



**Intelligent Information System Supporting
Observation, Searching and Detection for
Security of Citizens in Urban Environment**



European Seventh Framework Programme
FP7-218086-Collaborative Project

Deliverable 2.7 Proposed algorithms for mission planning for groups of UAVs

The INDECT Consortium

AGH — University of Science and Technology, AGH, Poland
Gdansk University of Technology, GUT, Poland
InnoTec DATA GmbH & Co. KG, INNOTEC, Germany
IP Grenoble (Ensimag), INP, France
MSWiA — General Headquarters of Police (Polish Police), GHP, Poland
Moviquity, MOVIQUITY, Spain
Products and Systems of Information Technology, PSI, Germany
Police Service of Northern Ireland, PSNI, United Kingdom
Poznan University of Technology, PUT, Poland
Universidad Carlos III de Madrid, UC3M, Spain
Technical University of Sofia, TU-SOFIA, Bulgaria
University of Wuppertal, BUW, Germany
University of York, UoY, Great Britain
Technical University of Ostrava, VSB, Czech Republic
Technical University of Kosice, TUKE, Slovakia
X-Art Pro Division G.m.b.H., X-art, Austria
Fachhochschule Technikum Wien, FHTW, Austria

©Copyright 2010, the Members of the INDECT Consortium

Document Information

Contract Number	218086
Deliverable Name	Proposed algorithms for mission planning for groups of UAVs
Deliverable number	Deliverable 2.7
Editor(s)	Paweł Lubarski, PUT
Author(s)	Krzysztof Witkowski, PUT
Reviewer(s)	Prof. DSc. PhD. Eng. Chief Mate Lucjan Gucma, Maritime University
Dissemination level	public
Contractual date of delivery	30-06-2010
Delivery date	30-06-2010
Status	v. 1.0
Keywords	mission, planning, swarm



This project is funded under 7th Framework Program

Contents

Document Information	3
Executive Summary	7
Introduction	9
1 Mission planning architecture overview	11
1.1 Basic terminology	11
1.2 Layers	11
1.2.1 Software architectures for robotics	11
1.2.2 Recent UAV software architectures	12
1.2.3 Proposed high-level architecture	13
1.3 Interfaced components	15
1.4 Related problems and fields	16
1.5 Mission simulation environment	16
1.5.1 Rationale	16
1.5.2 Components	16
2 Motion planning	17
2.1 Background	17
2.2 Assumptions	18
2.3 Path planning with no fly zones	18
2.3.1 Pre-processing of no fly zones representation	18
2.3.2 Visibility graph	19
2.3.3 Continuous Dijkstra	19
2.4 Trajectory generation	21
2.4.1 No wind case	21
2.4.2 Constant wind case	22
2.4.3 Multiple waypoints	23
3 Surveillance planning	23
3.1 Applications	23
3.2 Surveillance capabilities	24
3.3 Anatomy of surveillance	24
3.4 Pinpoint coverage	25
3.4.1 Minimum Spanning Tree Approximation for Traveling Salesman Problem	25
3.4.2 Min-Max Tree Covers of Graphs	25
3.5 Line coverage	27
3.6 Area coverage	27
3.6.1 Exact Boustrophedon decomposition	28
3.6.2 Approximate Spanning Tree Covering (STC)	29
3.6.3 Multi-Robot Boustrophedon Exact Coverage	31
3.6.4 Multi-Robot Forest Coverage	31
3.6.5 Backtracking Multi-Robot Spanning-Tree Coverage	32
3.6.6 Continuous Multi-Robot Approximate Spanning Tree Coverage	33
3.7 Tracking	33
3.7.1 Hopf Bifurcation Target Following	33
3.7.2 Mode-Switching Target Following	35
Conclusions	36

Executive Summary

In this document we describe a proposed algorithms for mission planning for groups of UAVs. It is a realization of deliverable D2.7 and complementary work to deliverable D2.6, that describes cooperation in groups of UAVs from broader perspective.

In Section 1 a high level software architecture designed for UAV is proposed. It is three-layer hybrid architecture, that enables robust coupling of deliberate planning and reactive control. Necessary simulation environment is also described, whose usage will supposedly decrease the development time and increase the outcome system quality.

In Section 2 algorithms for motion planning are described. Motion planning problem is decomposed in two stages: path planning and trajectory generation. This decomposition allows development of independent solutions for obstacle avoidance and wind compensation.

In Section 3 we extend planning algorithms to show how motion planning has to be arranged in order to guarantee accomplishment of surveillance mission goals.

Introduction

The aim of the INDECT project is the creation of an integrated information system to detect and prevent threats in order to ensure security of citizens in the urban environment. The integrated monitoring system with real-time detection of dangerous situation will facilitate the supervision of mass events, in particular during European Championships, which will hold in Poland & Ukraine in 2012. The monitoring system offered by INDECT, which consists of a set of subsystems (node stations, UAVs, cameras, etc.) working together, will analyze different factors in real-time and facilitate decision making by police and security services.

The Integrated Air Surveillance System for police departments will work on the basis of a family of autonomous, intelligent, unmanned vehicles of different types. Their purpose will be to make reconnaissance flights for police officers working in the field. Observation and tracking of objects (including dangerous materials like chemicals, biological and toxic substances in transit) will be facilitated by the use of a variety of technologies, including different types of UAVs.

Development of Unmanned Aerial Vehicles (UAVs), basis of the Integrated Air Surveillance System for police departments, is one of the three objectives for WP2. Numerous algorithms are required for successful planning of patrol mission, including path planning with obstacle avoidance capabilities, trajectory generation with wind compensation techniques, area coverage for spatial surveillance and finally efficient and robust methods for task balancing among multiple UAVs. Using modern methods and algorithms of artificial intelligence for autonomous groups of UAVs operations is essential in order to overcome current UAV group operations limitations.

Each UAV is equipped with camera, line of sight limited range narrow-band communication link to nearby fellow UAVs and has full knowledge about no fly zones in his region of interest. Some of the UAVs have gimbal mounted cameras. The camera is used for detecting potential object of interest during observation missions and for visual tracking and positioning followed objects during pursuit mission. Collision avoidance between members of team is asserted by dedicated negotiation procedures of approaching vehicles.

1. Mission planning architecture overview

1.1. Basic terminology

This report describes proposed algorithms for mission planning for groups of *unmanned aerial vehicles*. In the scope of INDECT project, the UAVs are limited to *fixed-wing* aircrafts, that exploit lift force generated as the wings move forward through the air, which differentiate them from the *rotorcrafts*, that exploit lift force generated as the rotor blades revolve around the mast. Important property of fixed-wing aircrafts is that they must maintain velocity above certain minimum to avoid *stall*. The groups are *heterogeneous*, because aircrafts can differ in minimum and maximum airspeed, potential operating range (depending on fuel consumption and fuel tank capacity), available sensors and their properties etc.

Autonomy means that vehicles are capable of accepting high-level task ordered by human operators and through sequences of low-level simple actions are capable of accomplishing the desired goal. The process of decomposing the high level goal or task into sequence of simple, basic actions is called *planning* and process of executing these actions is called *execution*. In order to plan and execute its actions autonomous robot might need to maintain the map of an environment in which it operates and estimate its position. These problems are called *mapping* or *map building* and *localization*, and if they are solved together they are referred to as *simultaneous localization and mapping* problem. In the scope of INDECT project, the necessary environment maps are available, therefore only localization problem applies.

As far as planning for UAVs is concerned, we are mainly dealing with *motion planning* problems. The domain of motion planning is continuous space (as opposed to discrete), which is the primary distinction from most other forms of planning. A motion plan describes route of vehicle in time that should be followed to displace from initial to final position without entering prohibited zones (e.g obstacles or enemy radar range in military applications). It should also be noted, that since the motion planning domain is infinite then planning algorithm must exploit implicit representations, and it can not simply evaluate all possible actions to make a decision.

From all potential motion plans a planning procedure must choose one that is *feasible* and, desirably, *optimal*. A plan is feasible if it obeys related constraints and therefore can be safely realized by vehicle, a physical agent. Optimality depends on the assumed criteria, that can be e.g to minimize traveled distance, fuel consumption or maximize area covered with spatial sensor or probability of capturing the evader. In the real world applications it is often computationally impossible to calculate optimal solutions and approximate or heuristic approaches are used.

Mission planning is a such motion planning, that guarantees meeting the *mission objectives* specified by the operator. If the objective is to search area we are dealing with *coverage planning*, i.e. planning a motion that guarantees observing requested area with available sensor. In *multi-agent or multi-robot planning* the goal is to find a set of single-agents plans, whose joint execution will satisfy mission requirements.

Multi-robot mission plans are used in *multi-robot cooperation*, a broader term that encompasses whole multi-robot mission lifespan: from submission, through planning, execution to reporting. It must therefore propose a general framework for communication and interactions between robots.

1.2. Layers

1.2.1. Software architectures for robotics

Several approaches have been proposed for implementing behavior of autonomous goal-driven robots. Early methodology, so called *Sense-Plan-Act* (Figure 1), advocated iterative three stage process in which input from sensors is passed to planning procedures, current state of the world environment is built and finally planner delivers a requested action for the action phase. However, the need to entwine sensing and acting with planning is inherently sequential and as a result it introduces unacceptable delays. Therefore architecture failed to deliver robust framework for real-time autonomous robots.

Alternative approach originates from an assumption, that complex behaviors are not necessarily products of complex control systems. Bottom-up combination of simple pre-wired

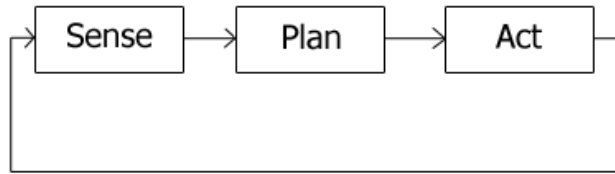


Figure 1: Schematic diagram of Sense-Plan-Act robotic paradigm



Figure 2: Schematic diagram of reactive robotic paradigm

reactive controllers (Figure 2), working asynchronously, without constructing any explicit representation of the surrounding environment can drive complex, seemingly intelligent emergent behavior. The reactive paradigm organize the components vertically so that there is a direct route from sensors to effectors, without explicit planning. Hence, there is the danger of conflicting controlling signals (as in the example of steering a car to exit from the highway while avoiding an accident with the car in the lane to your right). Brooks suggests solving this problem by allowing components at one level to subsume components at a lower level. Thus he called his approach the subsumption architecture.

Although subsumption architecture overcomes performance problems of SPA architecture thanks to its parallelism, designing a purely reactive controller for real world problem is in practice neither simple nor even sometimes possible. Therefore hybrid architecture (Figure 3) that combines strengths of reactive with deliberate models has been proposed.

It consists of three layers:

- the reactive layer, responsible for low-level control of the robot, which is operating in tight sensor-action loop,
- the executive layer, responsible for commanding the reactive layer in order to follow the plan produced by the deliberative layer,
- the deliberative layer, responsible for planning.

This approach combines strength of explicit problem solving through planning, while at the same time allowing real-time response.

1.2.2. Recent UAV software architectures

Center for Collaborative Control of Unmanned Vehicles

In their excellent paper about architecture and applications of UAV fleets Sengupta et al. [18] propose to decompose mission planning into three basic mission types. They argue that every application of UAV fleet can be composed from three basic behaviors: Traveling, Watching and Tracking. These behaviors are itself composed from basic actions: displacing and loitering. In the proposed semantics Traveling is visiting of set of points or open path exactly once and it has temporal objective. Watching is repeated visiting set of points or closed path, optimized for

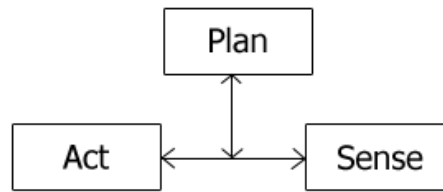


Figure 3: Schematic diagram of hybrid robotic paradigm

spatial objective. Tracking is following a single point (static or dynamic) infinitely, optimized to relate the UAV path with chosen object of the environment.

The concrete implementations of behaviors vary depending on the path dimensions. Zero-dimensional path is a point, which should be kept or followed. One dimensional path concerns displacing between two different positions. Two dimensional path means following path that covers the area, while three dimensional path means following path that covers the space. In order to define the path its dimension and parameters are specified.

In order to define the semantics of a concrete mission transition diagram between behaviors is created. The article presents example diagrams for patrol, convoy, road following and search & rescue missions. It is important to note, that presented model is suitable both for single and multi UAV missions.

Their software architecture (Figure 4) consists of Guidance, Control and Navigation components. Guidance is further decomposed into Collaboration and Trajectory Generation layers, Control into Safety and Regulation layers, while Navigation into Localization and Mapping layers. Collaboration decides which behavior should be activated in order to fulfill mission goals. Trajectory generation layers are responsible for actually planning this behavior. Collision avoidance prevents from collisions with unexpectedly encountered objects. Path tracking is responsible for keeping the aircraft on the planned trajectory during the flight.

Brigham Young University

Multi layer architecture implemented by Brigham Young University [19] is presented in Figure 5. The layers are decomposed to deal separately with:

- waypoint path planning, that guarantee fulfilling mission goals and avoiding obstacles,
- trajectory smoothing, that guarantee obeying differential constraints of non-holonomic vehicle motion,
- trajectory tracking, that generate concrete velocities, altitudes and turns during the mission execution in the presence of imperfection of the steering process,
- flight regulation, that tries to follow the settings of the trajectory tracking layer and directly communicates with platform devices.

1.2.3. Proposed high-level architecture

Autonomous flight of group of UAVs requires several subsystems to cooperate tightly. We adopt classic hybrid reactive-deliberate three-layer architecture for robotic software. Therefore we distinguish between

- the reactive layer, responsible for low-level control of the aircraft's servos and sensors,
- the executive layer,

and

- the deliberate layer, responsible for motion planning.

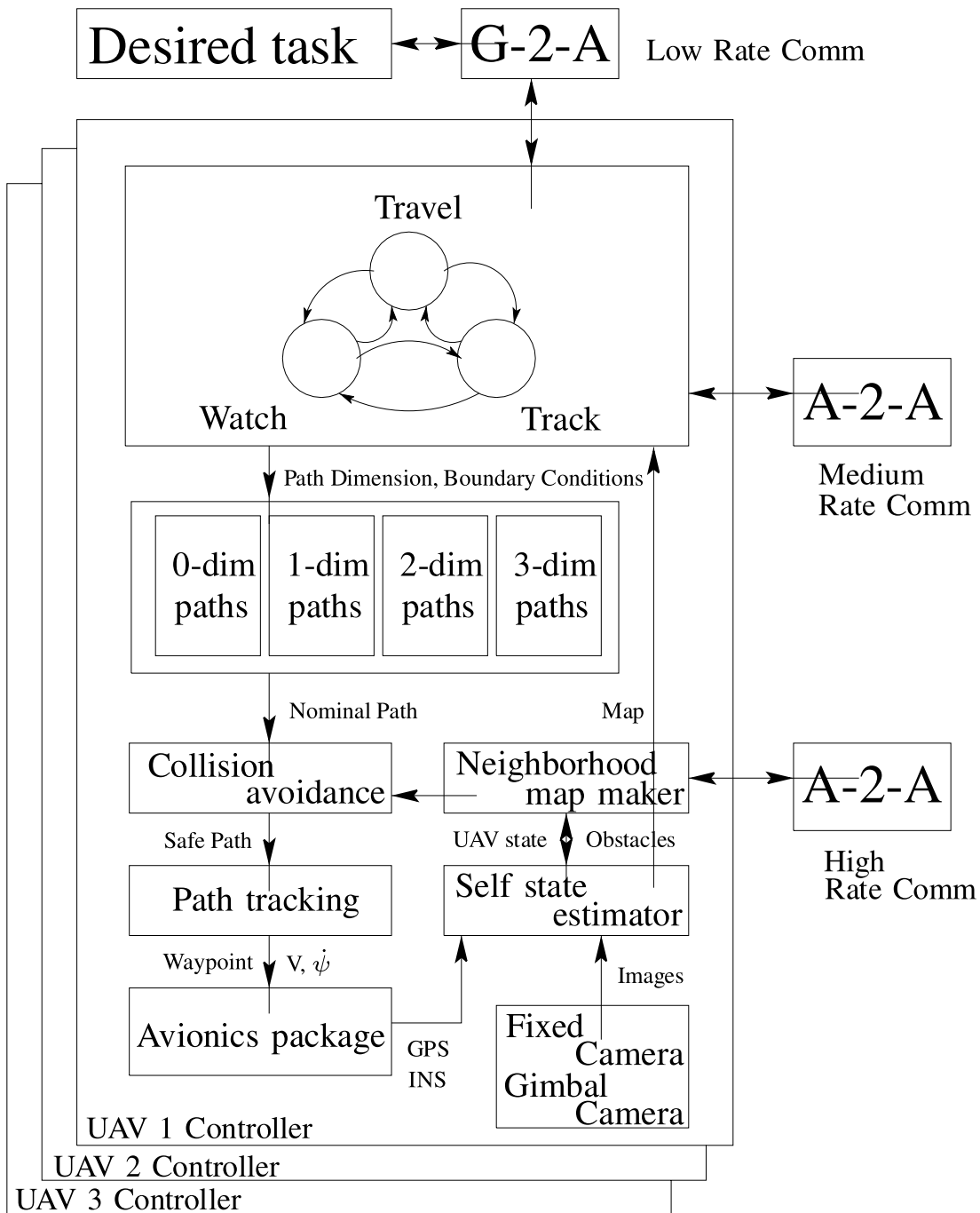


Figure 4: The architecture developed by the Center for Collaborative Control of Unmanned Vehicles (C3UV) at the University of California, Berkeley

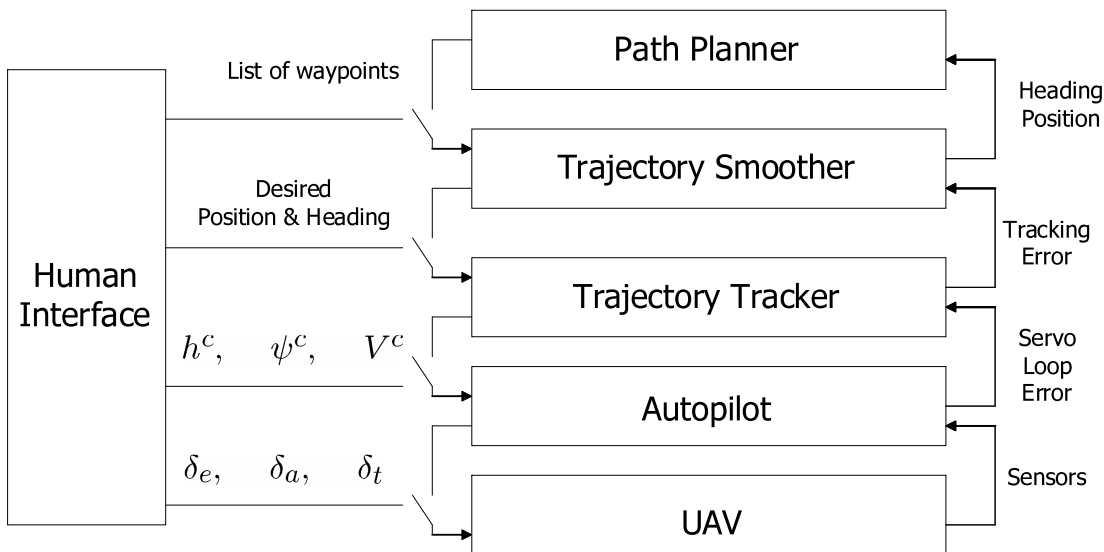


Figure 5: The architecture developed by the team at Brigham Young university

We propose adopting common in UAV literature even further, finer-grained decomposition of the deliberate layer into:

- cooperation manager layer, responsible for dividing task between multiple vehicles and other group-aware behaviors,
- path planning layer, responsible for producing list of characteristic waypoints according to commissioned task,
- trajectory smoothing layer, responsible for converting waypoints into time-parametrized trajectories fulfilling the differential constraints of the UAV.

Cooperation manager is the highest level planning component. Depending on the events received during mission execution, operator commands or state of the aircraft it activates proper planning procedures. Example event that can be received, independently from the type of currently executed mission, is an emergency event like reaching low fuel levels. Mission dependent events include detecting object that match current search criteria, failure to locate currently followed target, timer event, mission completion or an operator command according to the mission scenario.

1.3. Interfaced components

Each UAV is equipped in camera, line of sight limited range narrow-band communication link to nearby nodes (fellow UAVs or ground station) and has full knowledge about no fly zones in his region of interest. Some UAVs have gimbal-mounted cameras.

In order to fully exploit capabilities of the swarm, the UAVs need to be able to inform other members of group about their state, mission status (accomplished, failed etc). Each aircraft is equipped with communication system, which enables communication between this UAV and ground station and nearby aircrafts. UAVs maintain a routing network, therefore communication is possible also if it is possible indirectly (UAVs are out of communication range, but there is a sequence of nodes that can relay their communication).

The camera is used for detecting potential object of interest during observation missions and for visual tracking and positioning followed objects during pursuit mission. Although gimbal camera has multiple parameters that can be controlled, there is no need for separate deliberate planning of gaze direction, focus, zoom, contrast etc. Three degrees of freedom in orientation compensate for imperfect UAV path following during execution stage in order to minimize the deviation of the observed area from the one assumed in plan. Desired optical parameters are set by human operator or controlled by visual recognition subsystem. Motion planning is based on

an assumption, that the camera is pointing vertically downwards, in order to minimize occlusion problems, therefore camera is imaging a rectangle of a constant size.

Collision avoidance between members of team is accomplished by dedicated negotiation procedures of approaching vehicles.

1.4. Related problems and fields

Mission planning covers only a subset of problems that need to be solved in order to provide software for autonomous aerial vehicle.

The domain of *automatic control* aims to control the behavior of a dynamic system and is the basis for the reactive and execution layers. *Computer vision* methods allow for processing images captured with cameras in order to build map of environment and localize target objects. *Probability theory* is extremely useful in both of these domains to obtain trustworthy estimates of required data from multiple data sources in time, while *distributed artificial intelligence* and *decision theory* lay the foundations for the design of multi-agent systems.

1.5. Mission simulation environment

1.5.1. Rationale

Studying mission planning and cooperation mechanisms for multiple UAVs requires development of a dedicated simulation framework in which implemented algorithms can be tested and analyzed, before final validation during real world flight tests. Mission planning and execution software is composed of mutually dependent layers and robust interactions between cooperation, mission planning and execution layers should be thoroughly verified to eliminate malfunctions. In addition, real world experiments are time consuming, therefore performing large number of different surveillance scenario experiments in a simulated environment allows ensuring efficacy of developed algorithms in different tactical situations.

Each UAV is an independent physical entity with its own limited computational and communication resources. Aircrafts must cooperate in real-time within their capabilities, therefore the environment should be capable of testing also these non-functional aspects.

1.5.2. Components

Considering the requirements for the simulator, an agent based approach seems to be the most suitable. The environment should consist of three distinct components types:

- agents,
- agents' environments,
- communication mechanisms.

Agents

Agents represent acting entities in the simulation. It might be a human operator, an UAV or a ground vehicle under surveillance. Agents representing UAVs would be running the software stack under test. Human operator actions or ground vehicle behavior could be specified interactively, could be fixed in a scenario experiment description or could be determined by a simple heuristics. Two different experiment types could be therefore performed: interactive and non-interactive.

The UAV agents software stack should be running as an independent system process on a dedicated computing resource with computing power determined by the scenario description. It should be communicating with other agents only through the environment communication mechanisms, that would exhibit desired properties (bandwidth etc). In such a way non-functional requirements would be asserted.

Environment

The agents actions are performed in the environment. In the scope of the foreseen scenarios, actions would be limited to movement and camera observations. The environment must therefore provide motion and camera models, both of which depend on the spatial model.

The spatial model maintains the list of objects in the simulated world and the terrain shape.

The simulated motion model could be kinematic (motion is specified by velocity, acceleration etc.) or dynamic (motion results from the forces acting on the robot body), it should check for collisions between agents and between agents and obstacles. It might introduce factors that influence mission execution, such as constant or varying wind. Both motion simulation models will find its applications. Kinematic model will be useful for isolated tests of planning algorithms, while dynamic model will provide framework for thorough evaluation of full software stack including autopilot, trajectory tracking and planning layers and their interaction.

The camera model determines what part of the environment is captured by the agent camera, since part of the viewing field might be hidden by obstacles.

Communication mechanisms

Since mission plans are executed in the cooperative fashion, the agents must exchange messages. The requirement for this component is that the topology and characteristics of communication links during the simulation corresponds to topology and characteristics of communication during real missions.

2. Motion planning

2.1. Background

Continuous UAV motion planning is planning in a continuous space, with uncountably infinite number of possible states.

It can be divided into two types according to existence of available information. If perfect information about robot and environment is assumed, then we have *motion planning with complete information*. Contrary, if only local information from sensors is available we have *motion planning with incomplete information*. Even though assumption about perfect information is unrealistic, planning methods based on this assumption are justified as due deviation from plan can be compensated by an execution layer.

Methods for motion planning can be coarsely divided into combinatorial and sampling-based. Former operate on a discrete representation of the input and lead to complete planning approaches, which are guaranteed to find a solution whenever it exists, or correctly report failure if one does not exist. The latter sample the solution space and conduct discrete search that utilizes these samples. In this case, completeness is sacrificed, but it is often replaced with a weaker notion, such as resolution completeness or probabilistic completeness. Even though, in general, the drawbacks of combinatorial methods are computational complexity and implementation effort required, the trade-off between quality and required development time is acceptable in the case of UAV motion planning.

Another level of distinction concerns physical aspect of planning. Methods are called *dynamic*, if forces acting on a robot body are taken into account or *kinematic*, if the body dynamics is neglected. Reported domain experience suggests that kinematic modeling is sufficient at the mission planning level.

We say the robot is *non-holonomic*, if the number of controllable degrees of freedom is smaller than the total degrees of freedom. Otherwise it is *holonomic*. Non-holonomic planning involves obeying resulting differential constraints in planning.

Following long tradition [18, 19, 2] we decompose motion planning into path and trajectory planning. Path planning stage is responsible for determining characteristic waypoints that guarantee fulfilling goals of the particular mission, while trajectory generation routines can transform the path into continuous curve that meets the curvature constraints, together with desired time law of motion.

2.2. Assumptions

Although dynamic modeling is not used explicitly in motion planning, the specifics of aircraft dynamics introduces constraints on the minimum and maximum airspeed of the aircraft. Specifically, minimum admissible airspeed is required to avoid stall, while thrust limits the maximum airspeed of the vehicle.

Within these limits it is desirable to maintain a constant airspeed for keeping the fuel consumption low. For the same reason it is desirable to maintain constant altitude.

Another important assumption is that of flat-earth approximation. This approximation fails in the vicinity of poles and is reasonable for planning only limited distance missions. The UAVs designed for the INDECT project are small and capable of only limited range missions, therefore this assumption is justified. It should be noted, however, that the assumption is not essential for the proposed algorithms, but is rather necessary for preparing input parameters.

2.3. Path planning with no fly zones

As mentioned earlier, in the scope of our scenario we are given complete information about the environment, therefore the list of no fly zones is known in advance. This means that they can be incorporated into the path planning process from the beginning. When given a list of no fly zones, UAV must be able to plan its motion in such a manner, that these constraints are respected. We are assuming that availability of the no fly zones do not change during the time-span of a single mission. Another initial assumption is that no fly zones are specified to path planning algorithm as n -sided prisms, extending from the ground upwards. It is therefore possible to simplify the problem to two-dimensional, which is beneficial from the run-time efficiency point of view.

Different criteria can be placed for rating an algorithm for path planning among obstacles. Specifically, two conflicting goals are:

- maximum clearance - i.e. staying as far as possible from the obstacles,
- minimum distance - i.e. traveling along the shortest path.

In other words, we have to choose between the safest and the shortest path for an UAV. Maximum clearance solutions are obtained using Voronoi diagrams. However, in our project we focus on the shortest paths and assert safety in the manner described in the next section.

In the scope of our project we propose implementation of two algorithms with different conceptual origin. First algorithm is using a radial sweep principle to build a visibility graph (road-map) between obstacles vertices. This visibility graph is used afterward to compute the shortest path using the classic Dijkstra algorithm. Every algorithm that constructs an entire visibility graph to calculate the shortest path is bound to have at least quadratic time complexity (in terms of number of obstacle vertices), because the graph might have at most quadratic number of edges (this is the case with the full graph). Therefore a second method, whose usage is proposed, is an optimal wavefront propagation algorithm, that computes a planar map (in the form of a decomposition of space, and not a visibility graph), which later can be used to answer shortest path queries.

It is worth noticing, that intermediate structures for computing safe paths (whose description will follow), can be shared among all UAVs in the group operating in one area.

2.3.1. Pre-processing of no fly zones representation

No fly zones passed to motion planning procedures describe regions, which must not be entered. Planning the shortest path among these zones produces paths that lie on their borders. Due to the non-holonomic nature of aircrafts and unavoidable off-trajectory drift during the flight (execution phase), paths planned in such a way would inevitable result in entering the prohibited regions.

This can be easily and efficiently avoided by offsetting obstacle polygons with safety radius r , at least the size of the maximum turning radius (to compensate for non-holonomic nature of the aircraft). The exact necessary value to compensate for execution phase errors must be established experimentally and depends on the quality of the autopilot and trajectory tracking

subsystems. Evaluation of these subsystems will allow a trustworthy estimate of the final offset radius, that will guarantee safe execution of the path. Under the assumption of low density of no-fly zones, even an extremely cautious estimate (i.e. large value of the offset radius) shall result in paths shorter than in the maximum clearance approach.

Offsetting region by radius r is equal to performing Minkowski sum of that region with a disc of radius r centered at the origin. Given two sets $A, B \in \mathbb{R}^d$ their Minkowski sum, denoted by $A \oplus B$, is the set $\{a + b \mid a \in A, b \in B\}$.

The result of the Minkowski addition of a polygon and a disk is not a polygon. As chosen algorithms require obstacles in the form of polygonal regions, polygonal approximations instead of exact results of Minkowski addition should be calculated.

2.3.2. Visibility graph

The radial-sweep method finds the shortest path among polygonal obstacles in $O(n^2 \log n)$ time and $O(n^2)$ space, where n is the number of polygonal obstacles vertices. It is an example of an algorithm that initially constructs the entire visibility graph, that is afterward used to compute the shortest path, using the classic Dijkstra algorithm.

Visibility graph is such a graph $G(V, E)$, whose vertices represent obstacles vertices and characteristic points for each problem instance (starting, goal and possibly required intermediate positions). The edge between two vertices exists if they are mutually visible i.e. the ray between these vertices does not intersect the interior of any obstacle.

Naive approach to calculate a visibility graph is to directly check whether ray between every two distinct obstacle vertices crosses any obstacle edges. However, the algorithm based on a popular in computational geometry *plane sweep principle* can be applied. General idea of algorithms based on this principle is to imagine a line that is moved across the plane, intersecting some characteristic points. The solution is incrementally constructed as the line intersects these points and complete solution is available after the line has passed over all objects. The sweep line at each moment determines the *status* of the algorithm, most often the set of segments intersected by the line.

In the algorithm for computing a visibility graph a sweep line is rotated around the vertex for which the visible vertices should be found. For each candidate neighboring point (algorithm 1), if it is located behind any of the obstacle edges intersected by the sweep line, it is marked as invisible. Maintaining the intersected edges as the status of the radial sweep loop allows to compare point only with the relevant (closest) edge, instead of comparing with all obstacle edges (algorithm 2). In order to construct a complete visibility graph, the algorithm 1 must be repeatably called to detect neighbors of every vertex.

The repeating process of answering queries can be optimized by maintaining a visibility graph for obstacle vertices only. Then answering two point query requires to find visible vertices for starting and destination positions (in $O(n \log n)$ time) and running the Dijkstra shortest path algorithm, and therefore has only $O(n^2)$ time complexity.

2.3.3. Continuous Dijkstra

Hershberger and Suri [15] have presented an optimal algorithm based on an efficient wavefront propagation approach with the worst-case $O(n \log n)$ time complexity, that requires $O(n \log n)$ space. Wavefront propagation (or continuous Dijkstra method) simulates the expansion of a wavefront from a point source in the presence of polygonal obstacles. At time t it consists of all points whose shortest-path distance to the source equals t . It is composed from arcs generated by the source vertex and obstacle vertices already covered by the front. Simulating wavefront requires tracking events that change its topology, that fall into two categories: wavefront-wavefront collisions and wavefront-obstacle collisions. Hershberger and Suri optimal solution uses a specialized quad-tree-like decomposition of the plane, as they have observed that efficient advancement of a wavefront without a proper plane subdivision is difficult. Given the subdivision, the cells are the units for propagation and in each step the wavefront advances through exactly one cell. Another essential concept is propagation of only the approximate wavefront, that represents only the wavefronts hitting cell edge from one side, therefore two separate wavefronts are maintained, approaching the edge from opposite sides. At the end of

Algorithm 1 Radial sweep visible vertices (after Berg et al. [4])

```
1: procedure RADIALVISIBLEVERTICES( $p, S$ )  $\triangleright$  A set  $S$  of polygonal obstacles and a point  $p$ 
   that does not lie in the interior of any obstacle
2:   Sort the obstacles' vertices according to the clockwise angle that the half-line from  $p$  to
   each vertex makes with the positive  $x$ -axis. In case of ties, vertices closer to  $p$  should come
   before vertices farther from  $p$ . Let  $w_1, \dots, w_n$  be the sorted list.
3:   Let  $\rho$  be the half-line parallel to the positive  $x$ -axis starting at  $p$ . Find the obstacle
   edges that are properly intersected by  $\rho$ , and store them in a balanced search tree  $T$  in the
   order in which they are intersected by  $\rho$ .
4:   for  $i \leftarrow 1$  to  $n$  do
5:     if RadialVisible( $w_i$ ) then
6:       Add  $w_i$  to  $W$ 
7:       Insert into  $T$  the obstacles edges incident to  $w_i$  that lie on the clockwise side of
   the half-line from  $p$  to  $w_i$ 
8:       Delete from  $T$  the obstacle edges incident to  $w_i$  that lie on the counterclockwise
   side of the half-line from  $p$  to  $w_i$ 
9:     end if
10:  end for
11:  return  $W$ 
12: end procedure
```

Algorithm 2 Radial sweep visibility predicate (after Berg et al. [4])

```
1: procedure RADIALVISIBLE( $w_i$ )  $\triangleright$  A polygonal obstacle vertex  $w_i$ 
2:   if  $\overline{pw_i}$  intersects the interior of the obstacle of which  $w_i$  is a vertex, locally at  $w_i$  then
3:     return false
4:   end if
5:   if  $i = 1$  or  $w_i$  is not on the segment  $\overline{pw_i}$  then
6:     Search in the balanced search tree  $T$  for the edge  $e$  in the leftmost leaf
7:     if  $e$  exists and  $\overline{pw_i}$  intersects  $e$  then
8:       return false
9:     end if
10:  end if
11:  return true
12: end procedure
```

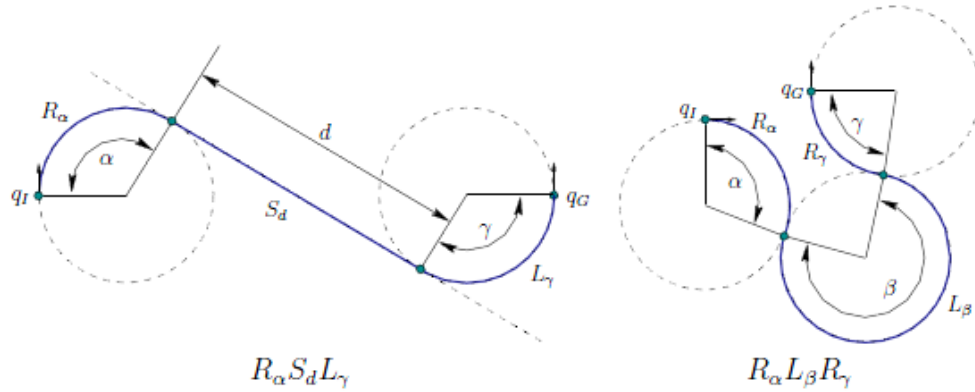


Figure 6: Two types of Dubins trajectories (after [2])

the propagation phase Voronoi diagram techniques together with collected wavefront collision information is used to compute the exact wavefront.

Although the algorithm offers better worst-case time and space complexity than previous one, its implementation is substantially more complex and time consuming. Early experiments shall answer the question whether its implementation is in fact necessary.

2.4. Trajectory generation

Standard assumption in the autopilot design is decoupling longitudinal and lateral dynamics. Consecutively, altitude is input for longitudinal autopilot and heading is input for lateral autopilot. Similarly, considering aircraft motion within limited distance, planar motion can be decoupled from altitude change, therefore leaving us with classic Dubins car path planning problem. This approach has been mentioned by La Valle [2].

The desired path is determined by a sequence of waypoints output by path planning algorithms. Motion planning must respect differential constraints of UAV motion, therefore time-parametrized curves of limited-curvature must be generated from that sequence. What follows, each point in the curve has attached an associated orientation (tangent to the trajectory curve at the point) and visiting time. It should be reminded, that the timing law is build on the assumption, that it is desirable for an aircraft to maintain a constant airspeed.

2.4.1. No wind case

Vehicle kinematic model

Two dimensional position will be specified by coordinates x , y and yaw angle ψ . Change of position is described by the kinematic model. Control signal u_ψ is the turning rate and is bounded $|\Delta\psi| < \Delta\psi_{max}$, while V_A is the constant aircraft velocity:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix}$$

$$\Delta\mathbf{x} = \begin{bmatrix} V_{UAV} \cos \psi \\ V_{UAV} \sin \psi \\ u_\psi \end{bmatrix}$$

In the no wind case the visiting time along the trajectory curve changes proportionally to the covered distance.

Algorithm

It has been shown by Dubins [7], that the time optimal path between two oriented positions contains only maximum rate turns and straight motion segments. Specifically, the shortest path

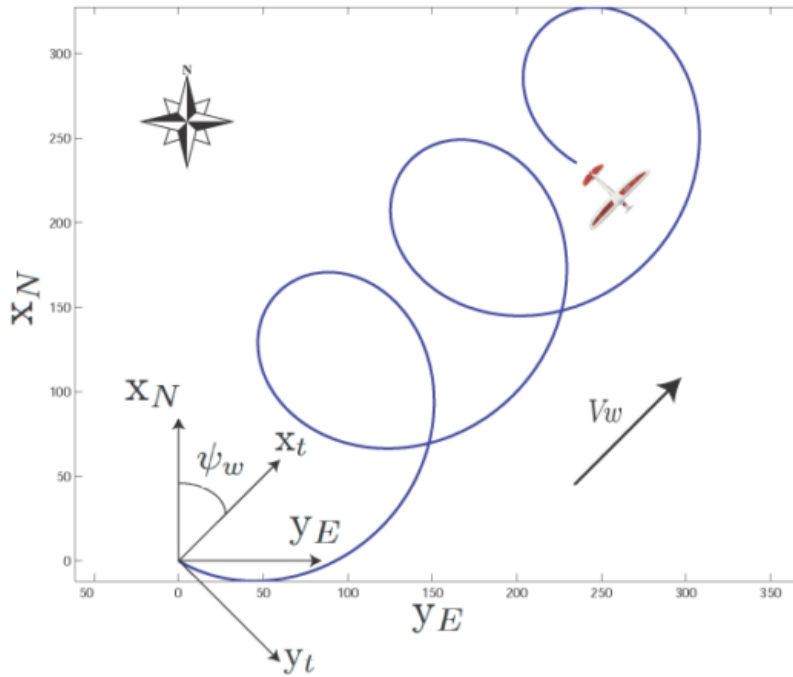


Figure 7: Trochoidal path and trochoidal frame (after [5])

transition from (x_1, y_1, ψ_1) configuration to (x_2, y_2, ψ_2) configuration can be always expressed as a combination of only at most three basic actions applied over an interval of time. These actions are straight flight (S), maximum turn to the left (L) and maximum turn to the right (R), and since two consecutive actions of the same kind can be merged into one, the optimal path structure will be always contained in the set:

- maximum left turn, maximum right turn, maximum left turn $(L_\alpha R_\beta L_\gamma)$,
- maximum right turn, maximum left turn, maximum right turn $(R_\alpha L_\beta R_\gamma)$,
- maximum left turn, straight flight, maximum left turn $(L_\alpha S_d L_\gamma)$,
- maximum left turn, straight flight, maximum right turn $(L_\alpha S_d R_\gamma)$,
- maximum right turn, straight flight, maximum right turn $(R_\alpha S_d R_\gamma)$,
- maximum right turn, straight flight, maximum left turn $(R_\alpha S_d L_\gamma)$,

where $\alpha, \gamma \in [0, 2\pi)$, $\beta \in (\pi, 2\pi)$ and $d \geq 0$.

Exact values of turn angles and straight flight distance can be obtained straightforward through the means of analytical geometry. Each configuration determines unambiguously left and right turn circle. Calculating candidate paths only requires finding tangents to circles for starting and final configurations. To determine unambiguously the motion, the interval duration for each action must be specified. Assuming a constant forward airspeed it is obtained directly from turn angles and straight flight distance.

2.4.2. Constant wind case

Vehicle kinematic model

Two dimensional position will be specified by coordinates x, y and yaw angle ψ . Change of position is described by the kinematic model. Control signal u_ψ is the turning rate and is bounded $|\Delta\psi| < \Delta\psi_{max}$, while V_A is the constant aircraft velocity and V_W is a constant velocity of wind. It is assumed that $|V_A| > |V_W|$.

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix}$$

$$\Delta \mathbf{x} = \begin{bmatrix} V_{UAV} \cos \psi + V_{W_x} \\ V_{UAV} \sin \psi + V_{W_y} \\ u_\psi \end{bmatrix}$$

Algorithm

It can be easily noticed, that circular (constant turn rate) paths in the air-relative frame correspond to trochoidal paths in the ground frame. Using Pontryagin's minimum principle it has been shown [6], identically as in no wind case, that the time-optimal path contains only turns of maximum rate and straight segments. However, contrary to the previous case, the action intervals duration must be determined numerically.

McGee et al. [6, 9] expressed the problem as finding the optimal path from an initial position and orientation with no wind to a final orientation while intercepting a moving virtual target. Techy et al. [5] have presented a different approach by expressing the motion in a trochoidal frame determined by the wind direction. We shall refer the first approach.

We assume existence of a virtual target, whose velocity is equal in magnitude and opposite in direction to the velocity of the wind. Its initial position equals to the desired destination position of aircraft. The final position of the aircraft is that of a virtual target moving in a straight line. Any point along this line can be expressed by a single variable d , which is the distance from the initial target position. Let define function $T_{UAV}(d)$ denoting the time it takes aircraft to fly the optimal no wind air path from its initial position and orientation to a final orientation at point d . Let also define function $T_{Target}(d)$ denoting the time it takes virtual target to fly to point d . To find optimal path we must find point d^* for which $T_{UAV}(d) = T_{Target}(d)$. Under the assumption that initial and final point are sufficiently far apart (for the simplicity of the algorithm presentation) the difference $T_{UAV}(d) - T_{Target}(d)$ is monotonically decreasing in the range $(0, d^*)$.

To calculate concrete d^* value standard iterative root finding methods like bisection or gradient descent algorithms must be used. Deciding on a concrete optimization method is not essential and is left for the future study and experiments.

2.4.3. Multiple waypoints

Considering above sections, we're given the ability to find optimal trajectory between two oriented points. In order to calculate optimal trajectory connecting a sequence of oriented points, an optimization over the orientation angle at these waypoints should be performed, according to the following formula:

$$\min_{\psi} \sum_{i=1}^n J_i(\psi_{i-1}, \psi_i)$$

where ψ is the vector of orientation angles at each waypoint and J_i is the time required to travel from waypoint $i - 1$ to waypoint i on the optimal path with fixed initial and final orientation angles.

This optimization can be achieved using gradient descent method or some other suitable optimization technique. As in previous section, deciding on a concrete optimization method is not essential and is left for the future study and experiments.

3. Surveillance planning

3.1. Applications

In the INDECT project, UAVs are required to perform surveillance tasks. According to Nehme et al. [23] surveillance is the process of monitoring the behavior of people, objects or processes for conformity with expected or desired norms .

Many of surveillance applications are contained in a family of car traffic surveillance, including:

- monitoring traffic intensity on highways and crossings in order to coordinate traffic lights,
- visiting traffic accident scenes,
- providing path hints for emergency vehicles,
- measuring patterns of typical road usage,
- checking parking lot utilization.
- vehicle following.

Road traffic monitoring can be performed without explicit knowledge of its coordinates using visual following, similarly to the vehicle following task, which decomposes into vehicle localization and path planning. As a vehicle changes its direction and speed, the following UAV might be required to stay at particular distance or angle from the target.

Different public services find multiple applications for air surveillance. Border guards describe a border patrol mission, in which a team of aerial robots is assigned to monitor long and narrow land strip in order to detect intruders. Another similar potential usage of air surveillance involves forest fire monitoring and is a subject that has received lately considerable interest [20]. Aircrafts are used in order to determine (using infrared sensors) and report the perimeter of the fire, which expands in an unknown direction.

3.2. Surveillance capabilities

According to Nygard et al. [11] surveillance can be distinguished depending on search patterns and a request origin. Search patterns sorted due to requested areas of responsibility are presented in Table 1.

Search type	Area of responsibility
Pinpoint search	Limited area around reference point
Strip search	Along straight line between two reference points
Line search	Along some characteristic curve (e.g. road or rails)
Area search	Large area (e.g. forest, urban district etc)

Table 1: Areas of responsibility and corresponding search patterns

As far as the request origin is concerned, tasks can be divided into:

1. Initial objective of original mission
2. Objective of operator modified mission
3. Autonomously requested by UAV, as a sub-goal of a more general mission

3.3. Anatomy of surveillance

Surveillance, as the process of monitoring the behavior of targets of interest, is a combination of area coverage and optionally tracking problem in terms of mission planning. UAVs fly over an area in order to detect any unexpected behaviors or objects. Various actions can be taken upon detection, depending on its nature and chosen policy. If UAV detects dynamic object, e.g. suspicious vehicle, it can start following it, until ground services intercept the target. Contrary, if detection concerns static object, e.g. car accident, after notifying the operator it might be preferable for UAV to continue its patrol.

So far we have presented algorithms for safe movement from start to destination position, which focus on the temporal objective, i.e. path should be shortest and therefore could be traveled in smallest amount of time. Unlike point-to-point planning, coverage path planning focuses on the spatial objective and aims to have an UAV pass over all points in the target

environment with a sensor tool of given range. In the spatial surveillance operations, search and rescue missions etc. this type of planning is necessary. By reduction to Traveling Salesman Problem (TSP) it can be shown that finding an optimal-length solution for area coverage path planning is NP-hard.

In section 3.2 surveillance requests have been sorted according to given area of responsibility into pinpoint, strip, line and area search. In following sections algorithms for planning coverage paths for each area of responsibility type will be presented, as well as trajectory generation techniques for tracking problem.

3.4. Pinpoint coverage

During the pinpoint surveillance mission, UAVs are required to visit a set of points and no particular order of visits is required, nor preferred. In the single robot case this is a well known and thoroughly studied Traveling Salesman Problem. Specifically, it is a metric version of TSP, i.e. the distances between points satisfy triangle inequality, for which constant-factor approximation algorithms exist [17]. We presents a popular algorithm with approximation factor 2, since many further presented algorithms build up on it.

If we want to extend the problem and incorporate k robots, we are interested in algorithm for covering the nodes of a graph with no more than k tours, so that the maximum length of a tour is minimized.

In order to apply following graph algorithms to point surveillance problem, the parameter graph G should be defined as a full graph, where vertices represent points that require visiting and edges cost correspond to shortest path distances between these points.

3.4.1. Minimum Spanning Tree Approximation for Traveling Salesman Problem

Algorithm 3 operates by constructing the minimum spanning tree for a graph and returning the vertices in the preorder walk. Since approximation property of this algorithm underlies further presented algorithms, the brief proof is referred.

Let T be the minimum spanning tree for the graph G . The cost of T is less than or equal to the minimal TSP tour, because if you remove one edge from the optimal TSP tour, you get a spanning tree. But T is the minimal spanning tree and the edge costs are non-negative. Consider a "twice around" tour of T called W , where starting with arbitrary vertex r , it traverses each edge of the tree twice once in every direction of the preorder walk. Clearly, the cost of W is twice the cost of T . Note that W visits all vertices, starting and ending in r . Let H be the tour obtained from W by short-cutting W as follows. Repeatedly, for any subsequent vertices u, v, w in W , where vertex v has already been visited, replace it by going directly from u to w (short-cutting v). The cost of H can only decrease due to the triangle inequality. The resulting sequence of vertices H is a traveling salesman tour of the graph, where each vertex is visited in the same order as the preorder walk of the tree.

Algorithm 3 Minimum Spanning Tree Approximation for Traveling Salesman Problem

- 1: **procedure** MST-TSP-APPROXIMATION(G) ▷ A graph G
 - 2: Construct the minimum spanning tree T for G .
 - 3: Pick an arbitrary vertex and call it r .
 - 4: Do the preorder walk (a walk in which each parent node is traversed before its children) of the tree T starting at r . Call L the list of vertices in preorder of visiting during the walk.
 - 5: **return** a tour that visits all vertices of G starting and returning to r in the order prescribed by L .
 - 6: **end procedure**
-

3.4.2. Min-Max Tree Covers of Graphs

Even et al. [13] presents a constant factor approximation algorithms for covering the nodes of a graph with no more than k tours, so that the maximum length of a tour is minimized. It covers the nodes of a graph with k trees, so that the maximum weight of tree is minimized.

Precisely, they construct approximation algorithms with performance ratio 4 for finding *k-tree covers* and *R-Rooted tree covers*.

Definitions

k-tree cover

Let $G = (V, E)$ denote an undirected graph with positive integral edge weights $w : E \rightarrow \mathbb{N}^+$. A tree cover of a graph $G = (V, E)$ is a set T of trees T_i such that $V = \bigcup_{i=1}^k V(T_i)$. The weight of a tree T is defined by $w(T) = \sum_{e \in T} w(e)$. The cost of a tree cover T is $\max_{T_i \in T} w(T_i)$. Note that trees in a tree cover may share nodes and even edges. The goal in the *min-max k-tree cover* problem is to find a minimum cost tree cover consisting of at most k trees.

R-Rooted tree cover

Let $R \subset V$ denote a set of roots. An *R-rooted tree cover* of a graph $G = (V, E)$ is a tree cover T , where each tree $T_i \in T$ has a distinct root in R . As before, trees in an *R-rooted tree cover* may share nodes and edges. In particular, the root of T_i may be in T_j , but the roots of T_j and T_i must be distinct. Given an edge weighted graph G and a set R of roots, the *min-max R-rooted tree cover* problem is to find a minimum cost R -rooted tree cover of G .

Algorithm

The algorithm is given a graph G , set of roots R and maximum acceptable weight of tree cover B and returns tree cover with weight at most $4B$. It begins by removing all edges with edge costs larger than B and contracting roots into a single vertex. After a minimum spanning tree for the resulting graph is found, the single vertex is uncontracted. As a result, the spanning tree is split into $|R|$ trees. Resulting trees are decomposed into sub-trees, which do not share edges. It is done in such a way, that weight of each sub-tree is in the range $[B, 2B)$ and the only possible exception is a leftover sub-tree containing the root of the tree. The weight of this leftover sub-tree is less than B . A maximum matching of all non-leftover sub-trees to the roots is found, under the constraint that a non-leftover sub-tree can only be matched to a root only if the non-leftover sub-tree and leftover tree of the root (or root itself) are at distance at most B . If some non-leftover sub-trees cannot be matched, this is the proof that none rooted tree cover of weight at most B exists. Finally, for each root, return a tree consisting of the root, the leftover sub-tree of the root (if any) of weight at most B , the single non-leftover sub-tree matched to the root (if any) of weight at most $2B$, and a cost-minimal path of weight at most B from the non-leftover sub-tree to the leftover sub-tree (or root). The weight of each tree is at most $4B$, resulting in a rooted tree cover of weight at most $4B$.

Using above algorithm and binary search *R-rooted tree cover* of weight at most 4 times bigger than optimal can be obtained. By doubling edges, each tree can be transformed into a tour (see previous section), the weight of which is at most twice the weight of the tree. Constant factor approximation algorithm for *k-tour cover* with approximation ratio of 8 is thus available, which can be used to plan pinpoint search plans for group of k robots.

Algorithm 4 *R-Rooted-Tree-Cover* (after [13])

- 1: **procedure** ROOTED-TREE-COVER(G, R, B) ▷ A graph G , a set of roots R and maximum weight B
 - 2: Remove all edges with weight greater than B
 - 3: Let $M \leftarrow$ MST of graph obtained from G by contracting roots in R to a single node
 - 4: $\{T_i\}_i \leftarrow$ forest obtained from M by uncontracting roots in R
 - 5: Edge-decompose each tree T_i into trees $\{S_j^i\}_j + L_i$ such that $w(S_j^i) \in [B, 2B)$ for every j and $w(L_i) < B$
 - 6: Try to match the trees $\{S_j^i\}_{i,j}$ to roots, subject to the constraint that a tree S_j^i may be matched only to roots of distance at most B from it
 - 7: **if** every tree is matched **then**
 - 8: **return** success: set of trees where each tree consists of S_j^i , its matched root r , and the leftover tree L (if any) that contains the root r
 - 9: **else**
 - 10: **return** fail “ B is too low”
 - 11: **end if**
 - 12: **end procedure**
-

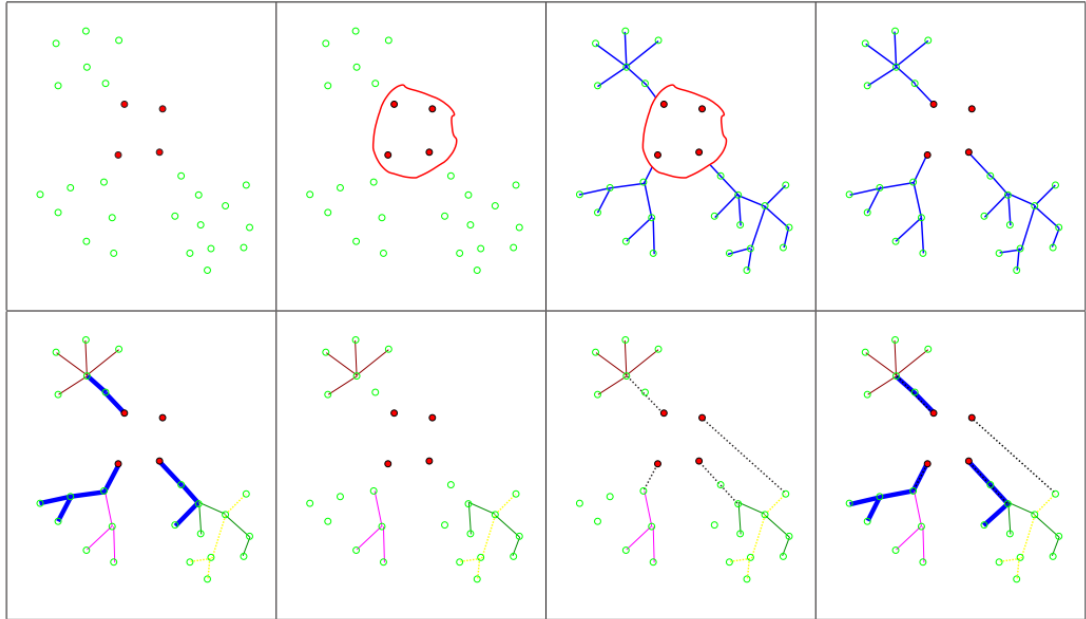


Figure 8: Example of R-Rooted-Tree-Cover execution (read from left to right). (1) The input; roots are denoted by filled nodes. (2) Contraction of the roots. (3) MST of the contracted graph. (4) Uncontracting the graph. (5) Edge decomposition of each tree; leftover sub-trees $\{L_i\}_i$ are denoted by dark thick edges. (6) The non-leftover sub-trees $\{S_j^i\}_{i,j}$. (7) matching of the non-leftover sub-trees to roots. (8) The final trees consisting of a leftover tree, a matcher tree, and a matching edge. After [13].

3.5. Line coverage

From combinatorial optimization point of view line search and strip search yield the same problem: we are given a set of linear structures with negligible width (e.g. roads, yields) that should be monitored. The problem can be presented in a graph based formulation, where endpoints of strips under observation are vertices and edges denote shortest paths between the endpoints, as well as strips that require monitoring. The problem of line coverage would be concerned with finding a set of tours within this graph, such that each edge representing strip under observation should be traversed in at least one tour. This has been defined as *k-Rural Postman Problem* and since we are interested in minimizing the maximum length of tours also as *Min k-Rural Postman Problem*.

However at this point little prior work has been found on the problem and ongoing internal research has not yet produced satisfactory results. However, line coverage can be solved using more general area coverage algorithms (albeit with optimality penalty) that are presented in next sections.

3.6. Area coverage

Most offline area coverage planners use a cellular decomposition of the free space to perform coverage, where the target region is divided into cells, whose individual coverage is straightforward. Cellular decompositions can be divided into:

- exact decompositions, where union of non-intersecting regions fills the target environment,
- approximate decompositions, where union of cells of the same size and shape approximate the free space.

For exact decompositions we have a guarantee that path completely covers the space, which might be necessary in certain applications (e.g. mine detection). Contrary, approximate approaches do not guarantee covering the whole target area, but they can provide time optimality guarantees for path that covers this limited area. Given concrete application, the operator

should decide whether longer-lasting, but complete decomposition is necessary or whether approximate (hence faster) solution is acceptable. We describe several algorithms, whose performance should be evaluated before concrete method will be chosen for deployment. They include exact boustrophedon decomposition and two approximate approaches using spanning trees to generate path. We start with single-robot versions of algorithms and extend them later on to multi-robot version.

3.6.1. Exact Boustrophedon decomposition

Boustrophedon coverage

Boustrophedon (literally “the way of the ox”) coverage or simply zig-zag back-and-forth motions is an example of a well known procedure that guarantees complete coverage of trapezoidal regions. Together with classical exact trapezoidal decomposition it provides a basic complete coverage approach for polygonal regions with polygonal obstacles.

Boustrophedon decomposition

Covering areas of arbitrary polygonal shape using basic algorithm for covering trapezoidal areas requires a method for decomposing any simple polygon (possibly with holes) into set of trapezoids. This can be achieved by trapezoidal decomposition [2] in worst-case $O(n \log n)$ time, where n is the number of obstacle vertices. It is based on the previously mentioned plane sweep principle. Lets imagine sweeping a vertical line l rightwards, starting from a position of the leftmost obstacle vertex. Whenever the sweep line touches an obstacle vertex, rays are extended upwards and downwards from the intersection point, until obstacle edge is hit. These rays divide the free space. Intersecting obstacle vertex triggers one of three types of events: *In*, *Out* or *Middle* (Fig. 11). *In* event closes one cell and opens two cells, *Out* event closes two cells and opens one, and finally *Middle* event closes and opens one cell. The newly created polygons on the left side of the sweep line are always trapezoids or triangle, since all but two edges of the polygonal are vertical. After all obstacle vertices have been visited the entire free space is divided in triangles and trapezoids.

Boustrophedon decomposition [22] is a variation on a trapezoidal decomposition, which allows elimination of superfluous motions. If we maintain the direction of the sweep line for the covering procedure, then it is enough for *Middle* event to just update the cell. Even though the resulting polygons are not even necessarily convex, their boustrophedon coverage based on vertical sweeps produces full coverage.

The classical approach decomposes polygons using a vertical sweep line. However, this is a completely arbitrary assumption and the algorithm can choose sweep line of any slope for decomposition, providing that later the slope is used consistently during the whole process. Huang [10] argues, that minimizing the number of turns during the coverage is a good criterion for suitable decomposition, as turning often requires more time than straight motion and can be generally more troublesome for execution layer. As number of turns is proportional to the altitude of the polygon measured along the sweep direction, a decomposition that minimizes the sum of polygon altitudes is optimal for coverage problem. He defines the diameter function $d(\theta)$ that describes the altitude of the polygon as a function of the sweep line direction. The sum of altitudes to minimize is then given by formula:

$$S(\theta) = d_B(\theta) + \sum_i d_{H_i}(\theta)$$

where $d_B(\theta)$ is the diameter function of the area boundary and $d_{H_i}(\theta)$ is the diameter function of hole i .

He has shown, that the sweep line which minimizes this sum must have slope equal to one of the polygons’ sides.

Application of the decomposition for area coverage

After the decomposition the order in which cells will be covered must be decided. Time spent during boustrophedon sweeps is determined by the covered area and does not depend on the order of cell visiting. Therefore we only need to minimize time spent on traveling between cells. We define a distance graph, such that each cell corresponds to a vertex in the graph and edge cost between adjacent cells depends on the distance between cells centroids. Since finding the

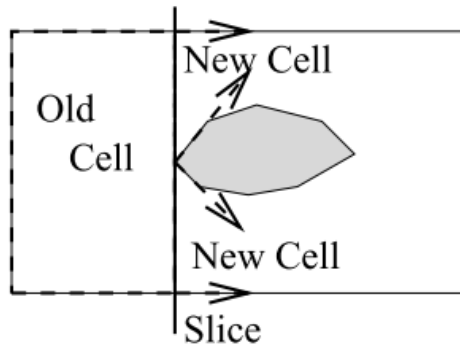


Figure 9: In Event (after [22])

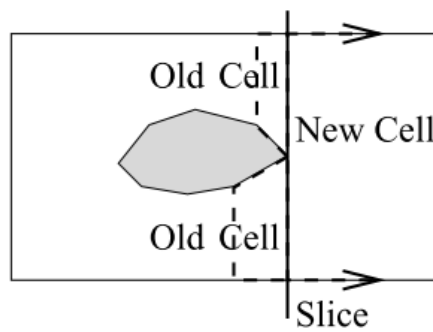


Figure 10: Out Event (after [22])

optimal path for visiting each node in the graph is a Traveling Salesman Problem, which is NP-complete, we must fallback to one of suboptimal solutions. In its definition, the TSP does not allow vertices to be visited twice, but in this application we do not need this constraint. Therefore, we replace the original distance graph with a complete graph in which the distance between cells is equal to the shortest path distance between these cells in the original graph. Since triangle-inequality applies, we can use minimal-spanning-tree TSP algorithm, that returns a tour whose cost is no more than twice the cost of an optimal tour.

3.6.2. Approximate Spanning Tree Covering (STC)

Gabriely and Rimon [3] have presented an algorithm for optimal coverage of a square tool based approximation of the covered area in $O(n)$ time, where n is number of cells making up

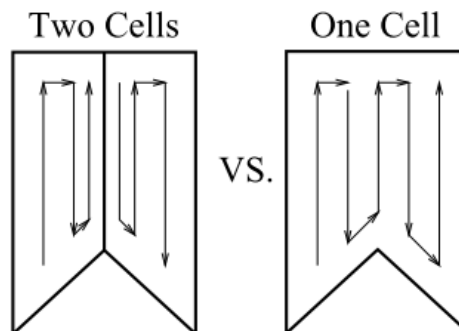


Figure 11: Fewer cells gives better coverage path (after [22])

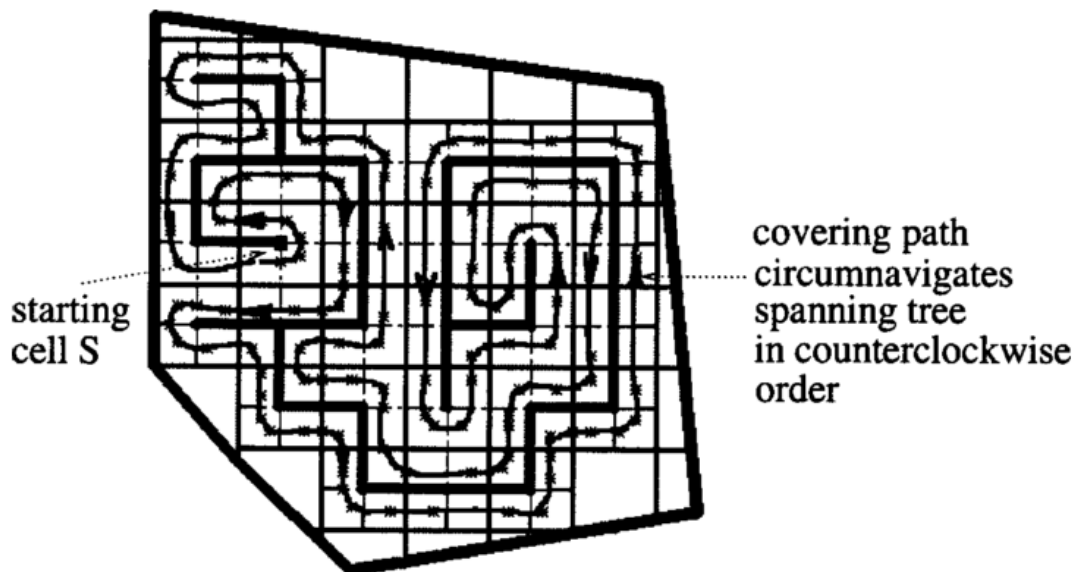


Figure 12: An example of the spanning-tree covering algorithm execution (after [3])

the approximate area.

Algorithm 5 takes a polygonal coverage area with holes and divides it into cells of twice the width of the coverage tool. All of the cells, which are even partially intersecting with holes are discarded. The cells induce a graph, whose vertices correspond to center points of each cell and edge exist for adjacent cells. The algorithm constructs a spanning tree for such graph and uses it for generating path, that covers every point exactly once. It should be noticed, that a closed path is generated, which allows repeating the coverage using the same path over and over. In fact, the path is a cycle in the graph induced by subdividing each $2D$ sized cell of the approximate area into four D -sized cells.

Each graph induced by the area definition can have many spanning trees. Simplest is one generated by depth-first search. However, optimization criteria might be introduced to find preferable tree by attaching edge weights and using minimum spanning tree algorithm (e.g. Kruskal's or Prim's). This way we can for example generate path along a particular coordinate direction. When minimal spanning trees are used, the generated path is a Hamiltonian cycle in the aforementioned induced graph.

Algorithm 5 Off-line Spanning Tree Covering (after Gabriely et Rimon [3])

- 1: **procedure** SPANNINGTREECOVERING(A, H, D, p) ▷ An area A , set H of holes, initial position p and coverage tool width D
 - 2: Subdivide the work-area A into cells of size $2D$.
 - 3: Discard cells partially covered by holes H .
 - 4: For remaining cells define a graph structure G , where nodes are the center points of each cell and edges are the line segments connecting centers of adjacent cells.
 - 5: Starting from the cell containing initial position p , construct a spanning tree for graph G using any spanning-tree construction algorithm.
 - 6: Subdivide every $2D$ -size cell into four identical sub-cells of size D .
 - 7: Starting from cell containing initial position p :
 - 8: **repeat**
 - 9: Move between neighboring sub-cells along a path which circumnavigates the spanning tree along a counterclockwise direction
 - 10: **until** The starting sub-cell is encountered again
 - 11: **end procedure**
-

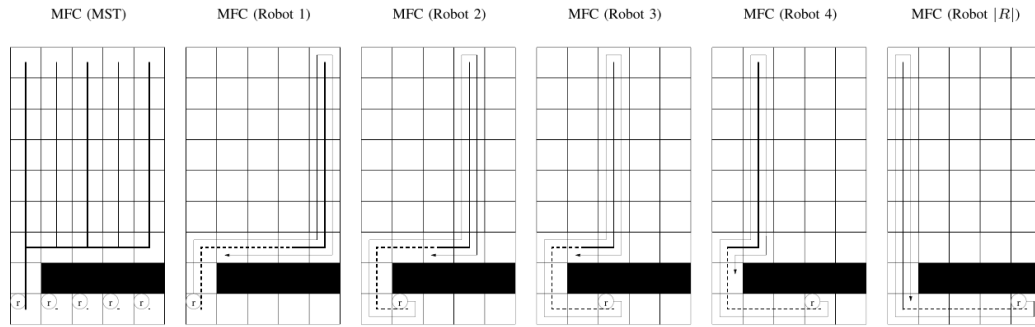


Figure 13: Example of Multi-Robot Forest paths

3.6.3. Multi-Robot Boustrophedon Exact Coverage

Adopting the Boustrophedon decomposition for multiple UAVs with the use of rooted tree cover problem is straightforward. Similarly to single robot case, we define a graph in which each decomposition cell corresponds to vertex in a distance graph and edges cost between adjacent cells equals to shortest path distance between cells centroids. Initial positions of UAVs are used as root vertices. Generated R -rooted trees are used as approximations for Hamiltonian tours, that determine the order of covering the cells.

3.6.4. Multi-Robot Forest Coverage

Multi-Robot Forest Coverage (MFC) [14] is a polynomial time multi-robot coverage heuristics. It extends approximate spanning tree coverage algorithm for a single robot from the previous section and is based on the earlier mentioned algorithm by Even et al. for finding tree covers with given roots. The work-area is decomposed into square sized cells and it is assumed that robots can occupy this cell simultaneously (this assumption can be met by placing UAVs at different altitudes).

Set of k trees defined by R -rooted tree cover determines the paths for each of k UAVs from an operating group (Algorithm 6). Similarly to the single robot spanning tree coverage, the UAVs circumnavigate their corresponding trees. This results in covering the whole area in a time interval no longer than eight times the optimal one, since trees induce a k -tour cover with approximation ratio 8.

Algorithm 6 Multi-robot Forest Coverage

- 1: **procedure** MULTIROBOTFORESTCOVER(A, H, D) \triangleright An area A , set H of holes and coverage tool width D
 - 2: Subdivide the work-area into cells of size $2D$.
 - 3: Discard cells partially covered by holes.
 - 4: For remaining cells define a graph structure, where nodes are the center points of each cell and edges are the line segment connecting centers of adjacent cells.
 - 5: Construct R -Rooted tree cover for G using initial robot positions as roots.
 - 6: Subdivide every $2D$ -size cell into four identical sub-cells of size D .
 - 7: **for** Each robot and corresponding tree from tree cover **do**
 - 8: Starting from cell containing initial position of robot.
 - 9: **repeat**
 - 10: Move between neighboring sub-cells along a path which circumnavigates the tree along a counterclockwise direction.
 - 11: **until** The starting sub-cell is encountered again.
 - 12: **end for**
 - 13: **end procedure**
-

3.6.5. Backtracking Multi-Robot Spanning-Tree Coverage

Hazon et Kaminka [12] present an algorithm that gives optimal allocation of sections among multiple robots for a given spanning-tree coverage path. They assume that:

- all robots move around the same closed path,
- the robots' paths can not cross,
- every robot backtracks only on its own steps,
- each robot is allowed to visit a cell twice, but no more.

The achieved optimality means that for a given closed path and initial positions of the robots, it produces subdivision of this path into subsections that requires least time to cover. However, it does not give any guarantee concerning unconstrained coverage optimality. The additional strength of the algorithm lies in universality and robustness of the produced plans. It guarantees completion of whole task using the original plan as long as at least one robot remains operational. It also takes into account that UAVs can have different ranges of potential speed and amount of fuel available.

The basic premise for this algorithm is that reasonable behavior of robot involves moving in only one direction (towards previous or next neighbor) or changing this direction exactly once (backtracking). Any additional change of direction is suboptimal. The robot motion is fully described by its initial direction, length of section to take in this direction and length of section to cover after backtracking. It must be also noted, that deciding on only one robot initial configuration determines best configuration for other robots. Therefore their approach performs an exhaustive search for the optimal solution in the polynomial space.

Robustness is guaranteed with a simple mechanism: after a robot finishes covering its sections, it waits for the neighboring robot to finish his coverage. If neighboring robot is not functional, it takes over its role. This backup guarantee is inherited, i.e. after completing the work of a nonoperational neighbor, it waits for it's neighbor to finish his coverage, etc. What follows, even if all robots but one fail, the remaining one will take over their assignments in turn.

Algorithm is guaranteed to return a solution that is optimal, even taking into account heterogeneous speed limits. Instead of calculating coverage time based on the length of the section, it should be calculated based on the length given the maximum speed of the robot in question. On the other hand, incorporating the available fuel to filter feasible solutions considered in exhaustive search is guaranteed to produce a solution that is feasible given the robots' fuel or battery constraints. This solution, however, still minimizes coverage time, rather than fuel consumption.

Algorithm 7 Optimal Backtracking MSTC (after Hazon et Kaminka [12])

```
1: procedure BACKTRACKINGMSTC( $P, R$ )           ▷ A closed path  $P$  and set of robots  $R$ 
2:   for every robot  $r \in R$  do
3:     for initial direction  $D \in \{ forward, backward \}$  do
4:       Find robot motion configuration  $\langle r, D, L_1, L_2 \rangle$  with minimum coverage time
       equal to  $2L_1 + L_2$ , where  $L_1, L_2$  are length of initial and backtracking subsections, that
       allows other robots to complete covering the rest of the area within this time.
5:       if the robot motion configuration coverage time is best found so far then
6:         end if
7:       Remember the configuration.
8:     end for
9:   end for
10:  return the best robots motion configuration found.
11: end procedure
```

3.6.6. Continuous Multi-Robot Approximate Spanning Tree Coverage

In continuous surveillance requests we may be interested not in minimizing the time of coverage, but in minimizing the maximum time between subsequent cell visits (of any robot). If multi-robot coverage is performed using approximate spanning-tree approach these can be achieved by dispersing the initial robots position uniformly along the Hamiltonian cycle. Afterward, each robot follows repeatedly the whole path.

3.7. Tracking

Unlike other motion planning tasks, trajectory for target following can not be computed in advance. This kind of motion planning does not follow two phase decoupling into path planning and trajectory generation, since path is implicitly determined by the path of the target. In following sections we describe how single aircraft can follow single moving target given its absolute coordinates. Further approaches for the case of cooperative following of many targets is described in deliverable “Proposed algorithms and methods for cooperation for groups of UAVs”.

3.7.1. Hopf Bifurcation Target Following

Unlike other, offline defined, motion planning tasks, trajectory for target following must be calculated in real time, therefore no computationally complex operations are feasible. Motivated by this assumption, Quigley et al [8] proposed an extremely simple to compute method based on Hopf bifurcation, that produce spiral trajectories converging to a cycle attractor of a given radius. Based on whether current UAV position is inside or outside of the limit cycle, the trajectory spirals outward from the center point and converges to the cycle or aims for the center point and later spirals in to the limit cycle as they approach it. The path is calculated with following formulas:

$$x_c = \hat{x} - x_t$$

$$y_c = \hat{y} - y_t$$

$$\frac{dx}{dt} = y_c + \frac{x_c}{\alpha r^2} (r^2 - x_c^2 - y_c^2)$$

$$\frac{dy}{dt} = -x_c + \frac{y_c}{\alpha r^2} (r^2 - x_c^2 - y_c^2)$$

$$\psi = \arctan\left(\frac{dy}{dx}\right)$$

where:

- (\hat{x}, \hat{y}) is the estimated position of the aircraft,
- (x_t, y_t) is the estimated position of the target,
- r is the desired orbital radius,
- and ψ is the desired aircraft heading.

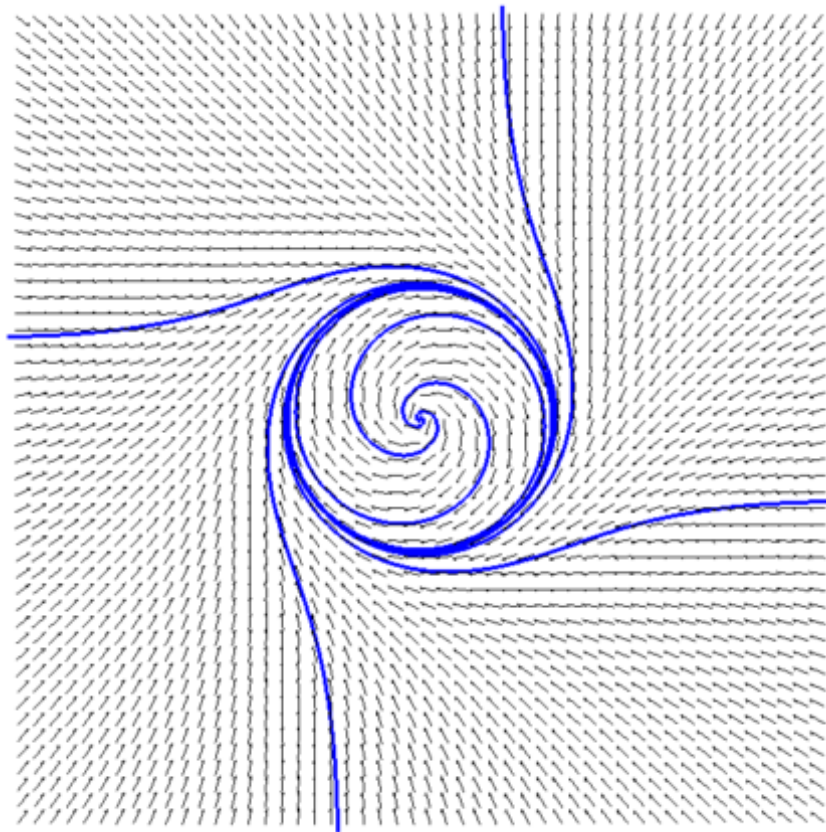


Figure 14: Trajectories through the slope field of a Hopf bifurcation always converge to the limit cycle (after [8])

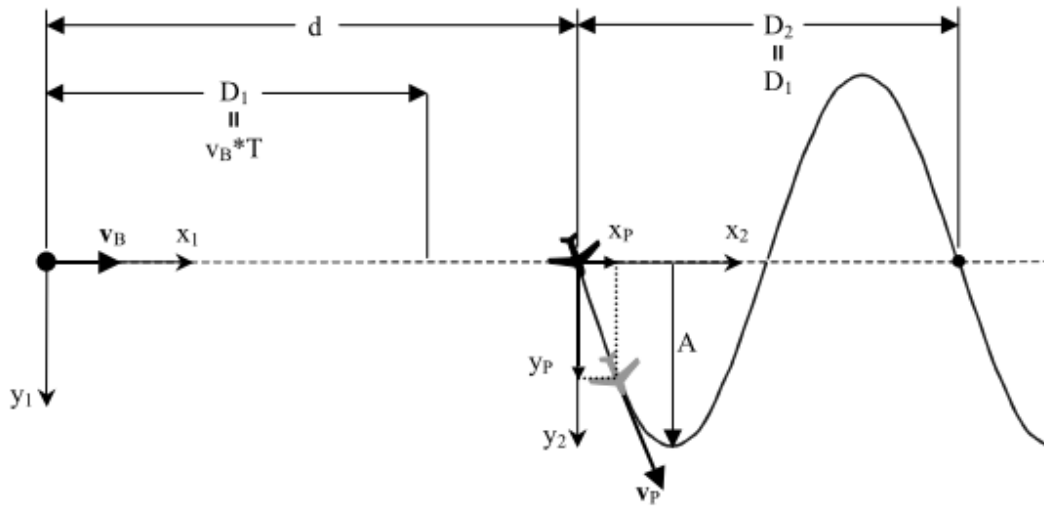


Figure 15: Top view of path-planner algorithms in sinusoidal mode (after [21])

3.7.2. Mode-Switching Target Following

Lee et al. [21] argues, that target following can be successfully performed using two modes of motion: *loitering* and *sinusoidal*. As the followed target might be moving within a range of velocities, the strategy for following must be adoptable to these various speeds. Therefore the tracking strategy changes depending on the ratio of the ground vehicle velocity V_G and following UAV velocity V_{UAV} . If the ratio $\frac{V_{UAV}}{V_G}$ exceeds certain threshold than UAV switches into loitering mode. When it is below the threshold, the UAV moves according to the sinusoidal mode and the ratio value is used to determine the amplitude of the sinusoid.

In the sinusoidal mode, the sinusoid amplitude is chosen so that in one arbitrarily chosen time period T , the distance traveled by the ground vehicle D_1 will be equal to a displacement D_2 of the UAV in the direction of target motion. Using the coordinate systems from Fig. 15, the sinusoid equation can be expressed in the the x_2y_2 coordinate system fixed at the beginning of the sinusoid:

$$y_p = A \sin\left(\frac{2\pi x_p}{D_1}\right)$$

where x_p, y_p are desired positions of the UAV.

The time derivatives used to calculate the desired UAV path become:

$$\dot{x}_p = \frac{v_p}{\sqrt{1 + A'^2 \cos^2\left(\frac{2\pi x_p}{D_2}\right)}}$$

$$\dot{y}_p = A \cos\left(\frac{2\pi x_p}{D_2}\right) \cdot \dot{x}_p$$

where $A' = \frac{2\pi A}{D_2}$.

The amplitude A is determined using the following formula:

$$\frac{V_{UAV}}{V_G} = \frac{1}{D_1} \int_0^{D_1} \sqrt{1 + A'^2 \cos^2\left(\frac{2\pi x_p}{D_1}\right)} dx_p$$

Conclusions

In this document we have described a proposed high level architecture for mission planning and concrete planning algorithms for motion and surveillance planning.

The robustness of the described architecture has been experimentally verified by recent years research. The presented algorithms are varied and have complementary strengths. Their behavior must be extensively tested in scenarios closely resembling the expected real life situations in which the Integrated Air Surveillance System, developed during INDECT project, will be used. This testing will be facilitated by the simulation environment proposed in section 1.5.

References

- [1] D. T. Lee. Proximity and reachability in the plane. Report R-831, Dept. Elect. Engrg., Univ. Illinois, Urbana, IL, 1978
- [2] Steven La Valle. Planning algorithms. Cambridge University Press, 2006
- [3] Gabriely Y, Rimon E. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*. 2001
- [4] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008
- [5] Tschy L, Woolsey Ca, Schmale DG. Path planning for efficient UAV coordination in aerobiological sampling missions. 2008 47th IEEE Conference on Decision and Control. 2008
- [6] McGee T, Spry S, Hedrick J. Optimal path planning in a constant wind with a bounded turning rate. In: 2006 AIAA Conference on Guidance, Navigation, and Control. 2006
- [7] Dubins L. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*. 1957
- [8] Quigley M, Goodrich M, Griffiths S, Eldredge A, Beard R. Target acquisition, localization, and surveillance using a fixed-wing mini-UAV and gimbaled camera. *IEEE International Conference on Robotics and Automation*. Vol 3. 2005
- [9] McGee T, Hedrick J. Path Planning and Control for Multiple Point Surveillance by an Unmanned Aircraft in Wind. 2006 American Control Conference. 2006
- [10] Huang WH, York N. Optimal Line-sweep-based Decompositions for Coverage Algorithms. *International Journal*. 2001
- [11] J. Nygard, P. Skoglar, J.Karlholm, R. Bjorstrom. Towards concurrent sensor and path planning. 2005
- [12] Hazon N, Kaminka G. On redundancy, efficiency, and robustness in coverage for multiple robots. *Robotics and Autonomous Systems*. 2008
- [13] Even G. Min-max tree covers of graphs. *Operations Research Letters*. 2004
- [14] Zheng X, Jain S, Koenig S, Kempe D. Multi-robot forest coverage. In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005.(IROS 2005).; 2005
- [15] Hershberger J, Suri S. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*. 1999
- [16] Lee J, Huang R, Vaughn A, et al. Strategies of path-planning for a UAV to track a ground vehicle. *AINS Conference*. 2003
- [17] Christofides N. Worst-case analysis of a new heuristic for the traveling salesman problem. Report 388. Carnegie Mellon University. 1976.
- [18] D. Caveney and R. Sengupta. Architecture and application abstractions for multi-agent collaboration projects. *IEEE CONFERENCE ON DECISION AND CONTROL*. 2005
- [19] R. Beard, D. Kingston, M. Quigley, D. Snyder, R. Christiansen, W. Johnson, T. McLain, and M. Goodrich. Autonomous vehicle technologies for small fixed wing UAVs. *AIAA Journal of Aerospace Computing, Information, and Communication*, vol. 2. 2005
- [20] J. Langelaan and S. Rock. Towards autonomous uav flight in forests. *Proc. of AIAA Guidance, Navigation and Control Conference*, 2005
- [21] 1. Lee J, Huang R, Vaughn A, et al. Strategies of path-planning for a UAV to track a ground vehicle. In: *AINS Conference*.; 2003

- [22] Choset H, Pignon P. Coverage Path Planning: The Boustrophedon Cellular Decomposition. Mechanical Engineering. 1994.
- [23] Crandall J, Nehme C, Cummings M. A UAV Mission Hierarchy. 2006.

Version	Date	Updates and revision history	Editor
0.1	09/03/2010	Document template	Grzegorz Sobański, PUT
0.1.1	17/03/2010	Table of contents draft	Krzysztof Witkowski, PUT
0.9	9/06/2010	Final draft	Krzysztof Witkowski, PUT
1.0	17/06/2010	Stable version for reviewers	Paweł Lubarski, PUT