Coverage Path Planning for 2D Convex Regions

Article *in* Journal of Intelligent & Robotic Systems · January 2020

DOI: 10.1007/s10846-019-01024-y

preprint submitted to J. of Intelligent & Robotic Syst. manuscript No.
(will be inserted by the editor)

Coverage Path Planning for 2D Convex Regions

Juan Irving Vasquez-Gomez · Magdalena Marciano-Melchor · Luis Valentin · Juan Carlos Herrera-Lozada

Received: date / Accepted: date

Abstract The number of two-dimensional surveying missions with unmanned aerial vehicles has dramatically increased in the last years. To fully automatize the surveying missions it is essential to solve the coverage path planning problem defined as the task of computing a path for a robot so that all the points of a region of interest will be observed. State-of-the-art planners define as the optimal path the one with the minimum number of flight lines. However, the connection path, composed by the path from the starting point to the region of interest plus the path from it to the ending point, is underestimated. We propose an efficient planner for computing the optimal edge-vertex back-and-forth path. Unlike previous approaches, we take into account the starting and ending points. In this article, we demonstrate the vertex-edge path optimality along with in-field experiments using a multirotor vehicle validating the applicability of the planner.

Keywords coverage path planning · unmanned aerial vehicle · drone survey · computational geometry · optimal path

1 Introduction

The applications of the unmanned aerial vehicles (UAVs), also called drones, have had dramatic growth in the last five years. Among the variety of applications, the surveying missions for two-dimensional (2D) coverage have shown a particular success. For example, 2D and 3D mapping [14], search and rescue [7], aerial sensor networks for disaster management [17] or precision agriculture [19]. Under those

J. I. Vasquez-Gomez
Consejo Nacional de Ciencia y Tecnología (CONACYT) - Instituto Politécnico Nacional (IPN), México, E-mail: jivasquezg@conacyt.mx

M. Marciano-Melchor and J. C. Herrera-Lozada
CIDETEC, Instituto Politécnico Nacional, México. E-mail: mmarciano@ipn.mx, jlozada@ipn.mx

L. Valentin
Consejo Nacional de Ciencia y Tecnología (CONACyT) - Centro de Investigaciones en Óptica (CIO), Aguascalientes, México. E-mail: lvalentin@conacyt.mx

circumstances, academia and industry are betting for an increment in the number of applications in the years to come.

A surveying mission is performed in two steps: i) a preparation stage, where the selection of the vehicle, camera configuration and mission planning is performed, and ii) an execution stage, where the drone flights autonomously and recovers the data. To fully automatize the task, it is essential to solve the mission planning, where the core is the so-called coverage path planning problem, defined as the task of computing a path for a robot so that all the points of a region of interest (ROI) will be observed [8]. The complexity of such problem has been demonstrated as NP-Hard [3], but under the assumption that flying in straight lines provides a fast and efficient way to cover the terrain, state-of-the-art approaches simplify the problem and compute the path that holds least flight lines, e.g. [9] and [20]. Even though, current planners have achieved remarkable results, they underestimate the connection path, composed by the path from the starting point to the ROI plus the path from the ROI to the ending point. Not taking into account the connection path usually increases the length of the full path as we shown in our experiments. The increment in length gets worse when the mission is composed of several ROIs and therefore several connection paths, as the cases proposed in [23] and [25]. For this reason, it is necessary to plan a path that takes into account the starting and ending points of the mission. In addition, due to the limited onboard resources it is also essential to compute the path efficiently.

As we mention before, flying in straight lines forming a back-and-forth path (BFP) has several advantages and decreases the dimensionality of the problem. However, the problem is still difficult because we need to compute the orientation of the lines and the there is an infinite number of possible orientations. Previous works have defined the path optimality only on the number of flight lines discarding the starting and ending (SE) points. However, whether we include such points, the best full path it is not necessarily the path with least flight lines. In general, if we want to maintain a BFP as the coverage pattern, then we have to perform a search of a BFP which combined with the SE points provides the minimum cost. A simply strategy is to rotate the BFP until a minimum appears, however, it is not an efficient strategy.

In this study, we propose and analyze the rotating calipers path planner (RCPP), an efficient algorithm for computing a coverage BFP for a convex ROI taking into account the starting and ending points. See an example of a computed path in Fig. 1. This method is well suited for scenarios where the SE points are fixed and the coverage must be made as quickly as possible but maintaining the BFP shape, for example, in search and rescue operations. The proposed algorithm reduces the search to a set of promising BFPs, the edge-vertex paths, and test them with the starting and ending points. The key idea of the algorithm is to analytically restrict the set of promising paths (least flight lines paths that start and end at the antipodal pairs). Under the mentioned reduction, we test at most $\frac{3}{2}n$ paths (where n is the number of vertices), which is the maximum number of antipodal pairs [16]. In addition, our algorithm has a $O(n)$ complexity, contrasting with some state-of-the-art approaches with $O(n^2)$ complexity [20].

This article represents a journal version of our earlier work that was presented in preliminary form in [22]. In summary, this paper includes the following new results:

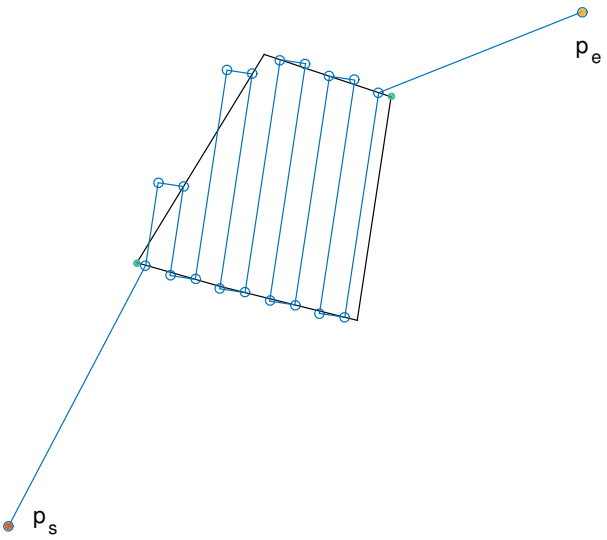


Fig. 1 Example of a full coverage path. The path (blue line) starts at the starting point p_s, follows a back-and-forth path inside the region of interest (black polygon) and moves to the ending point p_e (figure best seen in color).

- An extended description of the planner.
- Four mathematical proofs that validate the correctness of the proposed planner. In particular, we show that the RCPP computes the optimal edge-vertex path that takes into account the SE points. Consequently, this planner calculates shorter full paths than the current planners or in the worst case, it calculates the same path.
- In-field experiments using a multirotor vehicle validating the theoretical results.

2 Related Work

The coverage path planning (CPP) is a problem that has been studied several decades now, however, most of the literature addresses the problem from a ground mobile robot perspective. In this section, we review some relevant methods proposed for ground mobile robots and several methods for UAVs.

Depending on the representation of the ROI, the methods can be classified into solid and boundary representation approaches. Solid representation approaches focus on representing the interior of the region, however, due to the NP-hard nature of the problem [3], it is common to discretize the environment with a grid and assign a binary value or a probability of being occupied to each cell [13]. The

solid approaches, also known as grid based methods [8], are resolution complete algorithms and they could require high computational resources to deal with fine resolution maps. Some grid based approaches use bio-inspired neural networks [26], decompose the region in triangular cells and use a region growing algorithm to compute the path [4], use an Exploratory Turing Machine to supervise the coverage at multiple resolutions [18] or implement genetic algorithms to solve an epsilon-constraint optimization problem [1]. On the other hand, boundary representation methods define the ROI using its limits, the most common representation is by polygons. In this work, we use a polygonal representation, so we will discuss related approaches in the rest of this section. The reader can find a thorough review of alternative coverage methods in [8].

The Boustrophedon Cellular Decomposition (BCD) method [5] addresses the problem of planning a path to cover a polynomial world. The BCD performs a line sweep partitioning of the world. As a result, the world is dived into cells. Each cell is explored with a BFP, nevertheless, the robot always follows the same direction (from north to south and vice-versa). Later, Huang proposed a coverage method [9] for demining operations with mobile robots. In that work, Huang proposes that the sweep direction of the back-and-forth robot's movement must be perpendicular to the minimum width (span) of the ROI's polygon, however, his method does not consider the distance of traveling from the starting point to the ROI. Some works have studied the coverage of non-convex regions. In such approaches, the core problem is to get a quasi-optimal partitioning of the ROI and then to plan a coverage path for visiting one partition after the other [2], in some cases additional constraints like energy are part of the optimization [24]. In this work, we only study convex regions.

CPP using drones has been mainly addressed from a classical perspective of ground mobile robotics but new features have been included. The majority of the methods have followed the optimality criterion defined by Huang [9], where the optimal path is the one that holds least flight lines. In that sense, Li et al. in [12] introduce a method to cover a convex or non-convex polygon. The method decomposes the non-convex ROIs into convex cells, then it calculates the optimal orientation of the BFP by placing the flight lines perpendicular to the minimum width (FLPMW) of the cell and finally improves the full path by selecting the best joint points. In the study of Torres et al. [20], a method for covering convex regions is proposed. In addition to compute the path that holds least flight lines, their method finds the best connection from the starting and ending points to the BFP, however it does not change the flight lines orientation. A drawback of Torres's method is that the method has quadratic complexity, in fact, for each polygon's edge, it has to calculate the distance to each vertex.

In summary, previous approaches usually do not consider the connection path to reach the ROI, and the approaches that include it only find the way to connect the starting and ending points to the already computed BFP. Therefore, previous approaches are limited and a better path can be obtained if we combine the back and forth with the connection path in a single full path.

3 Preliminaries

In this section, we provide the assumptions and definitions that are needed in the proposed approach. Relevant or new definitions are remarked as block definitions.

3.1 General Definitions and Notation

In this study, we will address the problem from a 2D perspective, therefore the workspace will be $\mathbb{R}^2$ and it will be called "the plane". A point on the workspace is defined by a 2D tuple, (x, y). Given two points a and b, a line L is the linear combination $[a, b] = L(\alpha) = (1-\alpha)a+\alpha b$ so that $\alpha \in \mathbb{R}$ [16]. The distance between two points is the usual euclidean metric. The distance from a point a to a line L refers to the perpendicular distance from a to a point in L, namely, $d(a, L) = d(a, b)$ such that $b \in L, [a, b] \perp L$. The distance between two parallel lines L_1 and L_2 is defined by the perpendicular distance, $d(a \in L_1, L_2)$, between a point $a \in L_1$ and L_2 [15]. A line segment is the linear combination between two points a and b, $\overline{ab} = F(\alpha) = (1-\alpha)a + \alpha b$ so that $0 \leq \alpha \leq 1$.

3.2 Polygonal Region of Interest

We assume that the region of interest (ROI) is planar, so it is represented by a standard form convex polygon [10], $Q = \{V, E\}$, where $V = \{1, \ldots, n\}$ is a set of vertices (points in the plane) ordered in clockwise direction and $E = \{(1, 2), \ldots, (n-1, n), (n, 1)\}$ is the set of edges. The area of the polygon is represented as $\mathcal{A}(Q)$.

According to the study of Shamos [10] the following definitions are proposed.

Definition 1 A line L is a line of support if it meets the boundary of a polygon and lies entirely in one side of it.

Definition 2 An antipodal pair is a pair of vertices that admit parallel lines of support [10].

Given two parallel lines of support drawn in the same polygon, the distance between them is known as the span or width of the polygon [12].

3.3 Unmanned Aerial Vehicle

The surveying mission is performed by a drone which carries a pinhole camera that is pointing to the ground. We assume that the UAV maintains a constant altitude, therefore, we define the position of the drone, p, as a point in $\mathbb{R}^2$, namely $p = (x, y)$, where x and y are coordinates over the north-east-down (NED) frame. In this context, we define a drone path as a function $s(t) : \mathbb{R} \to \mathbb{R}^2$. It is worth to say that, the orientation of the drone is omitted and it is assumed to be tangent to the path. Inherently, we are assuming that there is a controller capable of following a given path. Although, it is a strong assumption our real world experiments show that it is possible to follow the given paths without noticeable loss.

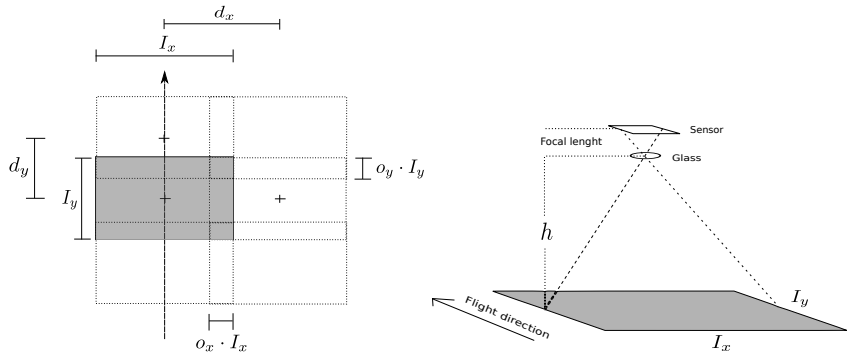


Fig. 2 Perception model for a drone coverage mission. Left figure shows a superior view. The dashed arrow indicates the drone's flight direction. During the flight, each photography is taken after a d_y distance. Gray filled rectangle draws the camera footprint $\mathcal{F}(p)$. Once that the drone has completed a flight line, it returns in the next flight line. Flight lines are separated by a d_x distance. Right figure shows the model from a lateral perspective, where h is the flight height.

3.4 2D Coverage

When the drone covers an area, it takes off at point p_s, moves to the region of interest, follows a path while it takes photos and finally moves to the landing point, p_e (see Fig. 1). At the moment that a picture is taken, the camera is able to capture a rectangle of the ground, also known as camera footprint, delimited by $I_x \times I_y$, where I_y is the length of the photo in the same direction of the path, and I_x is the width of the photo perpendicular to the path. See Fig. 2. Then, given a drone position p, the camera footprint is represented by $\mathcal{F}(p) = I_x \times I_y$.

Definition 3 Given a path s, the coverage region, $\mathcal{C}(s) = \bigcup_{p \in s} \mathcal{F}(p)$, is the area observed by placing the drone along s.

Now, let us define a coverage path as follows.

Definition 4 A coverage path, ρ, is a drone path so that every point of the region of interest, Q, is contained in the coverage region of ρ. Namely, $\mathcal{A}(Q) \subseteq \mathcal{C}(\rho)$.

3.5 Back-and-Forth Path

There is an infinity number of possible paths to cover an area. Even for single polygonal regions, to find the optimal coverage path has been proved to be NP-Hard [3]. To deal with the problem, several restricted paths have been proposed, such as spiral, zig zag, star or back-and-forth. The restricted paths are also known as search patterns. We use a back-and-forth path (BFP) where the drone follows straight lines that cross the region of interest (see Fig. 4). Some advantages are that the straight lines allow the UAV to maintain the camera pointing to the ground, the BFP guarantees complete coverage under the correct parameters, it has low spatial complexity and it is easy to be followed by autonomous vehicles.

Before constructing a BFP, let us establish several definitions.

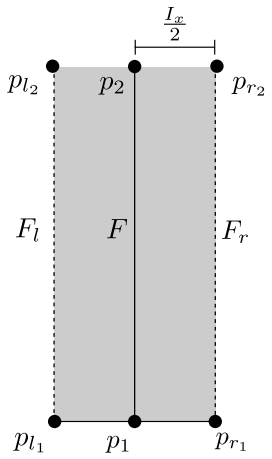


Fig. 3 Coverage region of the flight line F.

Definition 5 Given two distinct points p_1 and p_2, a flight line, F, is the linear combination

$$F(\alpha) = (1 - \alpha)p_1 + \alpha p_2 \tag{1}$$

so that $\alpha \in R$ and $0 \leq \alpha \leq 1$.

Since a flight line satisfies the line segment definition we will also write it as $\overline{p_1 p_2}$ indifferently.

Definition 6 The coverage region of a flight line, $\mathcal{C}(F)$, is the area delimited by two different lines F_l and F_r. So that F_l and F_r are parallel to F and they are at a distance $\frac{I_x}{2}$ from F.

Definition 6 is based on the fact that for each point $p \in F$ the limit of the footprint, $\mathcal{F}$, is defined by two points at a distance $\frac{I_x}{2}$, each one in one side of F. See the illustration presented in Fig. 3.

In most of the cases, the coverage region of a single flight line, $\mathcal{C}(F)$, is not enough to observe a region of interest. So that, the BFP adds parallel flight lines until the area is covered.

Definition 7 A back-and-forth path (BFP) is a series of parallel flight lines, $\mathbf{P} = \{F_1, \dots F_n\}$, so that the coverage region contains the region of interest, Q. Namely, $\mathcal{A}(Q) \subseteq \bigcup_{F \in \mathbf{P}} \mathcal{C}(F)$

Using the definition 5, the BFP can be represented by the ordered set of points that delimit the flight lines, such points are also known as waypoints. The connections between flight lines depend on the type of drone; for multirotor vehicles, the connections could be straight lines with sharp corners; for fixed wing vehicles, the connections could be Dubin's paths [6] in order to maintain the lift. For this work we will connect the lines with straight lines but the method is not restricted to straight lines. A BFP can have any orientation, but, there is a special case of a BFP that will be used later.

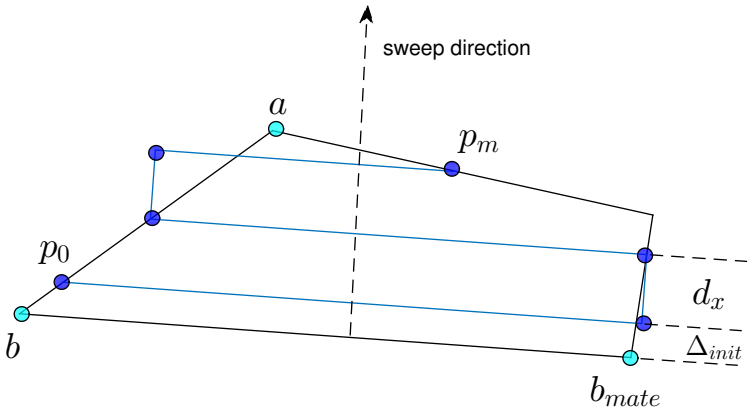


Fig. 4 Construction of a edge-vertex path. The algorithm iteratively adds waypoints by intersecting a line with the region of interest's polygon.

Algorithm 1: Get an edge-vertex path for a polygon ($\texttt{GetPath}(b, b_{mate}, a)$). The algorithm computes a back-and-forth path ($\rho = \{p_0, \ldots, p_m\}$) for a polygon ($Q = \{V, E\}$). The path starts from the initial vertex b towards b_{mate} and sweeps toward a. See the text for details.

Data: Q, d_x, b, b_{mate}, a
Result: ρ
$\Delta_{init} = \frac{d_x}{2}$;
$L_{flight} \leftarrow \texttt{CreateLine}(b, b_{mate})$;
$L_{flight} \leftarrow \texttt{OffSet}(L_{flight}, \Delta_{init})$;
$\rho \leftarrow \emptyset$;
while $\mathit{Intersects}(\mathcal{C}(L_{flight}), Q)$ **do**
  $ip_1, ip_2 \leftarrow \texttt{IntersectEdges}(L_{flight}, E)$;
  $\rho \leftarrow \texttt{CheckAndConnect}(\rho, ip_1, ip_2)$;
  $L_{flight} \leftarrow \texttt{OffSet}(L_{flight}, d_x)$;
return ρ;

Definition 8 An edge-vertex path (EVP) is a back-and-forth path where the flight lines are parallel to one edge of the region of interest's polygon.

In some cases, the EVP could be aligned with two edges, such case is considered an especial case [12] of the EVP.

3.5.1 Edge-Vertex Path Construction

To build an EVP, we add waypoints to a path by iteratively intersecting a line (L_{flight}) with the polygon (Q). The process is summarized in Algorithm 1 and it will be described next. The algorithm requires as input the polygon Q, an initial vertex b, an adjacent vertex named b_{mate}, the antipodal vertex to b, named a, and the distance between flight lines, d_x. d_x is calculated depending on the camera parameters and overlap needed (the reader is referred to [21] in order to calculate adecuate parameters). First, we create a line, L_{flight}, parallel to the edge (b, b_{mate}) which is perpendicularly displaced towards a (sweep direction) by an offset, $\Delta_{init} = \frac{d_x}{2}$ (see Fig. 4). Then, L_{flight} is intersected with the polygon Q, generating the points ip_1 and ip_2. After that, the points are attached to the path but they are connected in the correct order using the function `CheckAndConnect`, such function establishes a perpendicular constraint between the points that form a "return" of the drone, for multirotor vehicles, it could not be necessary, but it will facilitate the construction of Dubin's curves for fixed wing drones. We keep displacing and intersecting the line towards a, while the footprint intersects the polygon. Finally, the algorithm returns the path, $\rho = \{p_0, \ldots, p_m\}$. Note that, the constructed EVP always starts from b to b_{mate} and sweeps towards a. Whether we include the takeoff and landing points, the path is called full coverage path and it is written as $\tau = \{p_s, p_0 \ldots p_m, p_e\}$.

3.6 Problem Definition

In general, we would like to obtain the shortest path (in terms of flight time for this paper) that covers the ROI. However, due to the NP-Hard nature of the problem, we are restricting the paths to the set of edge-vertex paths. Then, we establish the problem as:

To calculate a full coverage path, $\tau = \{p_s, p_0 \ldots p_m, p_e\}$, so that the cost of traveling it is the minimal from all possible edge-vertex paths. Note that, p_s and p_e are the starting and ending points respectively, and $p_0 \ldots p_m$ form an edge-vertex path.

In other words, we want o obtain an edge-vertex path inside the polygon but at the time of combining it with the SE points we want to get the minimum total cost. Nonetheless, it could exist a shorter coverage path but that one will not have the edge-vertex shape. As we will see later the proposed planner guarantees an optimal path for the problem, since it efficiently tests all the possible edge-vertex paths.

4 Fundamentals of the Planner

As we mentioned before, there is an infinite number of BFP that could cover the ROI. However, there are certain properties on the BFP that can be exploited in order to reduce the possible solutions. In consequence, the searching time is also reduced. In this section, we provide a formal analysis of a key idea of the planner: any BFP starts and ends at an antipodal pair of the ROI's polygon. Later, based

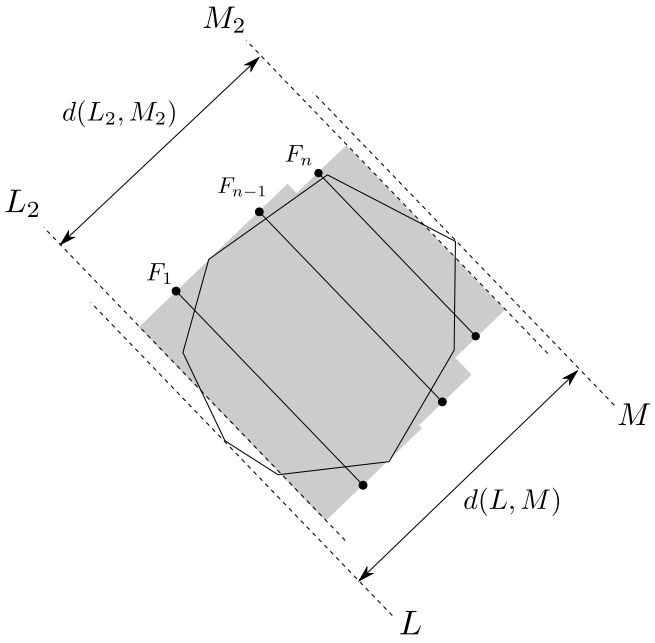


Fig. 5 Illustration accompanying demonstration of Lemma 1.

on this idea (in section 5) the work will be to find all the antipodal pairs and then to find the best path for each antipodal pair.

The coverage region of a BFP could cover a wider area than the polygon, but the minimal coverage region is delimited by two lines, as the following lemma establishes.

Lemma 1 *The minimal coverage region of any back-and-forth path,* $\mathbf{P} = \{F_1, \ldots, F_n\}$*, that covers a convex polygon,* Q*, is limited in two sides by the lines of support* L *and* M *parallel to the flight lines.*

Proof Let L and M be lines of support of Q and be the limits of the coverage region. See the accompanying Fig. 5. Namely, L is the limit of the coverage region for F_1 that is not inside on $\mathcal{C}(\mathbf{P})$. Similarly, M is the limit of the coverage region for F_N. Given that, by definition the limit L is parallel to F_1, F_1 is parallel to all other flight lines including F_n, and the limit M is parallel to F_n, then L is parallel to M. Now, suppose that there exist parallel lines, L_2 and M_2, delimiting a narrower area, namely, $d(L_2, M_2) < d(L, M)$. In consequence, $\epsilon = d(L, M) - d(L_2, M_2)$. Since, $\epsilon > 0$ the lines L_2 and M_2 are at a distance ϵ inside the polygon dividing it and leaving an uncovered area of width ϵ which contradicts the BFP definition (Def. 7). Therefore, the minimal coverage region is delimited by the lines of support parallel to the flight lines. □

Now, let us use Lemma 1 to propose the following definitions:

Definition 9 A limit line is a line of support that delimits a minimal coverage region.

Definition 10 A coverage extreme point is a point $p \in Q$ that admits a limit line and it is the first or the last visited during a coverage mission.

Lemma 2 *The coverage extreme points for any back-and-forth path are a pair of antipodal points.*

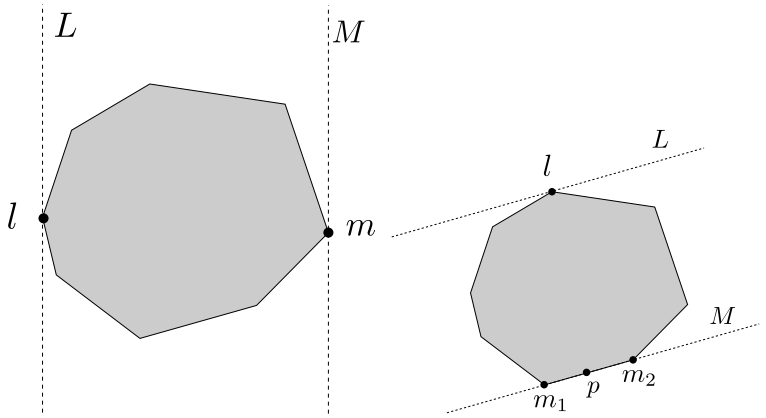
Proof Let us replace the coverage region of the BFP by two limit lines L and M (Lemma 1). Given that, the lines of support can only match with a vertex or an edge of the polygon ([12]), there are three possible cases for fitting the lines:

- Case 1. L and M touch a vertex of the polygon, let's say, $l \in V$ and $m \in V$ respectively.
- Case 2. L touches a vertex, l, and M touches an edge, $(m_1 m_2)$, so that $l, m_1, m_2 \in V$. For notation purposes, we will write the intersection with the edge as the line segment $\overline{m_1 m_2}$.
- Case 3. L and M touch edges of the polygon, let's say (l_1, l_2) and (m_1, m_2) respectively, so that $l_1, l_2, m_1, m_2 \in V$. The intersections are written as $\overline{l_1 l_2}$ and $\overline{m_1 m_2}$

Next, we will show that for these cases the coverage extreme points are antipodal pairs.

- The first case is straightforward. The unique touched points are the vertices l and m (Fig. 6(a)). Both vertices are extreme points because they belong to Q, admit limit lines and the coverage mission can visit l first and m later or in reverse order. Now, given that l and m are admitting limit lines and such lines are parallel lines of support (Def. 9), then l and m are an antipodal pair.
- The second case generates a point l and a segment line $\overline{m_1 m_2}$. l and the points in $\overline{m_1 m_2}$ belong to Q and admit a limit line. During a coverage mission l can be visited first or last therefore it is an extreme point; On the other hand, in the segment $\overline{m_1 m_2}$, the points m_1 and m_2 are the only that can be visited first or last, because any other point $p \in (m_1, m_2)$ will leave a set of uncovered points. Formally, suppose a point $p \in \overline{m_1, m_2}$, so that, $p \neq m_1 \neq m_2$ (Fig. 6(b)). If we run the coverage from m_1 to p, it covers the segment $\overline{m_1 p}$ (Definitions 5 and 6), leaving uncovered the open set of points $(p, m_2]$, contradicting the BFP coverage property; on the other hand, if we run the coverage from m_2 to p then it will leave uncovered the open set of points $[m_1, p)$, contradicting the BFP coverage property. Therefore, the points in $\overline{m_1, m_2}$ that can be visited first or last are only m_1 and m_2. In consequence, the valid pairs of extreme points are $\{l, m_1\}$ and $\{l, m_2\}$. Now, $\{l, m_1\}$ and $\{l, m_2\}$ are vertices of Q and both pairs admit parallel lines of support, therefore both pairs are antipodal pairs.
- In the third case, there are two line segments $\overline{l_1 l_2}$ and $\overline{m_1 m_2}$ with candidates to be extreme points. Using the same seasoning from the previous case, there are only four options for the extreme points $\{l_1, m_1\}$, $\{l_1, m_2\}$, $\{l_2, m_1\}$ and $\{l_2, m_2\}$. Now, each pair is composed by vertices of Q and admits parallel lines of support, therefore, each pair is an antipodal pair. □

Summarizing, we have shown that the coverage of a ROI starts or ends at an antipodal pair.



(a) Case 1. The extreme coverage points match with the antipodal points.

(b) Case 2. The extreme coverage points match with a pair of antipodal points. Any other point p is discarded as an extreme point.

Fig. 6 Illustrating the proof of Lemma 2

Algorithm 2: The rotating calipers path planner. The planner computes a full path (τ) composed by the starting point (p_s), the vertex-edge path (ρ), and the ending point (p_e).

Data: V, p_s, p_e
Result: τ
$A \leftarrow$ `computeAntipodalPairs`(V);
$c^* \leftarrow \infty$;
foreach $(i, j) \in A$ **do**
  $\rho \leftarrow$ `BestPath`(V,i,j);
  $\tau_{aux} \leftarrow$ `minCost`$(\{p_s, \rho, p_e\}, \{p_e, \rho, p_s\})$;
  if $cost(\tau_{aux}) < c^*$ **then**
    $\tau \leftarrow \tau_{aux}$;
    $c^* \leftarrow$ `cost`(τ);
return τ;

5 The Rotating Calipers Path Planner

In this section, we propose the rotating calipers path planner (RCPP), a method for computing the path that covers a convex ROI taking into account the mission's starting and ending points. We have called it the RCPP because it reuses some ideas from the rotating calipers algorithm proposed by Shamos [10] to calculate the antipodal pairs. The RCPP has a $O(n)$ complexity where n is the number of vertices of the ROI. In the following sections, we will provide the details about each part of the planner and we will analyze the quality of the solutions as well as the complexity of the proposed planner.

To deal with the problem we have established the BFP as the search pattern to cover the ROI and we are trying to find the BFP whose union with the starting and ending point provides the smaller path. If we do not consider the mission's starting and ending points, then the best path is the path whose flight lines are

perpendicular to the minimum width (FLPMW). However, the inclusion of the points affects the total path. In general, to solve the problem we would have to test all possible combinations of BFPs with the starting and ending points. However, we have found that the coverage of all BFP paths starts and ends at the antipodal pairs (Lemma 2). It is an advantage since the number of antipodal pairs is limited to $3/2n$ [16]. In consequence, if we know what is the best path for a given antipodal pair, then the combinations with the starting and ending points is limited by the number of antipodal pairs. Finally, to overcome the problem of finding the best path for an antipodal we propose a sub-method that finds the path that maintains the connection with the antipodal pair and guarantees the least flight lines. Note that in our method, the least flight lines optimality is used internally to obtain the path between antipodal pairs, and not to define the full path.

The RCPP takes advantage of Lemma 2 and proposes an strategy to efficiently compute all the antipodal pairs and the best path for each one, to finally select the path that has the lowest cost in combination with the SE points. Algorithm 2 summarizes the planner. In detail, at line 1 the set of antipodal points is computed. The `computeAntipodalPairs()` method is described in section 5.1. Then, for each pair (line 3), the best path is computed (line 4) and it is attached to the starting and ending points (line 5). The `BestPath()` method is explained in section 5.2. Since the order of the path matters, it is tested in straight and reverse order and the one with minimum distance is chosen (line 5). If the cost of the current path is smaller then it is kept as the best full path (τ). Finally, once that all the antipodal pairs have been tested, τ is returned.

5.1 Computation of the Antipodal Pairs

The procedure of `computeAntipodalPairs` returns the antipodal pairs for a convex polygon. In this procedure, we use the Shamos's algorithm [10] that computes all the pairs in $O(n)$ time, where n is the number of vertices of the polygon. The procedure receives a set of vertices and returns the antipodal pairs as a set of tuples, namely $A = \{(i_1, j_1), \ldots, (i_n, j_n)\}$. An interesting fact is that Shamos's method is known as the "rotating calipers algorithm" because it resembles to a dynamically adjustable caliper that is measuring the polygon as it rotates.

5.2 Calculation of the Best Path for Each Antipodal Pair

In this procedure, we compute the optimal BFP path for an antipodal pair i and j. See Algorithm 3. The main idea is to find the pair of parallel support lines that pass trough vertices i and j which have the minimum distance between them, and then, to get the BFP that matches those lines. To put it differently, the process is given an antipodal pair and a caliper that touches the pair, rotate the caliper in clockwise direction until and edge is touch and measure the height of the polygon, then rotate the caliper in counter clockwise direction until a second edge is touch and measure the height, finally, compare both options in order to find the minimum width.

Algorithm 3: Best path algorithm. It computes the best path (ρ) for an antipodal pair. It receives as input a set of vertices (V) and the antipodal pair, (i, j).

Data: $V = \{1, \ldots, n\}$,(i, j)
Result: ρ
`/* Rotate the caliper in clockwise direction */`
1 **if** $angle(i, j) - \pi < 0$ **then**
2 $\quad b \leftarrow j$;
3 $\quad a \leftarrow i$;
4 **else**
5 $\quad b \leftarrow i$;
6 $\quad a \leftarrow j$;
`/* Rotate the caliper in counter-clockwise direction */`
7 $\phi \leftarrow$ `angle(b,a)` $- \pi$;
8 $\gamma_b \leftarrow$ `angle(b-1,b)`;
9 $\gamma_a \leftarrow$ `angle(a-1,a)` $- \phi$;
10 **if** $\gamma_b < \gamma_a$ **then**
11 $\quad b_2 \leftarrow b - 1$;
12 $\quad a_2 \leftarrow a$;
13 **else**
14 $\quad b_2 \leftarrow a - 1$;
15 $\quad a_2 \leftarrow b$;
`/* Find the path that holds least flight lines */`
16 **if** $dist(b, a) < dist(b_2, a_2)$ **then**
17 $\quad \rho \leftarrow$ `getPath`$(b, b + 1)$
18 **else**
19 $\quad \rho \leftarrow$ `getPath`$(b_2 + 1, b_2)$

In the following sections, we assume that all vertex indices are reduced modulo n (so that an index $n + 1 = 1$) and `angle`(m, n) is a function that computes the clockwise angle swept out by a line as it rotates from a position parallel to the edge $(m, m + 1)$ to a position parallel to $(n, n + 1)$[10].

5.2.1 Caliper rotation in clockwise direction

Our target is to find the first edge of contact whether we rotate the caliper in clockwise direction. Such edge will be called $(b, b + 1)$ and it will be the base of one possible BFP. See Fig. 7 as an example and note that if we rotate lines L and M over the antipodal points i and j the first edge of contact will be $(j, j + 1)$ that matches with M'. The edge $(i, i + 1)$ is not feasible since a parallel line of support for L' that passes through j does not exist.

One way to find $(b, b + 1)$ is by finding the minimum angle between α_i and α_j, which are the angles from the imaginary support lines to the nearest edge of the polygon in clockwise direction. However, instead of measuring α_i and α_j, we measure the difference between the rotation from the line L' to M'. See lines 1-6 of the Algorithm 3. This procedure is validated by Theorem (1). If the difference of the angle minus π is less than zero then the edge $(j, j+1)$ is taken as the base of the path. Otherwise, the edge $(i, i + 1)$ is taken as the base of the edge. A special case is when both angles are equal. In such case, the number of flight lines is the same, then, it does not matter which base is selected in terms of the number of

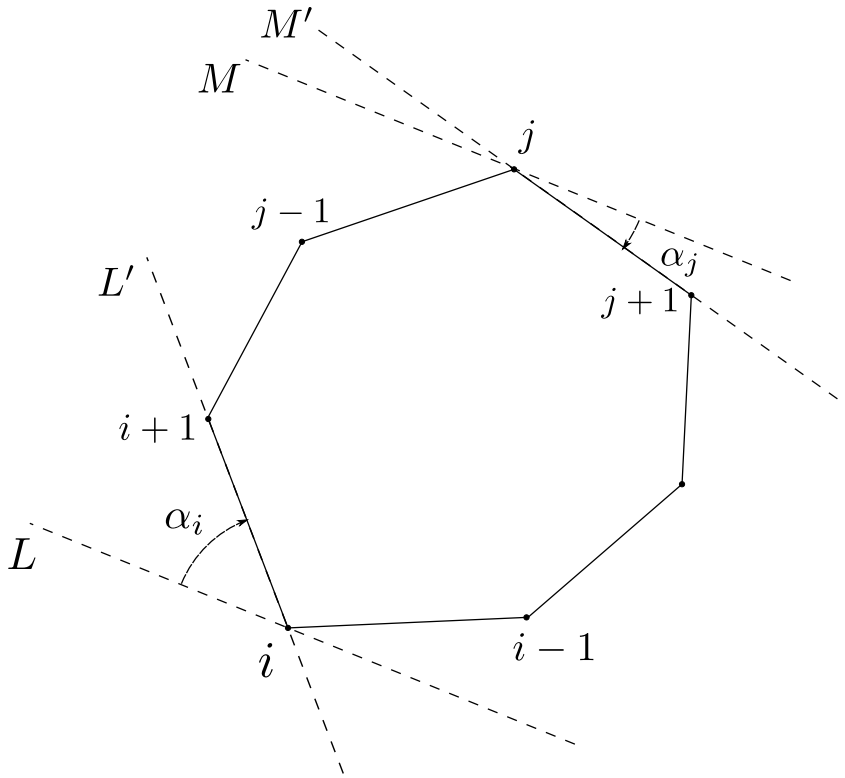


Fig. 7 Illustration of the clockwise rotation of the caliper. Lines L and M represent the caliper. The lines are rotated until one edge is touch.

flight lines. For this reason and simplicity in the algorithm, in case of equal angles we select the edge $(i, i+1)$ as the base. Once that the edge $(b, b+1)$ has been found, it is stored as the first possible base edge.

Theorem 1 *If* $\texttt{angle}(L', M') - \pi < 0$ *then* $\alpha_i > \alpha_j$.

Proof Let us construct a line K parallel to L' that pass trough the vertex j. So that

$$\texttt{angle}(L', K) = \pi \tag{2}$$

(See Fig. 8). The angle between K and M' is denoted by γ. So that,

$$\gamma = \texttt{angle}(L', K) - \texttt{angle}(L', M') \tag{3}$$

By alternate interior angles,

$$\alpha_i = \alpha_j + \gamma \tag{4}$$

Replacing (2) and (3) in (4), and reordering,

$$\alpha_j - \alpha_i = \texttt{angle}(L', M') - \pi \tag{5}$$

From Equation (5) we can observe that, when α_i is bigger than α_j the difference is negative in the left side of the equation, therefore, the right side must also be negative □

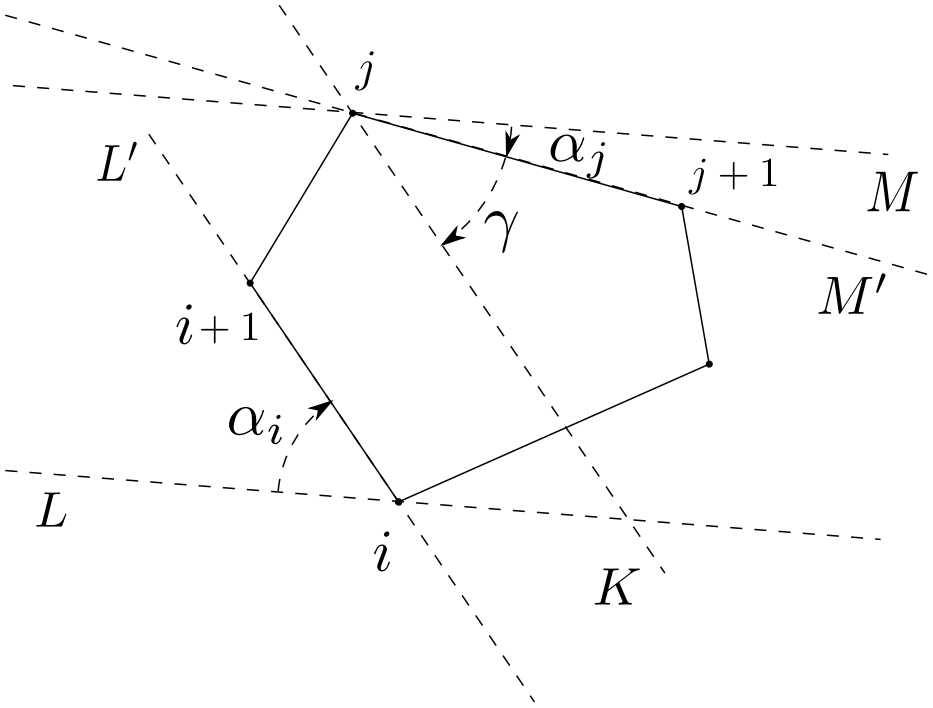


Fig. 8 Illustration of the imaginary caliper rotating in clockwise direction. The caliper is represented by the parallel lines of support L and M. In this step, the target is to find the first edge of contact.

5.2.2 Caliper rotation in counter-clockwise direction

In this step, we find the other possible path, with base edge (b_2, b_2+1), by rotating the caliper in counter clockwise direction. However, we do not repeat the same process the other way round because it will imply to create a reverse function to `angle`() and from the embedded systems perspective, we prefer to use the same function. For this reason, we measure the angles with respect to the line B, a line that matches $(b, b+1)$ (see Fig. 9). The measured angles are γ_b, the angle between $(b-1, b)$ and B, and γ_a, the angle with respect to A, a line of support parallel to B that pass thought the antipodal pair of b. The angle ϕ complements γ_a to reach the edge $(a, a+1)$. Note that, from the previous step, $(b, b+1)$ was touch before $(a, a+1)$, therefore the angle ϕ is bigger than or equal to zero.

The minor of the angles γ_b and γ_a will tell us the nearest edge from the two possibilities: $(b-1, b)$ or $(a-1, a)$. Once that the second base edge, (b_2, b_2+1), has been found, the farthest vertex is the antipodal pair. See lines 7-15 from the Algorithm 3.

5.2.3 Finding the path that holds least flight lines

So far, there are two possible paths: i) a path with base $(b, b+1)$ that ends at vertex a and ii) a path with base (b_2, b_2+1) that ends at vertex a_2. Then, we

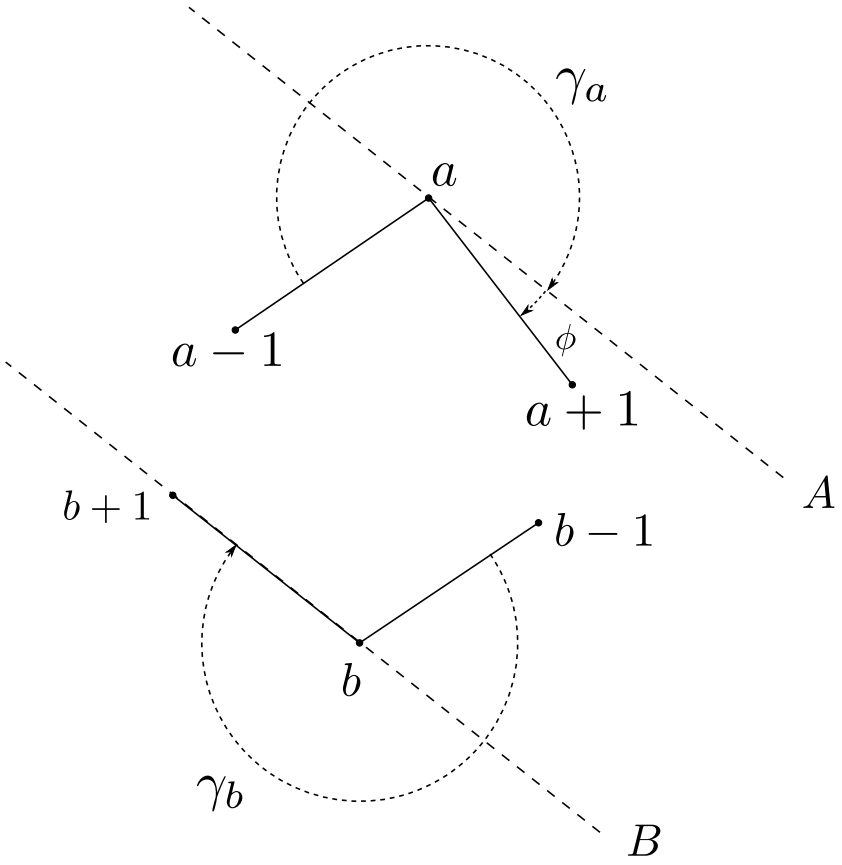


Fig. 9 Angles between the caliper last position and the previous polygon edges. These angles (γ_a and γ_b) define the first edge of contact if the caliper will be rotated in counter-clockwise direction.

select the path that holds least flight lines. See lines 16-19 of the Algorithm 3, where the function `dist`(b,a) computes the distance between the line that matches the edge $(b, b+1)$ and the vertex a, and the function `getPath`(b, b_{mate}, a) computes a BFP with origin at vertex b with sweep direction perpendicular to (b, b_{mate}). To prove the optimality of the algorithm in terms of the number of flight lines we provide the following lemma:

Lemma 3 *Algorithm 3 obtains the coverage back-and-forth path which holds least flight lines that pass trough the antipodal pair* (i, j).

Before proceeding with the demonstration let us state the following: "given any two points p and q and distinct parallel lines l_1 and l_2 passing through them, we can always decrease the distance d between them by rotating them about p and q in the preferred direction of rotation" [15]. Next, we provide the following proof:

Proof Let us establish two parallel lines of support L and M that represent any BFP (Lemma 1). The lines, L and M, pass trough i and j respectively. Then,

the minimum distance between L and M will appear in one of the two possible rotations (clockwise or counter clockwise) [15] until an edge is touched, it can not go further because the line of support definition will be contradicted. As can be seen, Algorithm 3 obtains the smallest distance between L and M because it performs a rotation of the parallel lines in both directions and selects the lines with the smallest distance between them □

5.3 Complexity Analysis

The computational complexity of the rotating calipers path planner, described in Algorithm 2, is $O(n)$, where n is the number of vertices of the polygon. It is due to the fact that the algorithm requires a call to the antipodal pairs computation and a loop that computes the best path for each antipodal pair. The first step has been demonstrated to be $O(n)$ [10]. The loop step calls the best path computation (Algorithm 3) for each antipodal pair. Given that Algorithm 3 performs a sequence of decisions and procedures that do not depend on the number of vertices we set its complexity as constant time, c. Therefore, the parameter to observe is how much antipodal pairs will be processed. According to [16], Shamos's algorithm generates at most $\frac{3}{2}n$ antipodal pairs. Therefore, Algorithm 2 performs in the worst case $\frac{3}{2}c \times n$ operations, simplifying, the complexity is $O(n)$.

5.4 The Quality of the Path

Considering that Lemma 2 establishes that the coverage regions from all possible BFPs start and end at an antipodal pair, the proposed RCPP (Algorithm 2) computes all the antipodal pairs and then, it finds the path with least flight lines for each pair (Lemma 3). Since, the computed paths for each pair are edge-vertex paths (they are aligned with an edge) and the best one is selected (otherwise Lemma 3 is contradicted), we can say that the proposed planner computes the edge-vertex path that covers the convex polygon Q whose distance added to the connection path to the starting (p_s) and ending (p_e) points is the minimum according to the metric $cost()$. For this reason, and given that the previous approaches, [9], [12], [20] only search for the edge-vertex path with least flight lines (see Theorem 1 from [11]) without taking into account the SE points, the RCPP usually calculates a better full path, some examples are provided in the experiments; otherwise, RCPP provides the same solution than previous approaches given that such paths are also tested.

6 Experiments

We present several in field experiments where we compare the costs of the paths computed by our algorithm (RCPP) versus the state-of-the-art algorithm of flight lines perpendicular to the minimum width (FLPMW) also known as least flight lines. The FLPMW implementation corresponds to the algorithm presented by Torres in [20]; in addition to the orientation of the flight lines, Torres's algorithm provides the connections to the starting and ending points by testing the order of

Fig. 10 3DR Solo quadcopter used during the in field experiments.

them while the flight lines orientation is kept. Both algorithms were implemented in Matlab and tested in a core i7 machine. Our implementation is available at `https://github.com/irvingvasquez/ocpp`. The experiments were carried out at the Instituto Politécnico Nacional campus Zacatenco in Mexico City. We used a 3DR Solo drone (see Fig. 10) to flight the missions. Mission parameters are drawn in Table 1.

Forward speed	5 m/s
Heading rate	$\pi/4$ rad/s
Distance between flight lines	10 m
Flight altitude	50 m

Table 1 Flight parameters used in the experiments.

The experiment workflow is as follows. First, we generate random polygons with random distances from the starting point. See Fig. 11 where we present the polygons (Q_A, Q_B and Q_C) and resulting paths for each algorithm (which will be analyzed below). Observe that for the polygons Q_A and Q_B, the UAV has to cover the terrain and then return to the home position. Therefore, the takeoff point and the landing point are the same. But, for the polygon Q_C, the UAV has to complete the mission reaching a waypoint different from the starting point. Once the polygons are set, we compute the paths using both algorithms (the computation is done offline on the ground). The resulting paths are drawn in blue in Fig. 11. Next, we upload the paths to the UAV using MAVLink protocol. Finally, the UAV follows the path using the onboard controller and reaches the landing point. See Fig. 12. The results are summarized in Table 2 where the processing time is the elapsed time to compute the path, the estimated flight time is the distance of the path divided by the speed set to the vehicle and the flight time is the amount of time that the UAV required to follow the path.

As shown in Table 2, the proposed algorithm reduces the flight time for the presented cases. Time reduction is about six seconds per polygon. Although it might look small, such saving is essential for several tasks where the coverage time is critical, for instance, search and rescue operations. In addition, savings will be accumulated in cases where more than one convex polygon is covered, e.g. the non-convex region coverage where the area is divided into convex regions and each one is visited one after the other. In the final analysis, RCPP required a small amount of extra processing time (in the order of milliseconds), this is due to the

	Processing time (s)		Estimated flight time (s)		Flight time (s)	
Poly.	FLPMW	RCPP	FLPMW	RCPP	FLPMW	RCPP
Q_A	0.0027	0.0066	189.3	186.2	174	171
Q_B	0.0021	0.0064	125.9	119.2	108	102
Q_C	0.0023	0.0077	167.1	164.5	142	136

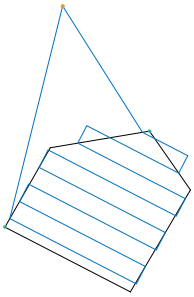
Table 2 Comparison of the proposed approach (RCPP) versus the state-of-the-art planner (FLPMW). The table shows the processing time, estimated cost and real flight cost for several polygons. Units are seconds. RCPP: Rotating calipers path planner, FLPMW: Flight lines perpendicular to the minimum width.

number of inner operations of the algorithm, but for polygons with a large number of edges, it is expected that RCPP will perform better due to its $O(n)$ complexity. It is worth to say that the processing time is spent on the ground, not during the mission.

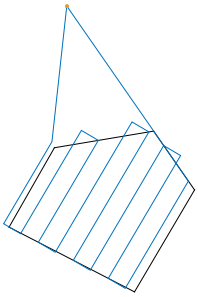
7 Conclusions and Future Work

The rotating calipers path planner, an algorithm that computes the full path to cover a convex region of interest, has been presented. The algorithm obtains the optimal edge-vertex path taking into account the starting and ending points. The optimality is validated through a mathematical analysis of the algorithm and with experiments using a quadcopter for an in-field surveying. In addition, the algorithm has a $O(n)$ complexity where n is the number of edges of the region of interest. The proposed algorithm reduces the flight time, or in the worst case, it provides the same flight time than previous state-of-the-art approaches. Future research directions are coverage planning for disjoint areas and coverage planning under external perturbations like the wind.

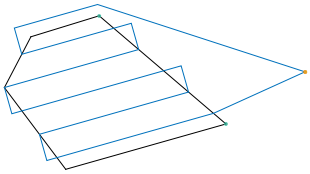
Acknowledgements The authors thank to SNI-México, CONACYT cátedra 1507 and IPN-COFAA program. The authors also thank to Efrén López Jiménez for his support during the flight experiments.



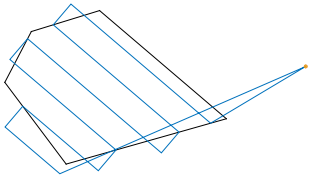
(a) RCPP's path for Q_A.



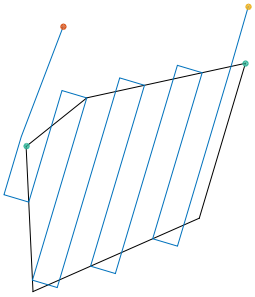
(b) FLPMW's path for Q_A.



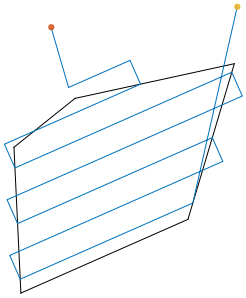
(c) RCPP's path for Q_B.



(d) FLPMW's path for Q_B.

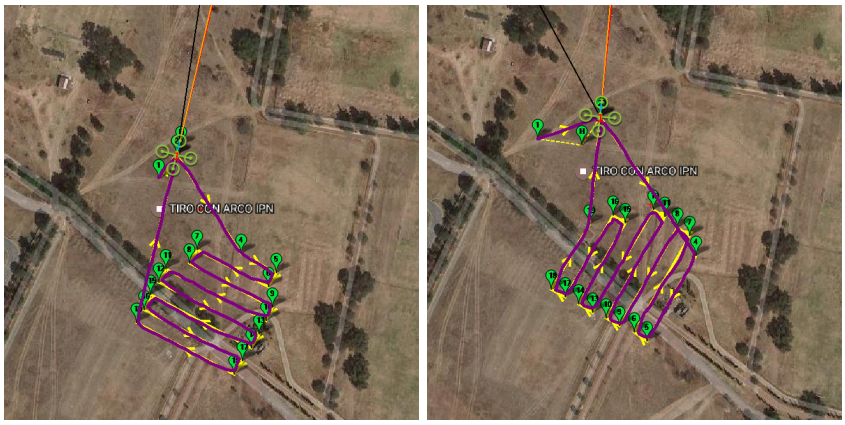


(e) RCPP's path for Q_C.

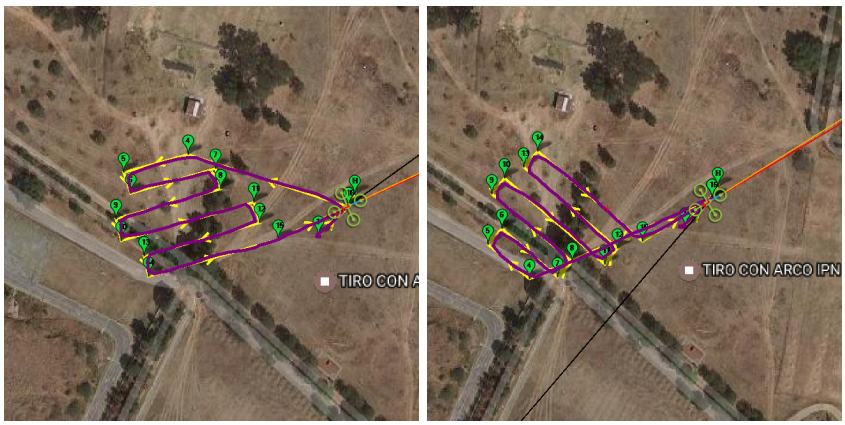


(f) FLPMW's path for Q_C.

Fig. 11 Comparison of the paths computed by the proposed algorithm (RCPP) versus the flight lines perpendicular to the minimum width (FLPMW) approach.



(a) RCPP's path for Q_A. (b) FLPMW's path for Q_A



(c) RCPP's path for Q_B. (d) FLPMW's path for Q_B

(e) RCPP's path for Q_C. (f) FLPMW's path for Q_C

Fig. 12 Comparison of the paths followed by the drone using the proposed algorithm (RCPP) versus the flight lines perpendicular to the minimum width (FLPMW) algorithm.

References

1. SM Ahmadi, H Kebriaei, and H Moradi. Constrained coverage path planning: evolutionary and classical approaches. *Robotica*, 36(6):904–924, 2018.
2. Vatana An, Zhihua Qu, Frank Crosby, Rodney Roberts, and Vithia An. A triangulation-based coverage path planning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2018.
3. Esther M. Arkin, Sndor P. Fekete, and Joseph S.B. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(12):25 – 50, 2000.
4. Fotios Balampanis, Ivn Maza, and Anbal Ollero. Coastal areas division and coverage with multiple uavs for remote sensing. *Sensors*, 17(4), 2017.
5. Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon cellular decomposition. In *Field and service robotics*, pages 203–209. Springer, 1998.
6. Matthew Coombes, Wen-Hua Chen, and Cunjia Liu. Boustrophedon coverage path planning for uav aerial surveys in wind. In *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on*, pages 1563–1571. IEEE, 2017.
7. D. Erdos, A. Erdos, and S. E. Watkins. An experimental uav system for search and rescue challenge. *IEEE Aerospace and Electronic Systems Magazine*, 28(5):32–37, May 2013.
8. Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.
9. Wesley H Huang. Optimal line-sweep-based decompositions for coverage algorithms. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2001*, volume 1, pages 27–32. IEEE, 2001.
10. Michael lail Shamos. *Computational Geometry*. PhD thesis, Yale University, 1978.
11. Deshi Li, Xiaoliang Wang, and Tao Sun. Energy-optimal coverage path planning on topographic map for environment survey with unmanned aerial vehicles. *Electronics Letters*, 52(9):699–701, 2016.
12. Yan Li, Hai Chen, Meng Joo Er, and Xinmin Wang. Coverage path planning for uavs based on enhanced exact cellular decomposition method. *Mechatronics*, 21(5):876–885, 2011.
13. H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation 1985*, volume 2, pages 116–121, Mar 1985.
14. Francesco Nex and Fabio Remondino. Uav for 3d mapping applications: a review. *Applied geomatics*, 6(1):1–15, 2014.
15. Hormoz Pirzadeh. Computational geometry with the rotating calipers. McGill University, 1999.
16. Franco Preparata and Michael Ian Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 1985.
17. Markus Quaritsch, Karin Kruggl, Daniel Wischounig-Strucl, Subhabrata Bhattacharya, Mubarak Shah, and Bernhard Rinner. Networked uavs as aerial sensor network for disaster management applications. *Elektrotechnik und Informationstechnik*, 127(3):56–63, 2010.
18. Junnan Song and Shalabh Gupta. $\varepsilon^\star$: An online coverage path planning algorithm. *IEEE Transactions on Robotics*, 34(2):526–533, 2018.
19. P. Tokekar, J. V. Hook, D. Mulla, and V. Isler. Sensor planning for a symbiotic uav and ugv system for precision agriculture. *IEEE Transactions on Robotics*, 32(6):1498–1511, Dec 2016.
20. Marina Torres, David A Pelta, José L Verdegay, and Juan C Torres. Coverage path planning with unmanned aerial vehicles for 3d terrain reconstruction. *Expert Systems with Applications*, 55:441–451, 2016.
21. J. I. Vasquez-Gomez, J. C. Herrera-Lozada, and M. Olguin-Carbajal. Spatial resolution optimization for terrain coverage with uavs. In *Proceedings of International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE) 2017*, pages 37–42, Nov 2017.
22. J. I. Vasquez-Gomez, M. Marciano-Melchor, and J. C. Herrera-Lozada. Optimal coverage path planning based on the rotating calipers algorithm. In *Proceedings of International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE) 2017*, 2017.
23. Y. Wang, T. Kirubarajan, R. Tharmarasa, R. Jassemi-Zargani, and N. Kashyap. Multi-period coverage path planning and scheduling for airborne surveillance. *IEEE Transactions on Aerospace and Electronic Systems*, PP(99):1–1, 2018.

24. Minghan Wei and Volkan Isler. Coverage path planning under the energy constraint. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2018*, pages 368–373. IEEE, 2018.
25. Junfei Xie, Lei Jin, and Luis Rodolfo Garcia Carrillo. Optimal path planning for unmanned aerial systems to cover multiple regions. In *AIAA Scitech 2019 Forum*, page 1794, 2019.
26. Daqi Zhu, Chen Tian, Bing Sun, and Chaomin Luo. Complete coverage path planning of autonomous underwater vehicle based on gbnn algorithm. *Journal of Intelligent & Robotic Systems*, Feb 2018.