Polygonal plot scaling method for plant protection of rotor-like UAVs

Erbiao Zou[1*]

[1]the School of Information Science and Engineering, Northeastern University, Shenyang 110004, China

Abstract. In this paper, a universal polygonal block boundary method is proposed to solve the problem of integral shrinkage and integral expansion of the boundary of the operation area when facing complex operation area boundary and obstacle boundary in the plant protection process of rotor-wing unmanned aerial vehicles (UAVs). When rotorcraft UAVs carry out plant protection operation, they cannot spray more or miss spraying, and they cannot have any collision with obstacles. Based on the method proposed in this paper, the boundary of the operation area and the boundary of the obstacles can be shrunk and expanded as a whole, so as to improve the safety factor of UAVs and the operation accuracy and efficiency.

1 Introduction

Because the aging situation in our country is more and more serious and the younger generation tend to go out to work in big cities, rural labour force will continue to reduce [1]. The most of domestic rural land is still in the small block. The reasons why the costs of labour force of producing unit weight food are relatively high and the process is inefficient are the number of land block and the current situation of mechanical work inadequate and so on. In the future, it is a general trend to combine small plots of farmland [2] and use unmanned aerial vehicle (UAVs) to spray pesticide and fertilize to achieve full autonomous operation. If we want to ensure that there is no some farmland missed and avoid obstacles safely when UAVs work, it is necessary to input data points of the boundary of obstacles and plots which are shrunk or extended properly according to the specific environment. In this way, the subsequent path planning and UAVs work will be safe and efficient.

2 Preprocessing process of boundary polygons and obstacle boundaries

Before the boundary polygon of the block or obstacle can be shrunk and expanded, it is necessary to pre-process the collection point of the boundary. Specific pre-processing steps are as follows:

Step 1: coordinate transformation. The polygonal plot boundary points and the obstacle boundary points in the plot operated by UAVs can be obtained by manual collection, aircraft collection and high-precision map collection based on aerial survey. These coordinates are represented by longitude and latitude. To facilitate calculation, Gauss - Kruger Projection is used to convert them into plane coordinates [3]. And this paper focuses on the algorithm directly using plane rectangular coordinate to demonstrate.

Step 2: removal of coordinate. Sometimes, by artificial collection, there are two points which are quite close between them. This situation will increase the amount of calculation and will bring some difficulties to the subsequent algorithm processing. Considering that working accuracy of UAVs don't have to be too precise, so we only remain the first point as a useful point if there are some points which are continuous and the distance of points is too close.

Step 3: removing collinear sides from polygons. Calculate the unit vector of each side of the boundary polygon of the block or obstacle. If the unit vector of the adjacent two sides is the same, we combine them into one side.

Step 4: uniform sorting. Obtaining the boundary vertices of the polygon of plots or obstacles may arrange storage in the clockwise or anticlockwise. To facilitate follow-up algorithm implementation, we need to unified them into clockwise to arrange storage here, namely: firstly, judge the arrangement of the original point is clockwise or anticlockwise; if it is anticlockwise, invert the order of all the points; Otherwise, nothing is done. Secondly, the original arrangement of points is stored in the variable $\mathrm{Clock\_Flag}$ and if that is clockwise, the $\mathrm{Clock\_Flag}$ is assigned a value of 1; If it is anticlockwise, the $\mathrm{Clock\_Flag}$ is assigned a value of 0.

2.1 Polygon integral indenting and integral outspread methods

After preprocessing, based on the plane rectangular coordinates arranged in clockwise order of the polygon boundary of the block or the obstacle, the polygon

*Corresponding author's e-mail: zouerbiao@163.com

coordinates can be used for the overall shrinking and expanding of the polygon. Before the operation, it is necessary to calculate the following quantities used in the overall shrinking and expanding:

Quantity 1: vector DP_i for each side. From the first point to the penultimate point, you can make the coordinates of the last point subtract the coordinates of the previous point; when you calculate the vector of the last side, you need to make the coordinates of the first point subtract the coordinates of the last point. The ordinal of the vector corresponds to the ordinal of the side and the ordinal of the starting point of the side.

Quantity 2: unit vector NDP_i for each side. Convert DP_i to a unit vector and store it in NDP_i.

Quantity 3: $\mathrm{Sin}\,a$ of the angle between the unit vectors of the two sides of each vertex. The cross product between the unit vector of the last side of each vertex and the unit vector of the previous side can calculate the sine of the angle between the adjacent two sides of the vertex.

2.2 Indenting algorithm

On the basis of the vertex coordinates of the polygon and the sine value of the angle between the two sides of each vertex, the overall indenting algorithm can be implemented. The steps are as follows:

Step 1: according to formula (1), calculate the coordinate Q_i of each new vertex after shrinking L meters of the polygon, in which P_i is the coordinate of the original vertex of the polygon.

$$Q_i = P_i + (NDP_{i+1} - NDP_i) * L / S\mathrm{in}\,a_i \quad (1)$$

Step 2: removal of coordinate and removing collinear sides from polygons. The specific steps are the same as the removal of coordinate and removing collinear sides from polygons in the pretreatment.

Step 3: removing simple invalid polygons [4][5]. A simple valid polygon is a polygon that contains no other polygons and whose vertex coordinates are in the same direction as the original polygon. After the polygon is shrunk inside, some invalid simple polygons may be formed. When UAVs are making route planning, these simple invalid polygons not only are useless but also increase the difficulty of subsequent route planning.

Step 3.1: calculate the intersection points of each side. First, invert the vertices of the polygon arranged clockwise into anticlockwise, and then loop over to judge whether there is an intersection point between each side and its subsequent sides. If there is any intersection point, record the intersection point coordinates and the sides numbers on both sides of the intersection (the side with a smaller side number is denoted as the starting side, and the side with a larger side number is denoted as the ending side).

Step 3.2: arrange all intersections in ascending order according to the size of the number of the starting side. The purpose is to facilitate the subsequent insertion of intersections in the polygon vertex set.

Step 3.3: insert the intersections into the polygon vertex set. Starting from the first intersection point: 1) locate the starting side and ending side of the intersection point in the polygon vertex set; 2) insert the intersection point respectively behind these two positions. If there are more than one intersection point which are going to be inserted behind the same vertex, insert the intersection points in the order of the distance from each intersection point to the vertex from small to large; loop until all intersections are inserted, forming a new 2-d array of polygon vertices which is stored in the 2-d array $PolygonList$.

Step 3.4: find simple polygons. A simple polygon is a polygon that contains no other polygons. Start with the first point of the 2-d array $PolygonList$ and save it in a new 2-d array $NewPolygonListPP$. Then, proceed to the next point (denoted as point A) : 1) the position of point A in the array is stored in array b in sequence and point A is saved in the two-dimensional array $NewPolygonListPP$ if point A is an intersection and it is different from the first point of 2-d array $NewPolygonListPP$. Start with step 1) again from the next point of point A in another position in the two-dimensional array $PolygonList$ (denoted as the new point A). 2) If the point A is different from the first point in the two-dimensional array $NewPolygonListPP$ and has not been accessed yet, the point A is stored in the two-dimensional array $NewPolygonListPP$ and the access flag variable of A is set 1. Then the next point of point A is denoted as the new point A and taken to begin with step 1) again. 3) If there are other cases, it means that a new simple polygon has been formed. So, Firstly, the access mark variable of the point is set 1. Secondly, the set of points in the two-dimensional array $NewPolygonListPP$ is stored successively in another two-dimensional array $newPolygonList$; Then, the number of valid elements of the 2d array $NewPolygonListPP$ is stored successively in the one-dimensional array C (the valid elements are the vertices which are in the simple polygons); Fourthly, empty the two-dimensional array $NewPolygonListPP$; Lastly, mark the corresponding point of the next element of the one-dimensional array b in the two-dimensional array $PolygonList$ successively as the new A point and use it as the next detection point to start the detection again from step 1). The loop does not stop until all of the point access variables in the two-dimensional array $PolygonList$ are set to 1 [3].

Step 3.5: keep the valid polygons. Calculate the order of arrangement of each polygon in the two-dimensional array $newPolygonList$. And calculate the area of the simple anticlockwise polygon. The vertex set of the simple anticlockwise polygon with the maximum area will be reserved.

Step 4: Ensure that the arrangement direction of polygon vertex set before and after shrinking is the same. If the $1 == Clock\_Flag$ is satisfied, invert the set of

points returned in step 3.5.

2.3 examples of indenting algorithm implementation

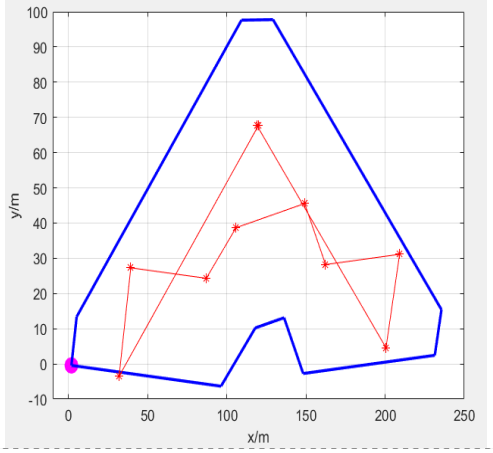

Figure 1. Comparison of the existing indenting algorithm before and after indenting.

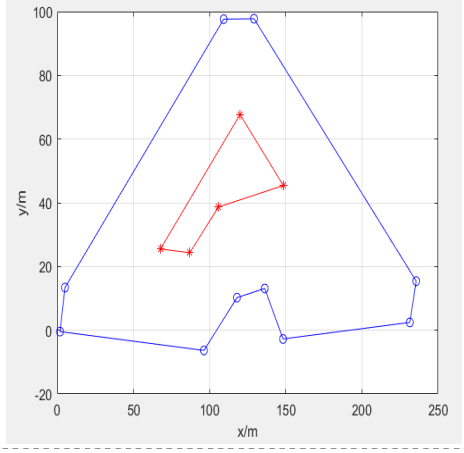

Figure 2. Comparison of the algorithm proposed in this paper before and after indenting.

The blue polygons in figure 1 and figure 2 are the polygons arranged clockwise starting from the coordinate $(0,0)$, and the red polygons in the two figures are the clockwise polygons shrunk 30 meters by using the existing indented method and the algorithm proposed in this paper respectively.

By comparing the two shrinking algorithms, it is found that the shrinking algorithm proposed in this paper can better deal with invalid polygons and valid polygons with small area, and effectively retain the main valid polygons, thus effectively reduce the difficulty of route planning caused by inaccurate sampling points.

2.4 Extended algorithm

On the basis of the vertex coordinates of the polygon and the sine value of the angle between two sides at each vertex, the overall outspread algorithm can be implemented. The steps are as follows:

Step 1: Calculate the cosine value of the angle between the unit vectors of two adjacent sides of each vertex and save it in one- dimensional array $\mathrm{Cos}\,a$. The cosine value of the last edge and the previous edge of each vertex can be calculated by the point multiplication of the unit vector of the last edge and the unit vector of the previous edge of each vertex.

Step 2: optimize small acute angles. If the sine value $\mathrm{Sin}\,a(1,i)$ and the cosine value $\mathrm{Cos}\,a(1,i)$ at each vertex satisfy the condition (2), then the vertex is replaced by three points when it expands out. The coordinates of these three points are able to be calculated by formula (3). Otherwise, formula (4) is used to calculate the new outspread vertex. The parameter $Theita$ in condition (2) is the angle value set according to the specific project needs, for example, you can set it to 15 or 10 degrees. In formula (3), the P_i is the original vertex coordinates; P_{ix} is the abscissa of P_i, and P_{iy} is the ordinate of P_i. Q_i, Q_{i+1} and Q_{i+2} are three new coordinates generated, similarly, Q_{ix} is the abscissa of Q_i, Q_{iy} is the ordinate of Q_i. NDP_i is the unit direction vector of the i^{th} side. L is the extended distance and we set it to 3.5 meters in our project. $\mathrm{Sin}\,a(1,i)$ and $\mathrm{Cos}\,a(1,i)$ respectively are the sine and cosine values of the i^{th} vertices. The parameters or variables in formula (4) are the same as those in formula (3).

$$\begin{cases} \mathrm{Sin}\,a(1,i) > 0 \\ \mathrm{Sin}\,a(1,i) \le \sin(Theita * 3.1415926/180) \\ \mathrm{Cos}\,a(1,i) < 0 \end{cases} \quad (2)$$

$$\begin{cases} p_i = -NDP_i + NDP_{i-1} \\ a_x = p_{ix} / \sqrt{p_{ix} * p_{ix} + p_{iy} * p_{iy}} \\ a_y = p_{iy} / \sqrt{p_{ix} * p_{ix} + p_{iy} * p_{iy}} \\ Q_{ix} = P_{ix} + L * \left(a_x * \sqrt{(1+\mathrm{Cosa}(1,\mathrm{i}))/2} - a_y * \sqrt{(1-\mathrm{Cosa}(1,\mathrm{i}))/2} \right) \\ Q_{iy} = P_{iy} + L * \left(a_y * \sqrt{(1+\mathrm{Cosa}(1,\mathrm{i}))/2} + a_x * \sqrt{(1-\mathrm{Cosa}(1,\mathrm{i}))/2} \right) \\ Q_{(i+1)x} = P_{ix} + a_x * L \\ Q_{(i+1)y} = P_{iy} + a_y * L \\ Q_{(i+2)x} = P_{ix} + L * (a_x * \sqrt{(1+\mathrm{Cosa}(1,\mathrm{i}))/2} + a_y * \sqrt{(1-\mathrm{Cosa}(1,\mathrm{i}))/2}) \\ Q_{(i+2)y} = P_{iy} + L * (a_y * \sqrt{(1+\mathrm{Cosa}(1,\mathrm{i}))/2} - a_x * \sqrt{(1-\mathrm{Cosa}(1,\mathrm{i}))/2}) \end{cases} \quad (3)$$

$$Q_i = P_i - (NDP_i - NDP_{i-1}) * L / \mathrm{Sin}\, a(1,i) \quad (4)$$

Step 3: removal of coordinates and removing collinear sides from polygons. This step is the same as steps 2 and 3 in the pre-processing process.

Step 4: keep outline. Start with the third side of the polygon and iterate successively, if the i^{th} side intersects the j^{th} side in the previous traversal from the first side to the $(i-2)^{th}$ side, then calculate the intersection point; and the set of points from the first vertex to the j^{th} vertex of the polygon is stored in the 2-d array $Points$ and the intersection points are also stored in the array $Points$. Lastly, the coordinates of the polygon from the $(i+1)^{th}$ vertex to the last vertex are also stored in the array $Points$ successively. Repeat this step until there is no intersections.

Step 5: the same third step.

Step 6: make the arrangement direction of polygon vertex set same before and after expanding. If the condition $0 == Clock\_Flag$ is satisfied, invert the set of points returned in step 5.

2.5 examples of implementation of outspread algorithm

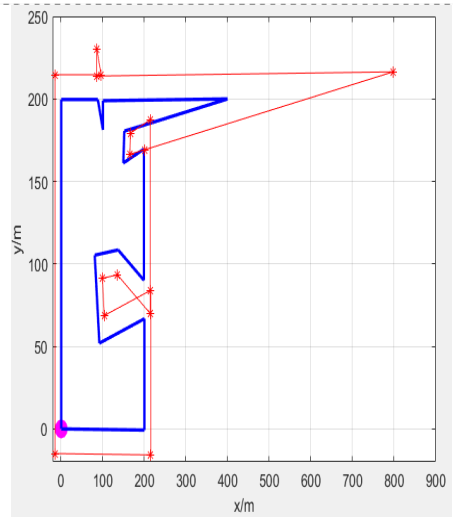

Figure 3. Comparison of existing methods before and after external expansion.

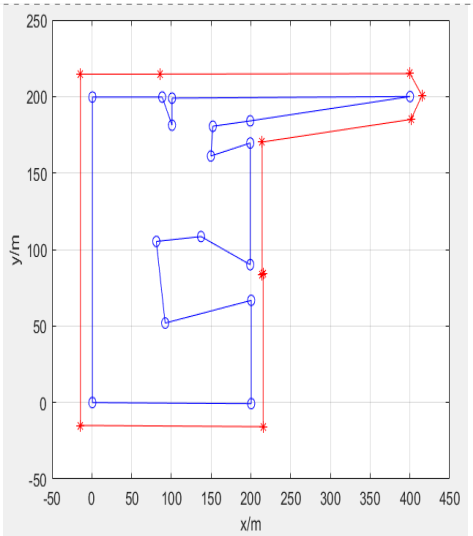

Figure 4. Comparison of the method proposed in this paper before and after external expansion.

The blue polygons in figure 3 and figure 4 are the clockwise polygons with coordinates $(0,0)$ as the starting point, and the red polygons in the two figures are the clockwise polygons expanded to 15 meters by the existing method and the method proposed in this paper respectively.

By comparing the two algorithms, it is found that the algorithm proposed in this paper can deal well with the case with small acute angles and the case with internal polygons resulted from the intersection of sides after expanding and the algorithm is able to preserve the external boundary contour effectively, thus effectively reduce the difficulty of route planning during the operation of UAVs.

3 Conclusion

In this paper, a universal method of integral shrinking and integral outspreading of polygonal block boundary is proposed. This method can eliminate the simple and invalid polygons caused by the polygon shrunk integrally and the phenomenon that the vertices of small acute angles extend too long and concave edges intersect to form complex inner polygons. Based on the method proposed in this paper, the boundary polygons of the operation area and the boundary polygons of obstacles can be well integrated to shrink or expand, which reduces the difficulty of UAVs route planning [6] and improves the operation accuracy and efficiency.

References

1. Luo, Z.Y., The impact of rural labour transfer on agricultural production efficiency in China[D]. Master. Chongqing Business University,2019.
2. He, F., Wu, L.Y., Lu, J.H. (2012) The power of Amalgamation -- Investigation Report on the land consolidation work of "small and large blocks" in Chongzuo City, Guangxi[J]. Journal of China Land. 2012(07):47-50.
3. Qi, X., 2019. Polygonal Plot Segmentation Method for Plant Protection of Rotor-like UAVs. In: 4th China Aviation Science and Technology Conference. Shenyang. 1204-1210.
4. Wu, Z.B., Zhao, C.X., Liu, B. (2019) Polygonal Approximation based on Coarse-grained Parallel Genetic Algorithm[J]. Journal of Visual Communication and Image Representation, 2019.
5. Zhao, J.B., Liu, W.J., Wang, Y.C. (2005) The algorithm of removing invalid ring in polygon OFFSET. Journal of engineering graphics, 03:44-49.
6. Erdi Dasdemir, Murat Köksalan, Diclehan Tezcaner Öztürk. A flexible reference point-based multi-objective evolutionary algorithm: An application to the UAV route planning problem[J]. Computers and Operations Research,2020,114.