Diploma Thesis

Coverage Path Planning in Non-Convex Polygon Areas for Orthophotomap Creation Using UAVs

Bc. Jan Bulušek

January 2017

Thesis supervisor: Ing. Milan Rollo, PhD

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student:	Bc. Jan B u l u š e k
Study programme:	Cybernetics and Robotics
Specialisation:	Robotics
Title of Diploma Thesis:	Coverage Path Planning in Non-Convex Polygon Areas for Orthophotomap Creation Using UAVs

Guidelines:

1. Study the theory of remote sensing and orthophotomap creation.
2. Study existing coverage path planning methods in non-convex polygon areas.
3. Choose the most convenient methods and modify them for deployment on unmanned fixed-wing aircraft.
4. Implement the methods and compare their performance on various polygons.
5. Integrate the methods into a framework for UAV command & control.
6. Verify the results in simulation with unmanned fixed-wing aircraft dynamics.

Bibliography/Sources:

[1] Franco, Carmelo Di, and Giorgio Buttazzo. "Energy-Aware Coverage Path Planning of UAVs." Autonomous Robot Systems and Competitions (ICARSC), 2015 IEEE International Conference on. IEEE, (2015): 111-117.
[2] Li, Yan, et al. "Coverage path planning for UAVs based on enhanced exact cellular decomposition method." Mechatronics 21.5 (2011): 876-885.
[3] Schenk, T. "Introduction to photogrammetry." The Ohio State University, Columbus (2005).

Diploma Thesis Supervisor: Ing. Milan Rollo, Ph.D.

Valid until: the end of the summer semester of academic year 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
Head of Department	**Dean**

Prague, January 5, 2016

Acknowledgement

I would like to thank all the people who helped me with my diploma thesis in any way. I thank my thesis supervisor Milan Rollo for valuable comments and remarks he had given me during the creation of this thesis. My thanks also goes to members of the Artificial Intelligence Center from the department of Computer Science, Czech Technical University, for all the help and patience they provided me during my work. Special thanks goes to my family and friends for supporting and encouraging me.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date January 8, 2017 ....

Abstract

Cílem této práce je implementace algoritmů pro pokrývání nekonvexních oblastí s využitím UAV (Unmanned Aerial Vehicle) a jejich experimentální ověření (jak v simulacích, tak na reálném bezpilotním prostředku). Z existujících algoritmů pro plánování příslušné letové trajektorie byly vybrány tři offline algoritmy využívající exaktní dekompozici oblasti – naivní, Enhanced Exact Cellular Decomposition planner a Energy-Aware planner. Zmíněné plánovací algoritmy byly dále upraveny tak, aby respektovaly kinematická omezení letounu a aby byly schopny plánovat i nad oblastí s přítomnými bezletovými zónami. Algoritmy byly včleněny do řídícího systému pro UAV a otestovány na sérii polygonálních oblastí se stupňující se složitostí. Algoritmy jsou následně na základě svého výkonu porovnány a dále analyzovány.

Klíčová slova

UAV; plánování trajektorií; plánování pokrývacích trajektorií; pokrývání nekonvexních oblastí

Abstract

The goal of this work is to implement coverage path planning algorithms over non-complex areas using UAVs (Unmanned Aerial vehicle) and their experimental verification (in a simulated environment as well as in the real one). Three coverage path planning algorithms employing the offline exact decomposition approach were chosen – the Naive planner, the Enhanced Exact Cellular Decomposition planner and the Energy-Aware planner. These planners were further modified to take into account UAV's kinematic constraints and to be able to plan over areas with no-flight zones present. These algorithms were integrated into a framework for UAV command & control and tested on a series of polygonal areas with increasing complexity. The selected algorithms are further analyzed and compared according to their performances.

Keywords

UAV; trajectory planning; coverage path planning; non-convex area coverage

Contents

1. Introduction **1**
1.1. Orthophotomaps 1
1.2. UAV 2
1.3. Tactical AGENTFLY framework 3

2. Basic concepts **4**
2.1. Coverage path planning 4
2.2. Aerial remote sensing 4

3. Aerial coverage algorithms **7**
3.1. Previous work 7
3.2. Selected approaches 9
3.2.1. Naive exact decomposition 9
3.2.2. Enhanced exact cellular decomposition 10
3.2.3. Energy-aware path planner 13

4. Algorithm implementation and modifications **17**
4.1. Common parts of algorithms 17
4.1.1. Polygon area storing 17
4.1.2. Convex area coverage 19
4.1.3. Stripe computation 20
4.2. Naive exact decomposition planner 21
4.3. Enhanced exact cellular decomposition planner 21
4.3.1. Polygon width computation 21
4.3.2. Visiting order determination 22
4.4. Energy-aware planner 22
4.4.1. Concave parts separation 23
4.4.2. Multiple paths problem 29

5. Algorithm enhancements **30**
5.1. Kinematic constraints application 30
5.2. No-flight zones implementation 36
5.3. Limited-range planner 39

6. Algorithm analysis and results **44**
6.1. Tactical AGENTFLY interface overview 44
6.1.1. UAV properties 45
6.1.2. Measurement tools 45
6.2. Software tests 46
6.2.1. Convex polygon coverage test 46
6.2.2. Simple concave polygon coverage test 50
6.2.3. Complex concave polygon coverage test 53
6.2.4. Polygon with no-flight zones coverage test 56
6.3. Field experiments 59
6.3.1. Convex polygon coverage 60
6.3.2. Concave polygon coverage 62
6.4. Test conclusions 64

7. Conclusion **65**
7.1. Thesis summary 65
7.2. Future work 66

Appendices

A. Contents of the enclosed CD **67**

Bibliography **70**

1. Introduction

Even though we live in a world where there are no entirely unmapped places left, the process of mapping an area still holds its relevance. For various reasons, such as agriculture crop surveillance, aerial archeology, photogrammetry or orthophotomap creation, it is desirable to update or to create more accurate map of certain area for further analysis.

One of the possible approaches is to take a series of aerial images of an area of interest. This process, being rather expensive in the past because of the need for manned aircraft (or even satellite imagery), got easier with the emergence of unmanned aerial vehicles (UAVs), which allowed to perform terrain mapping in reasonable time at relatively small cost. Also, with the possibility of flying in lower altitudes the mapping quality has increased. With the current rise of automated drones there is now a possibility to automate this process and thus to further reduce the cost of mapping.

Because the operation time of UAVs is limited, the effectiveness of path planning coverage algorithm is vital since it decreases the time necessary to perform the desired sweep. The motivation of this thesis is to contribute to this effort by comparing several coverage algorithms for fixed-wing UAVs on the Tactical AGENTFLY framework in order to choose the most efficient one for further usage.

In this chapter, I am going to briefly introduce objects of this thesis. Chapter 2 explains additional concepts that are needed for further discussions. Chapter 3 describes and explains the previous work made in this field along with the selected algorithms which are to be implemented. Chapter 4 describes the modifications made on the mentioned algorithms. Chapter 5 shows the improvements made to these algorithms beyond the original specifications. Chapter 6 presents the experimental results and their further analysis. Chapter 7 contains the thesis conclusion and future work suggestions.

1.1. Orthophotomaps

Orthophoto is an aerial image geometrically corrected (orthorectified) such that the scale is uniform, i.e. the image has the same lack of distortion as a map. This correction consists of adjusting to terrain relief, camera lens distortion and camera tilt. Unlike the perspective (oblique) images, it can be used for measuring real distances between objects due to the mentioned lack of distortion.

Ortophotomap or *ortophotomosaic* is a map created by merging orthophotos into one complex image. An example of such a mosaic is the Google Earth virtual globe surface.

According to [1], there is a five-step process of creating a digital orthophotos:

- **Aerial photography** for acquiring the imagery;
- **Ground control** in order to precisely match the images to sets of coordinates;

- **Image scanning** resulting in a continuous-tone image;
- **Digital elevation model production** (determining the ground elevation);
- **Image rectification** (correcting images for the aforementioned ground elevation).

The quality of the orthophoto depends, among other factors, on the recorder parameters. The resolution of the final orthophoto imagery is dependent on that quality as well as density of scanned images.

In this thesis, we are primarily focusing on the image acquisition combined with ground control. The latter image processing is out of the scope of this work.

1.2. UAV

As was already mentioned, the UAV abbreviation stands for Unmanned aerial vehicle. As the name suggests, it is an aircraft remotely controlled by human or with a certain autonomy.

As is presented in [2], UAVs can be of various forms and construction. There are two dominant categories nowadays, fixed wing and rotory wing. Each of these categories could be further subdivided according to their flight characteristics, but for our purposes it is sufficient to distinguish between these two.

A **fixed-wing UAV** consists of a rigid wing that generates lift by its forward movement. The forward movement is generated usually by a propeller attached. The overall movement is controlled by control surfaces built into the wing itself, such as ailerons, elevator and/or rudder. Example of such an aircraft can be found in Figure 1.

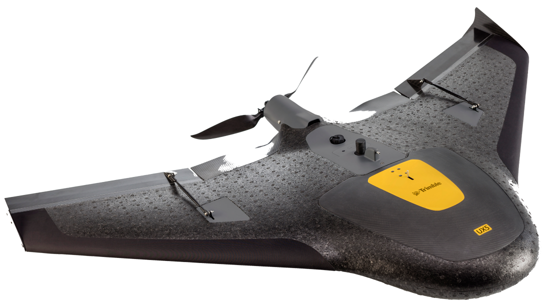


Figure 1. Fixed-wing UAV (Trimble UX5).

Rotory wing UAV is an aircraft with lift generated by one or multiple rotors. There is a variety of possible constructions depending on the number and positioning of each rotor – there are helicopters, quadcopters, hexacopters etc. On the contrary to fixed-wing UAVs, since the lift is generated by rotor movement, no forward movement is necessary. Example of this aircraft type can be found in Figure 2.

Figure 2. Rotory wing UAV (Parrot AR.Drone 2.0).

The advantages and disadvantages of each class are dependent on their construction. Simpler construction of the fixed-wing aircraft ensures less complicated maintenance and, more importantly, more efficient aerodynamics that provides us with longer flight duration and/or greater payload available on less power than the rotory wing UAVs. The main rotory wing UAV advantage is its VTOL (vertical take-off and landing) capability as well as the ability to hover and greater maneuvering options in general.

1.3. Tactical AGENTFLY framework

According to [3], AGENTFLY is a multi-agent system being developed at department of Computer Science at Czech Technical University, Faculty of Electrical Engineering. It enables large-scale simulations of civilian and unmanned air traffic. The system integrates advanced flight path planning, decentralized collision avoidance with detailed models of the airplanes and the environment.

Tactical AGENTFLY (or shortly **TAF3**) is an ongoing U.S. Army-sponsored research project developing agent-based coordination and planning techniques for multi-UAV information collection missions with special emphasis on complex urban environments.

2. Basic concepts

In order to fully discuss the topic, there are several terms and principles that need to be explained first.

2.1. Coverage path planning

In order to create an orthophotomap of an entire area, orthophotos have to be taken at various locations in such a way that every part of the area is captured on at least one photo. The coverage path planner needs to identify these locations and to determine their visiting order.

While the location identification process is throughly studied and various approaches are available (as seen in the Section 3.1), the effective visiting order determination is much more complex task. The effectiveness requires visiting each of these locations once while maintaining the shortest flight trajectory possible. This problem is often referred to as The traveling salesman problem (TSP) and as stated in [4], it can be described as a search for permutation $P = (1 \quad l_2 \quad l_3 \quad \dots \quad l_n)$ – i.e. the visiting order – that minimizes the sum of a given set of real numbers $\sum_1^{n-1} a_{l_k l_{k+1}}$ – distances between locations.

The traveling salesman problem is NP-hard. For a small amount of locations n, it can be solved using brute-force algorithms, but this is not applicable for larger n. As such, usually only approximate solutions are used (on the contrary to location identification).

2.2. Aerial remote sensing

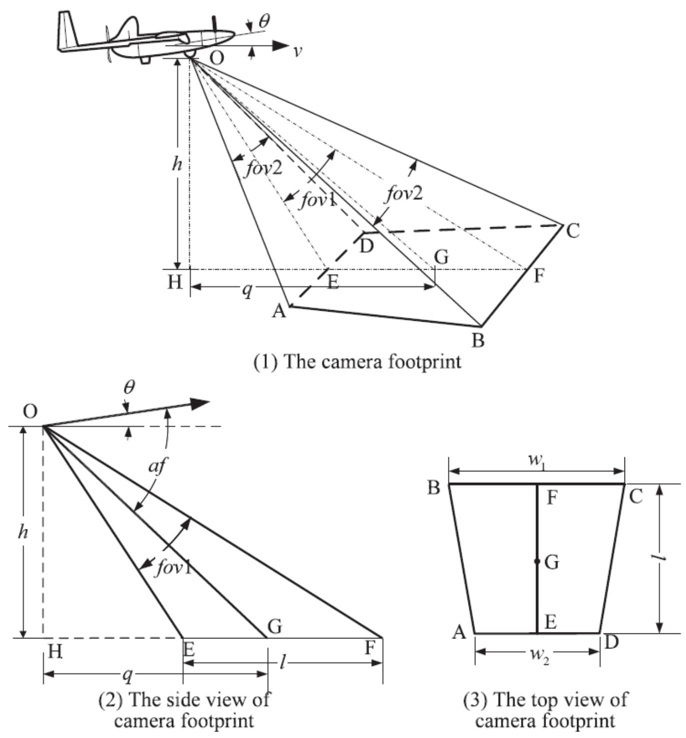
The sensing apparatus necessary for this task consists of a positioning device such as GPS and a camera capable of taking pictures of the desired properties.

As [5] suggests, the goal is to get detailed photos as sharp as possible. Blurry images caused either by the motion or by shallow depth-of-field are not suitable for the task since blurred details confuse the software. The blur is reduced by closing up the aperture. The image sharpness is improved by using high-definition cameras and shooting in RAW image format. The RAW file format is the full unprocessed output of the camera sensor without any compression applied and as such provides more options regarding the image processing. The camera should be able to take several photos in a continuous shooting mode (at least 1 photo every 2 seconds). Also, because of the fact that it is supposed to be carried by an UAV, the camera construction should be light-weight.

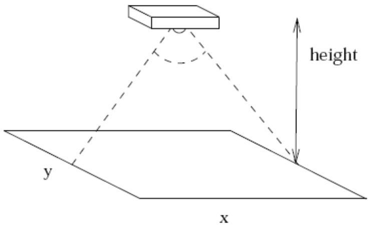
Examples of suitable cameras are listed below:

- **Canon**: S110, SX260
- **Sony**: QX1,DSC-RX100 A7R, A7, A7S, NEX-6, NEX-5R, NEX-5T, A5100
- **Panasonic**: GH3

The **projection area** is the term for an area that gets covered by a single photo taken by a UAV. As is suggested in [6], the projection area shape differs with the orientation of camera axis. In the most cases, such as in [7], the camera axis is assumed to be vertical or near-vertical, creating the rectangular or close to rectangular projection area. Certain authors, such as [8], presume the camera to be tilted towards the movement direction, creating oblique photographs. This in effect shifts the rectangular area into trapezoidal shape, but the wider part is omitted nonetheless due to the possibility of deformations, accepting back the rectangular shape (see Figure 3).



a)



b)

Figure 3. Projection areas construction. Stereoscopic image of trapezoidal shape is obtained if the camera is tilted towards the movement direction (a). The dimension w_2 is taken as sweep breadth, omitting the wider part of the trapezoid (corners C, D). In case of perpendicular mounting of a camera the projection area computing is simpler (b). Images are taken from [8] and [7] respectively.

Apart from the camera mounting on the UAV, the projection area is dependent mainly on two parameters: camera's field of vision and the altitude UAV flies at. With the camera axis considered to be aligned vertically to the ground, the dimensions of the projection area can be computed using trigonometry, as is shown in equations 1 and 2.

$$L_x = 2h \cdot \tan(\frac{\alpha}{2}), \tag{1}$$
$$L_y = L_x \cdot r. \tag{2}$$

In these equations, L_x and L_y denote the width and length of the projection area (with *length* corresponding with the height of the photo). h denotes the altitude and α means the camera's field of vision. r is the camera's aspect ratio, i.e. the ratio between the camera's width and its height.

The **spatial resolution** is the number of points in the photo corresponding to a ground distance. This number describes the required quality of the resulting orthophotomap and creates a constraint for the flight altitude according to formula 3.

$$h \leq \frac{I_x}{2R \cdot \tan(\frac{\alpha}{2})} \tag{3}$$

Here R denotes the required spatial resolution measured in pixels per meter and I_x the width of a photo taken by the camera measured in pixels. The inequality reflects the fact that the lower the altitude is, the better spatial resolution is obtained.

The **image overlap**, usually stated in percents, describes the required overlapping of the adjacent photos. Because of varying dimensions of photos, the image overlap is divided into forward and side overlap. These parameters (when converted from percentages into numbers ranging from 0 to 1) combined with the projection area dimensions give us the minimal distance between the photos taken:

$$d_x = L_x \cdot (1 - ov_x), \tag{4}$$
$$d_y = L_y \cdot (1 - ov_y). \tag{5}$$

where ov_x and ov_y in formulas 4 and 5 describe the side and forward overlap respectively. The resulting distances d_x and d_y create a constraint for the waypoint generation during the planning process.

3. Aerial coverage algorithms

3.1. Previous work

The problem of covering an area by a UAV is directly related to Coverage path planning problem (as displayed in Section 2.1). Unlike the general CPP, the resulting trajectory for UAV has to be subjected to additional constraints depending on the aircraft's flight capabilities, namely its maximal flight duration and maneuvering capabilities. The aim to reduce the flight time duration requires us to perform as efficient sweep as possible, avoiding already visited areas if possible. The maneuvering capabilities constraint expressed as aircraft's turning radius affects the shape of the resulting trajectory. To minimize this effect, it is desirable to reduce the number of turns within the trajectory, and thus to prefer straight flight elements over curvatures. However, certain factors are neglected for the sake of task simplicity, such as unfavorable weather conditions (e.g. windy environment).

There are two main approaches towards the algorithm execution: **online** and **offline**. Online algorithms are running on the aircraft's onboard computer in real time, adjusting the flight trajectory to static or even dynamic obstacles. This approach is more robust since it can react even to unpredicted conditions, but the drawback here is the increased onboard computational power required as well as the sensory apparatus necessary to perform the task. Also, due to its reactive structure, the efficiency of the resulting trajectory is limited. This approach was used by the authors of [9] or [10] for frontier-based exploration; another approach, potential field navigation, is presented by the authors of [11].

Offline algorithms, on the other hand, have increased trajectory efficiency, but they require complete information about the area (e.g. obstacles) prior to the sweep and can not react to a sudden change of conditions, such as the presence of another aircraft. Typically, they subdivide the area into smaller, less complex subareas which are covered using one of the simple sweep patterns. The entire area is then covered by visiting and covering all the subareas. There are two main categories regarding the subdivision: **exact** and **approximate** (see Figure 4).

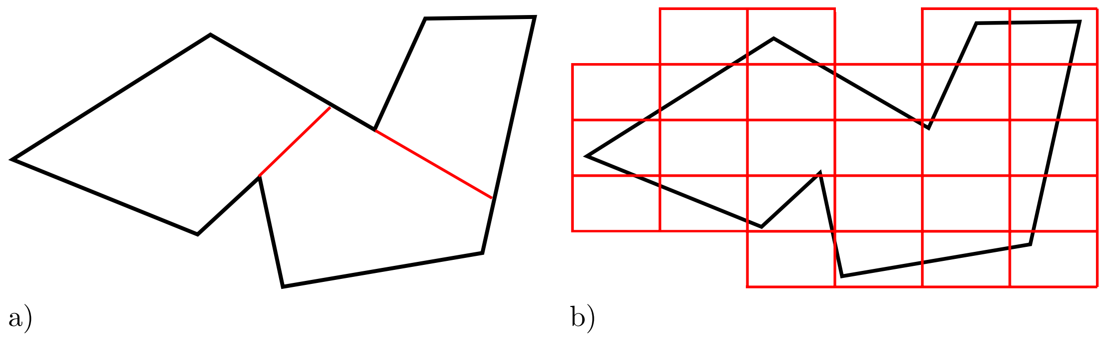


Figure 4. Exact decomposition (a) vs. approximate decomposition (b).

The approximate approach approximates the area by the grid of regular, usually rectangular cells (hence the name). The size of each cell is usually chosen in such a way that the UAV is able to cover it entirely from the position in its center. While this approach clearly does not require any sweep pattern to be used for the cell sweeping and is able to process even complexly-shaped areas, its drawback lies on the fact that the resulting coverage is very likely to cover parts of the outside regions, reducing the overall effectiveness. Algorithms based on this approach focus mainly on the cell visiting order. Numerous paths were taken to solve this problem; authors of [12] evaluate each cell according to its distance to the goal cell. In [13], the authors propose the usage of the wavefront planner while minimizing the number of steep turns by a cost function. Similar cost function is used in [14], where also the UAV's kinematics is considered by adding Dubins curves. Another approaches such as ant pheromones in [15], evolution algorithms in [16], or neural networks in [17], are also used.

The exact approach, on the other hand, decomposes the area into the set of subareas, union of which gives us exactly the original area. These subareas are usually of convex shape in order to sweep them with a simple pattern (e.g. back-and-forth motion known as the Lawnmower pattern); as such, the main goal of this subdivision is to obtain the smallest number of subareas possible to minimize the traveling salesman problem. The authors of [18] suggest the simple Boustrophedon decomposition for the use. [19] presents a cell decomposition based on the dynamic programming techniques. Another approach is used in [8], where the subdivision based on the geometrical properties of resulting subareas is proposed. The authors of [20] take into account the 3D properties of the scanned area. An uncommon approach is used in [7], where the subdivision based on the current flight trajectory is performed.

According to [21], the cell coverage for the exact approach can be done using several sweep patterns. The paper mentions spiral, spiral-like, lawnmower and zamboni patterns. With respect to the conditions stated in the beginning of this chapter, the most preferred pattern is the Lawnmower pattern, which employs the back-and-forth motion consisting of straight elements only.

3.2. Selected approaches

3.2.1. Naive exact decomposition

The naive exact decomposition planner employs a simplistic approach of subdividing the concave polygon area into a set of convex cells regardless of their other geometric properties. Each of these cells is covered using the Lawnmower pattern with the leading direction (see below) perpendicular to its polygon width to minimize the number of turns. The overall visiting order is determined by greedy search based on cell's proximity to UAV's actual position.

The decomposition process is done as follows: the algorithm iterates through polygon area vertices and from the location of its neighboring vertices the concavity is determined. If the vertex is found to be concave (its inner angle γ is greater than π rad), one of the connections with its neighbors is prolonged and performs a cut in the polygon. This action divides the angle into a straight one (π rad) and $\gamma - \pi$ rad, which has to be smaller than π rad and thus is not concave anymore. After inspecting all the polygon vertices, the area is successfully subdivided into a set of convex areas. An example illustration of such decomposition can be found at Figure 4a.

The algorithm structure is as follows:

1. Iterate through the polygon's vertices (V). Take both its neighbors and determine the concavity. If no concavity is detected, continue to the next vertex.
2. If concavity is detected, prolong the line connecting V and one of its neighbors to the inside of the polygon. Find the intersection (I) of this line with the polygon border closest to V.
3. Add the new edge connecting V and I to the polygon. Continue to polygon's next vertex.
4. Repeat steps 1-3 until all the original vertices are processed.
5. Subdivide the polygon into a set of convex cells along newly formed edges.

When the subdivision task is completed, the visiting order needs to be established. The Naive planner makes its decisions based on UAV's current position and simply selects the closest unsurveyed cell. The cell proximity to a location is resolved as the distance between this location and closest vertex of the cell. The resulting greedy search algorithm's structure is as follows:

1. Initialize the starting position P.
2. From a set of non-surveyed cells pick the one closest to P (C).
3. Sweep C using the Lawnmower pattern. Update P and remove C from the cell set. If the cell set is not empty, go to step 2.

Regarding the exact decomposition approaches in general, a sweep pattern needs to be used to sweep respective cells. As already mentioned, the most commonly pattern used is the back-and-forth motion, i.e. the **Lawnmower pattern**. As the name suggests, this flight pattern employs visiting a sequence of straight lines placed in parallel; when the endpoint of one line is reached, the aircraft flies to the beginning of another and flies along it usually in the opposite direction than the previous one (see Figure 5).

This pattern is favored mostly because of its effectiveness, simplicity and large portion of straight flight lines, which are generally more plausible for photogrammetric purposes.

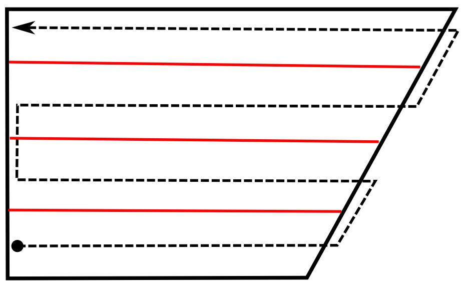


Figure 5. The trapezoidal cell coverage by the Lawnmower pattern and its decomposition into stripes. The area gets decomposed along the red lines. Each created stripe gets covered by flying along its dominant direction. The actual coverage movement is illustrated by the dashed line.

As displayed in this figure, there is one **leading direction** (i.e. the direction of stripe division) that describes the pattern. Each flight line then defines the **stripe** getting covered by following that line.

The illustration image also shows that in order to maintain the straight flight paths inside of the area of interest, the aircraft is permitted to take the turns necessary to reach the next stripe only after the current stripe is fully covered. Depending on the area's shape, it may also result in the aircraft reaching out of the area of interest.

The pattern's leading direction for the Naive exact planner can be set in parallel with the longest borderline (similarly to the approach displayed in Section 3.2.3) or further computations may be used (e.g. the polygon width similarly to the approach displayed in Section 3.2.2). To increase the algorithm effectiveness the latter approach was chosen.

3.2.2. Enhanced exact cellular decomposition

The Enhanced exact cellular decomposition approach (**EECD**), presented in [8], is quite similar to the Naive one (see Section 3.2.1) in terms of structure. However, its subtasks are treated differently.

Firstly, the polygon subdivision approach takes into account the geometrical properties of the resulting set of cells. To ensure that this set is more favorable for the sweep, a new cutting technique that produces cells with dominant directions is presented. The algorithm considers all the concave vertices present in the current part of the polygon and tries cutting from all of them in the directions of all border lines. For each cut, the sum of polygon widths (see below) of the resulting pair of cells is computed. After all the possible cuts are computed, the subdividing algorithm performs the one with the minimal sum. This process is repeated until all the concave vertices are resolved.

```
Data: polygon P
Result: set of convex cells C
C = {};
C.add(P);
for cell C_i in C do
    if C_i has concave vertices then
        minimal_width = MAX_VALUE;
        vertex_index = 0;
        line_index = 0;
        for concave vertices i do
            for border lines j do
                erupt a support line from i parallel to j;
                connect i with the closest reachable intersection;
                (SP1, SP2) = the new sub-polygons;
                width(i,j) = width(SP1)+width(SP2);
                if width(i,j) is smaller than minimal_width then
                    minimal_width = width(i,j);
                    vertex_index = i;
                    line_index = j;
                end
            end
        end
        cut the polygon in vertex_index with line parallel to line_index;
        C.add(c1);
        C.add(c2);
        C.remove(C_i);
    end
end
```

Algorithm 1: EECD subdivision method.

The function width(SP) in this algorithm computes the width of the *SP* polygon and is further described below.

The resulting set of cells is still not necessarily optimal, but their attributes are generally more favorable for our task. In order to further optimize this set, this decomposition method reduces the number of sub-polygons by applying the recombination process. As could be seen in Figure 6, when covering two adjacent sub-polygons by the Lawnmower pattern (displayed in Section 3.2.1) it is possible that the resulting flight trajectory is slightly longer than while covering one polygon of the same size. For this reason, it is possible to merge the adjacent polygons to create a concave one, but only if the leading flight direction (which is perpendicular to the polygon width – see below) in these polygons is the same. Even if the resulting merged cell is concave again, similarly to an example in Figure 6, the employed Lawnmower pattern is able to handle these types of concavities.

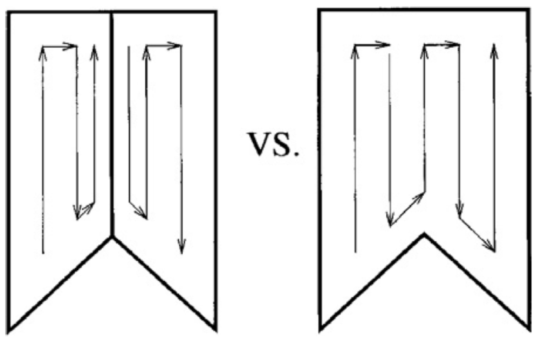


Figure 6. Flight length difference for fewer sub-polygons (taken from [8]).

To determine the visiting order, the set of sub-polygons is converted into non-oriented graph. Each node of this graph represents one sub-polygon while the edges are coincident with adjacency of the respective sub-polygons. The task to find the visiting order now transforms into traversing through the graph while visiting each node once. Depending on the nodes configuration this may or may not be possible, in which case we transform the task into visiting every node at least once.

Since the Lawnmower pattern is used for sweeping sub-polygons, the inner trajectory inside every cell consists of a set of stripes. As such, the trajectory can be entered from two peripheral stripes and exited in the opposite one (see Figure 7). The entering stripe is accessible from two directions, which gives us for each sub-polygon four possible trajectories in total. The graph edge weight is then computed as the minimal distance between the adjacent cells d_{ij}. In case the pair of cells is not adjacent, the edge weight is set to ∞.

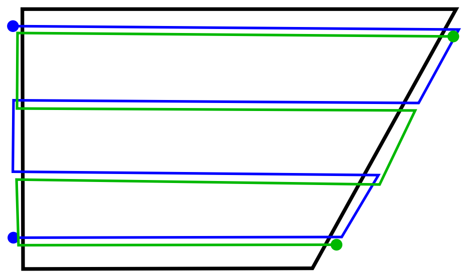


Figure 7. Two possible paths for Lawnmower pattern (blue and green) consisting of the same set of stripes. Each path is accessible via two entry points marked as dots.

The visiting order is determined by a recursive function employing dynamic programming:

$$f(i, S) = \min_{i \in S} f(j, S \backslash \{j\} + d_{ij}). \tag{6}$$

In this formula, $f(i, S)$ is the overall cost function of traversing a set of cells S with i elements inside. The terminal condition $f(0, \emptyset)$ is set as the initial distance of the UAV to the nearest cell. The visiting order is derived from the overall cost function minimalization.

When the visiting order is determined and the sweep of respective cells is performed, the UAV is supposed to fly even outside of the mapped area to ensure the full coverage of each stripe. [8] comments only briefly on this topic though, as it focuses on other planner performances.

Polygon width

The polygon width is its minimal span. There are numerous ways to compute it, e.g. the Rotating calipers method. It should be pointed out that the width computation for concave polygons can be done by computing over their convex hull (i.e. the smallest convex polygon that contains the selected concave polygon), which is also the task already solved (quick hull algorithm, Melkman's algorithm, Lee's hull algorithm etc.). All these algorithms are explained for example in [22].

Authors of [8] suggest for this subproblem a method with lower computational complexity. The main ideas are:

- To compute the span, it is necessary to measure the distance of every point from a given border line. The span is the maximal distance of the resulting set.
- It is not necessary to measure all the distances; for convex polygon it is sufficient while going along the border to measure the distance while it is increasing. When the distance starts decreasing, we have passed by the antipodal vertex and no other distance can be greater than the one already measured.
- To get the minimal span, all of the border lines have to be processed. To find the antipodal vertex however, we only need to increment from the antipodal vertex from the previous computation and not along the whole border.
- Polygon width is obtained as the minimal value of all the computed spans.

3.2.3. Energy-aware path planner

The algorithm presented in [7] employs the Lawnmower pattern to sweep the entire area. The main difference from other algorithms discussed here is that it does not analyze the area in advance in the depth others do.

In order to use the Lawnmower pattern, as discussed in Section 3.2.1, a leading direction has to be chosen. The presented approach chooses it simply by finding the area's longest borderline with the leading direction setting parallel to that. This longest borderline also serves as the starting point for the pattern; in case it is not coincident with the UAV's real starting position, the UAV is supposed to travel to this location along the area's border first. The part of the polygon that gets surveyed this way is subtracted from the main area that is being swept by the Lawnmower pattern.

While traversing along the border, the following approach to cover all the neighboring area is used. The UAV travels not only the path determined by adjacent vertices distance $|v_i - v_j|$, but also the adjustment dependent on the slope of the next borderline, as seen at Figure 8.

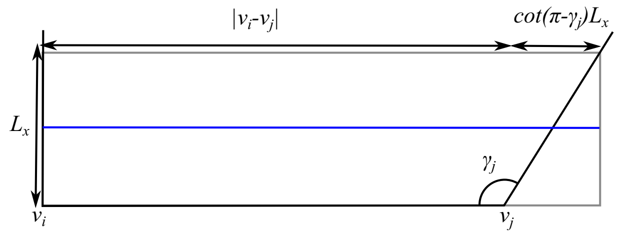


Figure 8. Adjusting the border traveling stripe according to the slope of the next borderline.

The flight stripe gets covered by photos taken from several specific locations. Since the aim is to reduce unnecessary coverage of the outlying area, waypoints enclosing the stripe on both sides are placed in the full distance from the area's border, i.e. $\frac{L_y}{2}$. To fully cover the rest of the stripe with the desired forward overlap, the distance Δy between m inner waypoints is computed using formulas 7 and 8:

$$m = \lceil \frac{d}{d_y} \rceil, \tag{7}$$

$$\Delta y = (d - L_y)(m - 1) \tag{8}$$

Here d denotes the stripe length, d_y is the minimal span between photos locations in forward direction (from equation 5) and L_y is one of the projection area's dimensions (computed using the equation 2). By using the ceiling function the satisfaction of the required overlap is ensured, since the result may produce even greater one.

The main sweep is organized in a matter similar to a convex area discussed above. The number of stripes is determined by formula 9, where d marks instead of the polygon width the polygon span measured from the longest borderline. Stripes neighboring the starting borderline and the farthest vertex are placed in the full distance from the area's border, i.e. $\frac{L_x}{2}$, while the rest of the stripes are placed equidistantly between them. The inner distances Δx are computed using formula 10. See Figure 9 for illustration of stripes placement.

$$n = \lceil \frac{d}{d_x} \rceil, \tag{9}$$

$$\Delta x = (d - L_x)(n - 1) \tag{10}$$

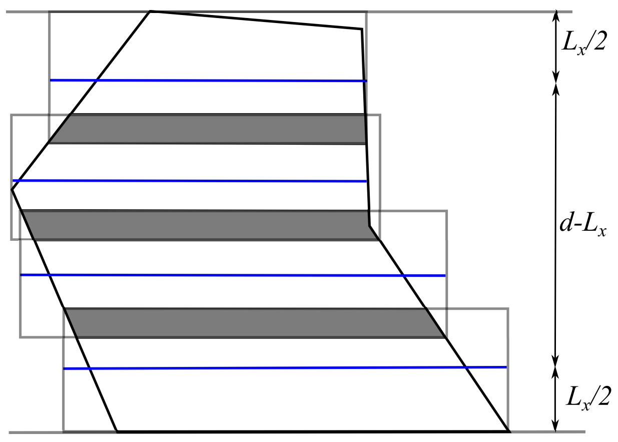


Figure 9. Stripes placement illustration.

Similarly to stripes placing, the waypoint placing inside each stripe (marking the locations the photos should be taken at) is performed. To ensure that all of the area is covered within the stripe, the stripe length is adjusted similarly as in the case of border traversing (see Figure 8). Once the stripe length is set, the waypoint generation process is performed in the same manner as with the border flight stripe, i.e. using formulas 7 and 8.

When the UAV reaches the pattern's end, it is supposed to return to its starting position. In case its current position is not coincident with the position required, it flies towards that position by following the borderlines, similarly to the beginning phase. The part of the area that gets surveyed this way should also be subtracted from the area reserved for the main sweep.

To handle possible concave areas during the sweep an uncommon approach is used – for each stripe the pair of edges being crossed is stored and compared with the previous one. If the edges from the actual pair do not match the previous one (or edges adjacent to them), then a concave area stretched between the previous and actual edges is detected. A new area covering the non-surveyed part of the polygon is formed and the algorithm is run recursively there from the nearest point of the main trajectory. An example of the procedure is displayed in Figure 10.

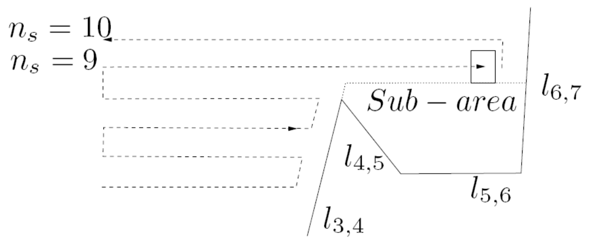


Figure 10. Handling concavities during on-flight decomposition (taken from [7]).

To sum it up, the algorithm structure is as follows:

1. Find the longest borderline of the area of interest. Travel there along the border if necessary; in that case also subtract the visited area from the rest of the sweep.

2. Compute the endpoint of the Lawnmower pattern starting from the longest borderline with the leading direction parallel to it. Connect it with the desired ending location of the UAV. If the travel along the border is needed, subtract the area to be visited from the rest of the sweep.
3. Start a Lawnmower pattern sweep across the area with the leading direction parallel to the longest borderline.
4. During each stripe compare its endpoints with the previous one. If there is a difference, isolate a concave subarea and recursively run a sweep there from the location closest to the subarea (L). After the sub-sweep is completed, return to L and continue the main sweep.

4. Algorithm implementation and modifications

For this thesis, the following algorithms from those listed in Section 3 were implemented for testing and comparison:

- Naive exact decomposition planner (from Section 3.2.1);
- Enhanced exact cellular decomposition planner (from Section 3.2.2);
- Energy-aware planner (from Section 3.2.3).

Due to theoretical structure of the listed algorithms several changes were made in order to make them usable. Also, certain parts of these algorithms were not displayed at all, since they are not directly related to the coverage problem, but are still critical for the correct functionality of these algorithms (e.g. the polygon area storing).

4.1. Common parts of algorithms

Certain parts of the implementation are shared by multiple coverage algorithms, such as the area storing. For this reason are these parts listed separately here.

4.1.1. Polygon area storing

In order to perform rather complex tasks over the area like its subdivision into several subpolygons or merging them back we need adequately complex data structure. For this reason, a structure similar to Doubly connected edge list (DCEL presented in [22]) was implemented.

Similarly to DCEL, the resulting structure (which we call the **hybrid DCEL** or shortly **HDCEL**) consists of sets of vertices, edges and faces. Additionally, it also contains a graph of its subpolygon structure as well as the border of the entire area.

The set of vertices is carried as a Hash map encoded by the vertex number (which is determined by the number of vertices already processed). Each vertex carries the information about its physical position as a Point.

The set of edges is also designed as a Hash map encoded by the pair of vertices it belongs to. Each edge carries the information about the mentioned belonging pair of vertices and about faces it is adjacent to. To distinguish which side of the edge belongs to which face, the pairs of vertices and edges are ordered; if we go from the first vertex to the second one, then the first face is on the left of this edge and the second one on the right (see Figure 11). This approach marks the main distinction to the original DCEL since there is no need for double edges anymore.

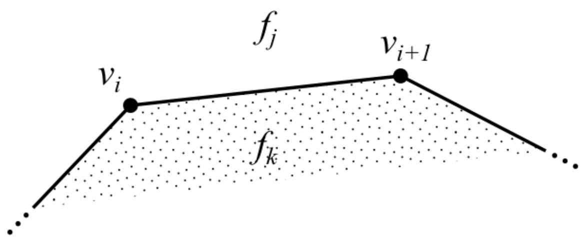


Figure 11. Hybrid DCEL edge illustration. When inspecting the edge (v_i, v_{i+1}), the returned set of faces is (f_j, f_k). On the contrary, if inspecting the edge (v_{i+1}, v_i), the returned set of faces is (f_k, f_j).

The set of faces is designed as a Hash map as well and it is encoded by the face number, which is determined similarly to the vertex number by the number of faces already processed. Each face carries the list of vertices that lie on its border in counter-clockwise order (carried as a Linked list) and a number that determines its type – common polygon, a hole inside another, processed etc.

There are two key operations that we require – subdividing polygons and their merging. The subdivision is performed from one already existing polygon vertex (V) into a newly created one (N) laying on one of polygon's edges (E). These vertices form a new edge between newly created pair of faces and set a dividing line between vertices of the original face (F). Both subsets of vertices are then formed into new faces along with the new pair, and all the edges belonging to the original face are updated. The whole process is displayed in algorithm 2.

```
Data: Face F, cutting Vertex V, cutted Edge E, new Vertex N
face F1, F2;
for Vertex ver in F do
    Vertex next = ver.next;
    Edge current_edge = (ver, next);
    if ver is between V and E.first_vertex then
        F1.add(ver);
        current_edge.face1 = F1;
    end
    else
        F2.add(ver);
        current_edge.face1 = F2;
    end
end
F1.add(N);
F2.add(N);
EDGES.add(new Edge(V,N,F1,F2));
FACES.add(F1);
FACES.add(F2);
FACES.delete(F);
```
Algorithm 2: Polygon cutting method.

Polygon merge is done inversely to the polygon subdivision. The edge (E) connecting two adjacent faces (F1, F2) is deleted. Sets of vertices of both polygons to be merged are combined. All the edges of the original pair are updated. The pseudo-code of the operation can be found as algorithm 3.

```
Data: Face F1, Face F2, Edge E
if F1 and F2 do not share Edge E then
    return;
end
int i = F1.vertex_position(E.first_vertex);
for Vertex v2 in F2 do
    F1.add(i,v2);
    Vertex next = v2.next;
    Edge(v2,next).face1 = F1;
end
FACES.delete(F2);
EDGES.delete(E);
```

Algorithm 3: Polygon merging method.

4.1.2. Convex area coverage

The Naive and Enhanced exact cellular decomposition (EECD) planners are using the same method to cover convex areas after their subdivision – the Lawnmower pattern. As mentioned in Section 3.2.1, this pattern has one distinctive direction. This leading direction is determined during the planning process (see below).

The general approach was taken from the Energy-aware planner, which describes the stripes subdivision and waypoint placing in greater detail (see Section 3.2.3) and efficiently increases the overlap of created photos.

To determine the number of stripes, we firstly generate a randomly placed line parallel to the leading direction. Then the distance of each area's vertex from this line is computed using formula 11:

$$d_i = \frac{a \cdot V_{ix} + b \cdot V_{iy} + c}{\sqrt{a^2 + b^2}} \tag{11}$$

Here, V_{ix}, V_{iy} are coordinates of i-th vertex and a, b, c are parameters of the generated line in the standard form (i.e. the line is characterized by $ax + by + c = 0$). Then we can compute span of the area in the direction perpendicular to the leading direction:

$$d = |d_{max} - d_{min}| \tag{12}$$

This value of d is used in the formula 9.

To compute each stripe more easily, the set of guiding points is generated. Each guiding point marks the position of the respective stripe and helps generating lines when finding intersections in further computations. The coordinates for i-th guiding point are computed as follows:

$$G_{ix} = V_x + (\frac{L_x}{2} + i \cdot \frac{d - d_x}{n-1}) \cdot \cos\alpha, \tag{13}$$

$$G_{iy} = V_y + (\frac{L_x}{2} + i \cdot \frac{d - d_x}{n-1}) \cdot \sin\alpha, \tag{14}$$

Here V_x, V_y are the coordinates of one of the extreme points, L_x is one of the projection area?s dimensions (computed by formula 2), d_x is the maximal distance between stripes computed using the equation 4, n marks the number of stripes and α is the angle held between the coordinate frame and line perpendicular to the leading direction. The exception for this rule is the case of having only a single stripe, in which case the guiding point can be found at $(V_x + \frac{d}{2} \cdot \cos\alpha, V_y + \frac{d}{2} \cdot \sin\alpha)$.

4.1.3. Stripe computation

When we have a guiding point for the stripe set, we can proceed to process the stripe itself. This includes setting its endpoints as well as determining positions of where the images should be taken at.

When setting endpoints, the intersections of the flight line with the area are computed, but these are not the endpoints yet. On the contrary to adjustments mentioned in Section 3.2.3, including only the segment intersected may not be sufficient since the area may be of irregular shape and thus would not get covered entirely (see Figure 12). What we need to do is to compute the farthest point belonging to the area that still lays inside the stripe, and to project that point onto the flight line.

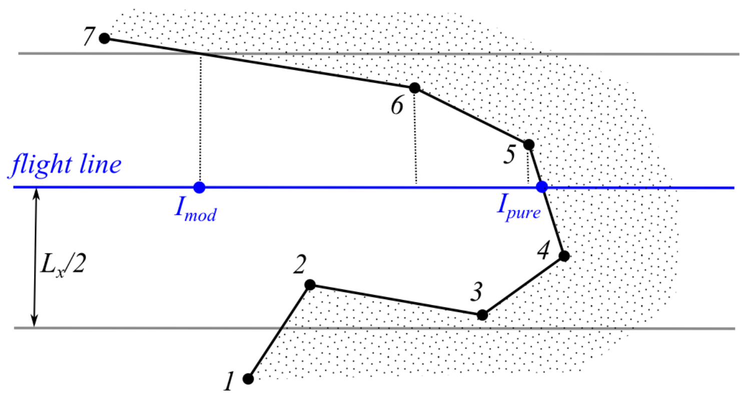


Figure 12. Stripe endpoint adjustment. The dotted area marks the inside of a polygon. The originally found intersection I_{pure} needs to expand to I_{mod} for complete coverage of the stripe. The adjustment is computed by adding the relevant parts of each segment (part of 4-5, 5-6 and part of 6-7). The other side of the borderline, i.e. 1-4 have to be considered as well, but the last relevant point in this array is only 3 and thus does not decide the final adjustment.

As seen in Figure 12, we need to iterate through the adjacent edges until we either reach out of the stripe or the direction changes too much (i.e. the vertex distance starts decreasing). For each edge within the range, we prolong the flight line with the following:

$$\delta d = |v_i - v_{i+1}| \cdot \cos\gamma_i \tag{15}$$

Here, v_i, resp. v_{i+1} denote the beginning and the end of the segment to be counted and γ_i is the angle between the mentioned segment and the flight line.

When the endpoints are found, the number of photos required is determined according to formula 7. The exact positions of the respective shots are computed as follows:

$$W_{ix} = E_x + L_x \cdot (\frac{1}{2} + i) \cdot \cos \alpha, \tag{16}$$

$$W_{iy} = E_y + L_x \cdot (\frac{1}{2} + i) \cdot \sin \alpha, \tag{17}$$

where W_{ix}, W_{iy} are the coordinates of i-th waypoint in the line ($i \in < 0, m >$); E_x, E_y are the coordinates of a line endpoint, α denotes the angle held between the flight line and the coordinate frame.

4.2. Naive exact decomposition planner

The Naive decomposition planner takes advantage of the common algorithm parts, as the cutting techniques and the Lawnmower pattern implementation was already discussed above. Since the cell visiting order is determined by the greedy search, there weren't any additional changes made and the algorithm was implemented according to the original specifications.

4.3. Enhanced exact cellular decomposition planner

While there were only minor changes made in this planner, its certain aspects were not explained in detail and as such they are listed here.

Similarly to the Naive decomposition planner, it heavily relies on the hybrid DCEL structure, which allows performing cuts and merging according to original specifications. The Lawnmower pattern used to sweep individual cells is not different from other planners listed here; the only unresolved tasks for this planner are the polygon width computation and the visiting order determination.

4.3.1. Polygon width computation

As mentioned in Section 3.2.2, there are numerous ways to compute the polygon width. If we are supposed to implement an approach with lesser computational complexity presented in [8] however, it requires us to implement a convex hull algorithm, too.

To compute the convex hull, Lee's algorithm presented in [22] was chosen due to its simplicity. The algorithm structure is as follows:

1. Select a polygon vertex with extreme coordinate value and add it to the convex hull.
2. Select the polygon's following vertex (in counter-clockwise order) and add it to the convex hull.

3. Form a line connecting the last two vertices of the convex hull.
4. Determine whether the polygon's next vertex in counter-clockwise order is on the left or on the right side of the formed line:
   - If the vertex is on the left, add it to the convex hull.
   - If the vertex is on the right, remove the last vertex of the convex hull and add this one instead.
5. Repeat from the second step until all the vertices are processed.

When the convex hull is obtained, further computations are processed according to the original specifications.

4.3.2. Visiting order determination

There was one modification made to EECD regarding the cell visiting order determination. After careful consideration the dynamic programming formula 6 was found to be effective only in scenarios when the visiting order is apriori known, acting otherwise as the greedy search algorithm. As such, the visiting order in our implementation is determined by the greedy search as well, taking all the possibilities created by the entry point existence into account (see Figure 7). By implementing this simple procedure, the existence of a non-oriented graph as suggested was deemed redundant, and as such was removed entirely.

4.4. Energy-aware planner

This planner, being substantially different from the others listed here, had several unique parts. Also, due to somewhat vague description of certain parts, some major changes from the original concept had to be made.

One of the unique features of this planner is the fact that it is not reliant on our hybrid DCEL structure, as the polygon area is not preprocessed in any way. As such, the polygon area is stored simply as a list of polygon vertices ordered counter-clockwise.

The border following feature that originally allowed to reduce the area for sweep was discarded. Not only it would be rather difficult to implement methods of area reduction by a single stripe, but considering more complex shapes, such an approach could significantly increase the flight duration.

As the planner is clearly designed for sweeping mainly convex areas, it originally determined the total number of stripes (as well as their equidistant placing) by computing the distance from the longest line (which is set as a basis for the sweeping) to the farthest vertex of the polygon. Unfortunately, this approach is applicable for convex areas only. For concave areas we might have multiple ending locations for our sweep, and thus the suggested approach cannot be used (see Figure 13). The possibility of recalculating stripes after reaching one of the endpoints was dismissed due to the fact that such recalculation could result in ending in a different endpoint, rendering such a recalculation pointless.

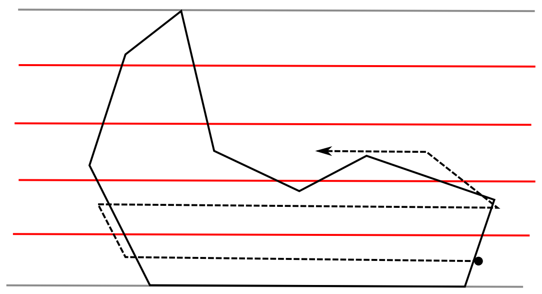


Figure 13. An example scenario that makes the original stripe calculations unusable.

Since we cannot compute the number of stripes in advance, we simply check after completing each stripe whether there is any unsurveyed area remaining. If so, another stripe is created in the distance equal to d_x (see equation 4). By doing so we ensure that the minimal overlap is satisfied. The process of stripe creating is repeated until there is no unsurveyed area left.

4.4.1. Concave parts separation

This subproblem was in [7] addressed to only briefly. The main idea of comparing the flight line intersections in subsequent stripes was used as suggested. This way we obtain an array of vertices that belong to the newfound concave area. However, to properly enclose it we need to determine the exact position of already surveyed area. Unfortunately, there are multiple possibilities regarding this position, and the original illustrating Figure 10 covered only one of the total of four possible scenarios. Additionally, there are some edge cases that needed to be resolved as well.

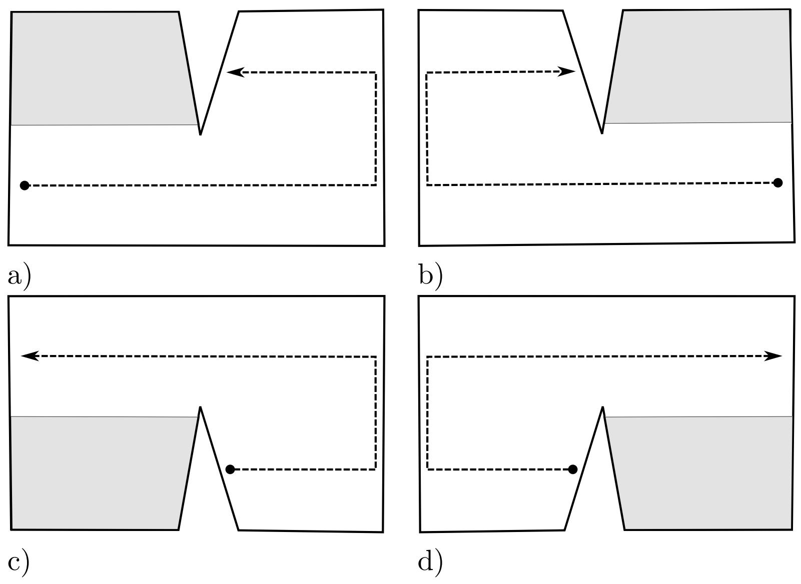


Figure 14. Four possible concave types for the Energy-aware planner. Upper-left (a), upper-right (b), bottom-left (c), bottom-right (d).

All the possible scenarios can be seen at Figure 14. Let us name the concave part that causes the area division as **the corner**. The **corner vertex** is the vertex which creates this concave part. Two cases of concave areas separation can occur – either we fly around the corner to discover a concave area adjacent to the one already swept (under the current stripe and thus called **bottom**) or we hit a corner that prevents us from reaching another part of the area adjacent with the one we are going to sweep (above the previous stripe, thus called **upper**). While the bottom concave area is accessible from the stripe it was detected at, the upper concave area was accessible during the previous stripe. Both these cases can appear on the **left** or **right** side of the stripe. While these situations appear to be quite similar in nature, their combinations of the individual patterns (which, individually, can be and are shared by some of them) make every one of them unique and thus have to be treated slightly differently.

	Bottom-right	Upper-right	Upper-left	Bottom-left
Array part belonging to the corner	beginning of the array	end of the array	beginning of the array	end of the array
Array part belonging to the far side	end of the array	beginning of the array	end of the array	beginning of the array
Corner vertex on the corner edge	first	second	first	second
Stripe to access the area from	current	previous	pevious	current

Table 1. Four scenarios differences.

To identify each of these situations we need to determine whether the detected concave area is upper or bottom as well as its position on left or right. Left/right positioning is determined by comparing the current and previous pairs of flight line intersections. Since these pairs are ordered, i.e. the first intersection in the pair lies on the left and the second one on the right, determining the left/right position of the found concave area is a straightforward task.

To determine the bottom/upper property, the left/right information is used along with starting and ending points of the newest stripe. We take the first edge of the newfound right-sided array (or the last one from left-sided) and determine whether the starting and ending point of the stripe lie within the same halfplane (with the edge acting as the dividing line). As displayed in Figure 15, if they lie in the same halfplane, the detected concave area is of the upper type; if not, the area is of the bottom type.

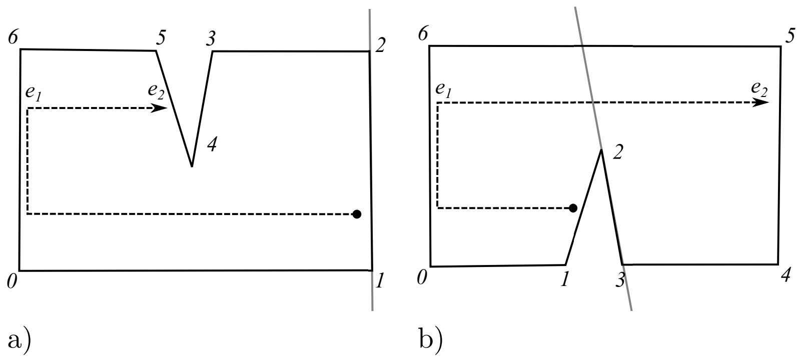


Figure 15. Determining the bottom/upper property for right-sided concave area. In the upper case (a), the first edge (1-2) of the concave area (1-2-3-4) leaves both endpoints e_1 and e_2 in the same halfplane, whereas in the bottom case (b) the first edge (2-3) of the concave area (2-3-4-5) leaves each endpoint in separate halfplanes.

When we have the left/right and bottom/upper properties set, we can safely determine the corner vertex to properly enclose the unsurveyed area (see algorithm pseudocode presented as Algorithm 4).

```
Data: Array concaveArray, previous stripe PS, current stripe CS
Result: Array concaveArray
if concaveArray is left-type then
    Edge defEdge = concaveArray.lastEdge;
end
else
    Edge defEdge = concaveArray.firstEdge;
end
if CS.leftIntersect and CS.rightIntersect lie on the same side of defEdge then
    concaveArray.corner = concaveArray.firstPoint;
    concaveArray.far = concaveArray.lastPoint;
end
else
    concaveArray.corner = concaveArray.lastPoint;
    concaveArray.far = concaveArray.firstPoint;
end
return concaveArray;
```

Algorithm 4: Corner determination method.

To fully enclose the area we have to find the intersections with the border of the previous stripe on the far (non-corner) side of the array and on the corner side and to trim the array accordingly (see Figure 16).

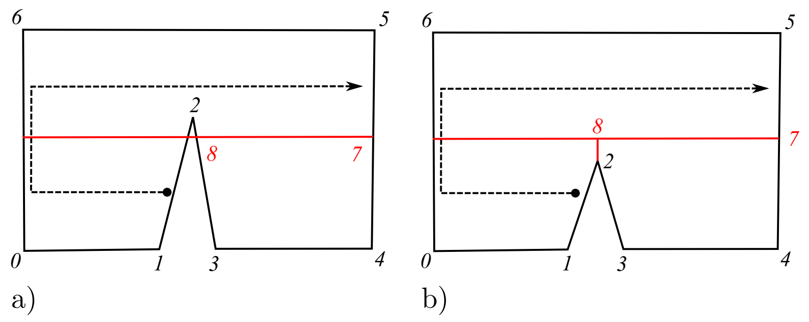


Figure 16. Two possible scenarios of concave area trimming depending on the position of the corner vertex (2). Whereas the far-side intersection is always present (creating a new vertex 7), corner-side intersection is present only in case (a), creating vertex 8 and enclosing the concave area (3-4-7-8). Corner-side intersection is nonexistent in (b), which leads to projecting the corner vertex onto the stripe boundary, creating vertex 8 and enclosing the area (2-3-4-7-8).

When searching for the intersection on the far side, we iterate through the array from its far side towards the corner side until the intersection is reached. All the vertices we pass by during these iterations belong to already surveyed area and thus are not taken as a part of the concave area. The edge case, when the intersection on the far side meets the intersection on the corner side, means that no concave area is present even while traversing multiple edges between stripes (see Figure 17).

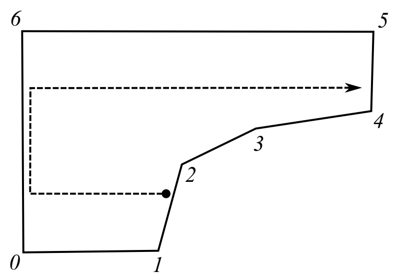


Figure 17. Even though we detect possible concave area at (2-3-4-5), none is actually present.

When searching for the intersection on the corner side, we iterate in a similar manner as in the previous case, only this time we have to keep in mind that no intersection has to be present, as could be seen in Figure 16. If that is indeed the case, we perpendicularly project the corner vertex onto the border of the surveyed area and use that instead. A pseudocode explaining the whole process is presented as Algorithm 5.

```
Data: Array concaveArray, previous stripe PS, current stripe CS
Result: Array concaveArray
for Edge e in concaveArray from far to corner do
    Point far_intersection = e.intersection(line between PS and CS);
    if far_intersection exists then
        break;
    end
    concaveArray.remove(e);
end
if far_intersection == corner then
    return null;
end
for Edge e in concaveArray from corner to far_intersection do
    Point corner_intersection = e.intersection(line between PS and CS);
    if corner_intersection exists then
        concaveArray.remove(Edges from corner to e);
        break;
    end
end
if corner_intersection does not exist then
    Point corner_intersection = corner.perpendicularProjection(line between PS
    and CS);
    concaveArray.add(corner_intersection);
end
return concaveArray;
```

Algorithm 5: Concave area enclosing method.

Another problem we may encounter during this phase is that this unsurveyed area may have multiple intersections with the border of already surveyed area, effectively creating not one, but several concave areas at once (see Figure 18). If we would enclose such a concave area as in aforementioned cases, we would get a self-intersecting polygon. To overcome this problem, we check for intersections within the new area, sort them along one or both coordinates to pair them, and create the list of newly found concave areas. See Algorithm 6 for detailed information.

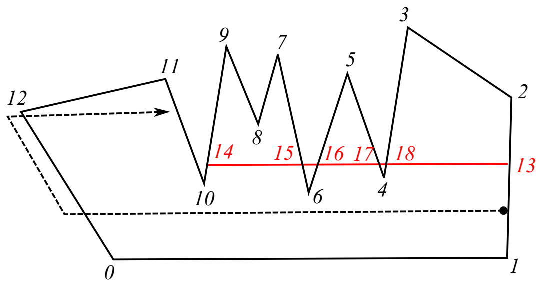


Figure 18. The originally found concave area (2-3-4-5-6-7-8-9-14-13) needs further subdivision. Intersections (15, 16, 17, 18) are found and paired along with the original intersections (14-15, 16-17, 18-13) to form concave areas (7-8-9-14-15), (5-16-17), (2-3-18-13).

```
Data: Array concaveArray
Result: Array concaveArrays
if concaveArray is not self-intersecting then
    return concaveArray;
end
Array intersections;
for Edge e in concaveArray do
    Point intersection = e.intersection(concaveArray.lastEdge);
    if intersection exists then
        intersections.add(intersection);
    end
end
intersections.sort;
Array concaveArrays;
for Point p1 in intersections do
    Point p2 = intersections.next;
    Array current_concave;
    current_concave.add(p1);
    current_concave.add(p2);
    for Point p in concaveArray do
        if p is between p1 and p2 then
            current_concave.add(p);
        end
    end
    concaveArrays.add(current_concave);
end
return concaveArrays;
```

Algorithm 6: Multiple concave areas separation method.

As the result of all the aforementioned steps, we now can determine the list of the concave area vertices and use it as an input argument for the recursive sweep. To sum it up, the whole process the of area subdivision can be described the following way:

```
Data: Area A, previous stripe PS, current stripe CS
for index i in between PS.leftIntersect and CS.leftIntersect do
    leftConcaveArray.add(i);
end
for index i in between PS.rightIntersect and CS.rightIntersect do
    rightConcaveArray.add(i);
end
if leftConcaveArray.length > 1 then
    findCorner(leftConcaveArray, PS, CS);
    enclose(leftConcaveArray, PS, CS);
    subdivide(leftConcaveArray);
end
if rightConcaveArray.length > 1 then
    findCorner(rightConcaveArray, PS, CS);
    enclose(rightConcaveArray, PS, CS);
    subdivide(rightConcaveArray);
end
```

Algorithm 7: Concave areas detection and identification method.

4.4.2. Multiple paths problem

In the case of concave areas emergence (regardless of their bottom/upper type), we now have multiple possibilities to continue our sweep to. In the original paper an idea of sweeping the concave area from the nearest point of the leading path is suggested; when the concave area was swept, the UAV was supposed to return to the original path and to continue on. This approach turned out to be impractical due to the fact that, because of the possibility of creating multiple concave areas at the same side (left/right) the meeting point does not necessarily have to be the endpoint of the current or previous stripe. In that case, the meeting point would have to be somewhere in the middle of the stripe. Since these stripes are expected to lead the plane in the straight lines for reasons stated in Section 3.1, adding a turn inside them would defeat their purpose.

To overcome this problem it was decided to sort the concave areas from one side to another, depending on the ending point of the previous stripe. If the previous endpoint is on the left side, then the set of concave areas is to be visited from left to right, and vice versa. If multiple upper concaves are present, then the most left (or right; it again depends on the position of the previous stripe endpoint) area is chosen as the continuation of the original path.

To illustrate this process, we take an example scenario presented in Figure 18. The previous line endpoint is on the left, which means that after completing a stripe in the main area (10-11-12-0) the set of concave areas is visited in left-to-right order, i.e. (7-8-9-14-15), (5-16-17) and finally (2-3-18-13).

5. Algorithm enhancements

In addition to implementing the selected algorithms, there were also some enhancements made in order to improve algorithm's behavior for various situations. In this section, these improvements descriptions are listed and their effect is discussed.

5.1. Kinematic constraints application

While applying a flight plan created by one of the aforementioned planners, we also have to keep in mind the kinematic constraints of the aircraft used. While the maneuverability of a rotory wing UAV allows us to take even very steep turns because of its ability to stop in midair, reducing the effect of such constraints, a fixed-wing aircraft can do no such thing. The general kinematic model for the UAV can be according to [23] described as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta s_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \begin{bmatrix} \dot{x_p} \\ \dot{y_p} \\ \dot{z_p} \end{bmatrix} \tag{18}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{19}$$

Here, $\dot{x}$, $\dot{y}$, $\dot{z}$ are time derivations of UAV's location in space measured in world's coordinate frame, whereas $\dot{x_p}$, $\dot{y_p}$, $\dot{z_p}$ are time derivations of UAV's location measured in its own coordinate system. ϕ, θ, ψ are roll, pitch and yaw, respectively, determining the UAV's orientation in their respective coordinate frames, whereas p, q, r are angular velocities of UAV's body. s, c in equation 18 are shortcuts for sin, cos respectively. An illustration of the situation can be seen in Figure 19.

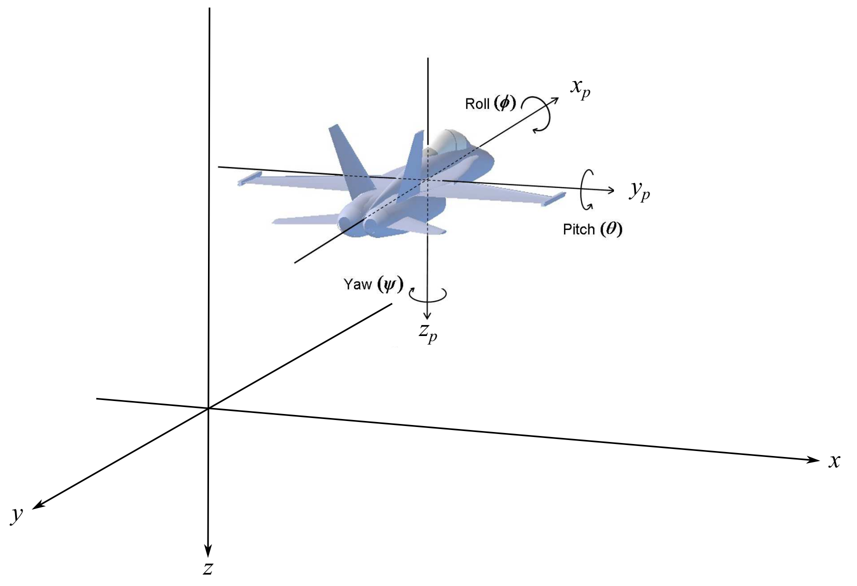


Figure 19. Coordinate frames and Euler angles illustration (image base taken from [24]).

It should be noted that this model takes into account UAV's movement only and omits any external effectors, such as weather conditions (namely wind effects).

Since we consider our problem to be planar only, we dismiss motion angles ϕ and θ as zero. We also assume the aircraft to be traveling in forward direction only (w.r.t. UAV's coordination frame) with a constant velocity v. These adjustments simplify our series of equations substantially:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix} \tag{20}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{21}$$

Now, p, q, r become redundant. Also, since we do not need to compute ϕ, θ (since they are constant and do not interfere with the rest of the model), we omit the respective equations along with $\dot{z}$, which is also considered constant. As such, we can assume that the UAV is controlled by setting its yaw angle as ψ_d and the whole system would collapse into these equations:

$$\dot{x} = v\cos\psi \tag{22}$$

$$\dot{y} = v\sin\psi \tag{23}$$

$$\dot{\psi} = k(\psi_d - \psi) \tag{24}$$

Here k presents regulator gain.

This model in effect ensures that the UAV can be considered as Dubin's vehicle. As such, it is moving with a constant forward velocity, and its turning is defined by its turning radius. Based on that, all the movement it is able to perform consists of curvy parts (C) and straight elements (S) only.

The planners used in this thesis employ flight patterns consisting of mostly straight segments that are to be visited in a specific order. Because of aforementioned kinetic constraints, the flight connections between those segments have to be computed using Dubins curves. Let us name the endpoint of the first segment P_1 and the starting point of the successive segment P_2 with segment orientations given by one-unit vectors l_1 and l_2 respectively. Our task is to connect these points using Dubins curves.

As we will see later on, this approach produces multiple solutions for our problem. Since the amount of solutions is within a reasonable range, the best possibility is to determine the length of each of these trajectories and to choose the shortest of them.

According to Dubins, the shortest path connecting two general points in 2D space with defined orientation in both can be of two types only: CSC and CCC. An illustration of these trajectories can be seen in Figure 20.

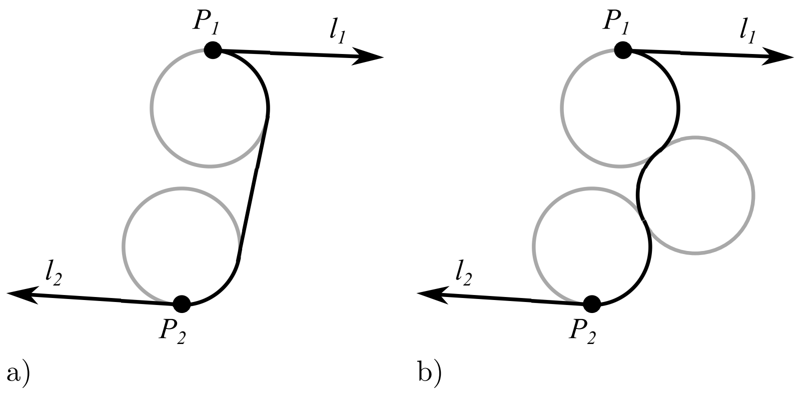


Figure 20. Dubins curve CSC type (a) vs. CCC type (b).

Since the Dubins curve considers only full turn or straight flight, it can be described via points in which the UAV changes its movement, i.e. tangent points in CSC path or contact points in CCC path.

Since the movement along the Dubins curve always starts and ends with a turn (i.e. C part), we firstly need to determine the movement circles at the beginning and the end of each trajectory. These circles can easily be defined via their centers, which lie on the line perpendicular to the required moving direction l_i in entry (P_1) and leaving (P_2) points, and in the distance of turning radius r_T from each (see Figure 21).

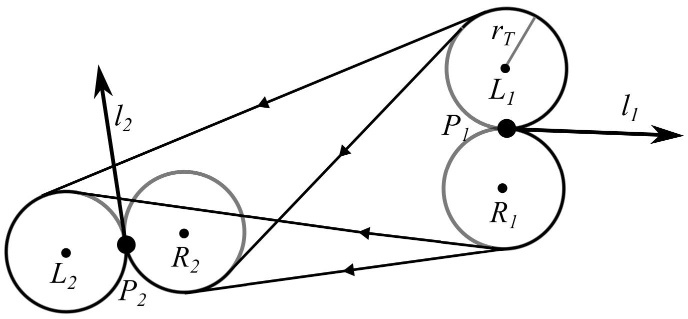


Figure 21. Four possible CSC curves from one waypoint to another.

Assuming that l_i vectors length is one unit, we can write the center's positions more specifically:

$$L_{ix} = P_{ix} - l_{iy} \cdot r_T \tag{25}$$
$$L_{iy} = P_{iy} + l_{ix} \cdot r_T \tag{26}$$
$$R_{ix} = P_{ix} + l_{iy} \cdot r_T \tag{27}$$
$$R_{iy} = P_{iy} - l_{ix} \cdot r_T \tag{28}$$

As we can see in Figure 21, there are two types of circle-circle tangent: external (connecting the same-side circles) and internal (connecting the opposite-side circles). Since all of these circles are based on UAV's turning radius, they have the same radius as well, and finding external tangent points in this case is a straightforward task. The tangent has to be parallel to the line connecting the circle's centers and the distance between it and the connecting line has to be equal to the circle's radius. In effect, tangent points lie on the line laying on circle's centers perpendicular to the line connecting these centers (see Figure 22). Mathematically it is written as follows:

$$T_1 = C_1 \pm l_c^\top \cdot r_T \tag{29}$$
$$T_2 = C_2 \pm l_c^\top \cdot r_T \tag{30}$$

Here, C_1, resp. C_2 denote centers of selected movement circles, l_c is a one-unit vector parallel to line connecting these centers and $^\top$ marks perpendicularity. The $\pm$ sign means that there are two external tangents for each pair of circles, and the one suiting our purposes is determined by the movement circle type (left or right).

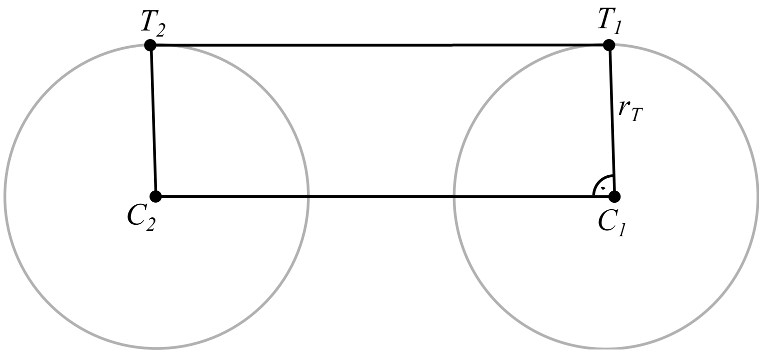


Figure 22. External tangent construction.

As for internal tangents, we use a similar approach. Since both circles are of the same radius, their tangent must cross the line connecting their centers in its middle. Also, the distance of the tangent point from the center has to be equal to r_T, and the tangent is perpendicular to center-tangent point direction (see Figure 23).

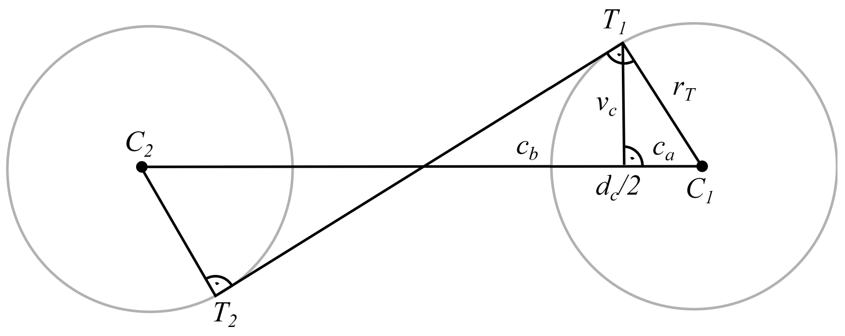


Figure 23. Internal tangent construction.

Seeing the right triangle, we can take an advantage of Euclides's theorems and compute c_a and v_c, which will allow us to compute the desired tangent points:

$$c_a = \frac{a^2}{c} = \frac{r_T^2}{\frac{d_c}{2}} \tag{31}$$

$$v_c = \sqrt{c_a \cdot c_b} = \sqrt{c_a \cdot (\frac{d_c}{2} - c_a)} = \sqrt{r_t^2 - \frac{4r_T^4}{d_c^2}} \tag{32}$$

$$T_1 = C_1 + l_c \cdot c_a \pm l_c^\top v_c \tag{33}$$

$$T_2 = C_2 - l_c \cdot c_a \mp l_c^\top v_c \tag{34}$$

Here, d_c denotes the distance between circle centers. Similarly to the previous case of external tangent, there are two available tangents, from which we choose the correct one by the movement circle type. It should be noted though that unlike the external tangent, the internal tangent can be found only if $d_c \geq 2r_T$. If this criterion is not met, then the formula 32 would produce complex numbers.

The CCC curves can be created on the same-side circles only, and only if their centers distance d_c is smaller than $4r_T$. The situation is illustrated in Figure 24.

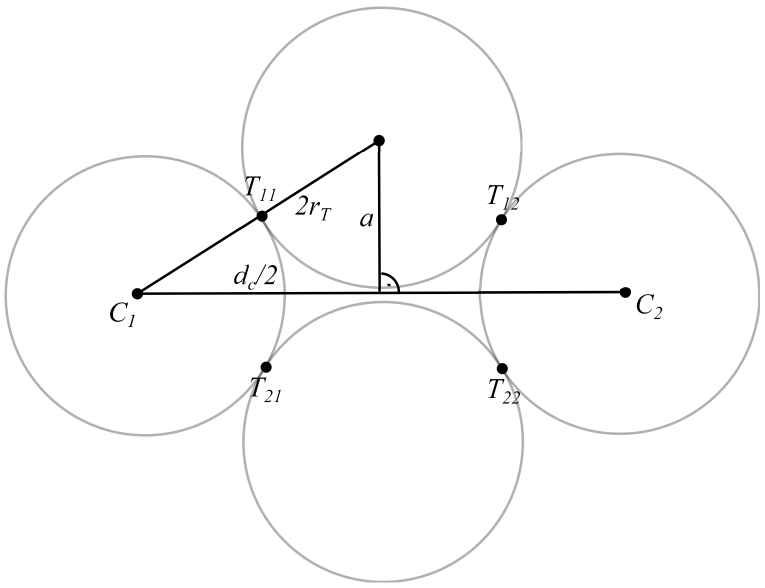


Figure 24. Finding the CCC curve contact points.

As we can see, there are two CCC curves for each pair of movement circles with centers in C_1, C_2 characterized by contact points T_{11}, T_{12} and T_{21}, T_{22} respectively. We can compute these contact points mathematically the following way:

$$a = \sqrt{4r_T^2 - d_c^2} \tag{35}$$

$$T_{11} = C_1 + l_c \cdot \frac{d_c}{4} + l_c^\top \cdot \frac{a}{2} \tag{36}$$

$$T_{12} = C_2 - l_c \cdot \frac{d_c}{4} + l_c^\top \cdot \frac{a}{2} \tag{37}$$

$$T_{21} = C_1 + l_c \cdot \frac{d_c}{4} - l_c^\top \cdot \frac{a}{2} \tag{38}$$

$$T_{22} = C_2 - l_c \cdot \frac{d_c}{4} - l_c^\top \cdot \frac{a}{2} \tag{39}$$

Finally, we determine the best solution of all the above by computing the trajectory length for each solution and choosing the shortest path. This can be done by measuring each of three parts of the curve and summing them up. For the S parts of curves, the length is computed trivially as

$$d = \sqrt{(T_{1x} - T_{2x})^2 + (T_{1y} - T_{2y})^2} \tag{40}$$

For the curvy parts C, we have to compute the angle T_1CT_2, resp. T_2CT_1 (that depends on the movement circle direction). If this angle α is computed in radians, then the length of the curvy path is as follows

$$d = r_T \cdot \alpha \tag{41}$$

Now that we have determined the best (shortest) path along all the found Dubins curves, we insert it into already found plan as a subset of waypoints. These waypoints will have the same coordinates as tangent/contact points of the Dubins curve and will be inserted between trajectory segments the curve connects (i.e. between waypoints with the same coordinates as P_1 and P_2).

5.2. No-flight zones implementation

Another problem that can be encountered in real-life situations is the existence of no-flight zones. Simply put, a no-fly zone (NFZ) is an area forbidden for the UAV to enter. In the real scenario it may represent e.g. a tall building or another obstacle which the UAV should avoid getting near to.

For our task, we consider NFZs to be areas of a polygonal shape, similarly to the definition of an area to be swept. In case of circular or otherwise curvy NFZs, these areas are approximated as polygonal ones.

Because sweep patterns as presented in Chapter 4 allow UAV to travel outside the area's boundaries (and applying kinematic constraints as presented in Section 5.1 amplify this even further), it might not be sufficient to consider the border of NFZ only and a safety margin around the NFZ has to be created. With this safe zone created, the UAV can travel outside the area's borders without necessarily reaching NFZs; the downside of this approach is a greater chance of leaving areas adjacent to NFZs unexplored. With respect to kinematic properties of the UAV, this safety margin was chosen to be r_T, i.e. aircraft's turning radius, which should allow the UAV to perform necessary maneuvers in the NFZ's proximity without reaching it.

With respect to the area of interest, there are three types of NFZs that may appear, sorted by their relative location to the main area. These are:

- **outlaying NFZs**, which lay entirely outside the area of interest and does not intersect with it in any way;
- **bordering NFZs**, which intersect with the area border;
- **inlaying NFZs**, which lay entirely inside the area of interest and does not intersect with it in any way.

An illustration of the situation can be found in Figure 25.

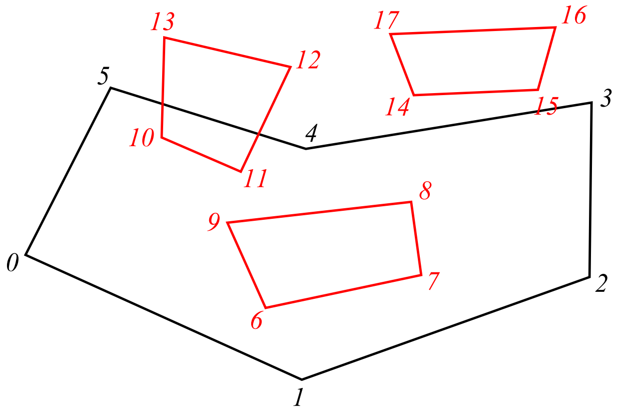


Figure 25. Various types of NFZs (red polygons) with respect to the area of interest (black polygon).

In addition, we allow NFZs to intersect each other, which brings another minor difficulties to our task.

To handle such a situation, we firstly need to merge NFZs to non-intersecting structures. To do that, the following approach was implemented:

1. Choose one unprocessed NFZ (NFZ_1). Find one of its vertices that does not lay inside any other NFZ. If none found, pick another NFZ.
2. Process its edges in counter-clockwise order starting from the outlaying vertex. For each edge find all the intersections with other non-processed NFZs.
3. If no intersection is found, continue with the next edge.
4. If the intersection(s) is found, sort them along the distance from the starting vertex of the current edge. Pick the intersection (with NFZ_2) with the shortest distance from the starting vertex. Add the intersection into NFZ_1 and continue along the border of NFZ_2.
5. Repeat steps 2-4 until the starting vertex is reached.
6. Add NFZ_1 into processed NFZs. Remove all NFZs that participated in its intersections.
7. Repeat from step 1 until no unprocessed NFZ is left.

When the set of non-intersecting NFZs is created, we can straightforwardly sort them into the three aforementioned categories using two criterions:

- if an intersection with the border exists;
- whether one of their vertices lies inside or outside of the area of interest.

The most complicated situation emerges when the first criterion is met and thus the bordering NFZ is detected. To handle that, an approach similar to the NFZ merging algorithm is used:

1. Find one of border vertices that does not lay inside any NFZ.
2. Process its edges in counter-clockwise order starting from the outlaying vertex. For each edge find all the intersections with NFZs.
3. If no intersection is found, continue with the next edge.
4. If the intersection(s) is found, sort them along the distance from the starting vertex of the current edge. Pick the intersection (with NFZ) with the shortest distance from the starting vertex. Add the intersection into the borderline and continue along the border of NFZ in the clockwise order while searching for intersections with border only.
5. If the intersection of NFZ and the border is found, continue along the border, again in counter-clockwise order.
6. Repeat steps 2-5 until the starting vertex is reached.
7. Remove all bordering NFZs.

If the first criterion is not met, the second one determines the exact NFZ type. If the outlaying NFZ is detected, it gets deleted, since it will not interfere with our task in any way. If the inlaying NFZ is detected, it is added into the HDCEL structure as a hole (see Section 4.1.1).

None of the planners presented is capable of dealing with a hole in the polygonal area on its own. To overcome this problem, a ’puncture’ approach was implemented. This approach basically cuts into the area and connects the hole with the border, effectively making the hole the part of the border (see Figure 26).

The process of creating the puncture inside the HDCEL is the following:

1. Choose one unprocessed NFZ (NFZ_1). From one of its vertices V erupt a line.
2. Find the closest intersection I, either with the border or with another NFZ (NFZ_2).
3. If the intersection belongs to the border, insert onto its intersected edge I, V, NFZ_1 in clockwise order (starting from V), a duplicate of V and a duplicate of I.
4. If the intersection belongs to NFZ_2, add into NFZ_1 (after V) I, NFZ_2 in counter-clockwise order (starting from its intersected edge), a duplicate of I and a duplicate of V. Remove NFZ_2.
5. Repeat from step 1 until no non-processed NFZ is left.

As a result, we obtain more complicated concave area, yet without holes. The simplistic puncture approach does not grant optimality regarding the area's shape, but enables planning processes to be used.

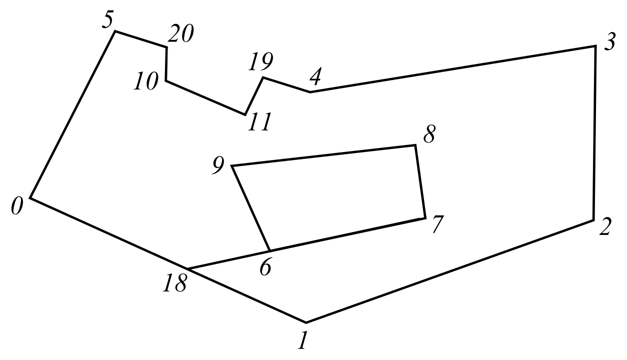


Figure 26. The area of interest absorbs the set of NFZs. While the main part of the polygon is preserved, the (19-11-10-20) segment is created by a bordering NFZ. A puncture of the inlaying NFZ is created by prolonging the (6-7) segment, resulting in the following addition into the polygon: (...-0-18-6-9-8-7-6-18-1-...).

Another problem that may appear during the planning process is accidental crossing of NFZ while traveling from one stripe to another. This problem can be reduced for parallel stripes by using the safe zones around NFZs, but when the UAV travels from one polygon part to another, crossing becomes possible.

We assume that the crossing emerges during longer Dubins curves only, i.e. CSC paths (as CCC curves usually connect neighboring stripes). For each CSC path the straight element is taken and prcessed the following way:

1. Iterate through NFZs. For each NFZ find its distance (D) from the straight element S connecting waypoints Wp_i and Wp_{i+1}.
2. if D is smaller than NFZ's radius R, create a line L perpendicular to S which goes through the NFZ's center. Determine whether S's endpoints lay on both sides of L.
3. If they both lay on the same side of L, skip this NFZ and go back to step 1.

4. If they lay on both sides of L, S crosses the NFZ. Create a point Wp_n on L in the distance of $\frac{3}{2}$ R from NFZ's center and insert it into S. Continue to step 1 with another NFZ.

The process is illustrated in Figure 27.

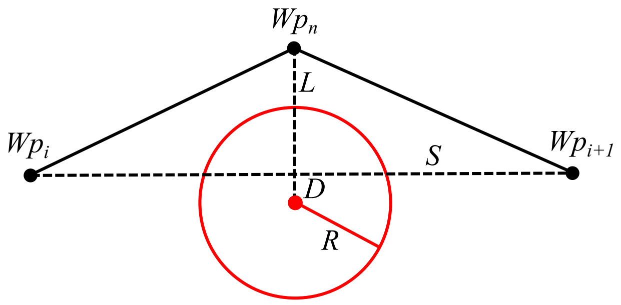


Figure 27. NFZ avoidance when traveling along straight elements. The straight element S connecting waypoints Wp_i and Wp_{i+1} crosses the NFZ, which is displayed as the red circle, in the distance D from its center. A new waypoint Wp_n laying on the line L in the distance of $\frac{3}{2}$ R is added.

5.3. Limited-range planner

While the majority of coverage planners aims to minimize the UAV's energy consumption, its strict energy limit is omitted. In other words, if there is an energy constraint for UAV's flight – usually in terms of limited flight time or flight range – and the planner is unable to minimize the energy consumption below this constraint, the produced flight plan might not be applicable. For this reason, a new planner was developed for this thesis to satisfy these conditions.

The value of the energy limit is dependent on multiple factors, mainly on power consumption rate of motor(s) used for UAV's propulsion and on the total power provided by batteries. While the amount of battery power can be found among the battery specifications, the power consumption rate is dependent on other factors, such as UAV aerodynamics, motor type etc. and thus the consumption rate has to be computed for each UAV individually. If professional UAVs are used, these informations can be usually found among the aircraft's specifications.

The general idea of this planner is following: at the beginning, an **airport** is set up (in our terminology it marks a location where the UAV is able to regain additional energy e.g. by manual battery change) and the UAV covers the area of interest from here. It performs its sweeps and regularly returns for recharging, until the area as a whole is surveyed.

The planning algorithm runs in iterations. In each iteration the UAV performs a sweep from the airport and returns back. The goal is to minimize the number of iterations and thus the energy consumption as a whole.

The key aspect of this planner is the existence of the airport. Its location is limited by one factor only – all the parts of area of interest have to be accessible from here (the term of accessibility is further discussed below). This restriction also limits the size of the area of interest, as the bigger areas cannot be covered from one airport only. If that was the case, an area subdivision would be required to assign smaller areas to respective airports, but such a task is beyond the scope of this thesis. For our task we assume that there is only one airport required, that is already chosen and that meets the aforementioned conditions.

The **accessibility** of a location means, in this case, not only the possibility to travel to this location, but also to return from it back to the airport (within some safety margin). To properly evaluate this attribute, the whole trajectory length has to be computed, including the connection made by Dubins curves. Mathematically, the location is accessible if the following condition is met:

$$d_{margin} < d_{spare} - (2 \cdot l + l_{dubins}) \tag{42}$$

In this formula, d_{spare} is the actual distance UAV is capable of flying, l marks the distance between UAV's current position and the inspected location, l_{dubins} is the length of a Dubins curve connecting in- and outgoing trajectories. Finally, d_{margin} is the safety margin. This setting creates the base of a double stripe explained below.

As the general planner description suggests, an area decomposition has to be made. When choosing from the decomposition possibilities (see Section 3.1), at the first look the approximate decomposition seems to be more suitable for this task as it allows irregular sweep patterns to be performed. However, even these patterns are affected by the regular decomposition grid orientation, which in certain edge cases could cause inaccessibility of some areas due to unnecessary path prolonging towards certain directions. As a result, an exact approach has to be used.

The exact approach has no advantage of automatically generated sweep patterns and as such has to use its own. To ensure that the accessibility condition is always met, we set a **double stripe** (d-stripe) as a basis for this pattern; this sub-pattern consists of two parallel adjacent stripes, one for traveling to the desired location P_d and one for the return to starting position P_s. This covers the maximal area while maintaining the straight direction (see Figure 28).

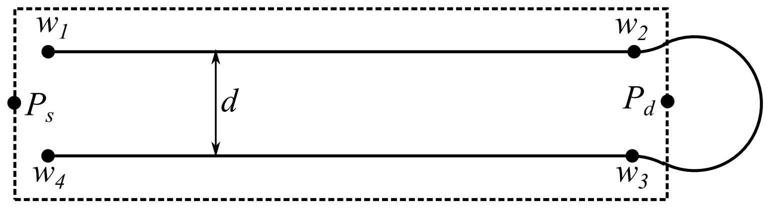
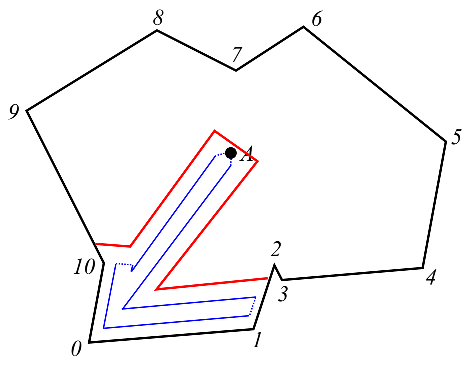


Figure 28. D-stripe connecting starting point P_s and desired point P_d with a coverage displayed by the dashed line. Its main parts are waypoints marking two adjacent stripes w_1-w_2 and w_3-w_4. The d-stripe width d is based on requirements presented in Section 2.2. The displayed Dubins curve connecting these stripes is only illustrative and is not part of a pattern.

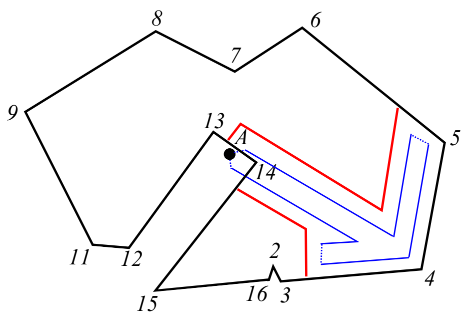
In order to avoid sweeping already surveyed areas as much as possible, we restrain from sweeping easily accessible parts at the beginning, since there is a chance we might need to travel through these areas again to reach the farthest parts of the area. For this reason the most difficult to reach areas are swept first, and the general sweep motion should be from the area's border towards its center. The sweep pattern for each iteration is assembled as follows:

1. Find the farthest vertex of the non-surveyed part of the area (F).
2. Travel to F via d-stripe and compute the current trajectory length. If there is a spare distance left, continue; otherwise return to the airport and move to step 7.
3. Inspect the nearest two non-surveyed area vertices for their distance from the airport. Pick the farther one (F_1).
4. Perform a d-stripe from F to F_1. If the spare distance is not sufficient to perform the task, fly as far as possible and return in the midway.
5. Recalculate the overall trajectory length. If there is a spare distance left, continue; otherwise return to the airport and move to step 7.
6. Move to step 3, but the consecutive d-stripe can be performed also from F_1 now. Repeat until there is no spare distance left.
7. Compose the area surveyed in this iteration by merging d-stripe areas.
8. Subtract the surveyed area from the area of interest.

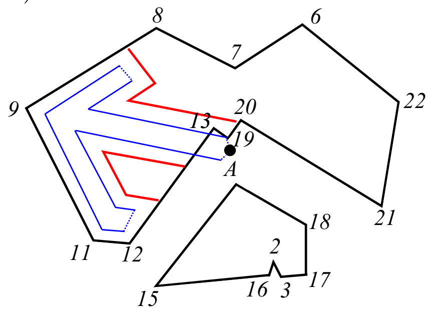
A theoretical performance of this planner is displayed in Figure 29.



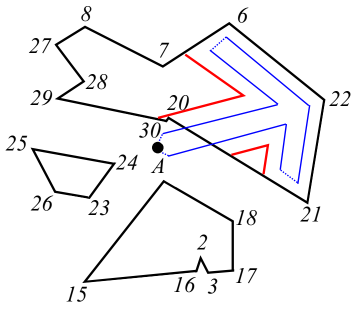
a)



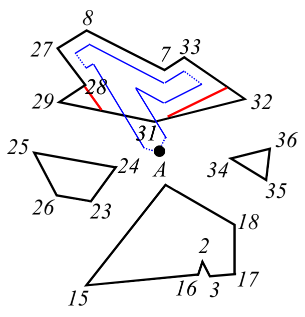
b)



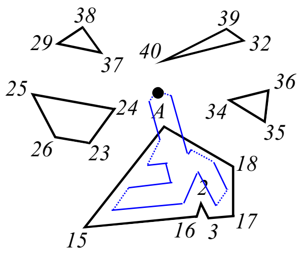
c)



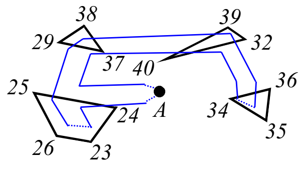
d)



e)



f)



g)

Figure 29. Theoretical performance of the Limited-range planner over the polygon circumscribed by the black line with the airport position in A. Each image shows the trajectory generated during one iteration as a blue line, which reduces the polygon area for the following iteration by the red line. Note the incomplete border following in images b) and c) due to reaching the distance limit.

While the algorithm does not employ any process unknown to previous planners, there are two cases of increased complexity in comparison to the other planners, presented in steps 7 and 8. Polygon merge and subtraction was already presented in Section 5.2, but the rudimentary methods displayed there cannot be applied. The main difference is that the polygon merge can produce the area with holes. Because of this, the polygon subtraction is also able to produce multiple polygons instead of a coherent one. These differences bring severe complications as these possibilities cannot be omitted similarly to the previous cases and the resulting product can now be a multi-parted mesh.

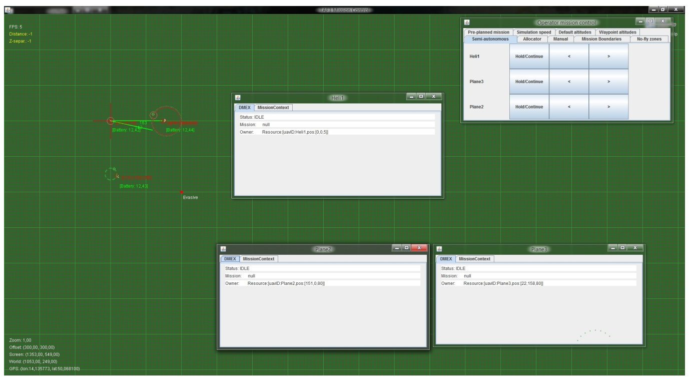
The algorithm was not implemented due to the insufficient time to focus on this task, since its completion was not part of this thesis assignment. The crucial part of the implementation is the development of improved merging and subtracting methods for aforementioned reasons as the versions developed earlier are not sufficient to perform this task.

6. Algorithm analysis and results

In this chapter the algorithm implementations are tested and compared for various scenarios using the Tactical AGENTFLY framework (**TAF3**), which was briefly introduced in Section 1.3. While the majority of these tests was software-based (see Section 6.2), we were also able to generate waypoints using TAF3 that were used for field experiments (see Section 6.3) to verify these simulations in real-life conditions.

6.1. Tactical AGENTFLY interface overview

Before we start with tests themselves, let us introduce the TAF3 interface first.



FPS: 5
Distance: -1
Z-separ.: -1

Status: IDLE
Mission: null
Owner: Resource:[uavID:Heli1,pos:[0,0,5]]

Status: IDLE
Mission: null
Owner: Resource:[uavID:Plane2,pos:[151,0,80]]

Status: IDLE
Mission: null
Owner: Resource:[uavID:Plane3,pos:[22,158,80]]

Zoom: 1,00
Offset: (300,00, 300,00)
Screen: (1353,00, 549,00)
World: (1053,00, 249,00)
GPS: (lon:14,135773, lat:50,068100)

Figure 30. Complete interface as displayed when starting the TAF3 framework.

As can be seen in Figure 30, the TAF3 interface consists of several windows. There is **Operator mission control**, which allows the user to set overall simulation properties – simulation speed, adjust altitudes for various objects (UAVs, waypoints etc.) and to send commands to respective UAVs.

The status of the respective UAVs can be observed on their own windows, in this case titled as **Heli1** for virtual helicopter and **Plane2** with **Plane3** as virtual fixed-wing aircrafts.

The most important part of the interface for our task is labeled **TAF3 Mission Control**, which provides us with grafical user interface. We can see the actual positions of UAVs in the local coordinate frame and we can set missions here manually. It should be noted though that while the (horizontal) X-axis values increase from left to right, the vertical Y-axis values increase from up to bottom. The scale of the grid is 100 meters per side of bigger square (enclosed by thicker lines) and 10 meters per side of each of its sub-squares.

The control mechanism can be viewed by pressing the F1 button, but we will use only a few functionalities of this interface:

- Left mouse button selects objects on the screen.
- Right mouse button shows a list of pre-set actions and scenarios, that can be selected by clicking left mouse button.
- While holding the right mouse button, we can drag the area within the window.
- Mouse scrolling wheel is used for zooming.
- Left mouse click while holding the Left Alt button sets a vertex for the area to be covered. The input ends when releasing Alt.
- By dragging mouse while holding Left Shift and Left Ctrl a no-fly zone is created.
- Enter starts the planning process over the area that has been set.

For reproduction purposes of this thesis, all the testing scenarios were added among the pre-set missions.

6.1.1. UAV properties

As one of this thesis guidelines suggests, a fixed-wing UAV was chosen for performance in the following tests. During the testing it was assumed that the UAV carries a camera modelled by the GoPro Hero4 camera model with the following parameters:

- Field of vision: 94,4 degrees.
- Resolution: 3000x2250 pixels.

The flight altitude was chosen to be 50 meters (with mentioned exceptions), which gives us according to formula 3 spatial resolution of 28 pixels per meter, meaning one pixel covers a square area with 3,6 cm long side. Forward and side image overlaps were picked up as 60 and 20 percent respectively.

According to simulations, the UAV's turning radius r_T is approximately 30 m. The same simulations also revealed that the in order for a waypoint to be marked as 'reached', a UAV has to reach the distance of 30 m and closer. This setting leads to imprecise trajectory following and waypoints marking Dubins curves are often insufficient to maintain the correct course. For this reason the turning radius for the use of the planner was set to 60 m.

6.1.2. Measurement tools

To evaluate planner's performance, two criterions are used – the total flight distance and the coverage ratio. The flight distance is computed three times; two times offline and once during the plan's execution.

The first offline approach sums up distances between the consecutive waypoints – this estimate is over-optimistic, since it does not take into account the existence of curvy elements. Let us name it as d_1.

The second offline approach computes the distances between waypoints in straight elements only. This distance is incremented by the length of Dubins curves used. This estimate is over-pessimistic, though, as we use greater turning radius instead of the real one for the reason already mentioned. Let us name it d_2.

The online approach begins its measurement during the planner's execution. It runs as an instance of `PolygonCoverageStatisticsThread`, which takes in every step UAV's previous position and computes distance to the current one. This value is added to already computed distance, the current position rewrites the previous one and the thread stops for 10 miliseconds. After that the process is repeated until the planner completes its task. Let us name the resulting distance d_3.

The same thread also maintains the area coverage ratio. In the beginning of the planner's execution, the area of interest gets approximated by a grid consisting of square cells one by one meter wide (cells inside NFZs are excluded). During each step of the thread the camera projection area is evaluated based on the UAV's position and orientation, and if it encounters any non-visited cell, this cell is marked as visited. When the planner finishes, the coverage ratio (number of visited cells divided by the total number of cells) is returned. Let us name the ratio r_C.

Again, due to the fact that the ending waypoint is marked as reached before it is actually reached, this ratio rarely shows 1,0 value even when the whole area is swept.

For visualization purposes all the non-visited cells are displayed on screen as yellow dots.

The illustrating trajectory visualizations were made using MATLAB.

6.2. Software tests

6.2.1. Convex polygon coverage test

The scenario can be loaded by selecting `Convex_polygon_coverage` from pre-set missions.

This scenario contains a simple pentagon shape only (see Figure 31). Four standard tests were made for this area – sweeps by the Energy-aware planner and the Naive planner, both with kinematic constraints turned on and off. The EECD planner was not part of this test due to the fact that without polygon decomposition (which does not take place in this scenario) it behaves exactly like the Naive planner.

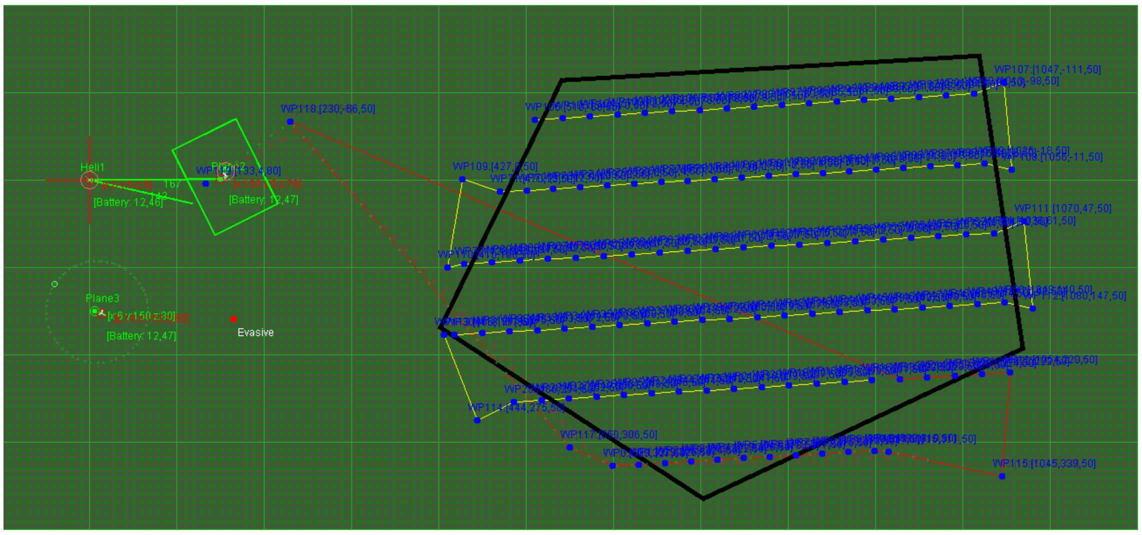
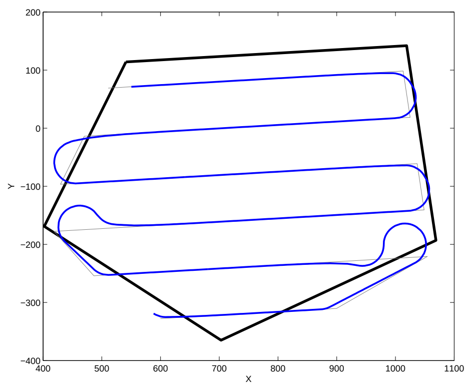
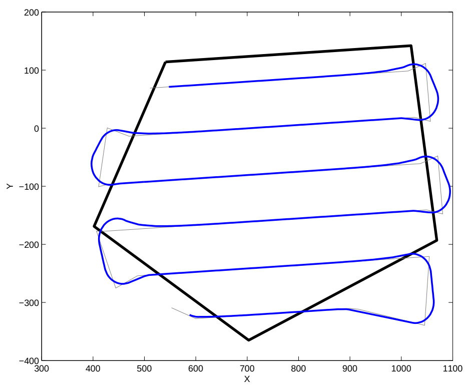


Figure 31. Starting position for the planner. Thick black line marks the area border, blue dots represent waypoints created by the planner. The yellow line marks the planned trajectory, while the red dashed line highlights its actual part. The green rectangle on the left represents UAV's projection area (w.r.t. its actual heading and position). Displayed trajectory belongs to the Naive planner.

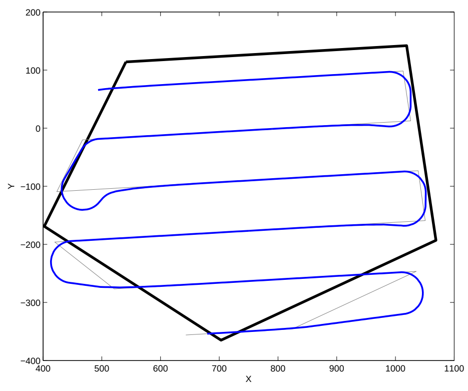
Trajectories generated for each possibility listed can be found in Figure 32.



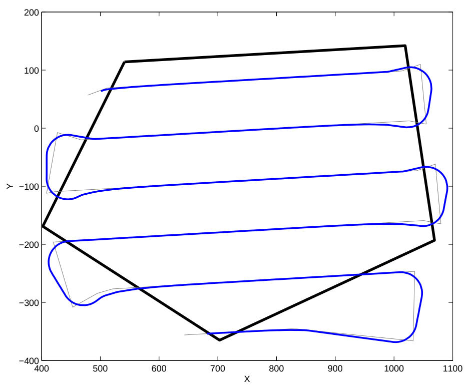
a)



b)



c)



d)

Figure 32. Trajectories generated for the Naive planner (a and b) and for the Energy-aware planner. Kinematic constraints are applied in scenarios b) and d).

The statistics measured for each scenario are the following:

	d_1 [m]	d_2 [m]	d_3 [m]	r_C [-]
Naive, non-kinematic	3701,859	3701,859	3833,083	0,999
Naive, kinematic	3544,362	5183,341	4469,186	1,0
Energy-aware, non-kinematic	3643,208	3643,208	3763,245	1,0
Energy-aware, kinematic	3487,130	5124,792	4206,922	1,0

Table 2. Convex polygon coverage test results.

As we can see, the value of r_C is not in all cases equal to 1. As was mentioned, the reason for this is the planner stopping before it reaches the last waypoint precisely, seemingly leaving the last portion of the area unsurveyed.

When comparing planner's performance, we are unable to find any significant difference. Both planners generate very similar trajectory for convex areas.

Interestingly, allowing and disallowing kinematic constraints did not affect any planner's performance regarding the coverage ratio. Transitions between sweep stripes were not smooth, as can be seen in Figure 32, but this problem's effect was diminished by proportions of the camera projection area. Even though the effect on the trajectory is recognizable, another scenario was set up to further highlight the difference.

The planning area remained the same, but the aircraft's altitude was lowered to 25 meters to make transitions even more difficult to follow while also reducing size of the projection area. The Naive planner was chosen for this task since it sweeps the area from less favorable direction, making the trajectory deviations more apparent.

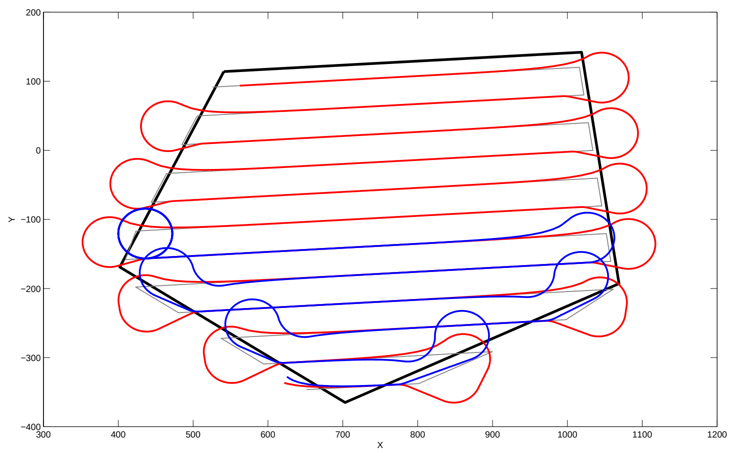


Figure 33. Trajectories comparison for the Naive planner with allowed and disallowed kinematic constraints. The area is marked by a thick black line, the desired sweeping trajectory is marked by light gray line. The red line represents the flight trajectory when the kinematic constraints are applied, the blue line shows the trajectory when no constraints are in effect.

Results of such a test can be seen in Figure 33. As we can see, neither trajectory follows the planner path precisely. In case of the red line (with the kinematic constraints applied), this is caused by the waypoint reaching distance, which becomes apparent at the end of the trajectory (upper left corner), when the planner does not reach its end. We can still observe that the red trajectory deviation from the planned course is not big, especially compared to the deviation of the blue trajectory (without kinematic constraints applied). During steep turns on the bottom of the area a significant overshoot is reached and the UAV even oscillates along the proposed stripe approximately during the first 200 meters. The imperfection of this approach is further exploited in middle part of the area, where the non-constrained UAV enters a stripe using such a bad angle that it is unable to reach the following waypoint, as it lies in the center of its turn. This effectively stops the planner from working, as it is unable to continue with its task further.

6.2.2. Simple concave polygon coverage test

The scenario can be loaded by selecting `Simple_concave_polygon_coverage` from pre-set missions.

This scenario contains a simple pentagon shape only (see Figure 34). Three tests were made for this area, one for each planner type. After the validation of kinematic constraints usefulness all planners are working with constraints allowed.

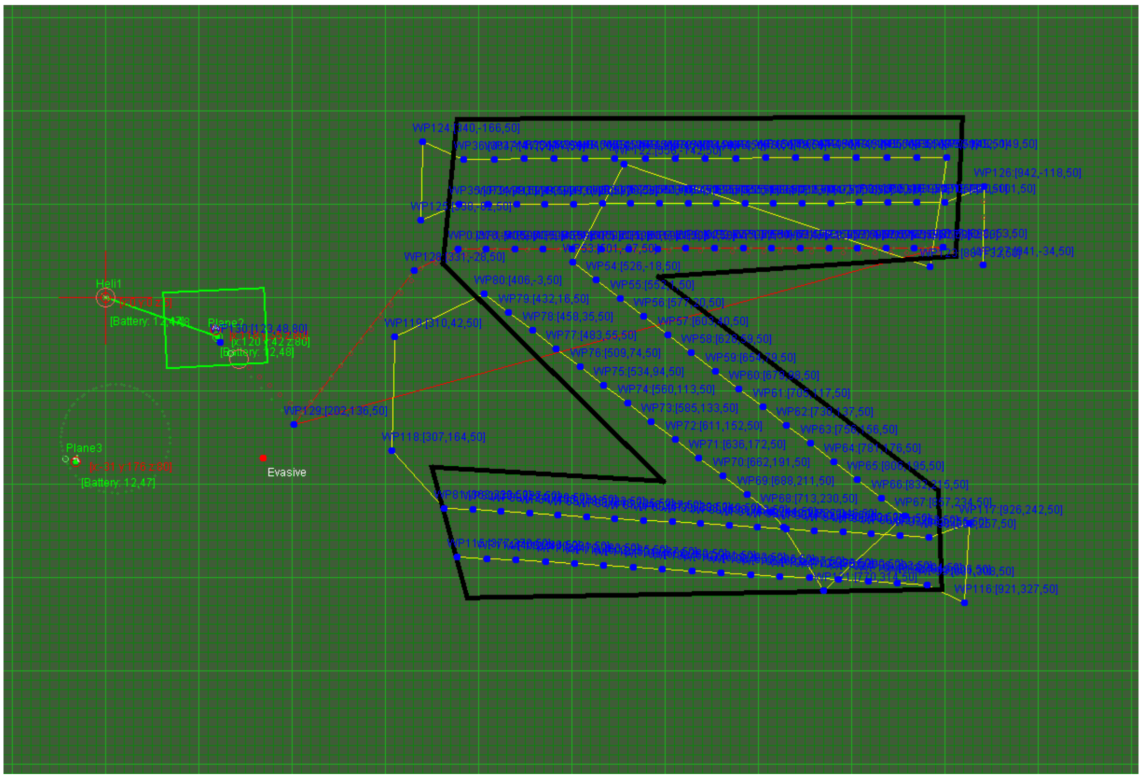


Figure 34. Starting position for the planner. Thick black line marks the area border, blue dots represent waypoints created by the planner. The yellow line marks the planned trajectory, while the red dashed line highlights its actual part. The green rectangle on the left represents UAV's projection area (w.r.t. its actual heading and position). Displayed trajectory belongs to the Naive planner.

Since trajectories for each planner were different this time, each result will be discussed separately.

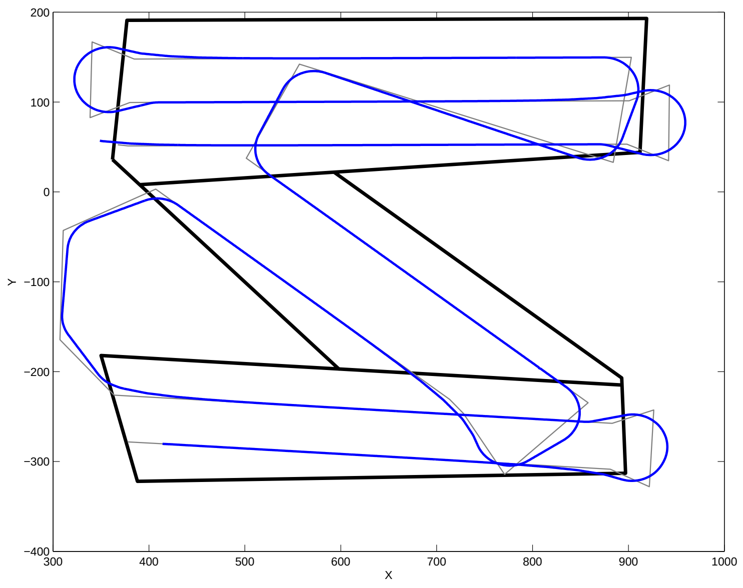


Figure 35. The Naive planner sweep trajectory generated for a concave polygon. The area is marked by a thick black line, as well as the subdivision performed by the planner. The desired sweeping trajectory is marked by light gray line. Blue line represents the actual flight trajectory.

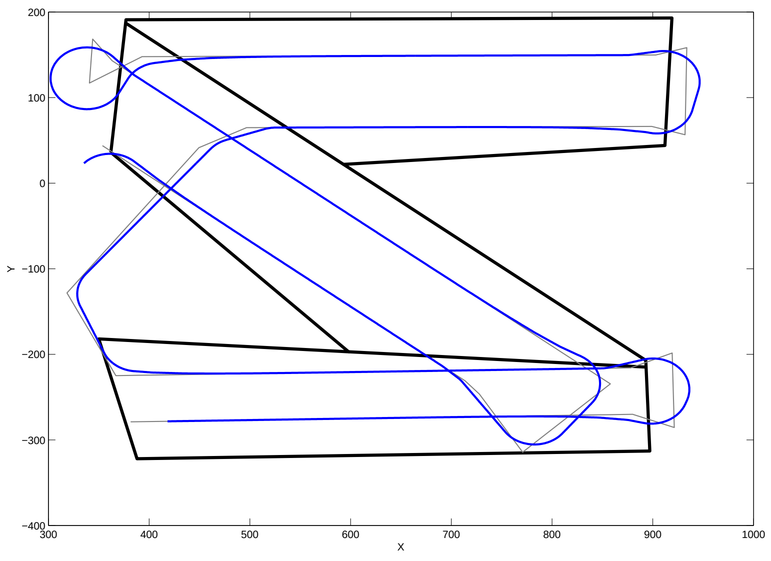


Figure 36. The EECD planner sweep trajectory generated for a concave polygon. The area is marked by a thick black line, as well as the subdivision performed by the planner. The desired sweeping trajectory is marked by light gray line. Blue line represents the actual flight trajectory.

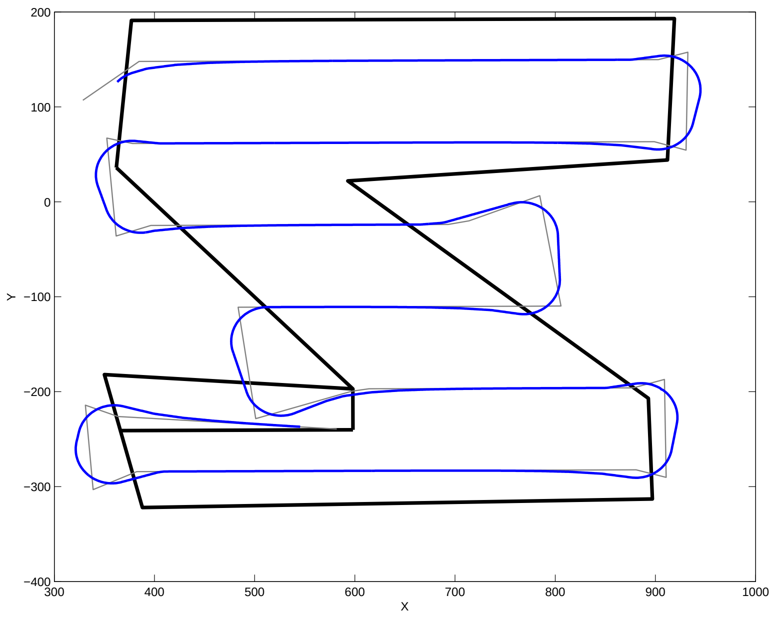


Figure 37. The Energy-aware planner sweep trajectory generated for a concave polygon. The area is marked by a thick black line, as well as concave parts separated by the planner. The desired sweeping trajectory is marked by light gray line. Blue line represents the actual flight trajectory.

The Naive planner separates given polygon into three parts, as can be seen in Figure 35. This low number of polygons is swept effortlessly.

Similarly to the Naive planner, the EECD planner also swept all the parts of the given area. Its subdivision method was slightly more efficient though and results in shorter trajectory.

The Energy-aware planner also swept the entire area using its Lawnmower pattern. Only one part on the bottom left side was considered out of reach for the traditional pattern and was swept recursively after the main part.

The statistics measured for each scenario are the following:

	d_1 [m]	d_2 [m]	d_3 [m]	r_C [-]
Naive, kinematic	4171,225	6682,902	5146,208	1,0
EECD, kinematic	3469,546	5297,487	4266,827	1,0
Energy-aware, kinematic	3074,977	5014,466	3893,623	1,0

Table 3. Simple concave polygon coverage test results.

The best results were given by the Energy-aware planner with the shortest trajectory. Arguably, it is caused by the shape od the polygon that allows it to be swept almost completely using a single pattern.

6.2.3. Complex concave polygon coverage test

The scenario can be loaded by selecting `Complex_concave_polygon_coverage` from pre-set missions.

This scenario contains a complex polygon with many concave vertices (see Figure 38). Three tests were made for this area, one for each planner type. Again, all planners are working with constraints allowed.

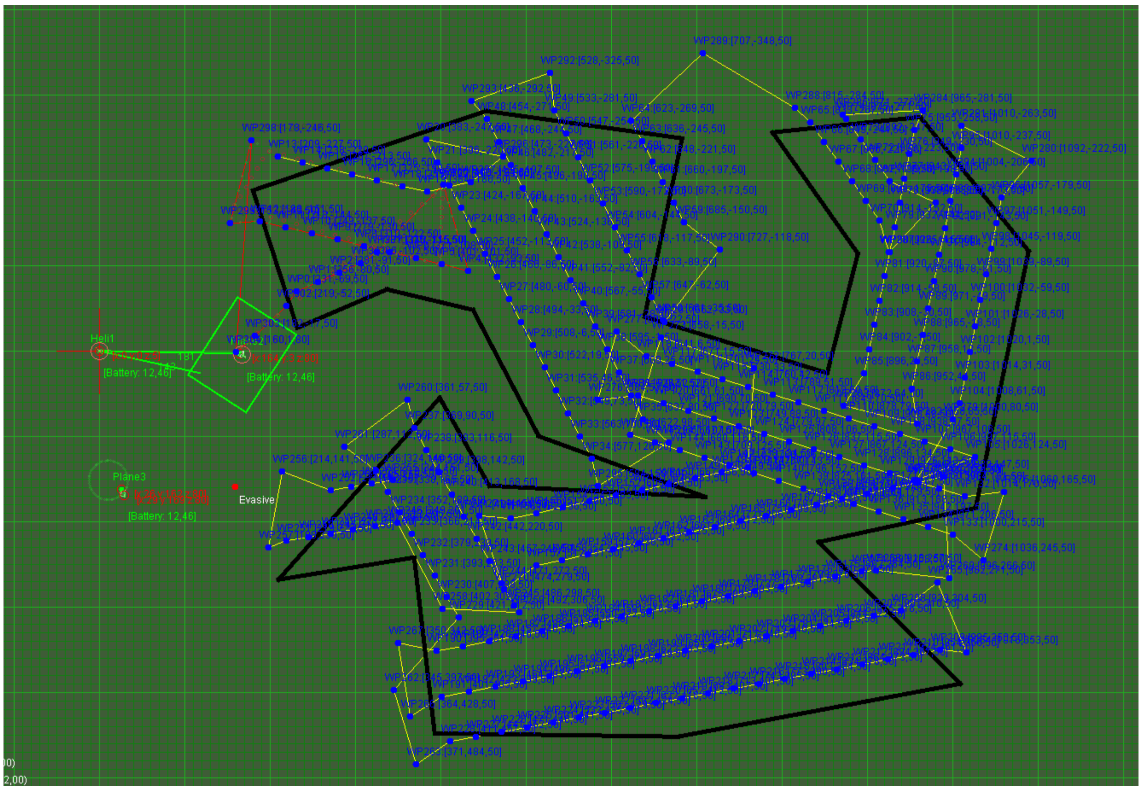


Figure 38. Starting position for the planner. Thick black line marks the area border, blue dots represent waypoints created by the planner. The yellow line marks the planned trajectory, while the red dashed line highlights its actual part. The green rectangle on the left represents UAV's projection area (w.r.t. its actual heading and position). Displayed trajectory belongs to the Naive planner.

Since trajectories for each planner were different, each result will be discussed separately.

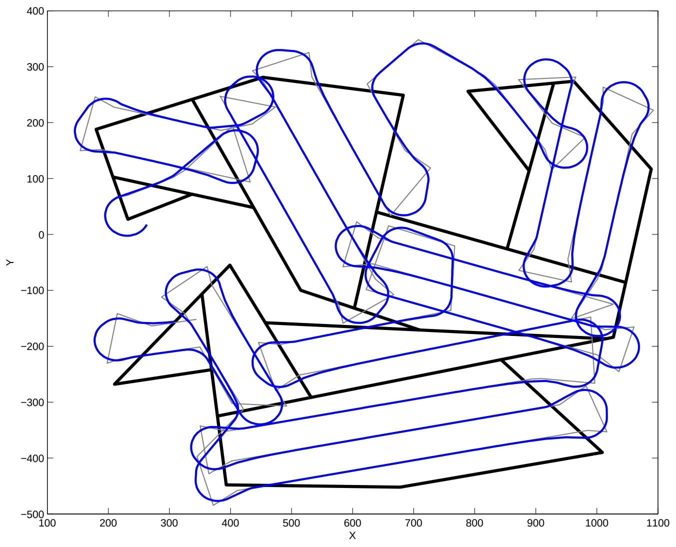


Figure 39. The Naive planner sweep trajectory generated for a concave polygon. The area is marked by a thick black line, as well as the subdivision performed by the planner. The desired sweeping trajectory is marked by light gray line. Blue line represents the actual flight trajectory.

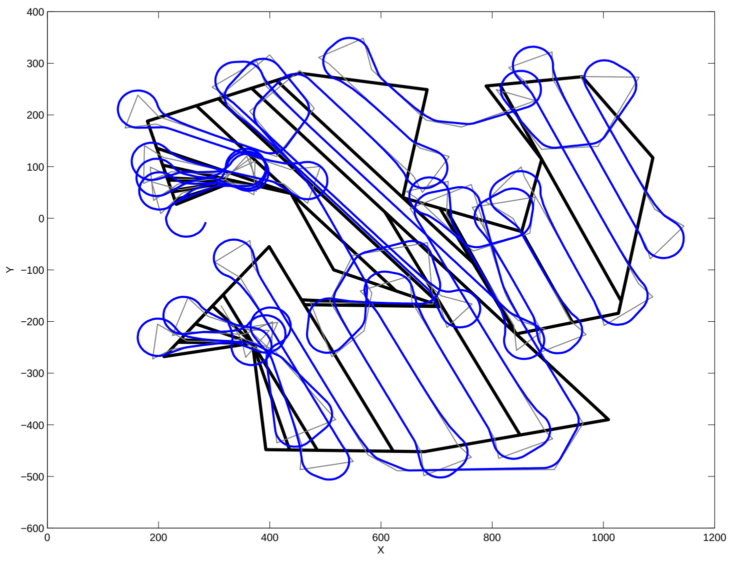


Figure 40. The EECD planner sweep trajectory generated for a concave polygon. The area is marked by a thick black line, as well as the subdivision performed by the planner. The desired sweeping trajectory is marked by light gray line. Blue line represents the actual flight trajectory.

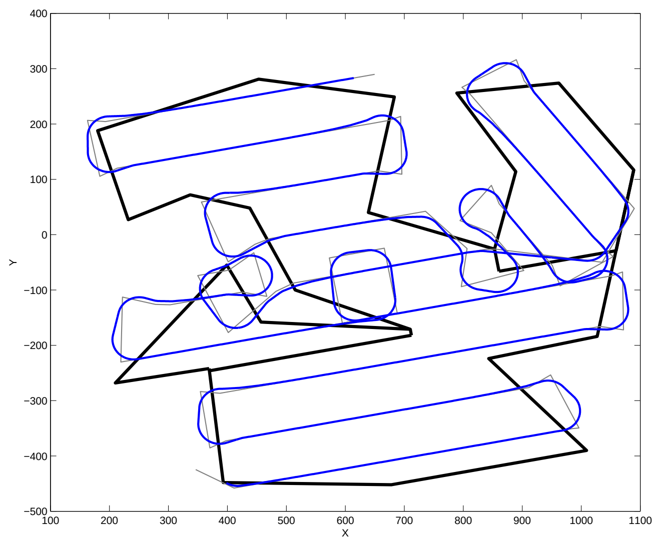


Figure 41. The Energy-aware planner sweep trajectory generated for a concave polygon. The area is marked by a thick black line, as well as concave parts separated by the planner. The desired sweeping trajectory is marked by light gray line. Blue line represents the actual flight trajectory.

The Naive planner separates given polygon into 10 parts, as can be seen in Figure 39. This amount of cells is greater than in the previous scenario, but still manageable, as the planner sweeps the entire area.

While the EECD planner also swept all parts of the given area, its biggest drawback becomes apparent in this scenario (see Figure 40). As it aims for creation of thin cells (i.e. with the least polygon width), it ends up inefficiently cutting many thin cells without actually solving the problem with concavity. Given the fact that its cutting directions are parallel to other polygon lines, increased complexity of the polygon allows performing multiple almost parallel cuts, as the probability of existence of almost parallel lines in the polygon is increased. This is clearly demonstrated in left parts of the area, where multiple almost parallel cells exist. These areas could have been swept in a single stripe, but the planned trajectory requires the UAV to sweep it several times instead. Nevertheless, the area gets swept completely.

The Energy-aware planner was not able to sweep the area with a single pattern this time, as can be seen in Figure 41. A subdivision had to be made dividing the area into three parts, one with completely different sweeping direction. Note the circular trajectory in the middle of the polygon, where two parallel sweep lines were too close to each other for smooth transition, which made UAV to make an extra loop.

However, this planner did not sweep the area entirely. The little concave area in the upper left part of the polygon got missed, apparently due to proximity to already swept area.

The statistics measured for each scenario are following:

	d_1 [m]	d_2 [m]	d_3 [m]	r_C [-]
Naive, kinematic	8467,553	16900,459	11854,008	0,999
EECD, kinematic	11880,357	27644,961	18975,799	1,0
Energy-aware, kinematic	6889,067	12026,980	8878,306	0,994

Table 4. Complex concave polygon coverage test results.

The best results regarding the trajectory length were again given by the Energy-aware planner with the trajectory more than two times shorter than the one given by the EECD planner. As was already mentioned however, the coverage ratio r_C was the lowest so far, unlike the rest of planners used. The best result regarding the coverage ratio belongs to the EECD planner, which had slightly greater coverage than the Naive planner, apparently due to its sweep redundancy.

6.2.4. Polygon with no-flight zones coverage test

The scenario can be loaded by selecting `Convex_polygon_with_NFZs_coverage` from pre-set missions.

This scenario contains a convex polygon with three no-flight zones present – inlaying, bordering and outlaying (see Figure 42). Three tests were made for this area, one for each planner type. Again, all planners are working with constraints enabled.

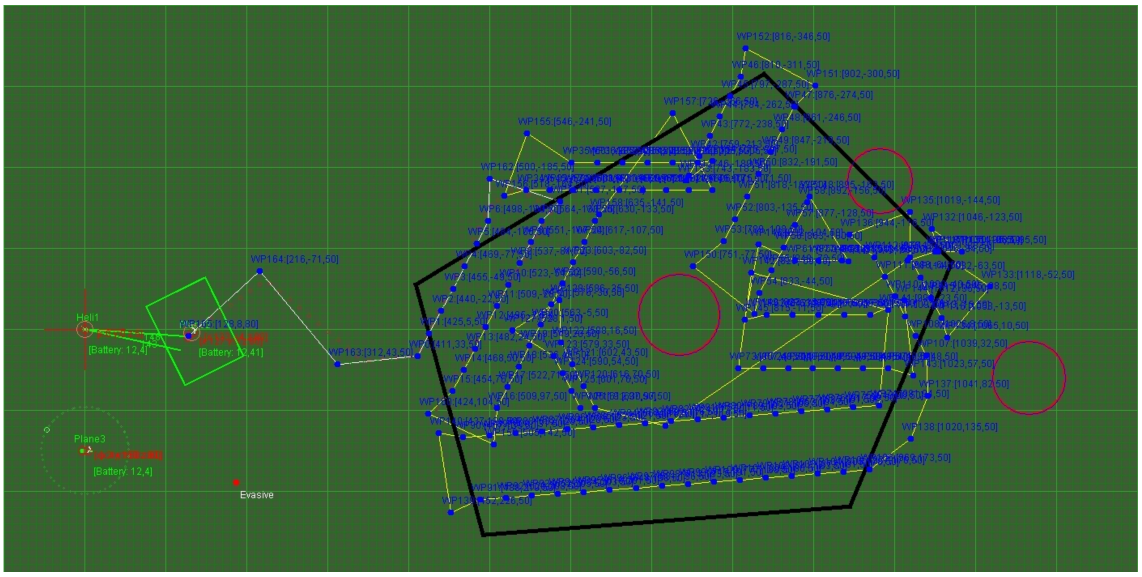


Figure 42. Starting position for the planner. Thick black line marks the area border, red circles represent no-flight zones. Blue dots represent waypoints created by the planner. The yellow line marks the planned trajectory, while the red dashed line highlights its actual part. The green rectangle on the left represents UAV's projection area (w.r.t. its actual heading and position). Displayed trajectory belongs to the Naive planner.

Each NFZ is replaced by a hexagon with the outer radius equal to the radius of the original NFZ enlarged by the UAV's turning radius. This creates a zone where the UAV should be able to turn safely. The effectiveness of this approach will be further discussed.

Since trajectories for each planner were different, each result will be discussed separately.

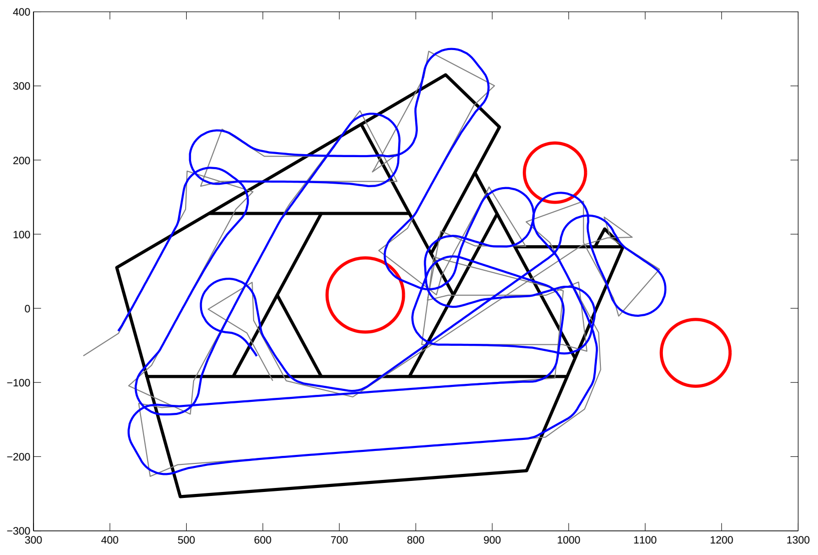


Figure 43. The Naive planner sweep trajectory generated for a convex polygon with no-flight zones. The area is marked by a thick black line, as well as the subdivision performed by the planner. No-flight zones are represented by red circles. The desired sweeping trajectory is marked by light gray line. Blue line represents the actual flight trajectory.

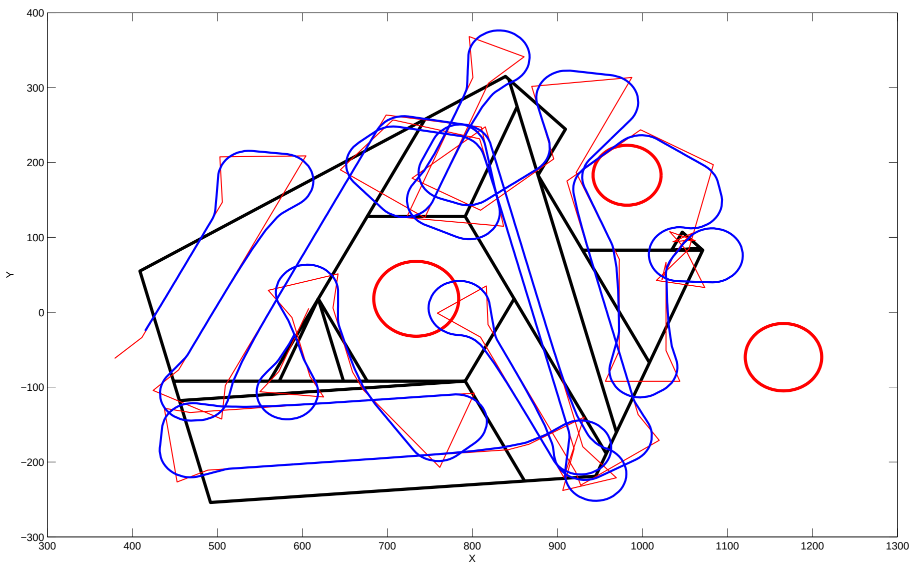


Figure 44. The EECD planner sweep trajectory generated for a convex polygon with no-flight zones. The area is marked by a thick black line, as well as the subdivision performed by the planner. No-flight zones are represented by red circles. The desired sweeping trajectory is marked by light gray line. Blue line represents the actual flight trajectory.

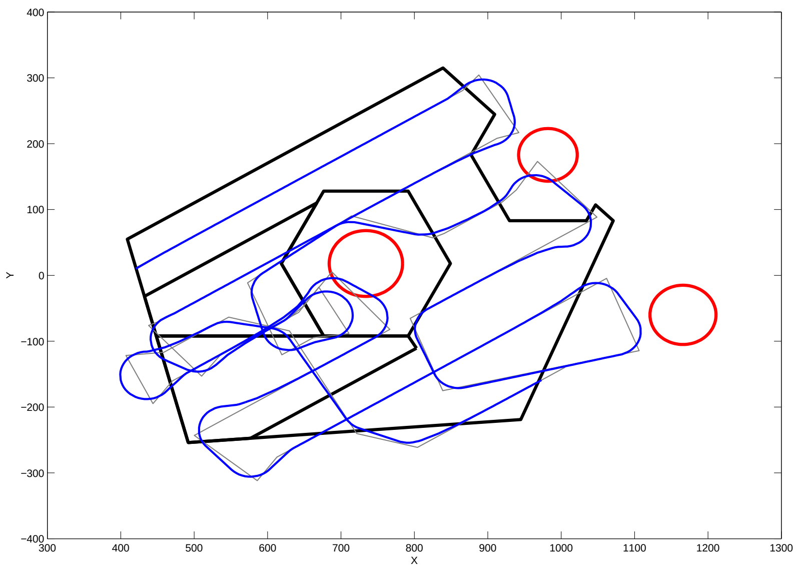


Figure 45. The Energy-aware planner sweep trajectory generated for a convex polygon with no-flight zones. The area is marked by a thick black line, as well as concave parts separated by the planner. No-flight zones are represented by red circles. The desired sweeping trajectory is marked by light gray line. Blue line represents the actual flight trajectory.

Due to the NFZs approximation, there are parts of the polygon that are excluded from the sweep, because they lay too close to NFZ, as can be seen in Figure 43 displaying the sweep pattern generated by the Naive planner. Also, the hexagonal approximation generates relatively big amount of cells for individual sweep. As seen in the picture, the sheer number of cells was not decisive for the planner's performance, but unfortunately the UAV reached no-flight zones during its maneuvering.

Apart from that, certain parts of the area were missed due to their exclusion from the polygon for safety reasons, namely the upper part of the area surrounding the central NFZ.

Similarly to the previous scenario, the EECD planner managed to subdivide the area into greater number of cells than necessary, which needlessly prolongs the overall trajectory (see Figure 44). Its coverage ratio was sufficient partially due to its coverage redundancy. Regarding the NFZ avoidance, the UAV's trajectory was successfully altered to avoid a NFZ on multiple occasions (see the bordering NFZ in the upper right part of the area), but similarly to the previous planner, it could not handle the avoidance while maneuvering near the central NFZ.

As seen in Figure 45, the Energy-aware planner creates its typically efficient sweep pattern and also manages to avoid the central NFZ while traveling from one cell to another, but similarly to its predecessors, it fails to avoid the NFZ while maneuvering.

The statistics measured for each scenario are the following:

	d_1 [m]	d_2 [m]	d_3 [m]	r_C [-]
Naive, kinematic	4568,063	11261,163	6845,101	0,970
EECD, kinematic	5329,757	13681,814	8777,589	0,999
Energy-aware, kinematic	4221,486	8005,343	5593,776	1,0

Table 5. Polygon with no-flight zones coverage test results.

Unfortunately none of the presented planners fully fulfilled our expectations regarding the NFZs handling. The approach we used was sufficient to avoid NFZs while traveling from one part of the area to another, but the maneuvers along CCC curves were able to reach the NFZ even inside its safe zone. It becomes apparent that either more sophisticated approach has to be developed to avoid reaching NFZs entirely, or greater safe zones have to be set – with the risk of leaving too much of an area unsurveyed.

6.3. Field experiments

In addition to planned software tests, three real experiments were also conducted to verify the performance of implemented planners in real-world conditions.

Simple flight scenarios were prepared using TAF3 framework. The set of waypoints produced by the respective planner was extracted using the `GeographyUtils` module from local cartesian coordination frame into GPS coordinates, which were used to navigate the UAV. Tests were conducted on a field near Unhošť in the county of Kladno, Czech republic. The used aircraft was TELINK GRID 720 (see Figure 46), which, in order to simulate a fixed-wing aircraft, was set with a constant forward velocity. This adjustment prevented the UAV from making turns on-spot which the usual rotory aircraft can do.

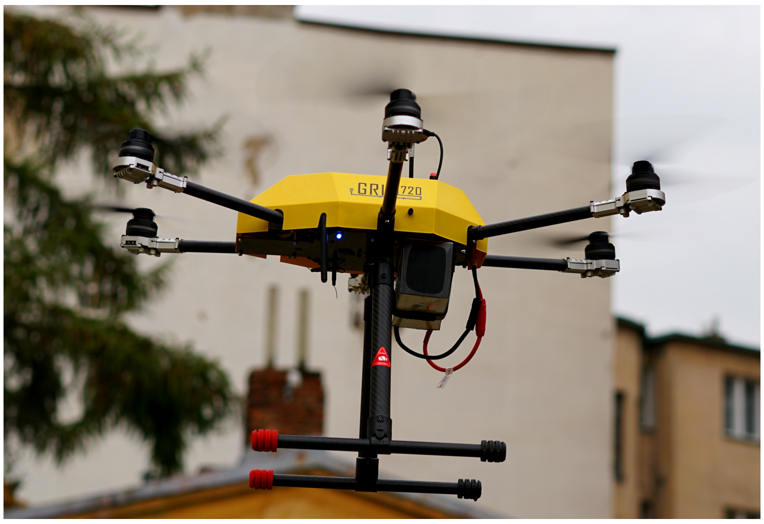


Figure 46. The TELINK GRID 720 UAV used for field experiments.

Tests were aiming for the usefulness of the flight pattern verification rather than for creating an actual orthophotmap. To evaluate the trajectory, the UAV flight log recorded at 25 Hz was retrieved. The log contains for each time stamp the information about UAV's GPS position and orientation of its body expressed in the Yaw, pitch, roll notation.

Since the UAV was not equipped with any camera type, the usage of the same camera model which was used during software tests was assumed. Two options were considered while evaluating the coverage made by this virtual camera: either its axis was fixed vertically to the ground (which in reality can roughly be achieved using gimbals) or its axis was fixed vertically to UAV's body, always heading in its Z-axis.

The GPS coordinates for each time stamp were transformed back to the local cartesian coordinate frame (with the log's initial position serving as its origin point). Given the UAV's position and orientation the projection area for the virtual camera was created and displayed along with the aircraft's location.

6.3.1. Convex polygon coverage

This test was designed for verification of basic planner's functions. A simple rectangular area was covered by the Naive planner using single Lawnmower sweep pattern.

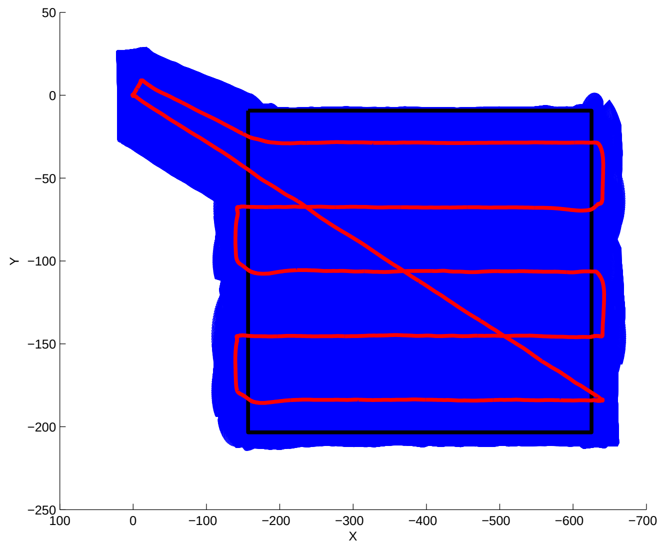


Figure 47. The Naive planner sweep trajectory generated for a convex polygon. The area is marked by a thick black line. The flight trajectory is represented by the red line. The area covered by the sweep is in blue. Camera axis is considered vertical to the ground and neglects the influence of UAV's body orientation.

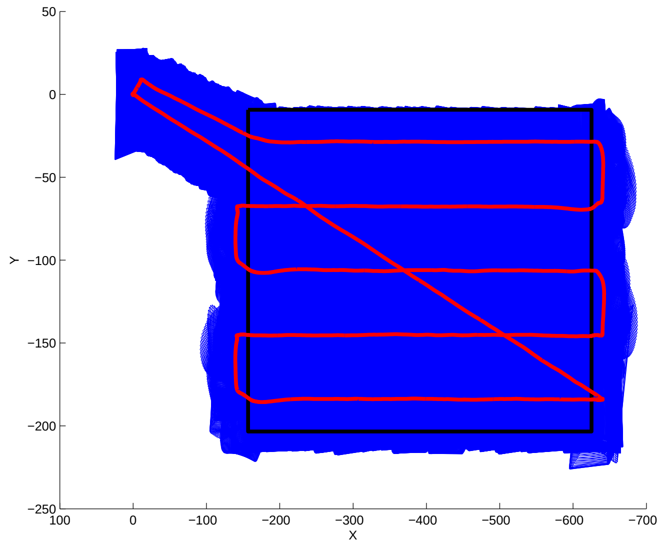


Figure 48. The Naive planner sweep trajectory generated for a convex polygon. The area is marked by a thick black line. The flight trajectory is represented by the red line. The area covered by the sweep is in blue. Camera axis is considered vertical to UAV's body orientation.

As can be seen in Figures 47 and 48, basic functions were verified as the planner had no difficulties sweeping the area. The orientation of the camera axis makes no visible difference in this scenario, as in both cases the area (marked by a thick black line) is swept completely.

6.3.2. Concave polygon coverage

The area containing two concave vertices was prepared for the EECD and Energy-aware planner to sweep. The aim of this test was to verify planner's performance even over more complex areas.

Firstly, the EECD planner was processed:

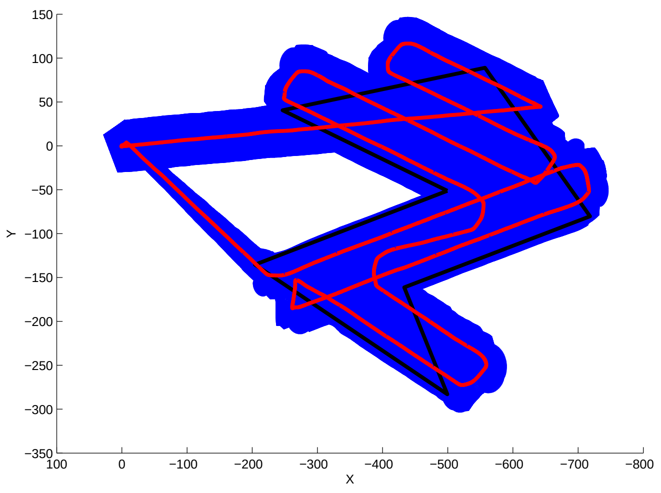


Figure 49. The EECD planner sweep trajectory generated for a concave polygon. The area is marked by a thick black line. The flight trajectory is represented by the red line. The area covered by the sweep is in blue. Camera axis is considered vertical to the ground and neglects the influence of UAV's body orientation.

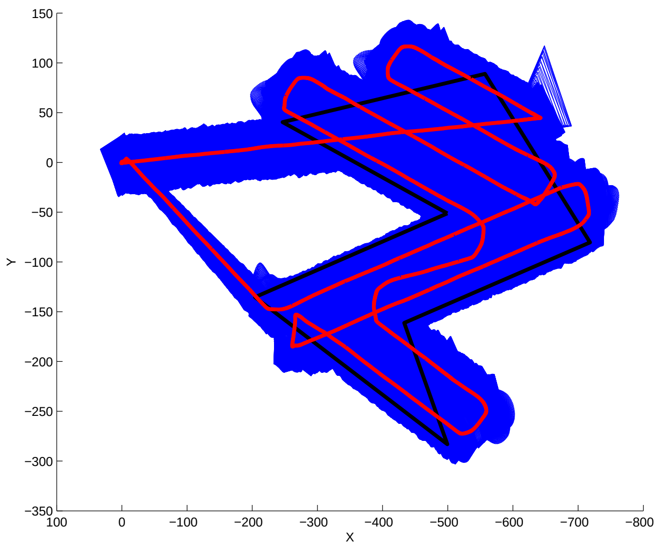


Figure 50. The EECD planner sweep trajectory generated for a concave polygon. The area is marked by a thick black line. The flight trajectory is represented by the red line. The area covered by the sweep is in blue. Camera axis is considered vertical to UAV's body orientation.

As can be seen in Figures 49 and 50, the planner had no difficulties sweeping the area. The orientation of the camera axis affects the shape of covered area more significantly, especially during steep turns, but does not affect the overall coverage ratio.

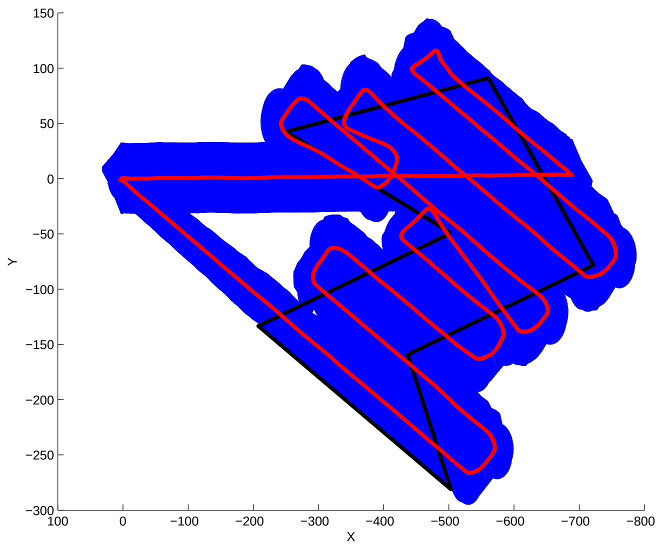


Figure 51. The Energy-aware planner sweep trajectory generated for a concave polygon. The area is marked by a thick black line. The flight trajectory is represented by the red line. The area covered by the sweep is in blue. Camera axis is considered vertical to the ground and neglects the influence of UAV's body orientation.

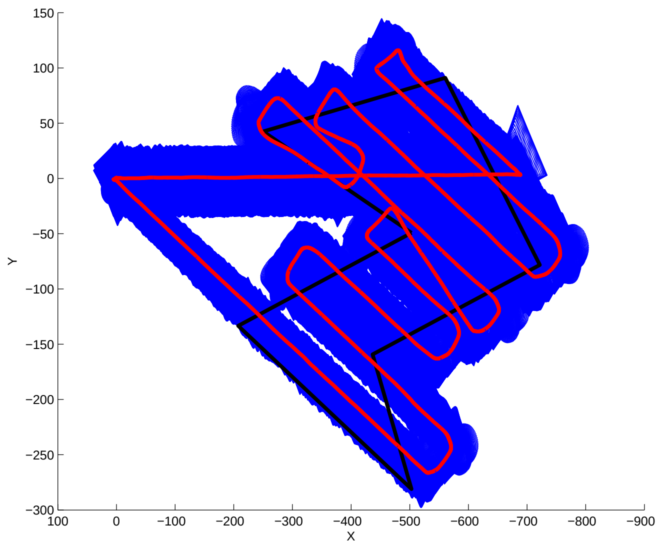


Figure 52. The Energy-aware planner sweep trajectory generated for a concave polygon. The area is marked by a thick black line. The flight trajectory is represented by the red line. The area covered by the sweep is in blue. Camera axis is considered vertical to UAV's body orientation.

As can be seen in Figures 51 and 52, the Energy-aware planner created a longer sweep trajectory than the EECD planner, which is due to unfavorable shape of the area. Another observed difference is that if the camera was fixed vertically to the UAV's body, the sweep would have not covered the entire area. There could be two reasons for that:

- Windy conditions. There might have been a wind breeze from one side, tilting the UAV away from the desired area.
- Flight altitude effect. The displayed plan was designed for the altitude of 25 meters, while the mean value of altitude along the trajectory ranges between 22 and 23 meters, which lowers the projection's area dimensions by 3-5 meters.

6.4. Test conclusions

We have verified that all the submitted planners are capable of successfully covering various areas. While software test results suggested the increased efficiency of Energy-aware planner in comparison with others, the field experiment revealed that its performance is highly dependent on the area's shape. The same can be applied to the EECD planner, which can provide us with very efficient plan during certain scenarios, while giving needlessly complex ones during others. The Naive planner, on the other hand, does not aim for efficiency, and as such provides us with not optimal, but generally usable solutions.

The NFZ avoidance addon was only partially successful as planners are able to create sweep trajectories that do not visit these zones. It is unable however to prevent the UAV from reaching the NFZ while making turns from one stripe to another.

7. Conclusion

7.1. Thesis summary

The main objective of this thesis was to implement coverage path planning methods and modify them for the deployment on fixed-wing UAVs, integrate them into a framework for UAV control and compare their performance during simulations. This objective was successfully completed. Two algorithms presented in [7] and [8] were implemented into the Tactical AGENTFLY framework along with intuitively formed naive approach. All three algorithms, differing in their structure, were thoroughly tested in various simulated scenarios and their performance was compared.

In addition, a method for including no-flight zones into the selected algorithms was implemented and tested and another planning algorithm was developed. The testing was performed not only in simulator, but also with a real aircraft.

The results of experiments displayed in Tables 2, 3, 4 and 5 showed that all the selected planners are able to fully cover the area of interest, even when complexly shaped. While specialized approaches towards polygon decomposition often yield more efficient results regarding the trajectory length, there are also cases when their performance becomes needlessly complex and redundant. The EECD planner based on the design presented in [8] proved to be efficient on polygonal shapes with relatively small amount of edges. The Energy-aware planner based on the approach presented in [7] is efficient in scenarios where the longest polygon edge defines its dominant dimension. The efficiency of the Naive planner is greater when used on simpler polygons and is reduced by growing complexity of presented scenarios. The no-flight zones implementation proved to be partially successful as the planner is able to create a sweep pattern without visiting these zones, but was unable to avoid them completely during the aircraft's maneuvering.

According to the data gathered, the difference in trajectory length can in some cases be even greater than 100 %, which makes the correct planner selection a crucial task. Choosing the optimal planner for the actual area can in effect greatly improve the task duration and thereby energy consumption, significantly reducing the overall cost of the mapping process.

All the thesis objectives were successfully completed:

- The theory of remote sensing was explained in Chapter 2.
- The overview of existing coverage path planning algorithms was presented in Section 3.1.
- The display of selected algorithms was presented in Section 3.2.
- The algorithm modifications are listed in Chapter 4.
- The functionality of selected approaches was verified in simulations presented in Chapter 6.

In addition, the following work not enlisted in the thesis assignment was included:

- The algorithm improvements are listed in Chapter 5.
- A new planning algorithm called the Limited-range planner was designed in Section 5.3.
- Field experiments were conducted and analyzed in Section 6.3.

All the simulation data and source codes are provided on the enclosed CD.

7.2. Future work

The work presented in this thesis can be improved and extended in multiple ways. One of the improvements could be trajectory alteration to avoid no-flight zones completely, even during UAV's maneuvers between its neighboring stripes. Another improvement could be implementing more complex polygon subdivision and merging methods, which would allow the Limited-range planner to be implemented, tested and evaluated.

The possible extensions for this work could contain adding another planning algorithms into the Tactical AGENTFLY framework. Another possibility could be converting the problem into 3D, which would on the one hand increase its difficulty, but which would on the other hand increase the realism during testing scenarios.

Appendix A.

Contents of the enclosed CD

```
/
├── diploma_thesis.pdf ............................this Thesis in PDF format
├── field_tests
│   ├── areas
│   │   ├── concave.txt ................vertices defining the concave testing area.
│   │   └── convex.txt ..................vertices defining the convex testing area.
│   ├── logs
│   │   ├── 2016_12_21_13_24_29_4.csv ..flight log of the sweep generated by the
│   │   │   EECD planner for the concave area
│   │   ├── 2016_12_21_13_38_14_5.csv ..flight log of the sweep generated by the
│   │   │   Naive planner for the convex area
│   │   └── 2016_12_21_14_18_18_1.csv ..flight log of the sweep generated by the
│   │       Energy-aware planner for the concave area
│   └── plans
│       ├── concave1.txt .list of waypoints generated by the EECD planner for the
│       │   concave area
│       ├── concave2.txt .list of waypoints generated by the Energy-aware planner
│       │   for the concave area
│       └── convex.txt ....list of waypoints generated by the Naive planner for the
│           convex area
└── simulations
    ├── areas
    │   ├── brd1.txt .................vertices defining the area from the first test
    │   ├── brd2.txt ..............vertices defining the area from the second test
    │   ├── brd3.txt .................vertices defining the area from the third test
    │   ├── brd4.txt ................vertices defining the area from the fourth test
    │   └── nfz.txt .....................no-flight zones present in the fourth test
    └── logs
        ├── t1-ea-k.txt .....flight log of the sweep generated by the Energy-aware
        │   planner with kinematic constraints for the first test
        ├── t1-ea-n.txt .....flight log of the sweep generated by the Energy-aware
        │   planner without kinematic constraints for the first test
        ├── t1-n-k.txt .flight log of the sweep generated by the Naive planner with
        │   kinematic constraints for the first test
        ├── t1-n-n.txt ..... flight log of the sweep generated by the Naive planner
        │   without kinematic constraints for the first test
        └── t2-ea.txt ......flight log of the sweep generated by the Energy-aware
            planner for the second test
```

    - `t2-eecd.txt` flight log of the sweep generated by the EECD planner for the second test
    - `t2-n.txt` . flight log of the sweep generated by the Naive planner for the second test
    - `t3-ea.txt` .... flight log of the sweep generated by the Energy-aware planner for the third test
    - `t3-eecd.txt` flight log of the sweep generated by the EECD planner for the third test
    - `t3-n.txt` . flight log of the sweep generated by the Naive planner for the third test
    - `t4-ea.txt` .... flight log of the sweep generated by the Energy-aware planner for the fourth test
    - `t4-eecd.txt` flight log of the sweep generated by the EECD planner for the fourth test
    - `t4-n.txt` . flight log of the sweep generated by the Naive planner for the fourth test
  - `plans`
    - `t1-ea-k.txt` .. list of waypoints generated by the Energy-aware planner with kinematic constraints for the first test
    - `t1-ea-n.txt` .. list of waypoints generated by the Energy-aware planner without kinematic constraints for the first test
    - `t1-n-k.txt` ...... list of waypoints generated by the Naive planner with kinematic constraints for the first test
    - `t1-n-n.txt` ... list of waypoints generated by the Naive planner without kinematic constraints for the first test
    - `t2-ea.txt` . list of waypoints generated by the Energy-aware planner for the second test
    - `t2-eecd.txt` .. list of waypoints generated by the EECD planner for the second test
    - `t2-n.txt` ...... list of waypoints generated by the Naive planner for the second test
    - `t3-ea.txt` . list of waypoints generated by the Energy-aware planner for the third test
    - `t3-eecd.txt` .. list of waypoints generated by the EECD planner for the third test
    - `t3-n.txt` list of waypoints generated by the Naive planner for the third test
    - `t4-ea.txt` . list of waypoints generated by the Energy-aware planner for the fourth test
    - `t4-eecd.txt` .. list of waypoints generated by the EECD planner for the fourth test
    - `t4-n.txt` list of waypoints generated by the Naive planner for the fourth test
- `source`
  - `PolygonPlanner.java` .... main java class for coverage planners
  - `Mesh.java` .... java class containing HDCEL structure
  - `EECDPlanner.java` .... java class containing the EECD planner

- `EnergyAwareCoveragePlanner.java` ....java class containing the Energy-aware planner
- `PolygonCoverageStatisticsThread.java` ....java class containing the measurement thread
- `demo.groovy` ......groovy configuration script containing pre-defined flight scenarios
- `mapping.groovy` . groovy configuration script containing UAV's parameters

Bibliography

[1] *Orthophotography (2016).* URL: `http://www.aerialmapping.net/Orthophotography.html`.

[2] *Fixed Wing Versus Rotary Wing For UAV Mapping Applications (2016).* URL: `http://www.questuav.com/news/fixed-wing-versus-rotary-wing-for-uav-mapping-applications`.

[3] *AgentFly | Agent Technology Center (2016).* URL: `http://agents.felk.cvut.cz/projects/agentfly`.

[4] MM Flood. *The Traveling-salesman problem (1955).*

[5] *Introduction To UAV Photogrammetry And Lidar Mapping Basics | DroneZon (2016).* URL: `https://www.dronezon.com/learn-about-drones-quadcopters/introduction-to-uav-photogrammetry-and-lidar-mapping-basics/`.

[6] Toni Schenk. "Introduction to photogrammetry". In: *The Ohio State University, Columbus (2005)* ().

[7] Carmelo Di Franco and Giorgio Buttazzo. "Energy-aware coverage path planning of UAVs". In: *Autonomous Robot Systems and Competitions (ICARSC), 2015 IEEE International Conference on.* IEEE, pp. 111–117.

[8] Yan Li et al. "Coverage path planning for UAVs based on enhanced exact cellular decomposition method". In: *Mechatronics (2011)* 21.5 (), pp. 876–885.

[9] PB Sujit and Randy Beard. "Multiple UAV exploration of an unknown region". In: *Annals of Mathematics and Artificial Intelligence (2008)* 52.2-4 (), pp. 335–366.

[10] Liam Paull et al. "Sensor-driven area coverage for an autonomous fixed-wing unmanned aerial vehicle". In: *IEEE transactions on cybernetics (2014)* 44.9 (), pp. 1605–1618.

[11] Hosein Khandani, Hadi Moradi, and Javad Yazdan Panah. "A real-time coverage and tracking algorithm for UAVs based on potential field". In: *Robotics and Mechatronics (ICRoM), 2014 Second RSI/ISM International Conference on.* IEEE, pp. 700–705.

[12] Joon Seop Oh et al. "Complete coverage navigation of cleaning robots using triangular-cell-based map". In: *IEEE Transactions on Industrial Electronics (2004)* 51.3 (), pp. 718–726.

[13] LH Nam et al. "An approach for coverage path planning for UAVs". In: *Advanced Motion Control (AMC), 2016 IEEE 14th International Workshop on.* IEEE, pp. 411–416.

[14] PB Sujit, BP Hudzietz, and Srikanth Saripalli. "Route planning for angle constrained terrain mapping using an unmanned aerial vehicle". In: *Journal of Intelligent & Robotic Systems (2013)* 69.1-4 (), pp. 273–283.

[15] Israel A Wagner, Michael Lindenbaum, and Alfred M Bruckstein. “Distributed covering by ant-robots using evaporating traces”. In: *IEEE Transactions on Robotics and Automation (1999)* 15.5 (), pp. 918–933.

[16] João Valente et al. “Aerial coverage optimization in precision agriculture management: A musical harmony inspired approach”. In: *Computers and electronics in agriculture (2013)* 99 (), pp. 153–159.

[17] Chaomin Luo et al. “Safety aware robot coverage motion planning with virtual-obstacle-based navigation”. In: *Information and Automation, 2015 IEEE International Conference on.* IEEE, pp. 2110–2115.

[18] Anqi Xu, Chatavut Viriyasuthee, and Ioannis Rekleitis. “Efficient complete coverage of a known arbitrary environment with applications to aerial operations”. In: *Autonomous Robots (2014)* 36.4 (), pp. 365–381.

[19] Wesley H Huang. “The minimal sum of altitudes decomposition for coverage algorithms”. In: *Rensselaer Polytechnic Institute Computer Science Technical Report 00-3 (2000)* ().

[20] Andreas Bircher et al. “Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots”. In: *Autonomous Robots (2015)* (), pp. 1–20.

[21] JF Araujo, PB Sujit, and João B Sousa. “Multiple UAV area decomposition and coverage”. In: *2013 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA).* IEEE, pp. 30–37.

[22] Franco P Preparata and Michael Shamos. *Computational geometry: an introduction.* Springer Science & Business Media (2012).

[23] *lecture | Small Unmanned Aircraft: Theory and Practice (2016).* URL: `http://uavbook.byu.edu/doku.php?id=lecture`.

[24] *Understanding Euler Angles | CH Robotics (2017).* URL: `http://www.chrobotics.com/library/understanding-euler-angles`.