees, genetic algorithms, and others.

There are also several works like [25] and [32] that used the Voronoi diagram for splitting a polygon. A set of lattice points is constructed inside the given polygon and several clusters are found based on the number of parts the polygon has to be split into. A centroid point is found for each cluster. The resulting set of centroids is then used to construct the Voronoi diagrams. While this approach produces good quality results for multi-robot workspace partition, it is not possible to specify the areas of the resulting parts. In [24] authors proposed a weighted Voronoi diagram for a partition that takes into account the different capabilities of the robots. The areas proportions of the resulting parts will be closer to the given capabilities proportions but it will be still impossible to specify precisely the areas of resulting parts. Moreover, these algorithms were tested only on convex and close-to-convex polygons. It is not clear how they will perform on more complex polygons.

In [30] it was proposed to split an area into a set of stripes and divide them according to the area requirements for each robot. This approach, though, has the drawback of producing disconnected areas in many cases with complex polygons.

Finally, [20] proposed an algorithm that represents a polygon as a directed region-adjacency graph with nodes as its convex parts. The nodes of the graph are then processed recursively, and reorganized into parts according to the given

The associate editor coordinating the review of this manuscript and approving it for publication was Heng Wang.

area requirements. It was noted in several research works that the resulting parts can be of shapes that make them a poor choice for being covered by robots. In [35] authors improved the algorithm for convex polygon partition by post-processing each obtained part. The lines that divided the obtained sub-polygon from the rest of the polygon were moved until the two angles introduced by each line were as close as possible to 90 degrees. The algorithm was tested only on a single rectangle, so more tests are needed to understand if it works for more complex cases. In our previous work [33] the same algorithm for dividing non-convex polygons was implemented with several extensions. It was shown that performing the initial partition into convex parts has a big effect on the quality of the resulting partition. The larger parts were yielding better results than when the polygon was split into triangles. Later when we will present the results, we will use this algorithm for comparison of qualities of obtained results and performance.

The contribution of this paper is a completely novel mathematics-based approach able to find the optimum solution to the problem of partitioning a convex polygon into various number of parts. In contrast with the other partitioning algorithms that need to apply numerical solutions by iterating towards the optimum, we find this directly by solving the equations that maximize the compactness. The core idea of the algorithm is that it is possible to calculate analytically the best possible way to divide a polygon into two parts for a given area requirement in terms of the perimeter of the corresponding part. We will present a theory behind the algorithm, and compare the results with two different approaches, one of which is taken from the aforementioned paper [33] and the other one is based on discretizing the polygon border into a set of vertices and brute-force search for the most compact solution.

II. OUTLINE OF THE ALGORITHM

Let us say we have a convex polygon with a perimeter P and an area requirement R that is less than the area of the polygon. The boundary of the polygon is defined by an ordered set of vertices $V_{0..n}$ arranged in counterclockwise order. Let us also say that a random point T was chosen on the polygon's border. Then there can be only one point H such that when the polygon is split by a line TH, the sub-polygon on the right would have an area R (see Fig. 1). We will call points T "tails" and points H "heads". We will also name "countervertices" those points that when connected with any given point will result in two parts where one of them has area R. So, for any given point, there are always two countervertices, except for when the R equals the half of the polygon area. And T is a countervertex of H and vice versa.

The task is to find the location of points T and H so that the corresponding right part with the area R would be the most compact. We define the compactness of a polygon as the ratio of the square root of its area to its perimeter P. Since the area requirement is fixed, and the only variable part is the

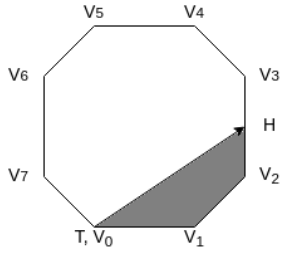


FIGURE 1. For a tail point T located on the border of the polygon, there can be only one head point H such that the area on the right of the line TH is equal to R.

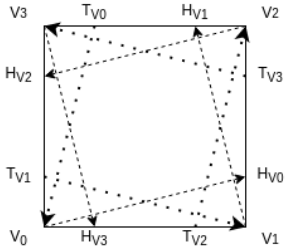


FIGURE 2. An example polygon defined by vertices $V_0 - V_3$. For an area requirement of R=12.5%, the image shows those heads (H_{V0}, H_{V1}, H_{V2}, H_{V3}) where the corresponding tails are original vertices and vice versa (T_{V0}, T_{V1}, T_{V2}, T_{V3}). Each directed line splits the polygon into two parts. Dashed lines connect the tails lying on the original vertices with their corresponding heads. Dotted lines connect the heads lying on the original vertices with their corresponding tails.

perimeter, the equation takes a form:

$$compactness = \frac{\sqrt{polygon.area}}{polygon.perimeter} = \frac{\sqrt{R}}{P(T)} \qquad (1)$$

where $P(T)$ is a function of the perimeter of the part on the right of the line TH. Therefore we can say that the problem of maximization of compactness is equivalent to a problem of minimization of a perimeter. As an example, the function $P(T)$ for the polygon depicted in Fig. 2 is shown in Fig. 3.

If we obtain this function, we can find its minima and the corresponding tail points T. These tail points will correspond to those lines TH that split the polygon in such a way that the parts on the right from these lines have the minimum perimeter.

In order to construct this function, two tasks should be solved:

1) Find the limits of the function domains: It is clear that $P(T)$ in a general case is a piecewise-smooth function (see, for example, Fig. 3). The domain of each constituent function is defined by those line positions where any of the endpoints T or H snaps from one segment to another. The task here is to find the limits of those domains. Our approach is presented in the next section.

2) Calculate the minimum of the perimeter function: For each part of this piecewise function, we then need to find locations of T and H that correspond to the part with the minimum perimeter. This can be done by calculating the derivative of a function $P(T)$ for the given domain.

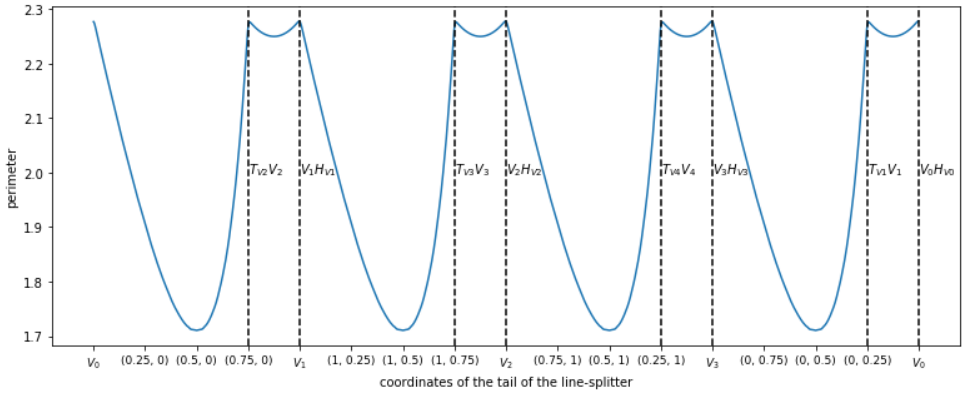


FIGURE 3. Function $P(T)$ for the polygon given in Fig. 2 assuming the length of each side to be equal to 1. Vertical dashed lines correspond to the locations of the lines TH containing the original polygon vertices. Each dashed line is annotated with the vertices of the corresponding line.

It is worth noting that the number of domains does not depend only on the number of vertices. One can imagine two cases with a square polygon and two area requirements: 12.5% and 50%. The case with 12.5% is depicted in Fig. 2 and the corresponding function in Fig. 3. One can see that, in total, there are eight domains. But for the case with the area requirement equal to 50%, the number of domains would be just four as the countervertices of the original vertices of the polygon would always point to the original vertices as well.

III. FINDING THE DOMAINS

Finding domains of $P(T)$ for a given area requirement R can be done in $O(n)$ time, where n is the number of polygon's vertices. For that, we can use linear equations for each segment of the polygon's border and the shoelace formula:

$$\begin{aligned} A &= \frac{1}{2}\left|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n\right| \\ &= \frac{1}{2}\,|x_1 y_2 + x_2 y_3 + \ldots + x_{n-1} y_n + x_n y_1 \\ &\quad - x_2 y_1 - x_3 y_2 - \ldots - x_n y_{n-1} - x_1 y_n| \end{aligned} \tag{2}$$

The algorithm for finding the domains of each part of the piecewise function goes as follows. First, we choose as the first tail T_0 one of the original vertices of the polygon, V_i. We iterate over vertices $V_{i+2}V_{i+3}\ldots$ in counterclockwise order until a vertex V_j is found such that the covered area $Area(V_iV_{i+1}\ldots V_j)$ is greater or equal than the area requirement R, i.e.:

$$Area(V_iV_{i+1}\ldots V_{j-1}) < R \leq Area(V_iV_{i+1}\ldots V_j) \tag{3}$$

In case the $Area(V_iV_{i+1}\ldots V_j)$ is equal to the area requirement R, then the head corresponding to T_0, H_0, is the last seen vertex V_j. Otherwise, we define the remaining area R' as the difference between the area covered in the last iteration and the given area requirement, i.e.:

$$R' = Area(V_iV_{i+1}\ldots V_j) - R \tag{4}$$

In such a case, the position of the head H_0 is located somewhere on the last seen segment $V_{j-1}V_j$ and we can find its exact coordinates using the shoelace formula.

Having the initial pair of tail and head vertices (T_0, H_0), we can proceed to locate the next tail T_1 that will delimit the first domain $[T_0, T_1]$ or the next head vertex H_1. Which one of them will be discovered first depends on the relative location of the vertices and the area requirement. The next tail T_1 will be searched on the segment V_iV_{i+1} excluding the V_i as it is already taken by T_0. Likewise, the next head H_1 may be located somewhere on the segment H_0V_{j+1} excluding H_0.

In order to find the locations of the next head H_1 and/or the next tail T_1, we can compare the areas of the triangles $T_0V_{i+1}H_0$ and $V_{i+1}H_0V_j$ as shown in Fig. 4. Three situations can arise:

1) $Area(T_0V_{i+1}H_0) = Area(V_{i+1}H_0V_j)$. In this case, the segment (V_{i+1}, V_j) is already a line that makes a right sub-polygon with the requested area. This is, T_1 will be located at V_{i+1} and the corresponding head point H_1 will be located at V_j.
2) $Area(T_0V_{i+1}H_0) > Area(V_{i+1}H_0V_j)$. In this case, the next head vertex H_1 will be located at V_j and we have to find T_1 somewhere on the segment T_0V_{i+1} by using the shoelace formula and the linear equation for this segment.
3) $Area(T_0V_{i+1}H_0) < Area(V_{i+1}H_0V_j)$. In this case, the next tail vertex T_1 will be located at V_{i+1} and its corresponding head vertex H_1 will be located on the segment H_0V_j and can be found by using the shoelace formula and the linear equation for this segment.

Having obtained the first domain T_0T_1 and its corresponding "countersegment" H_0H_1, we can now repeat the same procedure for the pair (T_1, H_1) and obtain the next domain T_1T_2 and its corresponding countersegment H_1H_2. This process repeats until all the perimeter is covered. While iterating in this manner over the polygon's vertices, we will also get those fixed parts of the polygon between the endpoint of the domain and the first point of the countersegment. So, for example, for a domain T_kT_{k+1} and its correspond-

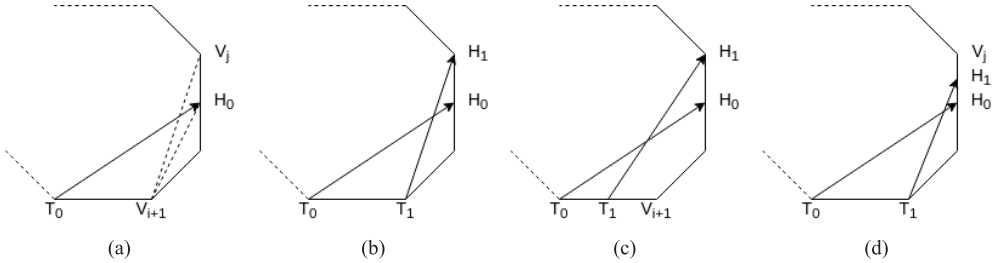


FIGURE 4. a) T_0H_0 is the initial line-splitter where $T_0 = V_i$ is some vertex on the polygon border, V_{i+1} is the next vertex that follows after T_0, and V_j is the next vertex that follows after H_0. Comparing the areas of triangles $T_0V_{i+1}H_0$ and $V_{i+1}H_0V_j$ can tell us about the location of the next tail point T_1 delimiting the current domain and its corresponding head point H_1. b) In the case when both triangles have equal areas, the next tail and head points are V_{i+1} and V_j respectively. c) If the area of $T_0V_{i+1}H_0$ is less than the area of $V_{i+1}H_0V_j$, T_1 will be found on segment T_0V_{i+1} and $H_1 == V_j$. d) If the area of $T_0V_{i+1}H_0$ is greater than the area of $V_{i+1}H_0V_j$, H_1 will be found on segment H_0V_j and $T_1 == V_{i+1}$.

ing countersegment H_kH_{k+1}, this constant area part will be delimited by vertices $T_{k+1}..H_k$ which can be empty or not. These polygon parts will be later joined with "flexible" parts, triangles, or quadrilaterals, built on the domain segments and countersegments.

The pseudocode of the algorithm is shown in Alg. 1. A part of the algorithm responsible for finding the initial partition is written out separately in Alg. 2. The function *to_segments* returns an iterator over segments built on pairs of input vertices. The *shoelace* function takes four parameters: an area requirement and three fixed points forming a triangle. It returns a point found on the segment built on the last two points such that the area of a new triangle built on the first point, the second point, and the newly found point will be equal to the area requirement. Finally, the *remove_collinear* function takes a list of points, checks if the last three points are collinear and removes the middle one if this is the case.

IV. FINDING MINIMUM FOR EACH DOMAIN

For each domain, we will have a situation like the one presented in Fig. 5. Let us denote the domain segment now as S_iS_{i+1} and its countersegment as E_iE_{i+1}. We search the position of the splitter TH with the tail T between the endpoints of the domain S_iS_{i+1} and the head H between the endpoints of the corresponding countersegment E_iE_{i+1} such that the polygon $TS_{i+1}E_iH$ with area R^* is the most compact. R^* is the difference between the area requirement R and the area of the grey region R_F. The grey region is fixed as well as $S_{i+1}E_i$. It is also possible that the grey area will be empty.

The following lemma will help later demonstrate that finding the minimum of the perimeter of a polygon can be done by parts.

Lemma: Given two functions $g(x)$ *and* $h(x)$ *with* $argmin(g(x)) = argmin(h(x)) = M$, *if* $f(x) = g(x) + h(x)$ *then* $argmin(f(x)) = M$.

Proof: For all x: $g(x) \geq g(M) \Leftrightarrow g(x) - g(M) \geq 0$ and $h(x) \geq h(M) \Leftrightarrow h(x) - h(M) \geq 0$. And since a sum of nonnegative functions is nonnegative $\Rightarrow (g(x) - g(M)) + (h(x) - h(M)) \geq 0$ which gives $g(x) + h(x) \geq g(M) + h(M)$ and, finally, $f(x) \geq f(M)$ which corresponds to the definition of minimum. ■

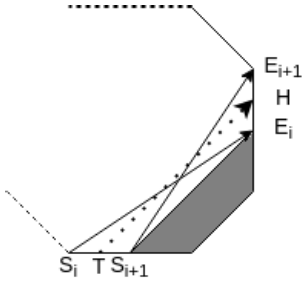


FIGURE 5. We search for the position of the line-splitter TH where the tail point T is located on the segment S_iS_{i+1} and the head point H is located on the segment E_iE_{i+1} so that the polygon $TS_{i+1}E_iH$ with area R^* is the most compact. R^* is the difference of the area requirement R with the area of the grey region R_F. Grey region is fixed as well as $S_{i+1}E_i$ for each domain.

In the following theorem, without loss of generality, we will rotate and translate the polygon such that the origin of the coordinates is fixed at the intersection of the two rays on which S_iS_{i+1} and E_iE_{i+1} are lying. Then, in the following subsections, the specific formulae will be given for the general case and for some specific cases where the formulae are undetermined.

Theorem: Minimizing the perimeter of a quadrilateral with a fixed area, one fixed side, and fixed nonparallel rays on which the adjacent sides are located is equivalent to minimizing the total length of the adjacent sides or minimizing the length of the opposite side.

Proof: Let us consider the example given in Fig. 5. We are searching for the location of points T and H so that the quadrilateral $TS_{i+1}E_iH$ has the smallest perimeter for a given area R^*. The side $S_{i+1}E_i$ is fixed as well as the rays on which the segments TS_{i+1} and E_iH will be built. Let us rotate and align the figure so that the intersection of rays based on the segments S_iS_{i+1} and E_iE_{i+1} would be in $(0, 0)$ and E_iE_{i+1} would be aligned with X-axis. See Fig. 6. Let us also rename the vertices S_{i+1} and E_i to S and E respectively to simplify the following equations.

From the shoelace formula (2) for the given case we have:

$$2R = T_xS_y + H_xT_y - S_xT_y - E_xS_y \tag{5}$$

Algorithm 1: *To_Domains*

Data: V – list of n vertices of polygon's border, R – area requirement
Result: iterator over domain segments D, their corresponding countersegments C, areas to find inside the quadrilaterals based on these segments, and "right" and "left" vertices V_R and V_L, where the "right" vertices are all the polygon border vertices located between the domain and its countersegment when iterating in counterclockwise order, and the "left" vertices are all the remaining vertices in counterclockwise order

```
// The following two are iterators;
tail_segments = to_segments(V_{0..n} + [V_0]);
head_segments = to_segments(V_{1..n} + V_{0..n});
D = next(tail_segments);
// See Alg. 2 for details;
V_R, V_L = initial_split(heads, V, D, R);
C = Segment(V_Rm, V_L0) // m - length of V_R;
while D is not None do
    // tail-based triangle;
    TT = Polygon(D.start, D.end, C.start);
    // head-based triangle;
    TH = Polygon(D.end, C.start, C.end);
    V_R.popleft();
    V_L.append(D.start);
    if TT.area < TH.area then
        C.end = shoelace(TT.area, D.end, C.start,
          C.end);
        V_L.prepend(C.end);
        head_segments.prepend(Segment(V_L0, V_L1));
    else if TT.area > TH.area then
        ΔA = TT.area - TH.area;
        D.end = shoelace(ΔA, C.end, D.end, D.start);
        V_R.prepend(D.end);
        TT = Polygon(D.start, D.end, C.start);
        tail_segments.prepend(Segment(V_R0, V_R1));
    yield D, C, TT.area, V_R, V_L;
    V_R.popleft();
    V_R.append(C.end);
    // if 3 last points collinear, remove the middle one;
    remove_collinear(V_R);
    V_L.popleft();
    V_L.append(D.end);
    remove_collinear(V_L);
    D = next(tail_segments, on_exhaustion=None);
    C = next(head_segments);
end
```

Algorithm 2: *Initial_Split*

Data: *head_segments* – iterator over "head" segments, V – vertices of the polygon's border ordered counterclockwise, D – initial tail segment, R – area requirement
Result: V_R – a list of "right" vertices, V_L – a list of "left" vertices

```
V_R = [] // right vertices;
V_L = V_{1..n} + [V_0] // left vertices;
A = 0 // accumulated area;
for C in head_segments do
    V_R.append(C.start);
    V_L.popleft();
    dA = Polygon(D.start, C.start, C.end).area;
    A = A + dA;
    if A < R then
        continue
    else if A == R then
        V_R.append(C.end);
        V_L.popleft();
        C = next(head_segments);
    else
        δA = dA + R − A;
        C.start = shoelace(δA, D.start, C.start, C.end);
        V_R.append(C.start);
    end
    return V_R, V_L;
end
```

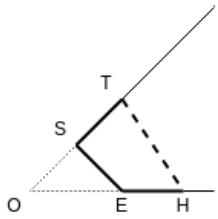


FIGURE 6. A part of the polygon. We search for the position of the line-splitter *TH* such that the quadrilateral *TSEH* with an area equal to a given area requirement is the most compact.

Let $T_y = kT_x$ and $S_y = kS_x$. Then:

$$H_x = \frac{\frac{2R}{k} + E_x S_x}{T_x} \tag{6}$$

We define A as $\frac{2R}{k} + E_x S_x$ to simplify the following equations:

$$H_x = \frac{A}{T_x} \tag{7}$$

Let us analyze perimeters of $TS + EH = P_1$ and $TH = P_2$ separately. We assume that if both minimal solutions are equal, then this solution will correspond to the minimum of the full perimeter:

$$P = |SE| + P_1 + P_2 \tag{8}$$

$$P_1 = |TS| + |EH| = \sqrt{(T_x - S_x)^2 + (T_y - S_y)^2} + H_x - E_x = (T_x - S_x)\sqrt{1 + k^2} + \frac{A}{T_x} - E_x \tag{9}$$

$$P_2 = |TH| = \sqrt{T_y^2 + (T_x - H_x)^2} = \sqrt{k^2 T_x^2 + (T_x - \frac{A}{T_x})^2} \quad (10)$$

Setting the derivative of P_1 equal to zero:

$$\frac{dP_1}{dT_x} = \sqrt{1+k^2} - \frac{A}{T_x^2} = 0 \quad (11)$$

gives the following T_x coordinate:

$$T_x = \pm\sqrt{\frac{A}{\sqrt{1+k^2}}} \quad (12)$$

For P_2:

$$\frac{dP_2}{dT_x} = \frac{2(T_x - \frac{A}{T_x})(1 + \frac{A}{T_x^2}) + 2k^2 T_x}{2\sqrt{k^2 T_x^2 + (T_x - \frac{A}{T_x})^2}} = 0 \quad (13)$$

$$T_x - \frac{A^2}{T_x^3} + k^2 T_x = 0 \quad (14)$$

$$T_x^4(1+k^2) = A^2 \quad (15)$$

we will get the same values for T_x as in Eq. (12). We are interested in the positive solution only as in our case $T_x \geq S_x \geq 0$.

Both derivatives of P_1 and P_2 are negative for T_x less than the obtained solution and positive for T_x greater than the obtained solution, meaning that $argmin(P_1) = argmin(P_2)$. And as it was proved in the lemma above, it means that $argmin(P) = argmin(P_1) = argmin(P_2)$. So we can say that minimizing all the perimeter is equivalent to minimizing either of these two parts, P_1 or P_2.

Note that for the case with the negative slope of the ray based on the vertices S_i and S_{i+1} the equation will stay exactly the same. ■

A. GENERAL SOLUTION

We have shown that minimizing the perimeter of the quadrilateral in our task is equivalent to minimizing just a part of the perimeter. We have used a simplified case where the area was rotated and translated so that we could lose some terms in the equations. Let us now solve the same task for a general case.

The shoelace formula (2) gives:

$$2R = T_x S_y + S_x E_y + E_x H_y + H_x T_y - S_x T_y - E_x S_y - H_x E_y - T_x H_y \quad (16)$$

Let $T_y = k_T T_x + m_T$ and $H_y = k_H H_x + m_H$:

$$H_x = (2R + E_x S_y + m_H T_x + S_x(k_T T_x + m_T) - T_x S_y - S_x E_y - m_H E_x) / (T_x(k_T - k_H) + k_H E_x + m_T - E_y) \quad (17)$$

Since $S_y = k_T S_x + m_T$ and $E_y = k_H E_x + m_H$:

$$H_x = (2R + E_x S_x(k_T - k_H) + (E_x + S_x - T_x)(m_T - m_H)) / (T_x(k_T - k_H) + (m_T - m_H)) \quad (18)$$

Let $\Delta k = k_T - k_H$ and $\Delta m = m_T - m_H$:

$$H_x = \frac{2R + E_x S_x \Delta k + (E_x + S_x - T_x)\Delta m}{T_x \Delta k + \Delta m} \quad (19)$$

Let $H = 2R + E_x S_x \Delta k + (E_x + S_x)\Delta m$:

$$H_x = \frac{H - T_x \Delta m}{T_x \Delta k + \Delta m} \quad (20)$$

Since minimizing perimeter is equivalent to minimizing the sum length of segments TS and EH, let us consider the part of the perimeter containing these two segments:

$$P = \sqrt{(T_x - S_x)^2 + (T_y - S_y)^2} + \sqrt{(H_x - E_x)^2 + (H_y - E_y)^2} \quad (21)$$

Substituting the ordinate coordinates gives:

$$P = |T_x - S_x|\sqrt{1 + k_T^2} + |H_x - E_x|\sqrt{1 + k_H^2} \quad (22)$$

Since $\frac{dP}{dT_x} = 0$:

$$\sqrt{1 + k_T^2} \pm \frac{dH_x}{dT_x}\sqrt{1 + k_H^2} = 0 \quad (23)$$

From the Eq. (20) we get:

$$\frac{dH_x}{dT_x} = \frac{-\Delta m(T_x \Delta k + \Delta m) - \Delta k(H - T_x \Delta m)}{(T_x \Delta k + \Delta m)^2} = -\frac{\Delta m^2 + \Delta k H}{(T_x \Delta k + \Delta m)^2} \quad (24)$$

And combining the Eq.(23)) and Eq.(24):

$$(T_x \Delta k + \Delta m)^2 = |\Delta m^2 + \Delta k H|\sqrt{\frac{1 + k_H^2}{1 + k_T^2}} \quad (25)$$

And in order to choose the sign, we can replace T_x with S_x in this equation since they lie on the same side of the ray relative to the intersection point of both rays:

$$k_T S_x - k_H S_x + m_T - m_H * \pm\sqrt{A} \quad (26)$$

Hence, if $k_T S_x + m_T > k_H S_x + m_H$ we choose the plus sign, and vice versa.

B. DOMAIN SEGMENT PARALLEL TO COUNTERSEGMENT

Still, a particular case exists when the domain segment is parallel to its countersegment. Then we can substitute some terms in the Eq. (16) with $T_y = kT_x + m_T$, $S_y = kS_x + m_T$, $H_y = kH_x + m_H$, $E_y = kE_x + m_H$ which will give:

$$H_x = \frac{2R}{m_T - m_H} + S_x + E_x - T_x \quad (27)$$

The perimeter of the quadrilateral containing the SE, ST, EH, and TH segments is as follows:

$$P = |SE| + \sqrt{1 + k^2}|(T_x - S_x) + (H_x - E_x)| + \sqrt{(T_x - H_x)^2 + (kT_x - kH_x + m_T - m_H)^2} \quad (28)$$

To calculate the location of T_x for the most compact partition, we calculate the derivative:

$$\frac{dP}{dT_x} = \frac{2(T_x - H_x) + 2k(kT_x - kH_x + m_T - m_H)}{\sqrt{(T_x - H_x)^2 + (kT_x - kH_x + m_T - m_H)^2}} = 0 \tag{29}$$

which gives:

$$(T_x - H_x)(1 + k^2) + k(m_T - m_H) = 0 \tag{30}$$

and therefore:

$$T_x = \frac{R}{m_T - m_H} + \frac{S_x + E_x}{2} - \frac{k(m_T - m_H)}{2(1 + k^2)} \tag{31}$$

$$H_x = \frac{R}{m_T - m_H} + \frac{S_x + E_x}{2} + \frac{k(m_T - m_H)}{2(1 + k^2)} \tag{32}$$

C. BOTH DOMAIN SEGMENT AND COUNTERSEGMENT ARE PARALLEL TO Y-AXIS

In case if both the domain segment and its countersegment are parallel to Y-axis, the area of the quadrilateral is calculated as:

$$R = (S_x - E_x)\frac{T_y - S_y + H_y - E_y}{2} \tag{33}$$

Therefore:

$$H_y = \frac{2R}{S_x - E_x} + S_y + E_y - T_y \tag{34}$$

The perimeter of the quadrilateral is calculated as:

$$P = |SE| + |(T_y - S_y) + (H_y - E_y)| + \sqrt{(E_x - S_x)^2 + (H_y - T_y)^2} \tag{35}$$

And to get the location of both the tail point and the head point that correspond to the most compact area, we calculate the derivative:

$$\frac{dP}{dT_y} = \frac{-2(H_y - T_y)}{\sqrt{(E_x - S_x)^2 + (H_y - T_y)^2}} = 0 \tag{36}$$

which gives us:

$$T_y = \frac{R}{S_x - E_x} + \frac{S_y + E_y}{2} = H_y \tag{37}$$

D. DOMAIN SEGMENT PARALLEL TO Y-AXIS

In case if it is only the domain segment that is parallel to Y-axis, we can take the Eq. (16) and perform substitution $H_y = kH_x + m$ and $E_y = kE_x + m$. And as $T_x = S_x$, we will get:

$$H_x = \frac{2R + E_x S_y - kS_x E_x - mE_x - S_x S_y + S_x T_y}{-kS_x - m + T_y} \tag{38}$$

Let $B = kS_x + m$ and $A = 2R - BE_x + S_y(E_x - S_y)$ then the previous equation can be rewritten as:

$$H_x = \frac{A + S_x T_y}{T_y - B} \tag{39}$$

Let us analyze only the part of the perimeter consisting of TS and EH:

$$P = |T_y - S_y| + |H_x - E_x|\sqrt{1 + k^2} \tag{40}$$

To get the locations of the tail and head points corresponding to the most compact partition, we calculate the derivative of the perimeter function:

$$\frac{dH_x}{dT_y} = -\frac{A + BS_x}{(T_y - B)^2} \tag{41}$$

$$\frac{dP}{dT_y} = \pm 1 - \frac{(A + BS_x)\sqrt{1 + k^2}}{(T_y - B)^2} = 0 \tag{42}$$

which gives:

$$T_y = B \pm \sqrt{|A + BS_x|\sqrt{1 + k^2}} \tag{43}$$

And the sign is chosen according to the location of S with respect to the intersection point B of the rays. If $S_y > B$ then we choose the positive solution and vice versa.

E. COUNTERSEGMENT PARALLEL TO Y-AXIS

In case if it is only the countersegment that is parallel to Y-axis, we take the Eq. (16) and substitute $S_y = kS_x + m$ and $T_y = kT_x + m$. And since $H_x = E_x$:

$$H_y = (2R + mS_x + kS_x E_x + E_x E_y - S_x E_y - kE_x T_x - mT_x)/(E_x - T_x) \tag{44}$$

Let $A = 2R + mS_x + kS_x E_x + E_x E_y - S_x E_y$ and $B = -kE_x - m$, then the equation above can be rewritten like:

$$H_y = \frac{A + BT_x}{E_x - T_x} \tag{45}$$

Next, we analyze the part of the perimeter consisting of TS and EH:

$$P = |H_y - E_y| + |T_x - S_x|\sqrt{1 + k^2} \tag{46}$$

By taking its derivative we can find the location of the tail corresponding to the most compact area:

$$\frac{dH_y}{dT_x} = \frac{A + BE_x}{(E_x - T_x)^2} \tag{47}$$

$$\frac{dP}{dT_x} = \pm\frac{A + BE_x}{(E_x - T_x)^2} \pm \sqrt{1 + k^2} = 0 \tag{48}$$

$$T_x = E_x \pm \sqrt{\frac{|A + BE_x|}{\sqrt{1 + k^2}}} \tag{49}$$

And the sign is chosen according to if T is on the left or the right side of E.

Finally, finding the most compact part out of all obtained parts is trivial. The algorithm is shown in Alg.3. The function *to_splitter* represents the algorithm discussed in this section for finding a minimum perimeter for each domain including special cases. It takes the domain segment, its corresponding countersegment, and an area requirement, and returns a segment connecting domain with the countersegment in such a way that the enclosed area has the smallest perimeter. The *add_splitter_vertices* is needed to include the endpoints of the splitter segment to the "right" and "left" vertices depending on if the endpoints coincide with vertices of the polygon border or not.

And to split the polygon into multiple parts we simply split the polygon by going over a list of area requirements following the order specified by the user and detaching the most compact areas one by one. This is, of course, not the most optimal solution. But the problem of finding the best order of splitting is complex and, hence, we will not go into details on how to solve it in this paper.

Algorithm 3: *Compact_Split*

Data: V – vertices of the polygon's border ordered counterclockwise, R – area requirement
Result: V_R – a list of vertices defining a contour of a part of the input polygon with the area equal to the given area requirement, V_L – a list of vertices defining a contour of remaining part of the polygon

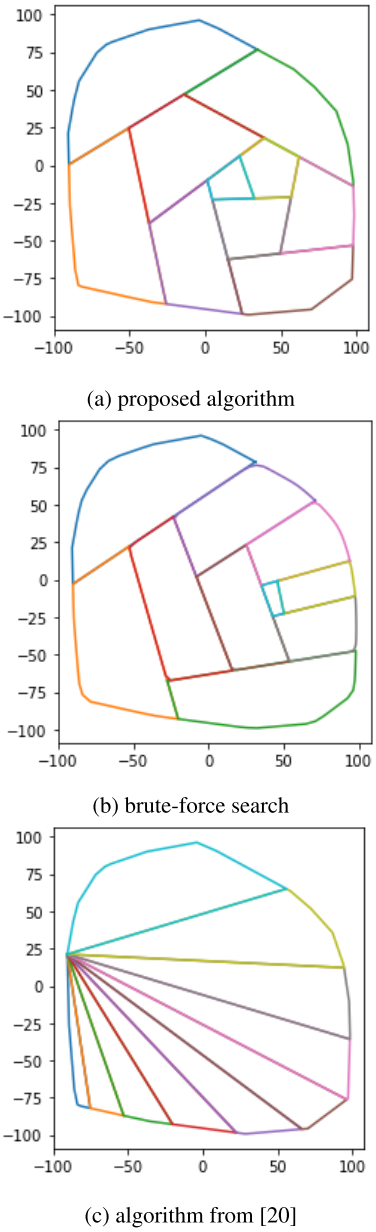
$P_{min} = \infty$ // shortest perimeter;
$C_R =$ None // shortest contour of a part with area R;
$C_L =$ None // corresponding remaining contour;
// See Alg. 2 for details on *to_domains* function;
for D, C, R^*, V_R, V_L **in** *to_domains*(V, R) **do**
  // find segment giving the most compact partition;
  $S =$ to_splitter(D, C, R^*);
  // add endpoints of S to V_R and V_L if necessary;
  $V_R, V_L =$ add_splitter_vertices(V_R, V_L, S, D, C);
  if *length*$(V_R) < P_{min}$ **then**
    $P_{min} =$ length(V_R);
    $C_R = V_R$;
    $C_L = V_L$;
  return V_R, V_L;
end

V. RESULTS

In this section, we show the quality of the results obtained with our algorithm compared against two different approaches. The first approach is the algorithm from [20] implemented in [33]. The second approach is a naive approach. The polygon border is discretized into a given number of vertices with a fixed distance between the vertices along the border. A brute-force search is performed over these vertices in order to find the most compact partition. We also show how the performance of our algorithm compares with these two approaches.

A. QUALITY

Let us see how the partition of a polygon into ten parts looks like for a different order of splitting. Fig. 7 shows the partition of the same polygon into ten parts using different approaches. In this case, the area requirements were ordered from the largest to the smallest and relatively scaled as $(10, 9, 8, \ldots, 1)$. We choose this order for demonstration purposes only. With any other order of area requirements or their values, the resulting partitions for each approach will be similar in terms of quality. And, as was mentioned previously, the problem of finding the best order of splitting is complex, and we leave it out of our work.



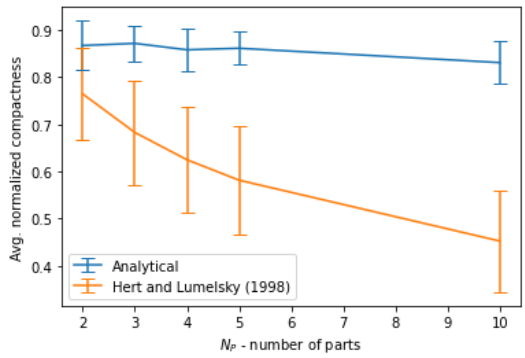
(a) proposed algorithm

(b) brute-force search

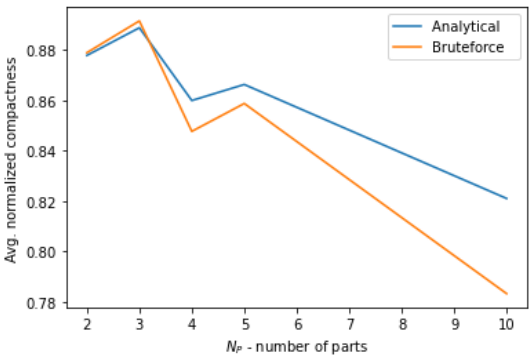
(c) algorithm from [20]

FIGURE 7. Comparison of polygon partition using three different approaches.

It can be seen in Fig. 7a and Fig. 7b that both the proposed algorithm results and the approach with brute-force search result in a much better partition than Fig. 7c built by the [20] algorithm. It is clear that the resulting areas in Fig. 7c are far from being compact. It can also be seen that in Fig. 7b there are some dents in the outer boundary of the polygon due to its discretization. In some cases this can be undesirable, and the effect will be more visible for smaller numbers of vertices the border is discretized into. It is also important to note that with the discretization it is impossible to have the exact partition, so the areas slightly differ from the initial area requirements.

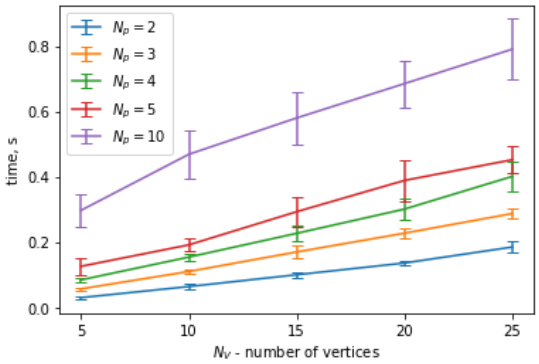


(a) proposed algorithm vs. algorithm from [20]

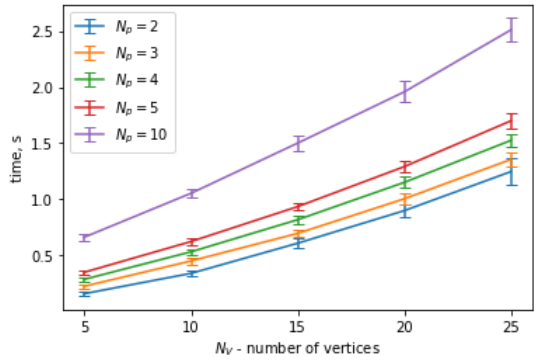


(b) proposed algorithm vs. brute force

FIGURE 8. Comparison of normalized compactness for three different approaches.



(a) proposed algorithm



(b) algorithm from [20]

FIGURE 9. Comparison of performance for the proposed algorithm and the algorithm from [20].

For a more general comparison, Fig. 8 shows comparisons of averaged normalized compactness for all three approaches. Normalizing compactness was done in order to get the same values for polygons with different areas but the same shapes, so that, for example, a square with an area R and a square with area $2R$ would have the same value of compactness. We normalized the values so that the maximum possible value would be 1. And the average value was taken then over all the resulting parts of the partition.

In Fig. 8a comparison of the compactness of the proposed algorithm with the algorithm of [20] is shown. Statistics over 100 random polygons are shown with a number of vertices ranging from five to 100. The order of splitting is chosen the same as before, starting from the largest part and ending with the smallest, where area requirements are relatively scaled as, for example (4, 3, 2, 1) for four parts. One can see that the quality of the resulting partition obtained by the [20] algorithm is significantly worse for this setup. We can also observe the quality of the obtained result degrades with the number of the parts, and more so for the [20] algorithm.

In Fig. 8b comparison of the compactness of the proposed algorithm with the brute-force approach is shown. Here we used only one polygon, the same as in the Fig. 7, as the brute-force search is very slow even for splitting into two or three parts. The partition into two, three, four, five, and ten parts were calculated. The area requirements are ordered and scaled in the same way as was explained in the description for Fig. 8a. For the brute-force approach the polygon was discretized into 100 vertices. One can see that the brute-force approach results in a bit worse results than the proposed here approach. For a split into three parts, one can notice, though, that the brute force yields a better result. This can be explained by the fact that it returns the parts with areas that slightly differ from the area requirements, and that due to discretization, there can be some discrepancies in the shape close to the original polygon vertices.

B. PERFORMANCE

Fig. 9 shows comparisons of the time to calculate the partition based on the number of vertices defining a polygon and the number of parts the polygon is being split into. The area requirements for each case were chosen equal. The statistics on 100 random polygons were collected for our algorithm and the algorithm of [20].

In Fig. 9a one can see the times to calculate the partitions using our algorithm. The times scale linearly depending on the number of vertices. We must note that after each iteration of finding the most compact area for a given area requirement, the coordinates of the part remaining for splitting were limited to 100 significant digits. If this was not done, the time to calculate the consequent parts could grow significantly due to the increase of significant digits after each split. So, for example, after the first iteration, if the remaining part contained a coordinate with 100 significant digits, the next

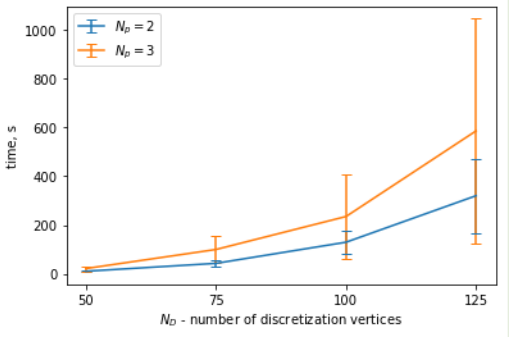


FIGURE 10. Performance of brute-force approach depending on the number of vertices the polygon border discretized into.

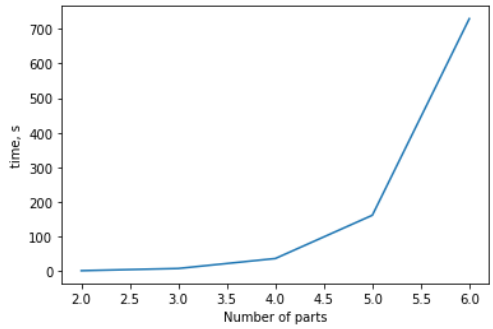


FIGURE 11. Performance of brute-force search for the most optimal order of splitting depending on the number of parts.

iteration could yield a part with a point that had ten thousand significant digits, and millions of significant digits on the next step.

Next, we also check the performance of our implementation of the algorithm of [20] implemented in [33]. Fig. 9b shows that it also linearly depends on the number of vertices the polygon is built on. But the algorithm spends around three times more time on the same polygons. This comparison, though, is not entirely fair since the calculations in that implementation are precise, while in the current paper we drop precision after each split.

Finally, we were also interested in how our approach scales compared to the approach with brute-force search over a discretized polygon border. In Fig. 10 one can see the statistics for five random polygons. We note that it does not matter how many vertices are contained in the border of a polygon. The performance depends only on the number of vertices the polygon border is discretized into. As it can be seen in the figure, the performance with this approach is much worse, even for the cases when the polygon is split into only two or three parts. This makes it barely usable in any real-life scenarios.

VI. CONCLUSION

In this work, we present a novel algorithm to split a convex polygon into a given number of parts depending on a list of area requirements for each part. This algorithm is based on the most compact partition of a polygon into two parts. The performance and the quality of the obtained results were compared against an algorithm described in [20] and [33], and against an approach with brute-force search over a discretized polygon border. It was shown that the algorithm presented in this paper is more efficient and the resulting parts show much better quality in terms of compactness.

For future work, it will be necessary to adopt this algorithm for partitioning any non-convex polygons with or without holes. In order to do that, a similar approach can be used as in [20]. We have already implemented that algorithm and it is described in detail in [33]. The idea will be the same – to represent a non-convex polygon as a directed region-adjacency graph and process it node by node.

Then, there is a problem of choice of the order of the area requirements. We saw that when the order is chosen as from the largest to the smallest, the partition has better quality than if the order was chosen from the smallest to the largest. But it is not clear what order will be actually the best in terms of compactness of the returned parts. What is clear is that it will be highly inefficient to search for the most optimal order of splitting. The total number of all possible ways to split a polygon into N parts when we split it into two on each step is equal to $\frac{(2n-2)!}{(n-1)!(n-2)!}$. This can be found using the Catalan numbers for the total number of ways to arrange an array into a binary tree and a total number of permutations of an array, where array is an array of area requirements. In this way, for two parts we can have only two possible ways to split a polygon, for three parts – 12 possible ways, for four parts – 60, for five parts – 250, for six parts – 1260, and so on. Fig. 11 shows that using the proposed algorithm if we search for the most optimal way to split polygon into just six parts, it will take more than ten minutes to finish the calculations which is not practical.

Finally, currently, our algorithm uses the compactness of parts on the right from the splitter to find the best partition. It is possible that taking into account the compactness of the left parts can improve the results. More experiments are needed in this regard.

REFERENCES

[1] J. J. Acevedo, B. C. Arrue, I. Maza, and A. Ollero, "A distributed algorithm for area partitioning in grid-shape and vector-shape configurations with multiple aerial robots," *J. Intell. Robotic Syst.*, vol. 84, nos. 1–4, pp. 543–557, Dec. 2016.

[2] J. J. Acevedo, I. Maza, A. Ollero, and B. C. Arrue, "An efficient distributed area division method for cooperative monitoring applications with multiple UAVs," *Sensors*, vol. 20, no. 12, p. 3448, Jun. 2020.

[3] A. Agarwal, L. M. Hiot, E. M. Joo, and N. T. Nghia, "Rectilinear workspace partitioning for parallel coverage using multiple unmanned aerial vehicles," *Adv. Robot.*, vol. 21, nos. 1–2, pp. 105–120, Jan. 2007.

[4] R. Almadhoun, T. Taha, L. Seneviratne, and Y. Zweiri, "A survey on multi-robot coverage path planning for model reconstruction and mapping," *Social Netw. Appl. Sci.*, vol. 1, no. 8, pp. 1–24, Aug. 2019.

[5] S. Ann, Y. Kim, and J. Ahn, "Area allocation algorithm for multiple UAVs area coverage based on clustering and graph method," *IFAC-PapersOnLine*, vol. 48, no. 9, pp. 204–209, 2015.

[6] F. Balampanis, I. Maza, and A. Ollero, "Area decomposition, partition and coverage with multiple remotely piloted aircraft systems operating in coastal regions," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, Jun. 2016, pp. 275–283.
[7] F. Balampanis, I. Maza, and A. Ollero, "Area partition for coastal regions with multiple UAS," *J. Intell. Robotic Syst.*, vol. 88, nos. 2–4, pp. 751–766, Dec. 2017.
[8] F. Balampanis, I. Maza, and A. Ollero, "Coastal areas division and coverage with multiple UAVs for remote sensing," *Sensors*, vol. 17, no. 4, p. 808, Apr. 2017.
[9] A. Barrientos, J. Colorado, J. Cerro, A. Martinez, C. Rossi, and and D. J. Sanz, "Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots," *J. Field Robot.*, vol. 28, no. 5, pp. 667–689, 2011.
[10] H. Bast and S. Hert, *The Area Partitioning Problem*. Princeton, NJ, USA: Citeseer, 2000.
[11] S. Bochkarev, *Minimizing Turns in Single and Multi Robot Coverage Path Planning*. Waterloo, ON, USA: Univ. Waterloo, 2017.
[12] J. Braga, F. Balampanis, A. Aguiar, J. Sousa, and I. A. Maza, and A. Ollero, "Coordinated efficient buoys data collection in large complex coastal environments using UAVs," in *Proc. OCEANS 2017-Anchorage*, Sep. 2017, pp. 1–9.
[13] T. Cabreira, L. Brisolara, and P. R. Ferreira, Jr., "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, no. 1, p. 4, Jan. 2019.
[14] D. Deng, W. Jing, Y. Fu, Z. Huang, J. Liu, and K. Shimada, "Constrained heterogeneous vehicle path planning for large-area coverage," 2019, *arXiv:1911.09864*.
[15] W. Dong, S. Liu, Y. Ding, X. Sheng, and X. Zhu, "An artificially weighted spanning tree coverage algorithm for decentralized flying robots," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, pp. 1689–1698, Oct. 2020.
[16] J. W. Durham, R. Carli, P. Frasca, and F. Bullo, "Discrete partitioning and coverage control for gossiping robots," *IEEE Trans. Robot.*, vol. 28, no. 2, pp. 364–378, Apr. 2012.
[17] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robot. Auton. Syst.*, vol. 61, no. 12, pp. 1258–1276, Dec. 2013.
[18] G.-Q. Gao and B. Xin, "A-STC: Auction-based spanning tree coverage algorithm formotion planning of cooperative robots," *Frontiers Inf. Technol. Electron. Eng.*, vol. 20, no. 1, pp. 18–31, Jan. 2019.
[19] D. C. Guastella, L. Cantelli, G. Giammello, C. D. Melita, G. Spatino, and G. Muscato, "Complete coverage path planning for aerial vehicle flocks deployed in outdoor environments," *Comput. Electr. Eng.*, vol. 75, pp. 189–201, May 2019.
[20] S. Hert and V. Lumelsky, "Polygon area decomposition for multiple-robot workspace division," *Int. J. Comput. Geometry Appl.*, vol. 8, no. 4, pp. 437–466, Aug. 1998.
[21] S. Hert and B. Richards, "Multiple-robot motion planning = parallel processing + geometry," in *Sensor Based Intelligent Robots* (Lecture Notes in Computer Science), G. D. Hager *et al.*, Eds. Berlin, Germany: Springer, 2002, pp. 195–215.
[22] X. Huang, M. Sun, H. Zhou, and S. Liu, "A multi-robot coverage path planning algorithm for the environment with multiple land cover types," *IEEE Access*, vol. 8, pp. 198101–198117, 2020.
[23] A. C. Kapoutsis, S. A. Chatzichristofis, and E. B. Kosmatopoulos, "DARP: Divide areas algorithm for optimal multi-robot coverage path planning," *J. Intell. Robotic Syst.*, vol. 86, nos. 3–4, pp. 663–680, Jun. 2017.
[24] J. Kim, C. Ju, and H. I. Son, "A multiplicatively weighted Voronoi-based workspace partition for heterogeneous seeding robots," *Electronics*, vol. 9, no. 11, p. 1813, Nov. 2020.
[25] J. Kim and H. I. Son, "A Voronoi diagram-based workspace partition for weak cooperation of multi-robot system in orchard," *IEEE Access*, vol. 8, pp. 20676–20686, 2020.
[26] E. Koutsoupias, C. H. Papadimitriou, and M. Sideri, "On the optimal bisection of a polygon," *ORSA J. Comput.*, vol. 4, no. 4, pp. 435–438, Nov. 1992.
[27] S. K. Lee, S. P. Fekete, and J. McLurkin, "Structured triangulation in multi-robot systems: Coverage, patrolling, Voronoi partitions, and geodesic centers," *Int. J. Robot. Res.*, vol. 35, no. 10, pp. 1234–1260, Sep. 2016.
[28] I. Maza and A. Ollero, "Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms," *Distrib. Auto. Robotic Syst.*, vol. 6, pp. 221–230, Jan. 2007.
[29] J. M. Palacios-Gasós, D. Tardioli, E. Montijano, and C. Sagüés, "Equitable persistent coverage of non-convex environments with graph-based planning," *Int. J. Robot. Res.*, vol. 38, no. 14, pp. 1674–1694, Dec. 2019.
[30] A. Pintado and M. Santos, "A first approach to path planning coverage with multi-UAVs," in *Proc. Int. Workshop Soft Comput. Models Ind. Environ. Appl.*, 2020, pp. 667–677.
[31] D. Puig, M. A. Garcia, and L. Wu, "A new global optimization strategy for coordinated multi-robot exploration: Development and comparative evaluation," *Robot. Auto. Syst.*, vol. 59, no. 9, pp. 635–653, Sep. 2011.
[32] B. Sun, D. Zhu, C. Tian, and C. Luo, "Complete coverage autonomous underwater vehicles path planning based on Glasius bio-inspired neural network algorithm for discrete and centralized programming," *IEEE Trans. Cogn. Develop. Syst.*, vol. 11, no. 1, pp. 73–84, Mar. 2019.
[33] G. Skorobogatov, C. Barrado, E. Salamí, and E. Pastor, "Flight planning in multi-unmanned aerial vehicle systems: Nonconvex polygon area decomposition and trajectory assignment," *Int. J. Adv. Robotic Syst.*, vol. 18, no. 1, Jan. 2021, Art. no. 172988142198955.
[34] Y. Tang, R. Zhou, G. Sun, B. Di, and R. Xiong, "A novel cooperative path planning for multirobot persistent coverage in complex environments," *IEEE Sensors J.*, vol. 20, no. 8, pp. 4485–4495, Apr. 2020.
[35] S. Xing, R. Wang, and G. Huang, "Area decomposition algorithm for large region maritime search," *IEEE Access*, vol. 8, pp. 205788–205797, 2020.
[36] M. Xu, B. Xin, and L. G. Dou, and G. Gao, "A cell potential and motion pattern driven multi-robot coverage path planning algorithm," in *Proc. Int. Conf. Bio-Inspired Comput., Theories Appl.*, Nov. 2019, pp. 468–483.
[37] J. Yackel, R. R. Meyer, and I. Christou, "Minimum-perimeter domain assignment," *Math. Program.*, vol. 78, no. 2, pp. 283–303, Aug. 1997.

GEORGY SKOROBOGATOV received the M.S. degree from the Moscow Institute of Physics and Technology, in 2016. He is currently pursuing the Ph.D. degree in aerospace science and technology with the ICARUS Research Group, Computer Architecture Department, Universitat Politècnica de Catalunya. He is the author of five articles and two proceedings.

CRISTINA BARRADO is currently an Associate Professor with the Computer Architecture Department, UPC BarcelonaTech University, Spain. She is also a member of the ICARUS Research Group and works in remote piloted aircraft systems (RPAS): their architecture, civil uses, and integration into the airspace. Previously, she worked in parallelization, compiling, operating systems, and embedded systems. She has been an adviser of five Ph.D. theses, four master's theses, and 26 final degree projects, since her integration into the ICARUS Research Group. She is also advising three Ph.D. students and five undergraduate/master's students' final projects. She has 17 indexed journal articles, two book chapters, and 68 conference papers.

ESTHER SALAMÍ received the M.S. degree in telecommunication engineering and the Ph.D. degree in computer architecture and technology from the Universitat Politècnica de Catalunya, Spain, in 1998 and 2007, respectively. She is currently an Associate Professor with the Computer Architecture Department, Universitat Politècnica de Catalunya. She is also a member of the ICARUS Research Group and her research focuses mainly on unmanned aerial systems (UAS), air traffic control (ATC), image processing, and artificial intelligence. Previous work, as a part of the High Performance Computing Research Group, UPC, includes multimedia processors, memory disambiguation, and digital video applications. She is the author of over 50 technical publications.