Project no. W911NF-08-1-0521 – final report as of 30th September 2010

Intelligent Software Agent Control of Combined UAV Operations for Tactical Missions
Final report

Michal Pěchouček[1], **Peter Novák, Lukáš Chrpa, Jiří Vokřínek and Antonín Komenda**
Agent Technology Center, Department of Cybernetics
FEE Czech Technical University in Prague
http://agents.felk.cvut.cz

This document provides a technical report on the work performed in the context of the Project W911NF-08-1-0521. The project researches and develops agent-based techniques for the control of cooperative autonomous aircrafts performing information collection operations in support of tactical missions.

[1] **Principal investigator**

Contents

Executive summary

This document provides a final technical report on the work performed in the context of the research project W911NF-08-1-0521 *Intelligent Software Agent Control of Combined UAV Operations for Tactical Missions* (TACTICAL-AGENTFLY). The project aims at investigation of problems and research challenges in the context of agent-based control of *teams of cooperative autonomous aircrafts* in information-collection missions. The main research foci of the project include i) modelling and integration of Vertical Take-off and Landing assets into the AGENTFLY framework, ii) investigation of problems of planning in dynamic and resource-constrained environments, and iii) prototyping and study of the issues in integrated coordination for mixed information-collection activities in tactical mission scenarios.

The main achievements of the project and innovations include

a) development and implementation of the spatio-temporal trajectory planning algorithms for Vertical Take-off and Landing assets and their integration into the AGENTFLY technological infrastructure,
b) development, implementation and evaluation of multi-agent-task-allocation-based solver for distributed vehicle routing problem and its utilization for target tracking and area exploration tasks, and
c) development and implementation of configurable technology for specification and simulated execution of missions and mission-centric information collection tasks.

In this report we provide a detailed account of the work performed in the context of the project, provide an extensive discussion of the technologies we proposed, designed and developed, or adapted for the purposes of the project. Where appropriate, we also provide a thorough evaluation of the proposed techniques in an example of a simulated ISTAR mission.

Chapter 1
Overview

This chapter provides an overview of the the final report for the work performed in the context of the project W911NF-08-1-0521. After an introduction into the broader context of the project and related research activities of ATG, we summarize the main objectives of the project and subsequently introduce and discuss the main achievements of the project team together with the main results of the project. We conclude this overview with a brief recapitulation of the project execution from a project-managerial view and finally outline the structure of the remainder of this report.

1.1 Motivation and context

One of the major research tracks of the *Agent Technology Center* (ATG) in the last years is development of novel techniques and approaches for distributed multi-agent control of Unmanned Aerial Vehicles of various types. In the past, the achievements of the center in this context included development of the AgentFly technology suite, a large-scale simulation framework for development and evaluation of automatic mechanisms for next-generation air traffic control. The AgentFly technology provides an infrastructure for a number of other research projects within ATG, including the here reported Tactical AgentFly technology.

On the substrate of the AgentFly suite of technologies, our group was in the past involved in several projects aiming at investigation of information collection techniques for team of coordinated UAVs. Besides here reported W911NF-08-1-0521 project, these include:

- Tactical AgentFly phase I (W911NF-08-1-0521_1312AM0), aiming at investigation of coordinated area surveillance and basic target tracking techniques supporting ground ISTAR missions,

- U-Scout project (BAA 8020902.A), focusing on multi-agent planning and task-allocation in missions involving a number of Unmanned Ground Vehicles and realistic simulation of physical dynamics thereof,
- Tactical AgentScout (W15P7T-05-R-P209), an on-going project aiming integration of aerial information collection ISTAR-type missions with different types of ground assets. This project is seen by the ATG, as well as the funding partner of the here reported Tactical AgentFly project as a related continuation project. Later in this report we discuss the relationships between the two, as well as the interactions of their corresponding workpackages and consequences on the here reported project.

We proposed the here reported project as a continuation of our long-term line of research towards deep understanding of information-collection tasks performed by a team of coordinated unmanned vehicles.

Finally, ATG currently performs several on-going research activities aiming at transfer of the research results resulting from the above mentioned projects, including the one reported upon here, and their related technologies from mid- and large-scale simulations in the AgentFly framework to real hardware. In particular, there is an on-going project in parts supported by the Czech ministry of defence aiming at integration real UAV (*Procerus*) into the AgentFly-based simulation and thus produce a mixed-simulation involving both simulated as well as real aircrafts. The objective is to develop a test-bed for multi-UAV air trajectory deconfliction by negotiation. Another, planned project will aim at transfer of the spatio-temporal planner for VTOLs (partly reported upon later in this document) to real quad-rotor UAV provided by *UAS Technologies*, Sweden, supported in part by the *SAAB* funded *LinkLab* - Center for Future Aviation Systems.

Fig. 1.1 The depiction of the *Procerus* UAV kit and the *LinkQuad* quad-rotor VTOL employed in the on-going research projects of the *Agent Technology Center*.

1.2 Project description

The research project W911NF-08-1-0521 *Intelligent Software Agent Control of Combined UAV Operations for Tactical Missions* (Tactical-AgentFly) aims at exploring the problems of agent-based control of teams of cooperative autonomous aircrafts, such as fixed wing UAVs and other rotorcraft, in information-collection operations and missions. In particular the research addresses the following main topics:

1. development of AgentFly model of Vertical Take-off and Landing type of vehicle,
2. development of multi-agent coordination algorithms for task allocation in resource-constrained and dynamic environments, with the specific focus on information collection tasks of ISTAR-type missions, and
3. investigation of challenges of integrated coordination for mixed information collection activities.

In the following, we provide a brief summary of the particular project objectives listed per workpackage.

WP1: Modelling Vertical Take-off and Landing Assets

The VTOL capability of rotor-type UAVs provides more flexibility and efficiency during information collection in urban unknown or partially known environments. This capability is particularly well suited for use where the ground surveillance site is remote, hazardous, and/or inaccessible by other means, while it is very complex to monitor from a high altitude. Rotor- type of UAVs are also better suited for planning coordinated flight activity of several UAVs of such a type in urban areas in comparison to coordinated fight of fixed-wing UAVs.

Firstly, the Tactical AgentFly project aims at development and subsequent integration of VTOL type of assets into the existing AgentFly platform. The second major objective of the project is to propose and develop suitable algorithms for trajectory planning of VTOL assets and integrate them with the VTOL model in AgentFly platform.

WP2: Planning in Dynamic and Resource Constrained Environment

One of the major challenges in autonomous control of teams of multiple UAVs performing information collection tasks is the adaptation to a dynamic environment on the ground. In particular, such a dynamics includes movements of the ground entities, be it adversarial targets, third-party entities, such as civilians, and most importantly interactions among the tasks allocated and

performed by other members of the team of UAVs. The effects of this environment dynamics are heavily amplified in scenarios involving constrained resources, in particular involving e.g., tracking of relatively high number of targets by a low number of aerial assets.

The main objective of this workpackage was development of techniques aimed at improving the performance of information collection by a team of UAVs. More concretely, this workpackage focuses on investigation of, what we coined $M \times N$ tracking scenarios, i.e., performing tracking tasks of M targets by a team of N UAVs, where $M > N$, while the speed of the UAVs is obviously greater than the speed of moving ground targets. One of the aims was also to investigate the methods of maximizing persistence of tracking of the objects and identifying how many assets are needed in different types of tracking scenarios.

WP3: Integrated coordination for mixed information collection activities

This workpackage aimed at studying mutual interactions between simultaneously performed heterogeneous information collection tasks. The workpackage breaks down to the following three subtopics.

Support for additional classes of information collection tasks. We proposed to extend the range of considered information collection task types with additional classes, in particular exploration and search.

Integrated coordination for mixed information collection. The aim of this subtopic was to study the theoretical interactions between mixed/heterogeneous information collection tasks performed simultaneously at the same time. In particular, the idea was to investigate the problems arising from automated techniques facilitating transparent switching between different information collection tasks, such as switch from surveillance to target tracking and back, alternatively a seamless hand-over of a tracking task from one UAV to another, depending on the local context and aiming and improving the overall efficiency of the team-performed set of information collection tasks in an operations theatre. In the project, we wanted to explore the possibilities of taking the interdependencies between various tasks into account explicitly and implementation of integrated coordination mechanisms which can produce (near)-optimum allocations for the overall mixed information collection task rather than for its individual constituent components only (as done by current methods).

Mission-centric/oriented information collection. The last, rather ambitious and speculative, topic of the workpackage WP3 was the objective to further extend the integrated coordination for mixed information collection so that the team of UAVs respects the plan of the ground mission it provides support to, takes a mission plan as an input and uses it for planning and coordinating information collection so that the information needs

of the mission are optimally covered. The activity aimed at research and possibly prototyping efforts towards considering temporal development and dependencies between the individual information collection tasks as the mission progresses. Explicit consideration and exploitation of the mission plan should enable more efficient allocation of the surveillance resources.

WP4: Testing, demonstration and deliverables

To enable thorough testing and evaluation of the techniques and technological prototypes developed in the context of this project, we proposed to perform extensive set of experiments, implement and finally deliver the corresponding technology demonstrators. Besides the incremental progress and final reports of the project, the source code of all the implemented software components and demonstration videos should become separate deliverables of the project.

1.3 Achievements & innovative claims

In the following, we highlight the main results of the individual workpackages.

WP1: Modelling and integration of Vertical Take-off and Landing Assets

Theoretical contributions:

1. we proposed spatio-temporal trajectory planning algorithm specifically tailored for use by VTOL type assets. The planner takes as an input a set of waypoints the aircraft should fly through, together with vectors of the flight in these waypoints and produces a 3D trajectory respecting the physical constraints of the aircraft e.g., w.r.t. its speed and turning radius in different speeds. The planning algorithm facilitates a significantly more efficient flight than similar trajectory planning algorithms for fixed-wing aircrafts.

Technological prototypes:

1. we implemented the planning algorithm for VTOL assets,
2. the planning algorithm was further extended with the model of VTOL physical dynamics what led to further improvement of the planner, and finally
3. we integrated the above mentioned modules into the AgentFly platform what allows us to implement simulations involving heterogeneous types of UAVs in integrated mission scenarios.

Evaluation:

1. we evaluated the performance of the implemented planner in various scenarios, in particular aiming at evaluation of the performance in scenarios involving several no-flight zones.

WP2: Planning in Dynamic and Resource Constrained Environments

Theoretical contributions:

1. we devised and designed a general-purpose *multi-agent problem solving architecture* specifically aimed at solving multi-agent task allocation and distributed vehicle routing problems by iterated negotiation employing Contract-Net protocol,
2. we formulated various information collection tasks, such as surveillance, tracking and exploration as task allocation problem instances and devised a method for solving of these using the above described general purpose multi-agent problem solver, and finally
3. we formulate the problem of tracking multiple mobile targets as a distributed vehicle routing problem (DVRP) within the above mentioned general-purpose multi-agent solver (involves a method for sub-optimal solving instances of the *travelling salesman problem*).

Technological prototypes:

1. we implemented the proposed multi-agent solver (MAS solver) and integrated it into the AgentFly platform,
2. we implemented the above formulated information collection tasks as problem instances for the MAS solver, and finally
3. we implemented and evaluated performance of various techniques for multi-UAV tracking of multiple mobile targets and compared them with the techniques based on the DVRP method described above. In particular, for comparison, we implemented the following techniques for tracking of multiple mobile targets:

   a. *greedy technique* based on an opportunistic selection of aircraft direction towards the closest region with high uncertainty about target position,
   b. a technique based on the *randomized traversal of last known positions of targets*, and finally
   c. a technique based on iterative solving of multi-vehicle DVRP problem instances fed with information about last known positions of the tracked targets and uncertainty distribution of their possible positions as they progressively evolve over time.

Findings:

1. the implemented multi-agent solver for multiple-vehicles routing technique performs and scales well w.r.t., number of planes with fixed number of targets,
2. we found that the DVRP-based technique performs fairly well in resource-constrained multi-target tracking scenarios, though does not scale very well w.r.t., the growing number of mobile targets. I.e., often the team of UAVs looses some of the tracked targets due to prolonged periods spent by flying between them. The $M \times N$ ratio thus varies with the overall size of the operations theatre.

WP3: Integrated coordination for mixed information collection activities

Theoretical contributions:

1. we propose a basic mission specification language in terms of sequences of high-level declarative goals. More concretely, the mission is specified for individual simulated units in terms of a list of *Prolog* terms,
2. we propose novel techniques for modular programming of simulated agents based on our past experience with agent-oriented programming languages for agents based on the *Belief-Desire-Intention* (BDI) architecture,
3. we surveyed the state of the art in problems of multi-agent mission planning and multi-agent plan repair,
4. we made first steps towards formulation of novel techniques for multi-agent mission plan repair algorithms.

Technological prototypes:

1. we extended the range of information collection tasks in the Tactical AgentFly simulation platform by implementation of the *exploration task*. For performance comparison, we actually implemented exploration task using the following two methods:

   a. Zig-zag technique based on a coordinated traversal of the explored area in a sweep-style zig-zag pattern from one side to another, and
   b. a technique based on formulation of the exploration task as an instance of the distributed vehicle routine problem and fed into the MAS solver described in the WP2.

2. we developed the *Jazzyk* programming language interpreter, an implementation of the framework of *Behavioural State Machines* and its integration with the AgentFly technological infrastructure including:

   a. *JazzykJVM* interpreter, a port of the original *Jazzyk* language interpreter to the *Java Virtual Machine* platform,
   b. a *Prolog*-based *JazzykJVM* plug-in, and

   c. a Java/BeanShell-based *JazzykJVM* plug-in, furthermore

3. we designed and implemented a framework for configurable mission simulation employing the *Jazzyk* programming language as a vehicle for encoding of interruptible hybrid reactive/deliberative behaviours of the individual simulated agents, and finally
4. we developed an integrated Tactical AgentFly mission scenario involving the various above mentioned information collection tasks, i.e., area exploration, surveillance, multiple target tracking and dynamic switching of these in a running, fully configurable ground mission.

Findings:

1. the evaluation the the implemented exploration information collection task led to the conclusion that informed exploration techniques (such as the DVRP-based approach) are significantly faster than uninformed techniques (such as e.g., the Zig-zag-based approach) in apriori known structured environment. The performance boost was more than 100% in comparison of the two.

1.4 Document structure

The remainder of this report is structured as follows. In the following three chapters, we provide a detailed account of the individual workpackages always structured into three main parts: 1) summary of the workpackage, detailed description of the technologies and theoretical frameworks involved in the solution methods and a description of the experiments and evaluation of the implemented techniques, together with their interpretation. We conclude the report by a discussion of the results and elaboration on possibilities of further improvements of the reported results and inspirations for future work directions. We finally include two appendices respectively describing the technological demonstrators included in the final deliverables package and an overview of the software architecture of the produced simulation platform.

Chapter 2
Modelling and integration of Vertical Take-off and Landing Assets

2.1 Summary of the workpackage

The Tactical AgentFly project aims at development and subsequent integration of VTOL type of assets into the existing AgentFly platform. The second major objective of the project is to propose and develop suitable algorithms for trajectory planning of VTOL assets and integrate them with the VTOL model in AgentFly platform.

The main contribution of our research work in this workpackage is a design of a spatio-temporal trajectory planning algorithm specifically tailored for use by VTOL type assets. The planner takes as an input a set of waypoints the aircraft should fly through, together with vectors of the flight in these waypoints and produces a 3D trajectory respecting the physical constraints of the aircraft e.g., w.r.t. its speed and turning radius in different speeds. The planning algorithm facilitates a significantly more efficient flight than similar trajectory planning algorithms for fixed-wing aircrafts. Subsequently, we implemented the planning algorithm and further extended with the model of VTOL physical dynamics what led to further improvement of the planner-fidelity. Finally, we report on the integration of the planner into the AgentFly platform. To evaluate the quality of the planner, we performed a set of experiments in various scenarios, which we describe below.

2.2 Technology description

2.2.1 Trajectory planning

Model without dynamics

Path planning deals with finding routes between a number of given waypoints satisfying a set of given constraints. Generally, we deal with a search problem in a continuous space. In general, this is an intractable problem. To simplify the task we use regular geometric tessellation lattices (fig. 2.1), where we allow movements only between centers of neighboring polygons.

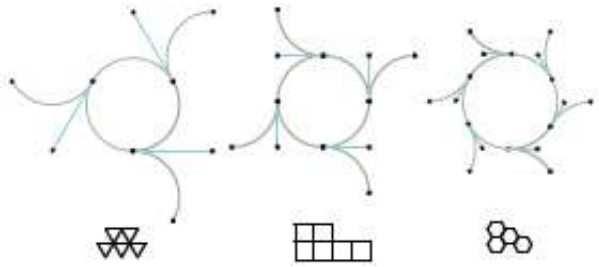

Fig. 2.1 Regular maneuvers for 2D Dubins curves with corresponding regular tessellation lattices (triangular, squarish and hexagonal).

Unlike the other lattices, the hexagonal one (cf. Fig. 2.1) provides the highest number of directions to move towards from a given point (keep in mind that regular lattices allow movements between the centers of neighboring polygons). The distance between adjacent polygons (hexagons in this case) is the same. However, by using the hexagonal lattice we can express only 2D space. In order to extend to the 3D case we use a sequence of parallel planes, again represented as hexagonal lattices, where the distance between two adjacent planes is the uniform. In our implementation, the distance between the planes equals the half of the distance between adjacent hexagons.

The *planning problem* can be defined in the following way. On the input we have two waypoints (starting and ending), a set of no-fly-zones (NFZs) and optionally the starting and ending directions. The task is to find a route from starting to the final waypoint, following the directions (if defined) and avoiding the NFZs. However, the set of input (starting and ending) waypoints needs to be modified so that every waypoint belongs to some plane and is placed into a center of some hexagon. Similarly for directions. NFZs are computed in such a way that every hexagon which intersects some NFZ is marked as blocked (otherwise is marked as free). We define several (primitive) maneuvers (according to planning theory, we can say actions):

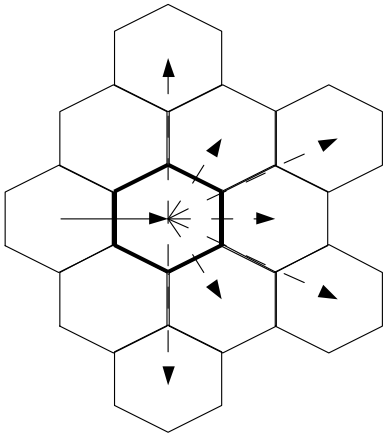

Fig. 2.2 Directions on Hex grid.

Straight — Moves to adjacent hexagon according to direction (note z-coordinate of any direction vector is set to 0).
Turn — Changes the direction.
Pitch — Moves to an adjacent upper (resp. lower) flight level (plane).

However, it turned out to be more appropriate to allow movements not only between adjacent hexagons but also 'diagonally' (cf. Fig. 2.2.1). In a consequence, the turning maneuver can change the aircraft's direction by $\pi/6$, $\pi/3$ and $\pi/2$ (resp. $-\pi/6$, $-\pi/3$ and $-\pi/2$). Obviously, the distance of 'diagonal' move is $\sqrt{3}$ times larger than the distance between adjacent hexagons.

Additionally, there are several restrictions that we require from the generated plans. Every turn maneuver must be applied after at least one straight maneuver. Every pitch maneuver can be applied only between two straight maneuvers. The planning procedure itself is implemented using the A* algorithm, where the actual distance from the starting waypoint g and the estimated cost to the ending waypoint (heuristic) h are computed in every open node. The heuristic h is computed by using Vancouver distance (analogical to Manhattan distance) plus the minimal number of pitch maneuvers that must be use to ascend (or descend) to the desired altitude. If 'diagonal' moves are allowed, then the heuristic h is inadmissible. Hopefully, the impact of the inadmissibility is low. An example of a valid plan in a hexagonal grid is showed in fig. 2.3 (note that the perspective is given orthogonally to hexagonal planes).

Model with simple Dynamics

The model without dynamics we introduced in the previous subsection might work well only for such VTOLs whose maximal speed is relatively low. To

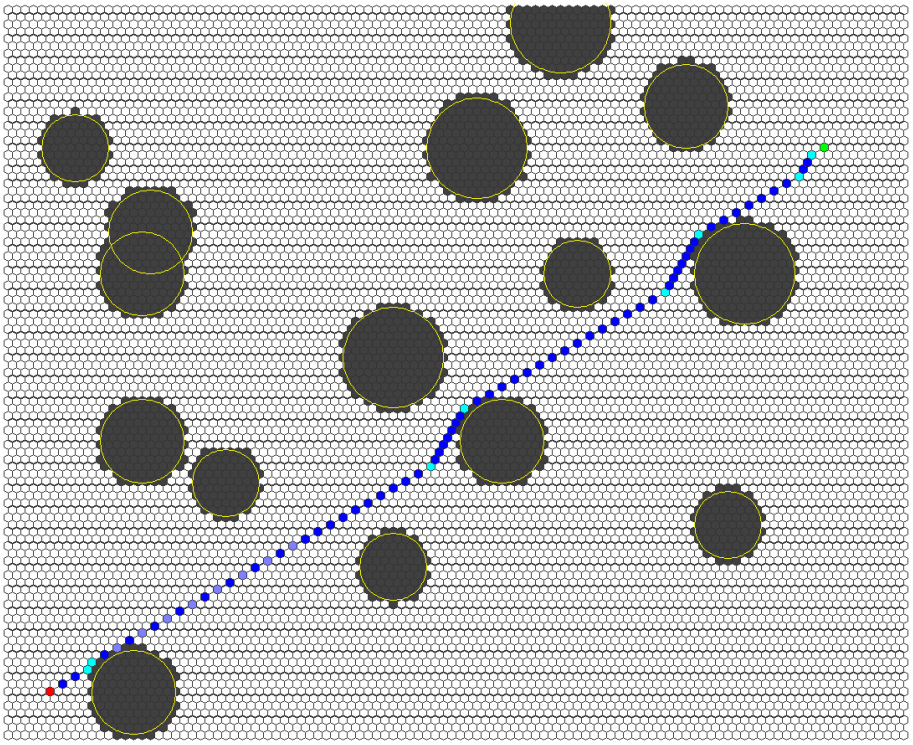

Fig. 2.3 Sample plan on hexagonal grid. Yellow circles represent NFZs. Grey-filled hexagons represent blocked ones. Blue-filled ones represent a path. Cyan-filled ones represent a place where turn maneuver applied. Light blue-filled ones represent a place where pitch-up maneuver applied. Red(Green)-filled ones represent starting (ending) waypoints.

model trajectories more realistically we introduced the *speed* as a new parameter. Relying on the common-sense model of mechanics, the minimal turn radius correlates with the speed. In fact, the minimal turn radius grows quadratically as the speed grows. We decided to keep the turn maneuver within the cell, where the direction changes. The turn radius is then computed in the following way (d stands for a distance between adjacent cells (not 'diagonal'), α stands for an angle of direction change):

$$r = \frac{d}{2} \cot \frac{\alpha}{2}$$

The *planning problem* described in the previous subsection can thus be extended in the following way. On the input we have a basic speed model, where we define maximal acceleration and deceleration, minimal and maximal speed, minimal and maximal turn speed and minimal and maximal pitch speed. Maximal turn speed defined in the basic speed model refers to the smallest turn angle ($\pi/6$). Maximal turn speeds for bigger turn angles are

updated with respect to previously mentioned physical relation (the minimal turn radius grows quadratically as the speed grows). On the input we also have the initial speed interval (in the starting waypoint) and optionally also the interval into which the speed in the destination waypoint should fall.

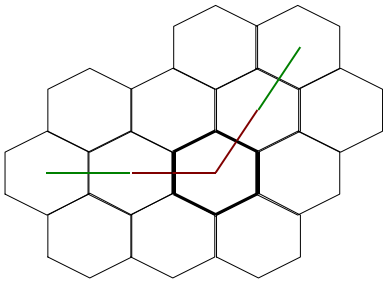

Fig. 2.4 Red line shows the part of trajectory (the Turn maneuver), where the speed cannot be changed.

The following restrictions apply in the resulting model of aircraft dynamics. During the Turn or Pitch maneuvers the speed remains constant (the aircraft cannot accelerate or decelerate). We assume that Turn (resp. Pitch) maneuver starts in a hex cell before and ends in a hex cell right after the hex cell in which the direction (or altitude) was changed (cf. Fig. 2.4).

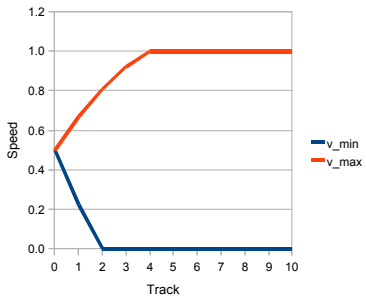

Fig. 2.5 A graph showing the aircraft an example of speed intervals in the simple model of VTOL dynamics. The blue line depicts the minimal speed and similarly the red line depicts the maximal speed of the aircraft. Note that the maximal speed is limited by 1.0, minimal speed is 0.0, acceleration is 0.1 and deceleration is -0.1.

Adding the speed as a new dimension into our model might results in an explosion of the state space. To prevent this we introduce a speed interval for every node (note that the node is specified only via position and direction). Speed intervals represent a range of possible speed values (see a simple example in fig. 2.5). Any limitations of speed (for instance for Turn or Pitch

Algorithm 1 Computing a speed interval for a newly opened node.

{Upper indices − (resp. +) stand for minimal (resp. maximal) speed values.}
if maneuver is straight **then**
    $v^-_{curr} := MAX\left(v^-, \sqrt{(v^-_{prev})^2 - 2as}\right)$
    $v^+_{curr} := MIN\left(v^+, \sqrt{(v^+_{prev})^2 + 2as}\right)$
end if
if maneuver belongs to Turn maneuver **then**
    if $\langle v^-_{prev}, v^+_{prev}\rangle \cap \langle v^-_{turn}, v^+_{turn}\rangle = \emptyset$ **then**
        return null {Speed constraint cannot be satisfied for the Turn maneuver}
    end if
    $v^-_{curr} := MAX(v^-_{prev}, v^-_{turn})$
    $v^+_{curr} := MIN(v^+_{prev}, v^+_{turn})$
end if
if maneuver belongs to Pitch maneuver **then**
    if $\langle v^-_{prev}, v^+_{prev}\rangle \cap \langle v^-_{pitch}, v^+_{pitch}\rangle = \emptyset$ **then**
        return null {Speed constraint cannot be satisfied for the Pitch maneuver}
    end if
    $v^-_{curr} := MAX(v^-_{prev}, v^-_{pitch})$
    $v^+_{curr} := MIN(v^+_{prev}, v^+_{pitch})$
end if
return v^-_{curr}, v^+_{curr}

maneuvers) can be handled via intervals quite easily. We can simply check the current speed interval and speed interval requested by the specific maneuver and if these intervals intersect, then the maneuver can be performed (otherwise the maneuver cannot be performed), for the formal description, see Alg. 1. It is sufficient for the generation of the final path (trajectory), but it is necessary to adjust the speed intervals in every point (hexagon center) of the trajectory such that the intervals follow the conditions of dynamics. We proposed a propagation algorithm (see Alg. 2) for such adjusting the speed intervals. The idea of the algorithm is based on the fact that we have to find a point (on the trajectory) from where we have to slow down (resp. speed up) to reach the maximal (resp.) minimal speed given by the limitation (for instance, the speed limitation for the Turn maneuver). After the point is found we compute the speed intervals in the following points with respect to the given limitation.

A feasible plan consists, like in the model without dynamics, a sequence of points (centers of the hexagons) and directions (in these points). Additionally, in this model we assign speeds to these points such that the speeds are maximal. We can, of course, simply compute time-stamps in these points as well. Time-stamps are important for plan execution.

Algorithm 2 Propagation for speed intervals.

{Upper indices $-$ (resp. $+$) stand for minimal (resp. maximal) speed values.}
$count_{min} := 0$
$count_{max} := 0$
$minspd := v^{-}_{curr}$
$maxspd := v^{+}_{curr}$
Select the predecessor node as a current node
while $v^{-}_{curr} < minspd \vee v^{+}_{curr} > maxspd$ **do**
    Push the current node into Stack
    if maneuver do not belong to the Turn or Pitch maneuver **then**
        $maxspd = MIN\left(v^{+}_{curr}, \sqrt{maxspd^2 + 2as}\right)$
        if $maxspd \neq v^{+}_{curr}$ **then**
            $count_{max}++$
        end if
        $minspd = MAX\left(v^{-}_{curr}, \sqrt{minspd^2 - 2as}\right)$
        if $minspd \neq v^{-}_{curr}$ **then**
            $count_{min}++$
        end if
    end if
    Select the predecessor node as a current node
end while
while Stack is not empty **do**
    Pop the element from the stack as a current node
    if maneuver do not belong to the Turn or Pitch maneuver **then**
        if $count_{max} > count_{min}$ **then**
            $maxspd := \sqrt{maxspd^2 - 2as}$, $count_{max}--$
        else if $count_{max} < count_{min}$ **then**
            $minspd := \sqrt{minspd^2 + 2as}$, $count_{min}--$
        else
            $maxspd := \sqrt{maxspd^2 - 2as}$, $count_{max}--$
            $minspd := \sqrt{minspd^2 + 2as}$, $count_{min}--$
        end if
    end if
    $v^{+}_{curr} := MIN(v^{+}_{curr}, maxspd)$
    $v^{-}_{curr} := MAX(v^{-}_{curr}, minspd)$
end while

2.2.2 Model of helicopter dynamics

The model of helicopter dynamics serves two purposes. Firstly, it is used as a smoothing method for the resulted plans from the hex-grid planner (cf. the previous project reports M18 and M21). Secondly, it is used during the simulation of the helicopter movement. These two cases differ in two main aspects. The smoothing process runs the model using larger time steps (improving the computational duration). On the other hand, the simulation phase may include additional errors in the movement caused by various real-world effects (e.g. wind, imperfections of the asset hardware, sensory errors and glitches, biased regulation and others).

We consider 6 DOF model consisting of three spatial and three rotational axes (cf. Fig. 2.6). The spatial position is described by translation vector

$$\mathbf{p} = (x, y, z),$$

and the rotation is described by three Euler angles φ, θ, ψ in a quaternion form

$$\mathbf{q} = (q_0, q_1, q_2, q_3)^T.$$

Furthermore, both the spatial and the rotational axes are completed by their first and second derivatives, i.e. velocities and accelerations. Spatial velocity and acceleration can be simply derived as follows

$$\begin{aligned} \frac{d\mathbf{p}}{dt} &= \mathbf{v}, \\ \mathbf{p} &= \int \mathbf{v} dt, \\ \mathbf{p} &= \mathbf{p_0} + \mathbf{v}t. \end{aligned}$$

The spatial acceleration respects the same pattern

$$\begin{aligned} \frac{d\mathbf{v}}{dt} &= \mathbf{a}, \\ \mathbf{v} &= \int \mathbf{a} dt, \\ \mathbf{v} &= \mathbf{v_0} + \mathbf{a}t, \\ \frac{d^2\mathbf{p}}{dt^2} &= \mathbf{a}, \\ \mathbf{p} &= \iint \mathbf{a} dt, \\ \mathbf{p} &= \mathbf{p_0} + \mathbf{v}t + \frac{1}{2}\mathbf{a}t^2. \end{aligned}$$

The first derivative of the rotation descried in a quaternion can be derived using differential equation for varying quaternion with angular velocity described in augmented quaternion $\overline{\boldsymbol{\omega}}(t) = (0, \omega_1(t), \omega_2(t), \omega_3(t))^T$ as follows

$$
\begin{aligned}
\frac{d\mathbf{q}}{dt} &= \frac{1}{2}\mathbf{q} * \overline{\boldsymbol{\omega}}, \\
\mathbf{q} &= \int \frac{1}{2}\mathbf{q} * \overline{\boldsymbol{\omega}}, \\
\mathbf{q} &= \mathbf{q}_0 \exp\left(\frac{1}{2}\overline{\boldsymbol{\omega}} t\right) = \\
&= \mathbf{q}_0 \exp\left(\frac{1}{2}\omega_0 t\right)\left(\cos\left(\frac{1}{2}\left|\boldsymbol{\omega}_{\mathbf{123}}\right| t\right), \boldsymbol{\omega}_{\mathbf{123}} \frac{\sin\left(\frac{1}{2}\left|\boldsymbol{\omega}_{\mathbf{123}}\right| t\right)}{\left|\boldsymbol{\omega}_{\mathbf{123}}\right|}\right)^T = \\
&= \mathbf{q_0} \begin{pmatrix} \cos\left(\frac{1}{2}\left|\boldsymbol{\omega}_{\mathbf{123}}\right| t\right) \\ \frac{1}{\left|\boldsymbol{\omega}_{\mathbf{123}}\right|}\omega_1 \sin\left(\frac{1}{2}\left|\boldsymbol{\omega}_{\mathbf{123}}\right| t\right) \\ \frac{1}{\left|\boldsymbol{\omega}_{\mathbf{123}}\right|}\omega_2 \sin\left(\frac{1}{2}\left|\boldsymbol{\omega}_{\mathbf{123}}\right| t\right) \\ \frac{1}{\left|\boldsymbol{\omega}_{\mathbf{123}}\right|}\omega_3 \sin\left(\frac{1}{2}\left|\boldsymbol{\omega}_{\mathbf{123}}\right| t\right) \end{pmatrix}.
\end{aligned}
$$

The inference used definition of quaternion exponential

$$
\exp(\mathbf{q}) = \exp(q_0)\begin{pmatrix} \cos(|\mathbf{q_{123}}|) \\ \frac{\mathbf{q_{123}}}{|\mathbf{q_{123}}|}\sin(|\mathbf{q_{123}}|) \end{pmatrix},
$$

where $\mathbf{q_{123}} = (q_1, q_2, q_3)^T$. The second derivative is based on vector derivation of the augmented quaternion describing the angular velocity as follows

$$
\begin{aligned}
\frac{d\boldsymbol{\omega}}{dt} &= \boldsymbol{\alpha}, \\
\boldsymbol{\omega} &= \int \boldsymbol{\alpha} dt, \\
\boldsymbol{\omega} &= \boldsymbol{\omega_0} + \boldsymbol{\alpha} t.
\end{aligned}
$$

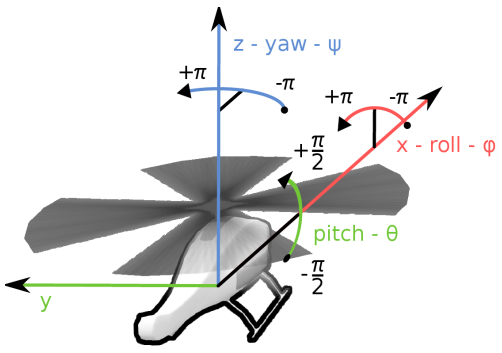

Fig. 2.6 Spatial and rotational axes of the model.

The mentioned equations describe only dynamics of a mass object in space. Following formulas describe a simplified physical model of a helicopter based on the air lift formula

$$L(\eta) = \frac{1}{2}\rho v^2 S C_L(\eta),$$

where $L[N]$ is the lift force, $\rho[\frac{kg}{m^3}]$ is air density, $v[\frac{m}{s}]$ is velocity of the rotor disk causing the lift force, $S[m^2]$ is area of the rotor disk, and $C_L(\eta)$ is coefficient of lift. The lift coefficient C_L is a function of the attack angle η of the rotor blades and it is for the purposes of the model linearized by partially linear function with an ascending part with coefficient k_1 and a descending part with coefficient k_2:

$$C_L(\eta) = \begin{cases} |\eta| < \frac{1}{6}\pi : & \eta k_1, \\ |\eta| \geq \frac{1}{6}\pi : & (\eta - \frac{1}{6}\pi)k_2 + \frac{1}{6}\pi k_1. \end{cases}$$

Spatial acceleration of the object with weight m is affected by two forces (i) gravity, and (ii) the lift force of the main rotor which is parametrized by the *collective rotor tilt* ν:

$$\overline{\mathbf{a}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -g \end{pmatrix} + \mathbf{q} \begin{pmatrix} 0 \\ 0 \\ 0 \\ \frac{L(\nu)}{m} \end{pmatrix} \mathbf{q}^*.$$

Augmented vector $\overline{\mathbf{a}}$ is defined as $\overline{\mathbf{a}} = (0, \mathbf{a})$, $\mathbf{q}^*$ represents an inversion of a quaternion $\mathbf{q}$, and $\mathbf{qpq}^*$ describes rotation of augmented vector $\mathbf{p}$ by a quaternion $\mathbf{q}$.

The angular acceleration is affected by *cyclic rotor tilt* in two dimensions σ_x, σ_y and *anti-torque tail rotor tilt* τ in a following ways

$$\boldsymbol{\alpha} = \begin{pmatrix} \frac{2L_r(\sigma_x)}{mr} \\ \frac{2L_r(\sigma_y)}{mr} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \frac{L_t(\tau)}{mr_t} \end{pmatrix}.$$

The radius of the main rotor is denoted as r, distance of the tail rotor from the main rotor axis is denoted as r_t. The lift force of one half of the main rotor used by the cyclic tilt is denoted as L_r and the lift force caused by the tail rotor is denoted as L_t. The parameters for the particular lift formulas differs in area of the rotor S and mean velocity of the rotor v. The caused lift force of the rotor cyclic is positioned in the center of the rotor radius $\frac{r}{2}$.

Model stabilization

The only explicit stabilization is used for horizontal stabilization of the model. The stabilizers ensure the model will always has tendency to stay horizontally even. The stabilizers are two PID regulators. Each regulator stabilizes the model in one axis, i.e. roll and pitch. The target state of the model rotation is zero angles of rotation.

The regulation equations are based on the equation of ideal PID regulators

$$\sigma_x = P_s(0-\varphi) + I_s \int (0-\varphi)dt + D_s \frac{d(0-\varphi)}{dt},$$
$$\sigma_y = P_s(0-\theta) + I_s \int (0-\theta)dt + D_s \frac{d(0-\theta)}{dt},$$

the equations are discretized and used in iterative manner. A step characteristics of the regulator is presented in Figure 2.8.

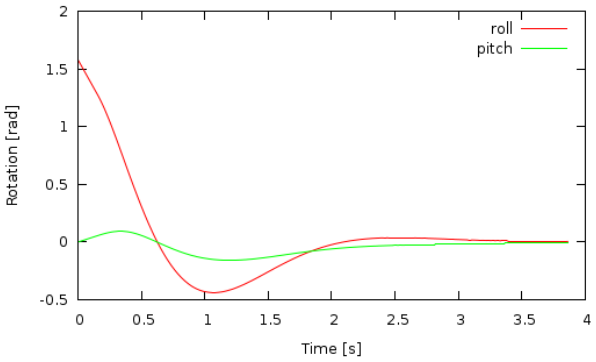

Fig. 2.7 Step characteristic of roll stabilization using the PID regulator.

Movement regulation

The movement of the model in the main three axes (spatial axes x, y, z and rotational axis ψ) is handled by four movement PID regulators. On the contrary of the stabilization regulators, the movement regulators are designed to fulfill requested height, velocity, direction, or yaw rotation.

The first regulated value is the height z of the model to target height z_t. It is based on PID regulator described by following relation

$$\nu = P_z(z_t - z) + I_z \int (z_t - z)dt + D_z \frac{d(z_t - z)}{dt}.$$

The affected action control is the collective rotor tilt ν, i.e. strength of the main rotor climbing/descending. A step characteristics of the height regulator is presented in Figure 2.8.

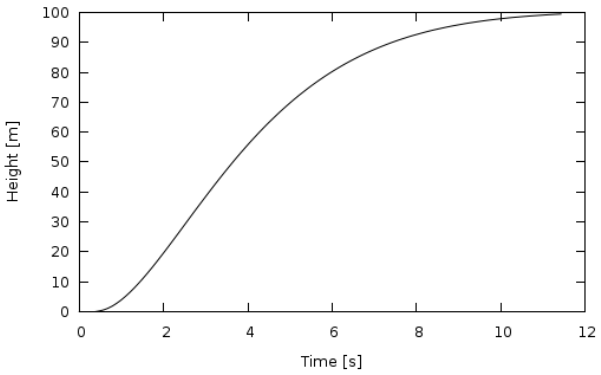

Fig. 2.8 Step characteristic of height PID regulator.

The second two regulators affect the cyclic rotor providing horizontal movement of the helicopter. The formulas for the PID regulator differs in the used constants and parameters

$$\sigma_x = P_v(v_{tx} - v_x) + I_v \int (v_{tx} - v_x)dt + D_v \frac{d(v_{tx} - v_x)}{dt},$$
$$\sigma_y = P_v(v_{ty} - v_y) + I_v \int (v_{ty} - v_y)dt + D_v \frac{d(v_{ty} - v_y)}{dt}.$$

They regulate the current velocity $\mathbf{v}$ towards the target velocity $\mathbf{v_t}$ in the required directions to the target point (typically a trajectory waypoint). A resulting trajectory using all the previous regulators is depicted in Figure 2.9. The waypoints are switched after reaching a predefined distance to the next waypoint.

The fourth and last regulator is used for pointing of the head of the model towards the target point. Although the model can reach any position using only the previous three spatial axes, the yaw rotation is important if the model carries an orientation dependent sensor (e.g. a range finder, of a camera). The torque regulator is also PID and it is defined as

$$\nu = P_r(0 - \angle\mathbf{d}) + I_r \int (0 - \angle\mathbf{d})dt + D_r \frac{d(0 - \angle\mathbf{d})}{dt}.$$

The vector $\mathbf{d}$ points towards the target point and $\angle\mathbf{d}$ denotes an angle between the direction vector (e.g. velocity vector) and the vector $\mathbf{d}$. The demonstration of the torque regulator is presented in Figure 2.10.

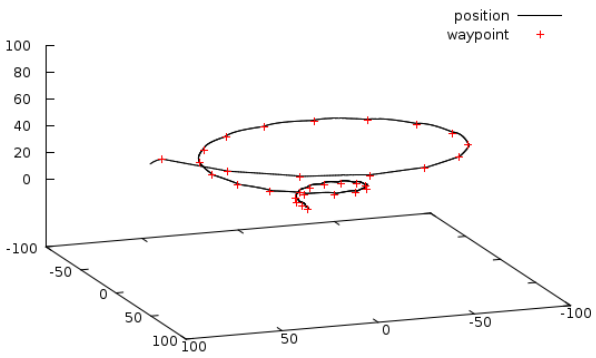

Fig. 2.9 Spiral movement of the helicopter using the height and velocity regulators.

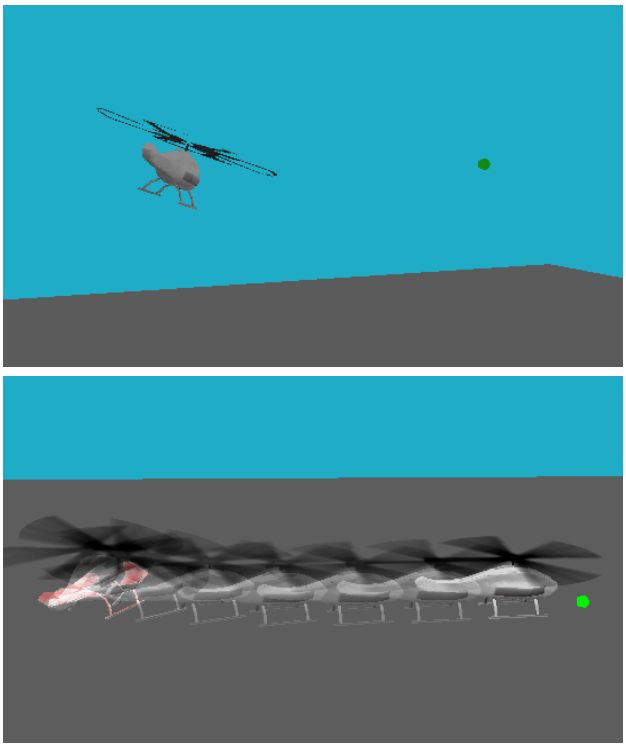

Fig. 2.10 Application of the torque regulator.

Trajectory smoothing using the model

The model used for the smoothing of the trajectory produced by the hex-grid planner uses all the introduced regulators together with the model of helicopter dynamics. The resulting plan trajectory is depicted in Figure 2.11. The plan begins at ground and increases the height together with forward movement towards the target. During the flight a no-flight zone is avoided provided that the hex-grid planner planned the waypoint around the zone.

All the used regulator parameters $P_s, P_z, P_v, P_r, I_s, ..., D_r$ were identified using the Ziegler-Nichols method. The model parameters including weight, rotor area, distance to the tail rotor and others are based on the Skeldar 200 VTOL UAV from SAAB Aerosystems.

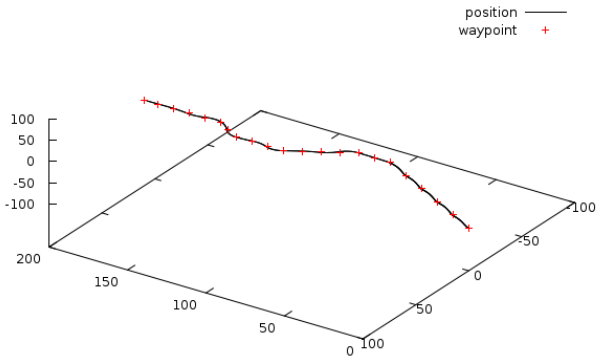

Fig. 2.11 Smoothed plan trajectory using the dynamics model of the VTOL.

2.2.3 Trajectory visualization

A new visualization module had to be prepared to be applicable with the new trajectory description using the list of VTOL states. The visualization is based on partially linear description of the trajectory. Each segment between two waypoints is represented by a cuboid with cut ends under a normal angle in the waypoint. The angle cuts the segments in the middle and creates a connection to successive segment. Each segment is visualized in 3D and creates a tunnel representing the path which is followed by the asset.

An example of a path plan visualized using the presented method is depicted in Figure 2.12. The turn maneuvers are represented by arcs (parts of circles) whose radiuses are computed with respect to cell sizes and turn angles (as discussed before).

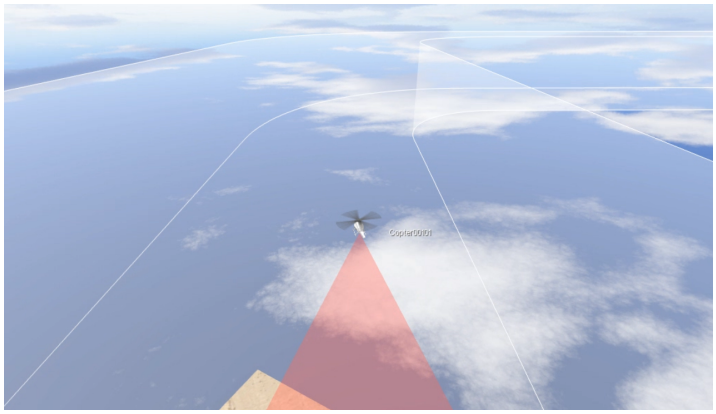

Fig. 2.12 An example of the visualization of the new plan representation.

2.3 Evaluation and experiments

We experimentally compared both the models in terms that we wanted to find out how including a simple dynamics affects the planning process. We randomly generated 100 points within the area of size (100,70,5), where the distance between adjacent hexagons was 2 and the distance between planes (of hexagonal grids) was 1. We also defined 20 cylindrical NFZs, where all the centers lied within the area and altitude was unlimited. The area served for restriction of starting and ending waypoints and NFZs, but not for the planning procedure. It means that any feasible path could leave the area for a while.

	Time [ms]	% Solved	% Unsolved in 1000ms	Median [ms]	Nodes per [ms]
basic	102295	99.6	0.4	0	97
dynamic	114742	99.7	0.3	1	94
basic+NFZs	81965	51.3	0.3	0	75
dynamic+NFZs	79857	39.8	0.3	0	70
basic+NFZs+vecs	442023	45.7	6.0	1	64
dynamic+NFZs+vecs	356730	35.5	4.6	1	62

Table 2.1 Comparison of the basic model and the model with a simple dynamics.

For all pairs (i, j) such that $j > i$ we were looking for plans (trajectories) from i-th to j-th waypoint. It gave us 4950 problems total for a particular case. We tested our basic model, model with a simple dynamics, firstly without any restriction, then we added NFZs and then we added starting and ending direction vectors. Of course, the waypoints, NFZs and direction vectors remained the same for all the tested cases. The time limit for every planning task was set to 1000ms, because for practical purpose, it is quite inappropri-

ate to look for the plan longer. The planner was implemented in JAVA SE 1.6. The experiments were performed on Core2Duo 1.86 GHz, 2 GB RAM, Windows 7.

The results are presented in the table 2.1. At First, it is advisable to mention that when NFZs are added, then some planning tasks become unsolvable because starting or ending waypoint lies within some of NFZs. It clarifies the significantly lower number of solved planning tasks, where NFZs are applied. Moreover, the model with a simple dynamics brings additional constraints (i.e., the UAV must accelerate to achieve minimal speed for the Pitch maneuver etc.) which resulted in the lower number of solved tasks (according to these constraints some tasks solvable in the basic model became unsolvable). Adding direction vectors caused significant slow down of the planning process, especially when required ending direction was (almost) opposite to direction we come from. It also caused that quite a lot of problems were not solved within the time limit (1000 ms). Considering the comparison of our two models without any additional restriction (NFZs, direction vectors) we can clearly see that the total time spent on all the planning tasks were about 12% higher in the model with a simple dynamics. This slow down was caused mainly by two factors, maintaining the speed intervals in every node and slightly longer plans. However, in the other comparisons the total time spent on all the planning task is not very informative, because in the model with a simple dynamics we solve much less tasks. On the other hand, we can compare how many nodes were explored per millisecond (in average). We can see that in this aspect the model with a simple dynamics does not stay much behind.

The experimental evaluation is in fact targeted to provide a rough view how the simple dynamic model can affect the planning process with respect to solving time. The results gave us a positive outlook, since the running time needed for solving did not increase much when used the simple dynamic model. However, we should investigate this more thoroughly in future to provide more general claims.

Chapter 3
Planning in Dynamic and Resource Constrained Environments

3.1 Summary of the workpackage

The planning and coordination in dynamic resource constrained environment has been analyzed with respect to scalability and limited resources. The multi-agent solver has been utilized for cooperative surveillance and tracking under undervalued numbers of aircrafts towards the numbers of ground targets. The problem is to find an optimal allocation of the airplanes in cases the number of the targets exceeds the number of the assets in the dynamically changing environment. This allocation has to be dynamic (i.e. airplanes have to hand over the targets when another plain is loosing them) and minimize the chance of target loss, e.g. maximize the persistence of targets tracking.

We have defined the surveillance and tracking domain in the form of tasks usable by the distributed allocation architecture (i.e. by the solvers). The objective function has been designed to fulfill the requirements of the missions. The algorithms developed has been integrated with the deliberative modules of the aircraft agents. The algorithms has been extended to be able to handle heterogeneous teams of the assets including CTOL UAVs, VTOL UAVs, and prospectively also ground units. The implemented algorithms have been evaluated in AgentFly simulation with hi-degree of environment dynamism.

Firstly, the focus was put mainly on a multi-agent problem solving architecture based on a task allocation and local resource planning. The architecture is usable for surveillance and tracking under undervalued numbers of aircrafts towards the numbers of ground targets. The later task delegation is used for solution improvement. The multi-agent solver architecture uses the principles of problem decomposition and delegation to autonomous agents that solve individually the parts of the problem. The overall solution is then obtained by merging the individual agents' results. The solution is improved using a task sharing approach. The principle is based on delegation of tasks from a busy agent to a vacant agent using a set of improvement strategies.

The algorithms have been implemented with proposed heuristics and strategies and tested in synthetic environments. In a multi-agent NP-hard problems (logistics-like), the solver provides solutions with the quality of 81% compared to the optimal solution on 115 benchmark instances in polynomial time. Such an efficiency of the algorithms gives us a good reason to believe that the principles will be usable for other distributed problems including allocation of aircrafts for surveillance and tracking.

Secondly, the general-purpose multi-agent coordination algorithms with a sharp focus on heterogeneous teams (i.e., those consisting of various types of UAVs capable to perform different tasks and/or performing them in different environments) was designed and implemented. The main thrust of our work was the realization of coordination techniques required to successfully implement the demonstration scenarios.

We have evaluated the multi-agent solver in the various scenarios of cooperative surveillance and tracking presented in Section 3.3.1. The experimental evaluation shows the influence of the team composition and coordination level to the surveillance and tracking efficiency. The validation in the surveillance and tracking domain in the form of tasks usable by the distributed allocation architecture (i.e. by the solvers) has been performed. The stability and self-balancing ability of the multi-agent solver has been experimentally proved according to the requirements of the missions.

One of the most important results of the project research track leading to implementation of multi-target tracking by a coordinated team of UAVs is the proposal of a multi-agent problem solving architecture based on task allocation techniques and task decomposition with subsequent merging of partial results. Firstly, we have focused on the multi-agent problem solving architecture extension towards MxN surveillance and tracking. We introduced the techniques of (i) tasking extension, (ii) task injection, and (iii) task groups implementation and finally we described a set of experimental evaluation scenarios on which the performance of the individual algorithms should be evaluated. Secondly, we have implemented the demonstration scenarios and subsequently performed an evaluation of the proposed techniques on them together with a subsequent analysis and interpretation of the results.

In the last project period, we have performed further evaluation of multi-agent coordination algorithms with a sharp focus on heterogeneous teams, i.e., those consisting of various types of UAVs capable to perform different tasks and/or performing them in different environments. In particular we were working on the last MxN project scenario in which we evaluate techniques for emergent team coordination in contrast to analytical techniques, such as as greedy target seeking on one side, and (not exclusively) targets clustering and subsequent vehicle routing among within the cluster.

3.2 Technology description

In this section, we focus present a multi-agent problem solving architecture based on a task allocation and local resource planning. The architecture is usable for surveillance and tracking under undervalued numbers of aircrafts towards the numbers of ground targets. The later task delegation is used for solution improvement. The section is divided into two parts presenting (i) abstract multi-agent solver, and (ii) its algorithms.

In the second part of this section the extension of the multi-agent problem solving architecture towards MxN surveillance and tracking is presented. We introduce the techniques of (i) tasking extension, (ii) task injection, and (iii) task groups implementation.

Section 3.3 then presents a set of experimental evaluation scenarios on which the performance of the individual algorithms are be evaluated. The implemented demonstration scenarios and evaluation performed are discussed.

3.2.1 Multi-Agent Solver

Multi-agent planning approaches are used for solving a wide variety of planning problems. One of the related strategies is a *task sharing* approach. The principle is based on passing of tasks from a busy agent to a vacant agent(s). The process can be summarized in four basic steps:

1. *Task decomposition:* The tasks of the agents are decomposed into subtasks and subtasks, which can be shared are selected.
2. *Task allocation:* The selected tasks are assigned to the vacant agents or agents, which ask for them.
3. *Task accomplishment:* Each agent tries to accomplish its (sub)tasks. The tasks, which need further decomposition, are recursively processed and passed to other agents.
4. *Result synthesis:* The results of the tasks are returned to the allocating agent since it is aware of the way to use it in the context of the higher tasks.

From the perspective of distributed problem solving, the task allocation and the result synthesis are the most crucial parts. On the other hand, from the planning perspective, the other two phases are more important. The allocation problem is usually solved by contraction and negotiation techniques, which implies problems related to the resource allocation domain, e.g., cross-booking, over-booking, backtracking, and others. In the allocation phase, a hierarchy of agents is established, which may not be effective in heterogeneous multi-agent systems.

The decomposition and delegation principle is widely used in agent-based approaches for problem solving and planning and shows great applicability to realistic problems.

We can define the abstract multi-agent solver architecture as a composition of three types of agents (see Figure 3.1):

- **Task Agent** for preprocessing of the problem. This agent should use domain specific heuristic, generic ordering strategy, or randomized method.
- **Allocation Agent** for problem decomposition and delegation of the subproblems to Resource Agents. This agent also maintains the task allocation and result synthesis.
- **Resource Agent** for individual resource planning. In case of further decomposition, the subproblem is handed over to another Task Agent.

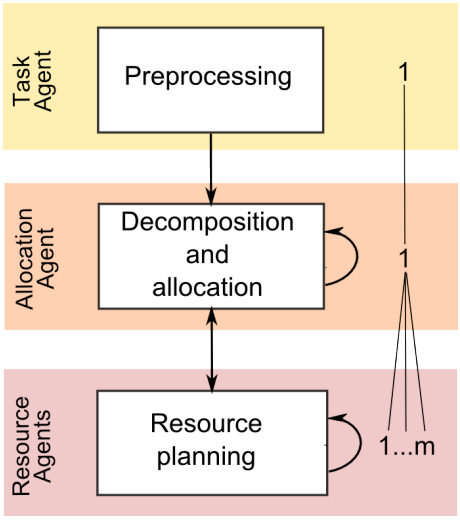

Fig. 3.1 An abstract architecture of the agent-based solver/planner.

The multi-agent system built upon this architecture is composed of one Task Agent, one Allocation Agent and a set of Resource Agents. For complex hierarchical systems, the abstract architecture can be reflected in the multi-agent system recursively, it can be reduced (i.e. one agent undertakes a role of more than one abstract agent type), or it can be parallelized (more abstract solvers are instantiated with potential overlapped agents – e.g., several Task Agents or Allocation Agents handling various problems in parallel). In a big systems, there may arise concurrent interactions that needs to be handled. The agents' interactions are guided by the interaction protocols, which are mostly built on contract net protocol (CNP) [11].

3.2.2 Abstract Algorithm

The multi-agent solver uses the principles of problem decomposition and delegation to autonomous agents that solve individually the parts of the problem. The overall solution is then obtained by merging the individual agents' results. The optimization based on CNP interactions in cooperative environment is usually described as utilitarian social welfare maximization. So the abstract algorithm objective function can be defined as maximization of social welfare, which is

$$sw = \sum_{a \in \mathcal{P}} u_a, \tag{3.1}$$

where $\mathcal{P} = a_1, \ldots, a_n$ is population of agents and u_a is utility of agent a. In our case, the social welfare can be computed as a sum of Resource Agents utilities that can be defined as

$$u_a = \sum_{t \in \mathcal{T}_a} (rew(t) - cost(t)) = \sum_{t \in \mathcal{T}_a} rew(t) - cost(\mathcal{T}_a), \tag{3.2}$$

where $\mathcal{T}$ is a set of tasks delegated to the agent a, $rew(t)$ is a reward for fulfilling task t, $cost(t)$ is a cost of agent a to perform task t, and $cost(\mathcal{T}_a)$ is cost of the overall plan of an agent. The total reward for fulfilling a set of all tasks $\mathcal{T}$ is

$$rew(\mathcal{T}) = \sum_{a \in \mathcal{P}} rew(\mathcal{T}_a) = \sum_{a \in \mathcal{P}} \sum_{t \in \mathcal{T}_a} rew(t), \tag{3.3}$$

so the social welfare can be expressed as

$$sw = rew(\mathcal{T}) - \sum_{a \in \mathcal{P}} cost(\mathcal{T}_a) = rew(\mathcal{T}) - \sum_{t \in \mathcal{T}} cost(t). \tag{3.4}$$

Since the reward $k = rew(\mathcal{T})$ is not generally influenced by the allocation of tasks to the agents (we assume the same quality of task fulfilling by any agent), the maximization of social welfare is the same as minimization of solution cost, e.g.

$$\begin{aligned} \max sw &= \max rew(\mathcal{T}) - \sum_{t \in \mathcal{T}} cost(t) \\ &= k - \min \sum_{t \in \mathcal{T}} cost(t) \end{aligned}$$

The *objective function* of the abstract solver is then

$$\min \sum_{t \in \mathcal{T}} cost(t), \tag{3.5}$$

where $cost(t)$ is evaluated by the Resource Agent undertaking task t.

Algorithm 1 The abstract algorithm of a multi-agent solver.

```
Input: Set of tasks T, set of Resource Agents R
Output: T allocated on R
          and local plans of Resource Agents exist

function solve(T, R) begin
   apply ordering heuristic on T
   forall t : T begin
      allocateCNP(t,R)
      if allocation not successful then
         exit with failure or
         mark t as not allocated and continue
      end
      forall r : R begin
         apply dynamic improvement strategy
      end
   end
   forall r : R begin
      apply final improvement strategy
   end
end

function allocateCNP(t, R) begin
   forall r : R begin
      find winner with the lowest insertion
        estimation of t
   end
   if winner is found then
      assign t to the winner
   else
      allocation not successful
   end
end
```

The abstract algorithm representing the presented multi-agent solver minimizing objective function denoted by Equation 3.5 is captured by Algorithm 1. It contains three phases: (i) task preprocessing provided by the Task Agent, (ii) allocation performed by the Allocation Agent with taking into account Resource Agents' computations of insertion costs and (iii) dynamic and final improvement based on the cooperation of Allocation Agent and Resource Agents.

The algorithm is based on local optimization of a single task insertion and improvement. Each iteration of the algorithm provides locally-optimized solution of resources utilization and order-dependent task allocation. The algorithm does not use any backtracking mechanism or exhaustive search of the state space. It has a significant impact on the algorithm's computational complexity, but it is susceptible to finding locally efficient solution only. The

Algorithm 2 The abstract algorithm improvement strategies.

```
function improveDW(r,R) begin
   t = the worst task of agent r
   forall a : R \ r begin
      find winner with the lowest cost^estI of t
   end
   if cost^estI of winner is lower then cost^estR of r then
      delegate t from r to winner
   end
end

function improveDA(r,R) begin
   forall t : tasks of agent r begin
      forall a : R \ r begin
         find winner with the lowest cost^estI of t
      end
      if cost^estI of winner is lower then cost^estR of r then
         delegate t from r to winner
      end
   end
end

function improveRA(r,R) begin
   forall t : tasks of agent r begin
      remove t from agent r
      allocateCNP(t,R)
   end
end
```

global solution quality is improved by execution of improvement strategies, which are:

- **Delegate worst** (DW) – each Resource Agent identifies its worst task and tries to delegate it to another agent if the removal cost (savings) is higher than the insertion cost.
- **Delegate all** (DA) – each Resource Agent delegates all its tasks (only if the removal cost is higher than the insertion cost).
- **Reallocate all** (RA) – each Resource Agent successively removes all its tasks from the plan and allocates them again using the CNP strategy. The result of the allocation can be the same as before task removing, or a change of the position of the task in the current agent plan, or delegation to another agent.

The Resource Agent has to provide the functions for insertion cost computation used by the algorithm for determining the *winner* in the CNP allocation. The Resource Agent uses a case-dependent resource planning heuristic for these computations. The functions for allocation are:

- **insertion estimation** $cost^{estI}$ – The estimation of the cost of the task insertion. It represents the increase of the agent's cost function caused by undertaking the task.
- **insertion** $cost^{insert}$ – The real cost of the task insertion. This value is determined by adding a new task to the plan in the current state. It is the result of the particular resource planning algorithm of the Resource Agent.

The opposite functions used by improvement strategies are:

- **removal estimation** $cost^{estR}$ – The estimation of the cost of the task removing. It represents the decrease of the agent's cost function caused by dropping the task.
- **removal** $cost^{remove}$ – The real cost of the task removal. This value is determined by removing the task from the plan in the current state. It is the result of the particular resource planning algorithm of the Resource Agent.

When using standard CNP we require the estimation functions of Resource Agent to provide the accurate estimations

$$cost^{estI} = cost^{insert},$$
$$cost^{estR} = cost^{remove}.$$

The RA estimation functions are *admissible* only if

$$cost^{estI} \leq cost^{insert}, \tag{3.6}$$
$$cost^{estR} \geq cost^{remove}. \tag{3.7}$$

The improvement strategies described by Algorithm 2 (improveDW and improveDA) uses the task delegation on condition that $cost^{estI}$ of *winner* is lower than $cost^{estR}$. This can be defined as *improvement condition*:

$$\forall i, j : cost^{remove}_{RA_i}(t) - cost^{insert}_{RA_j}(t) \geq 0 \tag{3.8}$$

for admissible delegation of task t from agent i to agent j. The case $i = j$ is achievable by improveRA strategy only.

The *admissible Resource Agent strategy* (its internal heuristics and estimation and allocation functions) has to:

- use admissible estimation functions according to Equation 3.6 and 3.7, and
- fulfill improvement condition defined by Equation 3.8.

The algorithms has been implemented with proposed heuristics and strategies and tested in synthetic environments. In a multi-agent NP-hard problems (logistics-like), the solver provides solutions with the quality of 81% compared to the optimal solution on 115 benchmark instances in polynomial time. Such an efficiency of the algorithms gives us good reason to believe the principles

will be usable for other distributed problems including allocation of aircrafts for surveillance and tracking.

3.2.3 Cooperative Surveillance and Tracking

The presented multi-agent solver has been used for multiple goals allocation to the team of heterogeneous agents. The goals represents surveillance and tracking tasks and the agents represents the unmanned vehicles capable to provide surveillance, tracking, or both.

We have a set of heterogeneous agent capabilities

$$\mathcal{C} = c_1 \dots c_n, \tag{3.9}$$

where each agent a is able to provide a subset of capabilities $\mathcal{C}_a$. We divide a set of task $\mathcal{T}$ to subsets $\mathcal{T}_1 \dots \mathcal{T}_n$, where the tasks in $\mathcal{T}_i$ can be undertook by agents providing the capability c_i and

$$\bigcup_{i=1}^{n} \mathcal{T}_i = \mathcal{T},$$
$$\bigcap_{i=1}^{n} \mathcal{T}_i = \emptyset.$$

According the agent capabilities, the agents population $\mathcal{P}$ can be divided into sets of Resource Agents able to undertake particular task set, so

$$\mathcal{R}_i = a_1 \dots a_n, a_i \in \mathcal{P},$$
$$\left| \bigcap_{i=1}^{n} \mathcal{R}_i \right| \geq 0,$$

and all agents from $\mathcal{R}_i$ provide capability c_i, i.e. are able to undertake a task from $\mathcal{T}_i$. The multi agent solver algorithm modification is described by Algorithm 3.

The allocation is executed for all tasks on different sets of Resource Agents according their capabilities and the improvement strategies are applied to this set of Resource Agents as dynamic improvement strategy and to all Resource Agents in the phase of final improvement strategy application. The functions *allocateCNP*, *improveDW*, *improveDA*, and *improveRA* of the Algorithm 1 and Algorithm 2 have to be modified to find the *winner* agent from the appropriate set of Resource Agents $\mathcal{R}_i$ corresponding to actual task t. It correspond to infinite *cost* of insertion estimation of the agent r iff

Algorithm 3 The modified abstract algorithm of a multi-agent solver for heterogeneous agent capabilities and tasks.

```
Input: Sets of tasks T_1 ... T_n, sets of Resource Agents R_1 ... R_m
Output: all tasks allocated
        and local plans of Resource Agents exist

function solve(T_1 ... T_n, R_1 ... R_m) begin
  for t : T_1 ... T_n begin
    R = R_i, where t ∈ T_i        allocateCNP(t,R)
    if allocation not successful then
      mark t as not allocated and continue
    end
    forall r : R begin
      apply dynamic improvement strategy
    end
  end
  forall r : R_1 ... R_m begin
    apply final improvement strategy
  end
end
```

$$t \in \mathcal{T}_i, r \notin \mathcal{R}_i. \tag{3.10}$$

The objective function of the solver (see Equation 3.5) holds even it is not explicitly handles the heterogeneous tasks and agent capabilities. In case of multiple capabilities of a Resource Agent the weighting factor for the different task prioritization has to be captured by the local agent planner. This weighting factor has not to be the same for all agents.

This multi-agent solver extension has been used in the implementation described in Section 3.2.4 with the following setting:

- $\mathcal{R}_1$ is a set of high altitude, long endurance UAVs able to provide high-level surveillance,
- $\mathcal{R}_2$ is a set of tactical UAVs able to provide Zig-zag surveillance,
- $\mathcal{R}_3$ is a set of micro UAVs able to provide tracking and close surveillance.

3.2.4 Tasking Extensions

We have extended the current principles and implementation of the UAV tasking system. Firstly, we have enabled injection of tasks directly into the agent community by sending an explicit topic message and secondly, we have added an ability to form a distinct groups of the UAVs to conduct different types of the tasks simultaneously (see Figure 3.2).

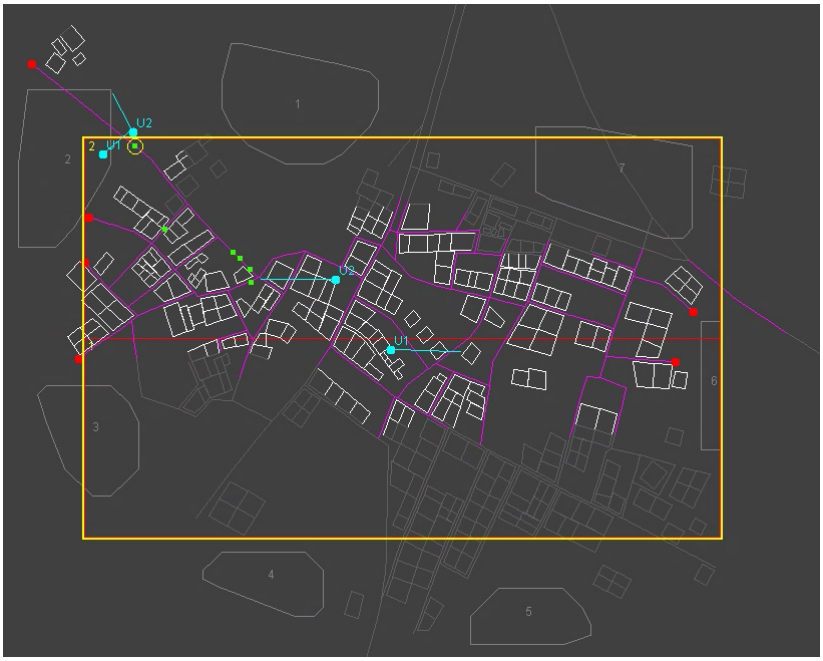

Fig. 3.2 Common Operational Picture showing two groups of UAVs. One group (U_1 and U_2 in the top-left corner) is providing tracking of a ground entity and the other one (U_1 and U_2 in the center) is doing a wide Zig-zag surveillance over the mission area.

3.2.5 Task Injection

The requirement for more complex autonomous control of the missions eventuates a need for tasking of the UAVs not only from the Common Operational Picture (COP) GUI by the user, but also by a simple-to-use interface usable within the system. The interface has to be usable both by another agents, which can require a task to be performed for them, and by low-level system parts running the scenarios (e.g. by the `ScenarioPlayer` – a object orchestrating the course of the scenario, creating the entities, measuring several parameters, and the like).

Since the COP GUI uses *topic messages subsystem* for tasking of the UAV agents, we have reused the principle and adapted the behaviour of the agents for few of unanticipated cases. The tasking process consists of creation of a instantiated `TaskBatchAssignment` data structure that describes a batch of tasks (a batch can contain more than one tasks for compound tasking). Afterwards, the instance is sent by a `TOPIC_ASSIGN_TASK_BATCH` topic by the topic subsystem. Since the topics are broadcasted, the batch is received by all the `TacticalModules` in the pilot agents. Following processing of the batch respects the algorithms for task handling of the system.

3.2.6 Task Groups

For missions where various types of UAVs are required the task groups are an essential functionality of the system. For an instance, we can have a High Altitude, Long Endurance (HALE) UAV providing a high-level surveillance of the target area, two Tactical UAVs (TUAVs) providing Zig-zag surveillance of the area and several Micro UAVs (MUAVs) providing tracking of various objects of interest, or providing tight-area surveillance (e.g. of a building or entry/exit points to/from an perimeter). For such a mission, we need to be able to distribute the task batches according to the types of the aerial vehicles.

The general idea of the extension is to provide an additional parameter, together with each task batch, describing the type of the UAVs which should perform the batch. In particular, the `TOPIC_ASSIGN_TASK_BATCH` topic can be sent with a string identifier of *a plane type*. The plane type is an AgentFly specification of the airplane by means of the model parameters (including safety range, turn and pitch radii, velocity constraints, and others) with additional TAF parameters (surveillance and tracking altitude, and camera angle). The type is, on the side of the `TacticalModules`, checked against the current plane's type and the incompatible batches are filtered out. This principle naturally distributes the task batches according to the plane types into several groups. Since the differentiation is based on the plane types, the groups are homogeneous and each consists of several numbers of same UAVs. In the former example, we would have three groups, one containing only one HALE UAV, second containing the TUAVs, and the last only the MUAVs. If the plane type is omitted, the task batch is comprehended like a overall batch and it is adopted by all the UAVs regardless the type.

Since the algorithm for task allocation of a task batch is distributed and the initial phase is based on a peer-to-peer communication among the agents about the same tasks, the groups are formed automatically with a minor modification of the underneath allocation principles (see Section 3.2.3). Tangibly, each group identifies its Coordinator agent, and because the agents work only with the batches filtered for them, they receive only particular tasks from the batches according to their groups. As each plane can be only of one type, the groups are disjoint and their conjunction is the overall group.

The COP GUI is updated independently by each Coordinator agent of each group and batches added using the COP are presumed to be the overall task batches (performed by all UAVs).

3.3 Evaluation and experiments

The general-purpose multi-agent coordination algorithms has been developed and evaluated on the series of scenarios introduced in this section. The focus

has been put on heterogeneous teams, i.e., those consisting of various types of UAVs capable to perform different tasks and/or performing them in different environments.

We carried out a number of experiments in three different setups according to scenarios and developed algorithms. The overview of the settings is given on Figure 3.3.

The experiments focus on the level of cooperation in information collection missions under resource constraints. The team of UAVs provides continuous coordinated surveillance using technologies and algorithms developed in previous phases of the project. Moreover, non identified ground units, the targets, traverse the surveyed area. The goal of the UAVs is to (i) secure continuous surveillance of the whole area minimizing the information age, (ii) identify important targets as soon as they enter the area, and (iii) perform tracking of the targets until they leave the area. The quality of the goals fulfillment (information age, time of non-tracked movement of a target in the area, etc.) is evaluated according to the type and intensity of coordination techniques employed.

A group of 10 moving ground targets (simulated people) was employed. These simulated people walked in a loop along predefined trajectories through an urban area. In all scenarios, the group of people and their trajectories were the same so that the obtained results were comparable.

During the experiments, various numbers of surveying and tracking UAVs were used to monitor the area and to perform tracking of as many ground targets as possible. The overall goal was to minimize the average information age, i.e. the age of the data obtained by surveillance, and the average information error, i.e. the difference between the actual and the last-observed positions of moving ground targets. All experiments were executed for the same amount of simulation time, long enough to ensure sufficient stabilization of average values of information age and information error.

3.3.1 Demonstration scenarios

The level of cooperation in information collection missions under resource constraints will be demonstrated using the following scenarios. The team of UAVs will provide continuous coordinated surveillance using technologies and algorithms developed in previous phase of the project. Moreover, non identified ground units, the targets, will traverse the surveyed area. The goal of the UAVs will be to (i) secure continuous surveillance of the whole area minimizing the information age, (ii) identify important targets as soon as they enter the area, and (iii) perform tracking of the targets until they leave the area. The quality of the goals fulfillment (information age, time of non-tracked movement of a target in the area, etc.) will be evaluated according to the type and intensity of coordination techniques employed. An overview

of the demonstration scenarios is depicted in Figure 3.3. We have defined following scenarios

We have defined the following demonstration scenarios:

1. *two homogeneous teams of the UAVs* – one for surveillance goal and one for tracking goal. The surveillance team uses the zig-zag algorithm implemented in the first phase of the project utilizing all members of the team to minimize information age. Once the team notices new/unserved target, new tracking task will be generated for the second team.
2. *a single heterogeneous team of multiple capabilities UAVs* – the scenario is similar to the previous one, except that every UAV is able to perform both surveillance and tracking tasks. Initially, all UAVs perform surveillance as in first scenario. When an UAV notices a non identified ground unit, target, it switches to tracking this target. The rest of the UAVs adapt their trajectories to further perform surveillance and minimize the information age of the covered area among the team members. When the target leaves the area, the tracking UAV re-joins the surveillance team.
3. *heterogeneous team with handover* – the scenario is very similar to the second one, but the team members dynamically balance two concurrent goals: 1) keeping the continuity of the team surveillance, and 2) tracking of previously identified targets. The handover of the tracking task, i.e., a UAV delegates tracking of a target to another UAV, will be executed as the target moves through the area to minimize disturbances of the individual UAVs surveillance routes.
4. *limited resource tracking* – this scenario is different from previous ones. The goal is to keeping the continuity of tracking ground targets. The problem is the number of ground targets may exceed the number of plains. If the target is lost it position is subject of uncertainty and has to be updated as soon as possible. We use uniform position uncertainty model – the plains are aware of potential positions of the lost targets and are able to decrease the uncertainty by making the surveillance over such positions.

3.3.2 Scenario A: Homogeneous teams

Four zones are defined in this scenario (see Figure 3.4): the central and largest zone A is the principal area of interest where the actual mission takes place. It is surrounded by four small auxiliary zones B1, B2, B3 and B4 where all currently idle UAVs are located.

There are two types of UAVs involved in this scenario. The first type can only perform surveillance while the other can only perform tracking. Each of the two groups of UAVs need to rely on capabilities of the other group in order to fulfill their mission.

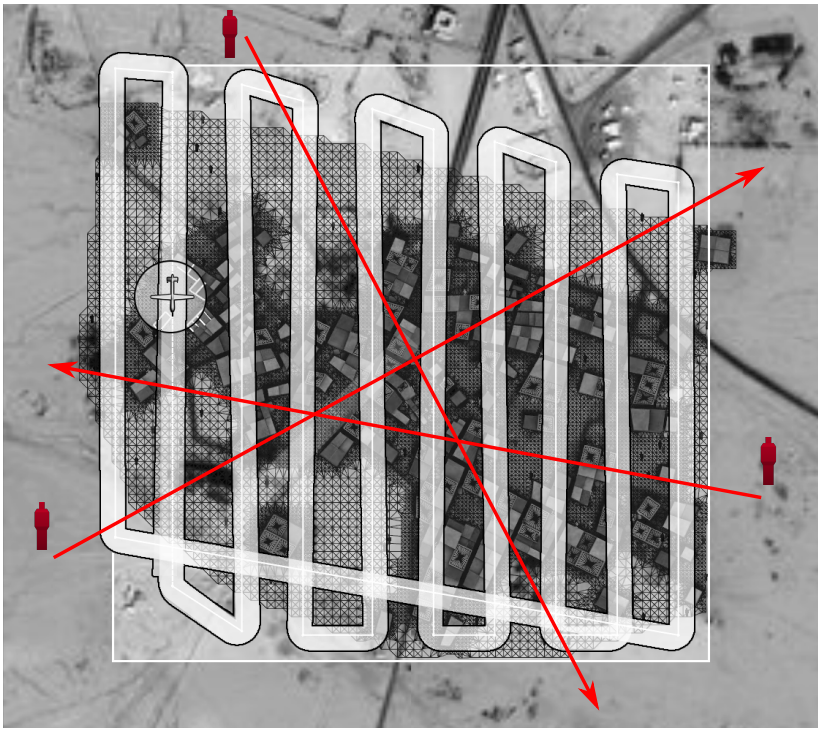

Fig. 3.3 Overview of the demonstration scenarios.

Both surveillance and tracking takes place in zone A. Surveilling UAVs are deployed throughout the mission. They plan their trajectories such that zone A is fully covered in terms of surveillance. Upon noticing an unattended moving target on the ground, the particular surveilling UAV requests the unassigned tracking UAVs to focus on that target. The unassigned tracking UAV (located in zone B1, B2, B3, or B4) closest to the ground target undertakes the given task and performs the tracking in zone A until the target leaves the zone. In such a case the tracking UAV terminates its task and returns to its default zone B1, B2, B3, or B4 where it awaits future tracking tasks.

Two sets of experiments were measured for this scenario. In both cases we investigated how the ratio of numbers of deployed surveilling vs. tracking UAVs influences the values of obtained average information age (area surveillance) and average information error (target tracking) while the total number of UAVs remains the same. In the first set of experiments, the total number of UAVs was 5, in the second it was 10.

The combinations of numbers of surveilling and tracking UAVs for both cases are listed in Table 3.1.

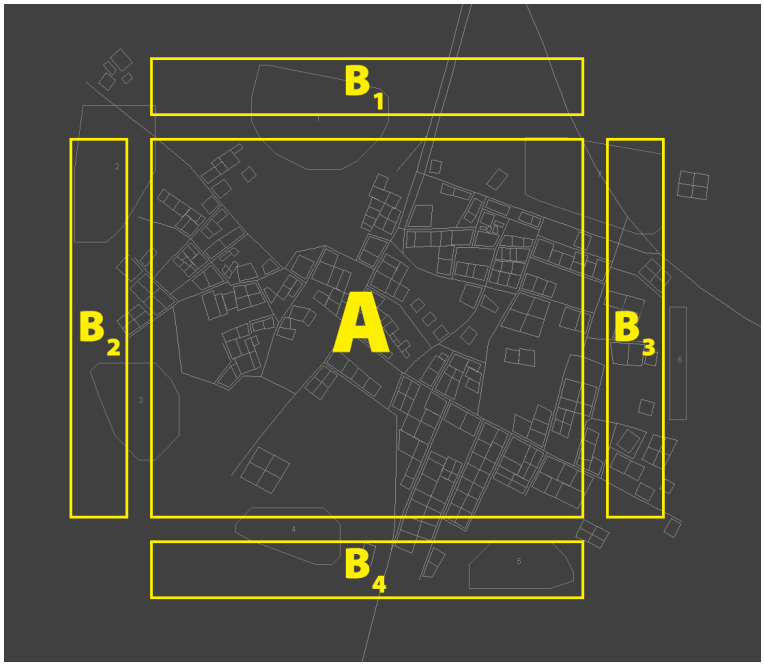

Fig. 3.4 Overview of scenario A – two homogeneous teams of the UAVs.

5 UAVs				
surveillance	4	3	2	1
tracking	1	2	3	4

10 UAVs					
surveillance	9	7	5	3	1
tracking	1	3	5	7	9

Table 3.1 Scenario A UAVs teams settings.

The results of both setups are presented in Figures 3.5, 3.6 and 3.7, 3.8 respectively.

3.3.3 Scenario B: Single team with multiple capabilities

All UAVs involved in this scenario are capable of both surveillance and tracking but they can only perform one of those operations at a time.

There is only one zone defined in this scenario: zone A which is identical to the zone of the same name introduced in Scenario A (see Figure 3.9). All UAVs operate within this zone throughout the whole mission.

All available UAVs start out as surveilling UAVs. As any of them locates an unattended ground target, it requests tracking for that target. In such a case the surveilling UAV closest to the target switches to tracking mode and follows the moving target until it leaves zone A upon which the UAV switches

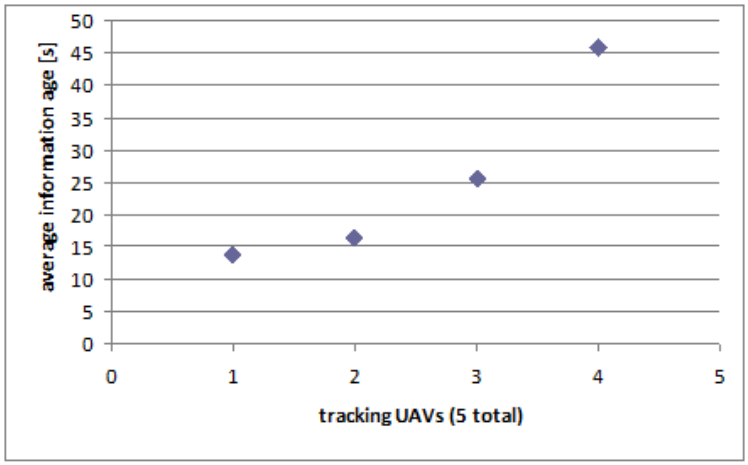

Fig. 3.5 Scenario A (5 UAVs) – average information error.

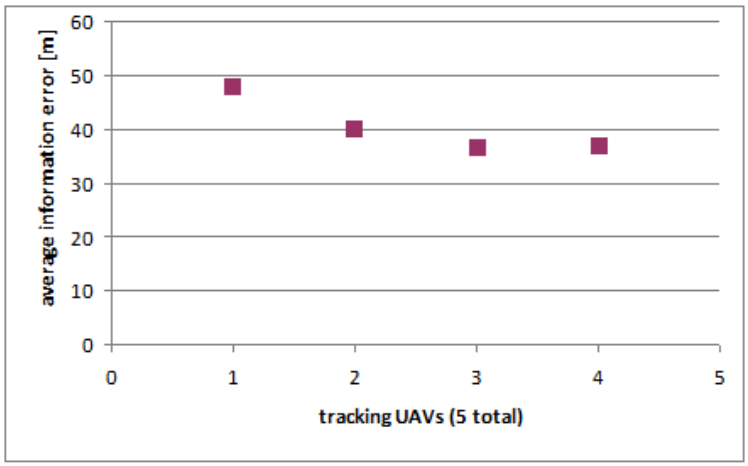

Fig. 3.6 Scenario A (5 UAVs) – average information age.

back to surveillance mode. As the number of surveilling UAVs dynamically changes due to UAVs undertaking tracking tasks, the UAVs participating in surveillance (re)plan their trajectories such that they cover the entire zone A.

Measurements of average information age (area surveillance) and average information error (target tracking) were performed for various numbers of UAVs: 2, 3, 4, 5, 6, 8 and 10, allowing us to evaluate the influence of the total number of UAVs on the quality of both surveillance and tracking of the same area.

The results are presented in Figures 3.10 and 3.11.

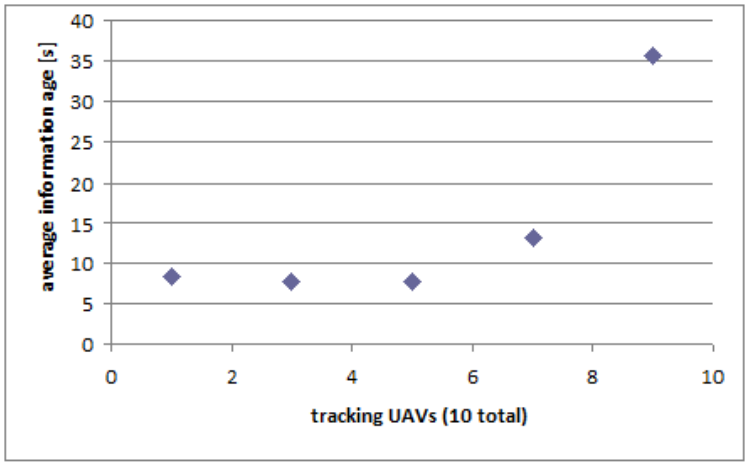

Fig. 3.7 Scenario A (10 UAVs) – average information error.

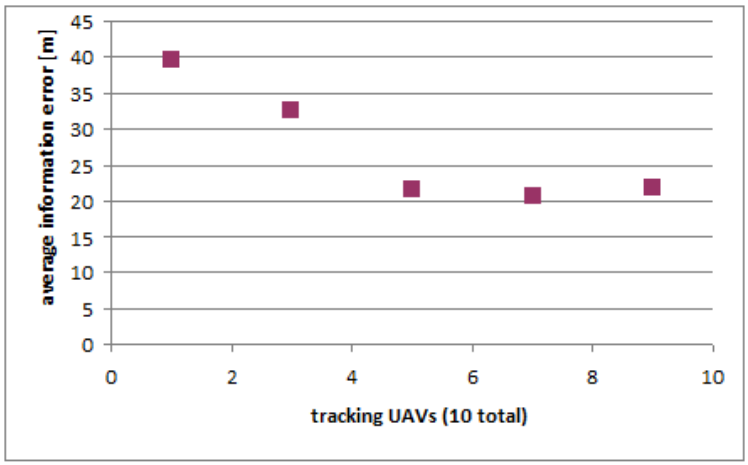

Fig. 3.8 Scenario A (10 UAVs) – average information age.

3.3.4 Scenario C: Heterogeneous team with handover

This mission is derived from Scenario B. All UAVs are capable of both surveillance and tracking and they follow the same logic as in Scenario B. In contrast with Scenario B, zone A is now split into three smaller sub-zones A1, A2 and A3 (see Figure 3.12). Each UAV is assigned to one of these smaller zones and it operates only within that particular zone throughout the mission.

Therefore, when a tracked ground target leaves a particular zone, it is abandoned by the original tracking UAV and is in turn taken over by another

Fig. 3.9 Overview of scenario B – a single heterogeneous team of multiple capabilities UAVs.

tracking UAV from the zone it has newly entered – unless the target moved outside all three zones A1, A2 and A3 in which case it is not tracked anymore.

Measurements of average information age (area surveillance) and average information error (target tracking) were performed for various numbers of UAVs: 3, 4, 6 and 9.

The results are presented in Figures 3.13 and 3.14.

3.3.5 Scenario D: Limited resources tracking

Limited resource tracking scenario has been tested on three algorithms. There is a set of (more or less) cooperating plains and a set of randomly moving ground targets. The number of plains and targets vary from 1 to 10 in the experiments. The algorithms tested provides various levels of planning and coordination and we evaluate their robustness to the changing number of plains and targets. The tested methods are following.

1. *Random walk* – each plane goes randomly through last known positions of the targets. There is no cooperation between planes. See Figure 3.15 for average ground target position information age for various combinations of

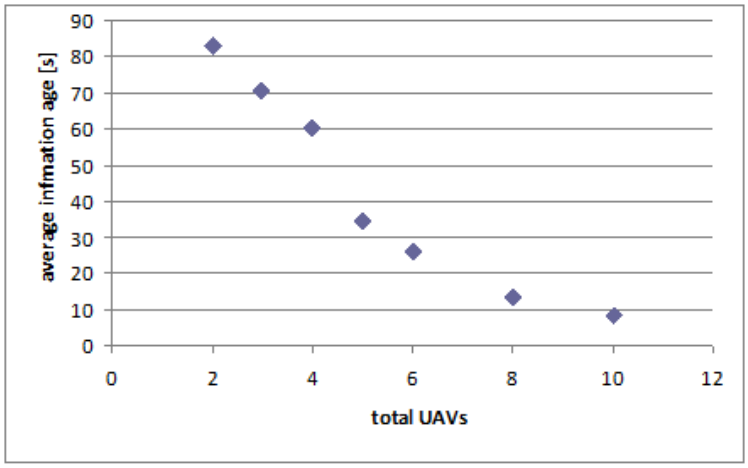

Fig. 3.10 Scenario B – average information error.

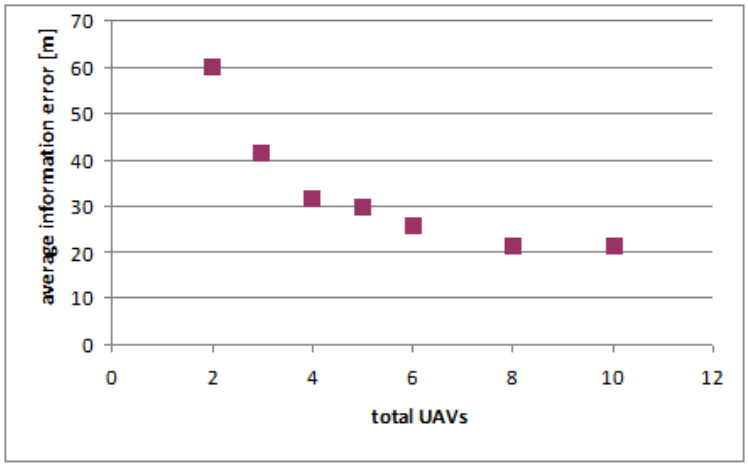

Fig. 3.11 Scenario B – average information age.

number of plains and targets. This method is capable to track 1-2 targets only regardless of number of plains.

2. *Greedy cooperative allocation* – each plane finds closest last known position or position uncertainty area and goes to explore it. The plains use simple voting method to avoid conflicts in selection of areas. It leads good utilization of resources and provides good implicit path conflict avoidance. See Figure 3.16 for average ground target position information age for various combinations of number of plains and targets. This method provide robust n to n tracking with very low communication needs (communication

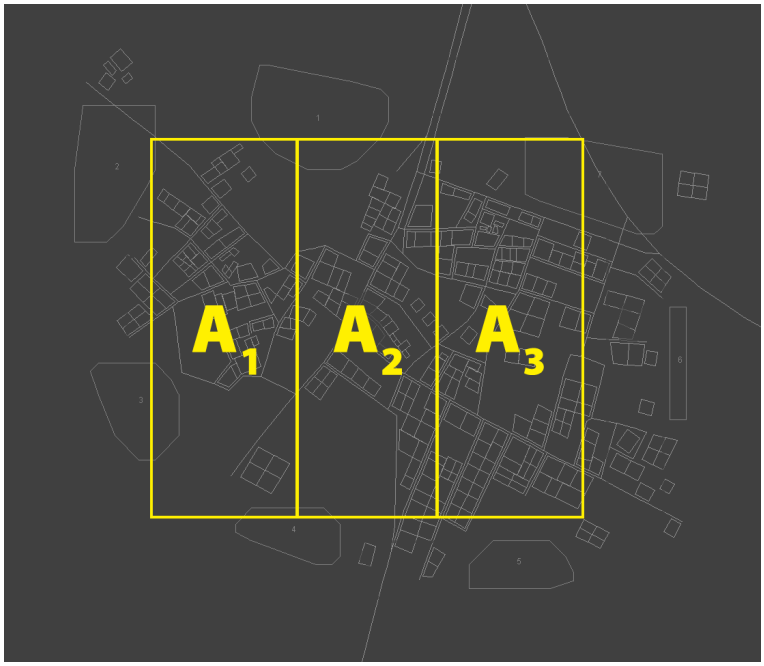

Fig. 3.12 Overview of scenario C – heterogeneous team with handover.

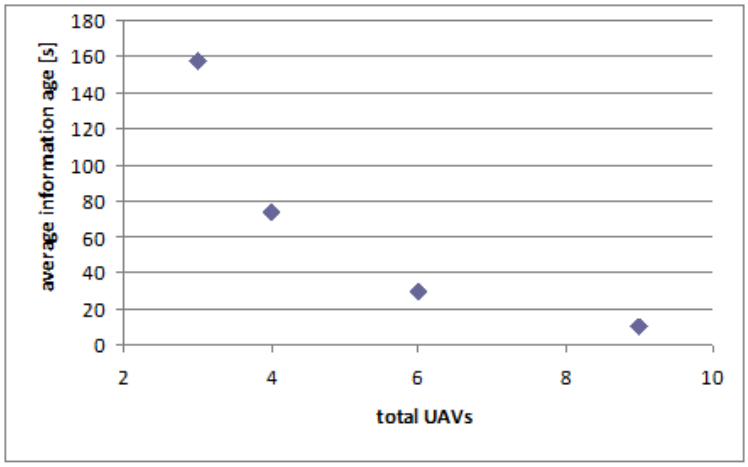

Fig. 3.13 Scenario C – average information error.

is linear to the number of plains). With 8 plains the scenario is saturated and plains are able to track any number of ground targets.

3. *Multi-agent solver allocation* – the plains utilize the solver introduced in Section 3.2.1. The tasks to allocate are last known positions of the ground

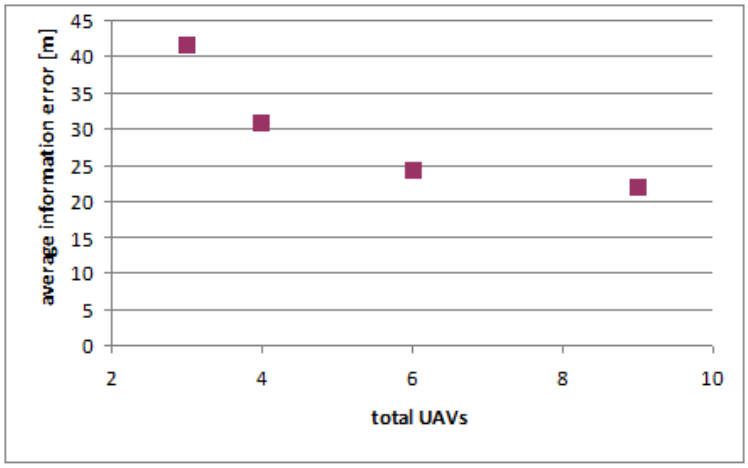

Fig. 3.14 Scenario C – average information age.

targets and position uncertainty areas. Each plane builds local route for allocated tasks using nearest-neighbor traveling salesman heuristics (with quadratic complexity to the number of tasks). Global routes cost minimization is secured by the task allocation algorithm of the solver. This approach provides efficient dynamic task delegation and effective handovers. The complexity of this method grows with the number of target points (i.e. size of area of uncertainty), so in case of high number of lost targets the communication overhead increase. In such case it should be better to change the behaviors of the planes to generic area surveillance. See Figure 3.17 for average ground target position information age for various combinations of number of plains and targets. This method provide good n to $n + k$ tracking (4 plains are able to track 5 targets or 6 plains successfully track 8 targets in our scenario). With 7 plains the scenario is saturated and plains are able to track any number of ground targets.

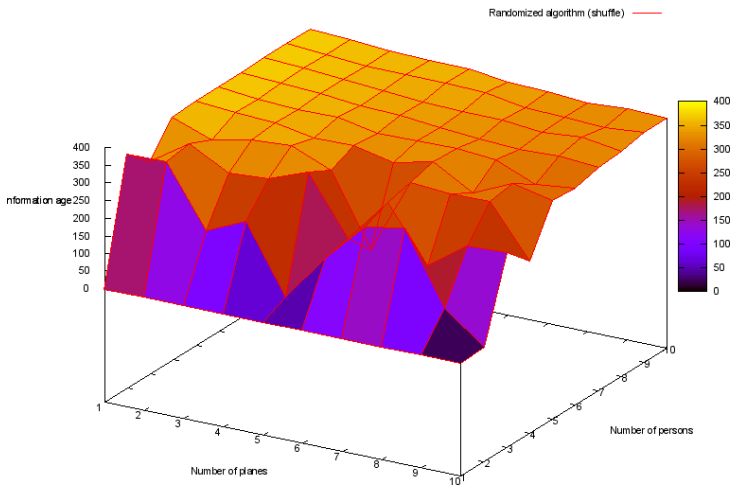

Fig. 3.15 Scenario D – average information age for random method.

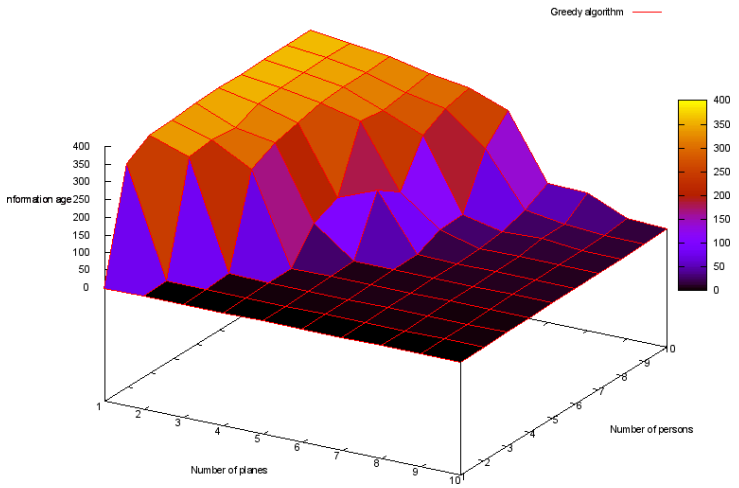

Fig. 3.16 Scenario D – average information age for greedy method.

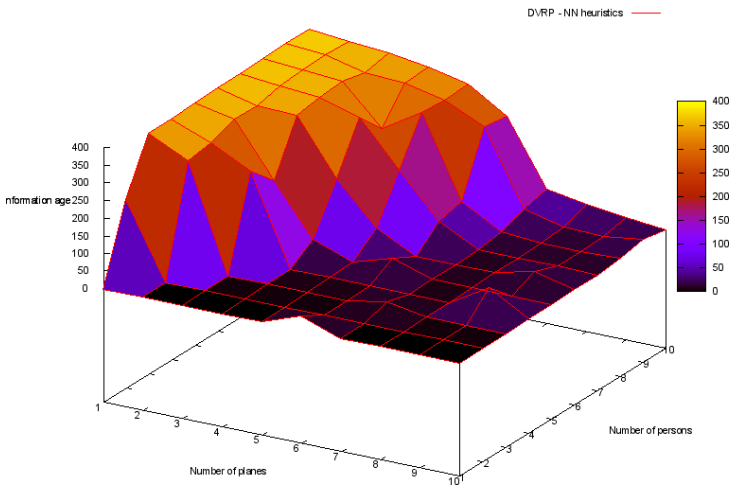

Fig. 3.17 Scenario D – average information age for DVRP.

Chapter 4
Integrated coordination for mixed information collection activities

4.1 Summary of the workpackage

The main objectives of the workpackage *Integrated coordination for mixed information collection activities* were: firstly, to extend our previous work on multi-UAV surveillance and tracking information collection activities and add support for tasks such as exploration and search; secondly, we aimed at investigation of issues arising from execution of mixed/heterogeneous information-collection missions, i.e., such in which a the individual members of a team of UAVs dynamically switch between different information collection tasks during the mission execution; thirdly, we intended to open the problem of mission-centric information collection, i.e., techniques of dynamic (re-)tasking of members of a team of UAVs according to the needs of an on-going ground mission. In the following we summarize the results gained from the work on this workpackage, the findings and the status of the individual objectives w.r.t. the work proposed in the original project proposal.

Support for additional classes of information collection tasks

One of the objectives of this workpackage was to extend the implemented TAF2 scenarios with new types of information collection tasks and investigate the issues arising from them. In particular, our aim was to implement *coordinated multi-UAV area exploration* and *coordinated target search*.

While we implemented and evaluated the *exploration* task into the TAF2 software framework and thus contributed to the integrated demonstration scenario implementation, after an initial problem analysis we concluded that due to its complexity, the task of implementing the coordinated target search lies beyond the scope of this project. In fact, we recognized that a proper

implementation of coordinated target search problem inherently features an adversarial-reasoning element, a topic which is beyond the scope of this project. Instead, the topic of adversarial reasoning and planning became one of the main foci of the related project TACTICAL AGENTSCOUT, already discussed earlier in this report. We discuss the details of our approach to implementation and evaluation of the exploration information collection task below in Section 4.2.

Integrated coordination for mixed information collection

The main objective of this sub-package was to investigate the techniques enabling efficient dynamic task switching and transparent coordination of multi-UAV team performing a mixed-task mission in a theatre. In particular, we aimed at tackling the interdependencies between various information-collection tasks and exploiting them in order to produce an integrated coordination mechanism allowing a switch between surveillance and tracking mode, as well as a seamless handover of target tracking task between different UAVs. In both cases we aimed at improving the overall mission efficiency, i.e., to perform local task switches and target handovers in order to optimize the overall multi-UAV team performance.

The proposed solution is based on integration of techniques of i) interruptible hybrid reactive/deliberative agent behaviours and ii) the universal multi-agent task allocation introduced in the Section **[TO DO]**. Below in this chapter, we discuss the details of the employed approach.

Mission-centric/oriented information collection

Finally, one of the most ambitious challenges proposed to tackle in this work-package was the *integrated coordination of mixed-information collection*. In particular, we aimed at development of methods that take a mission plan as an input and use it for planning and coordinating information collection so that the efficiency of information coverage of the mission is significantly improved.

The analysis of this objective led to two vectors of further investigation. Firstly, we identified a need for an approach/technology enabling highly flexible mission specifications and their subsequent execution according to a precise semantics by the entities involved in the simulation. Besides capturing temporal aspects of missions, such a specification technique should ideally feature a high degree of flexibility w.r.t. the specification granularity and rigidity respecting varying needs of different mission scenarios. Furthermore,

such mission specifications should enable also capturing various contingencies for alternative mission evolutions, should the situation require such changes.

Secondly, we aimed at development of multi-agent planning and re-planning techniques enabling planning for mission-specific information collection support according to the provided mission specification and the contingencies involved therein. This vector of investigation leads to the requirement to perform a full-fledged multi-agent planning taking a mission specification as an input. Again, the problem of efficient multi-agent mission planning is a hot topic of the MAS research, tackling which in a deep manner is beyond the scope of this project.

The problem of a *mission specification language* is still an open challenge of the Artificial Intelligence research community. To tackle this issue, we employed the state-of-the-art techniques from the field of reactive-planning and agent-oriented programming. In particular, our approach is based on the theoretical framework of *Behavioural State Machines* developed by Novák in [6] and further extended in [7]. Concretely, we propose a very basic mission specification language enabling description of missions in terms of ordered sequences of high-level declarative goals which a unit involved in the mission should achieve sequentially. Subsequently, we employed the methodology of agent-oriented design patters [9] which gave us a flexible tool for encoding a range of reactive behaviours which guarantee satisfaction of certain run-time conditions - in this case achievement of the specified high-level declarative goals. We adopted and further extended the BSM-accompanying programming language called *Jazzyk* originally introduced by Novák in [7]. Below, in the Section 4.2 we provide a more detailed account of the BSM framework and the concrete approach to mission specification language and the techniques used for implementation of the simulated entities in the integrated demonstration scenario. The described implementation is to be understood as a prototype design, which we intend to further extend and improve upon in the context of the related project Tactical AgentScout.

Furthermore, while not fully implementing a multi-agent mission planning and re-planning framework, we provide a brief survey of the recent state-of-the-art works of the field. The bulk of the prototyping work in this area was shifted into workpackages of the related Tactical AgentScout project where we will further expand this line of research. Below, in the Section 4.2 we summarize the state of the art in multi-agent planning and report on the advances we did in our studies of multi-agent re-planning and plan-repair. The provided text is a compilation of an extended version of the corresponding sections of the M1 report of the Tactical AgentScout project.

4.2 Technology description

4.2.1 Coordinated multi-UAV exploration

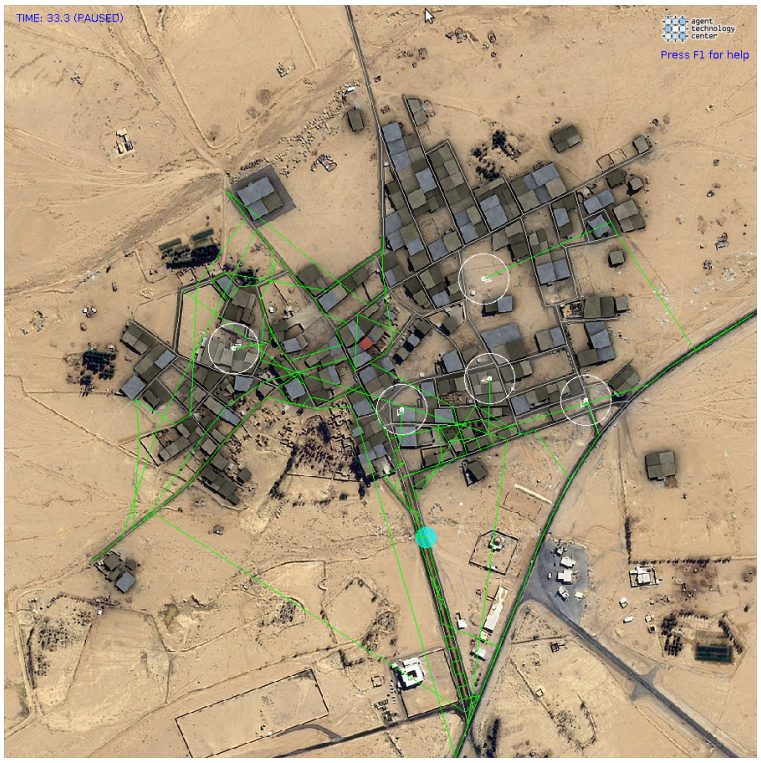

Fig. 4.1 Depiction of a situation during the DVRP-based area exploration by a team of 5 UAVs. The green lines represent the actual plans of the UAVs. The area in the upper right segment of the map was already explored by the team. In consequence the team does not plan to return to this area any more and pays most of its attention to the south-west segment of the map. The plans include all the important points on the map, i.e., all the crossroads as well as straight segments of the map of length over 10 metres. The field of view of a UAV is 50 metres in the simulated scenario.

Our main approach to implementation of the exploration information collection task is based on the utilization of the multi-agent solver introduced is Section 3.2.1. In particular, we use the solver to compute solutions of the Distributed Vehicle Routing Problem. The input is the set of all nodes of interest on the input map and a set of n UAVs. While treating each waypoint request as an independent task, the agents negotiate among themselves and using the heuristics described in the previous chapter divide the task among the team

members. This method is similar to multi-agent solver allocation method used in Scenario D for limited resources tracking (see Section 3.3.1). The surveillance tasks to allocate represents the points of interests on the map. Each plane builds local route for allocated tasks using the same method as in tracking scenario – the nearest-neighbor traveling salesman heuristics. Again, global routes cost minimization is secured by the task allocation algorithm of the solver. The Figure 4.1 depicts a snapshot of an example exploration mission employing the multi-agent solver.

Fig. 4.2 Depiction of a situation during the coordination zig-zag exploration of the target area (the complete map) by a team of 5 UAVs.

In order to perform a thorough evaluation of the proposed exploration method, we utilize the zig-zag algorithm developed in the first phase of this project (cf. the corresponding final report). The implementation is flexible in the number of participating UAVs, which uniformly divide the explored area among themselves and fly in a coordinated formation through it. The Figure 4.2 depicts a snapshot during such an multi-UAV exploration mission.

We discuss the details of the evaluation results in the following section.

4.2.2 Integrated scenario

As a test-bed scenario enabling evaluation of the prototype of the mission-centric information collection techniques, i.e., exploration and simultaneous surveillance and tracking, we developed the following mission specification.

A mission is configure with a number of blue teams, each consisting of a given number of ground troops, deployed in the area of interest, a team of UAVs providing airborne support by performing various information-collection tasks, and a number of insurgents randomly distributed in the area of interest (red force). Furthermore, the blue force knows the GPS coordinates of their corresponding landing points, the map of the urban area and the coordinate of a joint meeting point where the team will be lifted-off at the end of the mission. Finally, the area contains a point of interest called safe house. The task of the ground blue force is to explore the area of interest, find the safe house, approach it and secure its perimeter, and finally, the mission ends with a transport to the meeting lift-off point. The team of UAVs provides airborne support to the blue ground teams performing the area exploration, continuous overhead surveillance of the area in a wider vicinity of the discovered safe-house and most importantly, on request, the UAVs provides overhead tracking support to blue troops pursuing red insurgents throughout the main part of the ground mission, i.e., the approach and securing of the safe house. Finally, the team of UAVs should provide surveillance support over the meeting point during the final phase of the mission.

Exploration phase

The mission starts with a number of blue teams deployed to the area of interest and the team of UAVs taking off the airport outside the area of interest. The objective of the initial phase of the mission is to explore the area of interest and discover the location of the safe house. The safe house is a fixed location on the map, *apriori* unknown to the blue force, nor to the team of UAVs. We assume that the safe house is recognized by the UAV operator (automatic detection mechanism) continuously analysing the image stream from the team of UAVs. The red insurgents are randomly scattered in the urban area of interest, with the highest density around the safe house (about one third of all the red troops in the actual implemented simulation scenarios).

Since the location of the safe house is initially unknown to the ground blue force, the teams secure their deployment locations and wait for the broadcast signal sent by the UAVs upon discovery of the safe house. The UAVs actually employ the DVRP-based exploration already introduced above in Subsection 4.2.1. The Figure 4.3 depicts a snapshot of a situation during the exploration phase of the mission. The example mission involves two blue

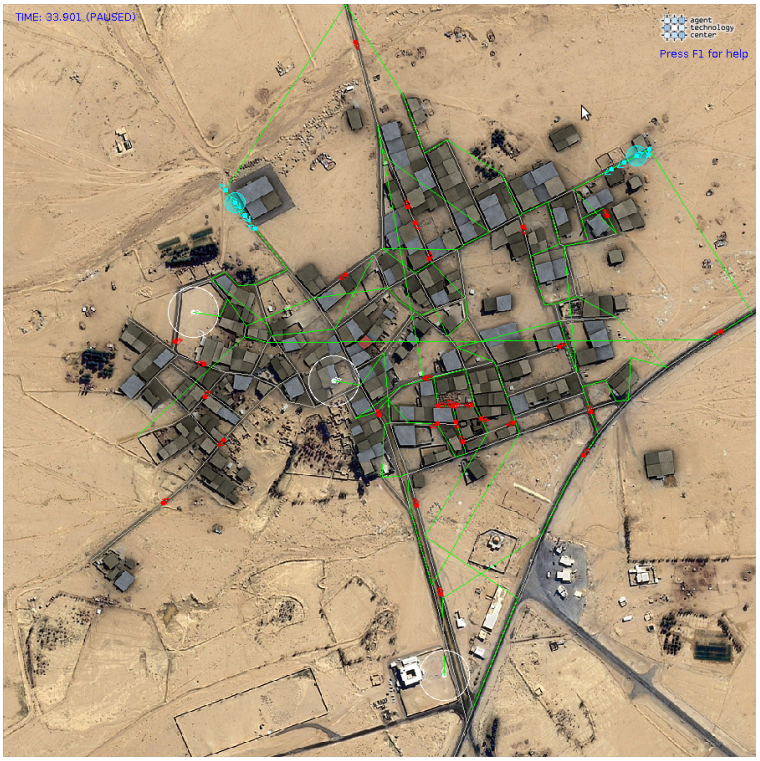

Fig. 4.3 Snapshot of a situation during the initial exploration phase of the mission. The blue teams are waiting for the safe-house discovery, while the team of UAVs explores the area of interest in a coordinated fashion. The green lines depict the actual plans of the UAVs, initially covering all the important nodes on the map (crossroads, etc.).

teams deployed at location in the north-west and north-east of the example map and a team of 3 UAVs performing the area exploration. The blue teams consists of 10 ground troops each.

Support of an on-going ground mission

The end of the exploration phase and a transition to a mode of active support of the ground mission by the team of UAVs is marked by the discovery of the safe house location. The Figure 4.4 depicts exactly this moment in an example mission.

The main objective of the blue force during the middle phase of the ground mission is to reach the safe house, secure the are and perform short, further unspecified operation in the immediate vicinity of the safe house. Furthermore, upon encounter, the blue troops engage the red troops, track them on

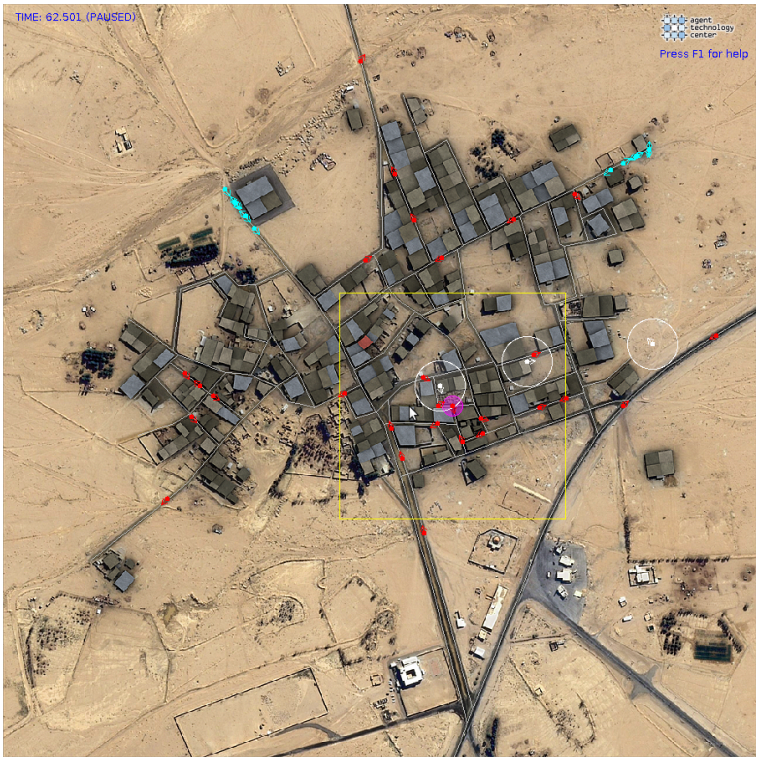

Fig. 4.4 At the end of the exploration phase, the safe house is found by one of the UAVs. The discovery of the safe house is a trigger for the blue force to starts its approach towards the house. On an encounter, during the safe-house approach, the blue troops track and attempt to capture red insurgents. Whenever such a pursuit is started, the blue troops involved call for airborne support by the autonomous team of UAVs. The yellow square depicts the surveillance area around the safe house. The UAVs fly in a zig-zag formation over the area unless providing a tracking support to the blue force.

the map and finally capture them (note for illustration purposes, that blue troops are moving about 5% faster than the red troops). During such pursuits, the team of UAVs provides on-request tracking support to the involved blue forces. In fact. during this mission phase, the team of UAVs aims to continuously perform surveillance of the area in a wider vicinity of the safe house and some of the team members switch to tracking mode only upon request from the blue force.

The surveillance of the area is implemented using the already described zig-zag algorithm implemented in the first phase of the Tactical AgentFly project.

The concrete implementation of the tracking support by the team of UAVs utilizes the DVRP-based technique of tracking introduced in the previous chapter. I.e., the blue force creates a new task in the multi-agent task al-

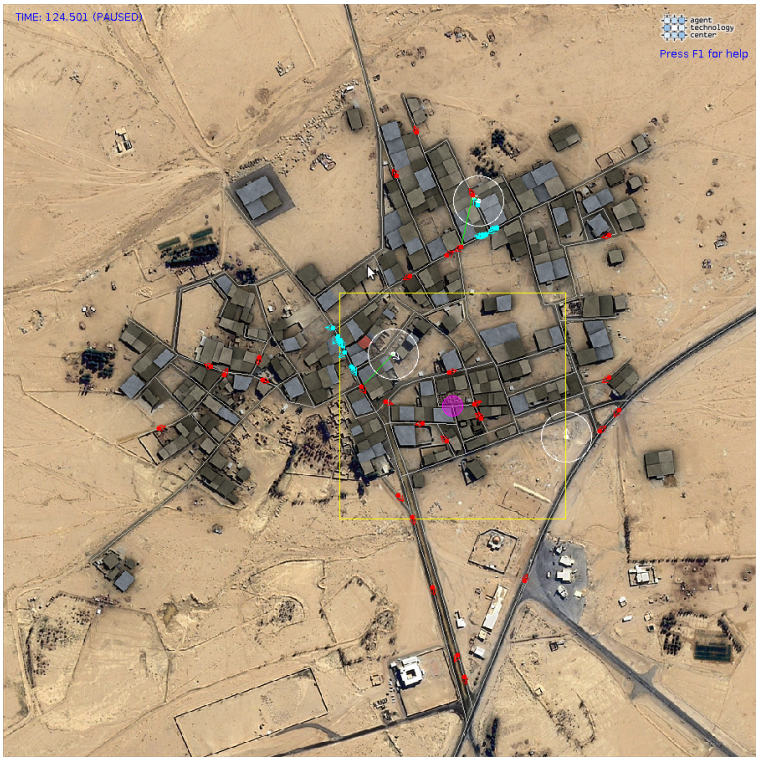

Fig. 4.5 Depiction of a situation during the running mission. The two blue force teams are approaching the safe-house location, while the UAVs provide airborne information collection support. The screen-shot depicts a situation in which two UAVs provide tracking support to the blue force (the green lines depict the tracking target _, while the third UAV performs zig-zag surveillance of the area around the safe house.

location solver what subsequently leads to negotiation and re-negotiation of the set of current tasks among the UAV team members. The task allocation algorithm finally allocates the task to the UAV which can fulfill the request in the most efficient manner w.r.t. the overall team performance. The remaining UAVs continue to perform the surveillance task of the area. Note that in situations when only few UAVs are involved in the ground mission support and when the blue force pursues a number of targets simultaneously (i.e., information collection under constrained resources), the task allocation leads to "overtasking" of some of the UAVs in the fashion described and evaluated in the previous chapter. I.e, a single UAV receives a number of tracking targets which it is supposed to serve simultaneously. The Figure 4.7 depicts such a situation during the mid-phase of the ground mission.

Upon successful capture of a red insurgent, the tracking request is cancelled and the involved UAV continues in whatever tasks it is engaged in: either returns to the surveillance of the safe-house area, or switches to tracking of

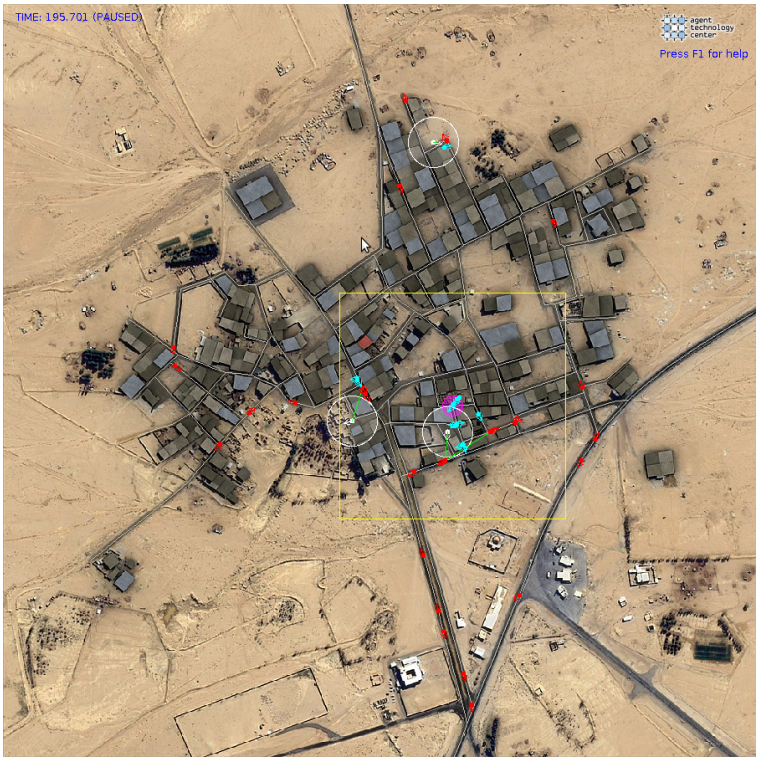

Fig. 4.6 Another situation during during the running mission. All the UAVs provide tracking support to the blue force which just arrives to the safe-house location. Some of the blue troops are still engaged in tracking and capturing the red insurgents. Note, the UAV nearest the safe house provides tracking support to two targets simultaneously. The green lines depict the current plans of the UAVs in terms of the next waypoints.

the still active targets allocated to it. In fact, in every moment, the multi-UAV team renegotiates the task allocation among the team members. This often leads to re-allocation of tasks, i.e., tracking target handover from one UAV to another one, or switching between surveillance and tracking between several UAVs while continuously providing tracking of all the requested targets. This emergent behaviour is a result of the dynamic re-allocation of the requested tasks according to the local conditions and the actual mission evolution.

Meeting at the lift-off location

Upon reaching the safe house, the blue teams secure the area in the immediate vicinity of the target location. They still actively engage the encountered red troops what eventually leads to clearing and securing the neighbourhood. After a fixed time spent at the target location, the mission switches to its

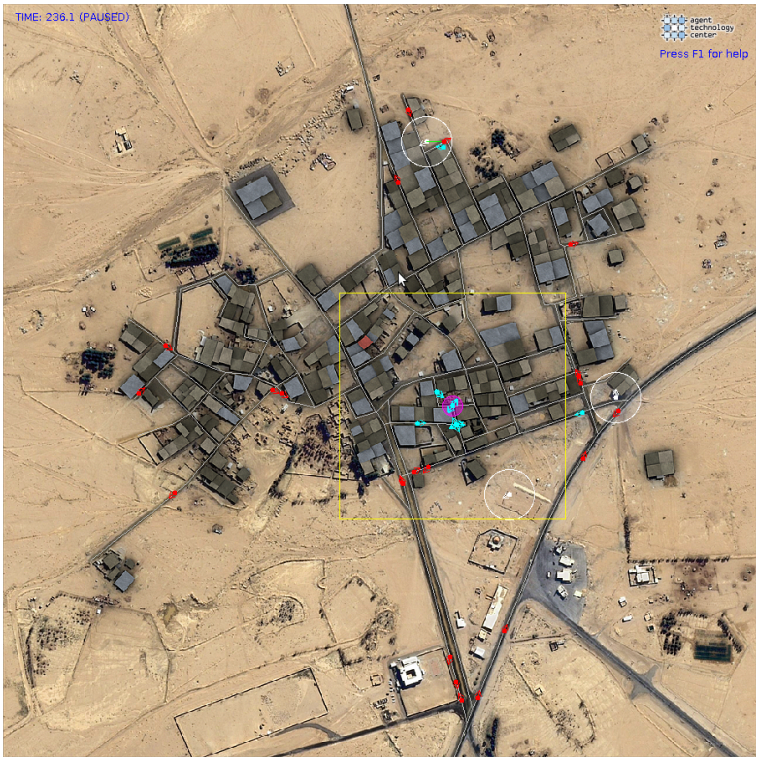

Fig. 4.7 The blue force captures the safe house. The are in the immediate vicinity of the safe house is clear of red insurgents and secured by the blue force in a guard formation around the location. One of the UAVs is still engaged in tracking support to blue troops. Note the left-most UAV which was just disengaged from a tracking support for a blue troop and is returning to the area of interest to join the last UAV team member and perform the surveillance task.

final phase. The blue forces disengage all the, still pursued, red insurgents, leave the safe house area and head towards a meeting point where they are supposed to be lifted off after the mission completes. Due to the red troops disengagement, the UAVs stop all the tracking tasks and head towards a predefined airport location outside the area of interest. The blue team however requests continuous surveillance of the meeting point. Usually, this task is allocated to a single UAV which finally continuously circles the target location. The allocation is again handled by the already discussed multi-agent task allocation algorithm. So, in fact, the UAV team member finally serving the task is not pre-determined, nor is it fixed until the end of the mission. However, the algorithm usually leads to an optimal allocation of a single UAV to the task. The Figure 4.8 depicts the approach of the blue force to the meeting point and the single-UAV surveillance of the meeting point area. Finally, the

Figure 4.9 depicts the situation at the very end of the mission where all the blue troops are collected at the meeting point.

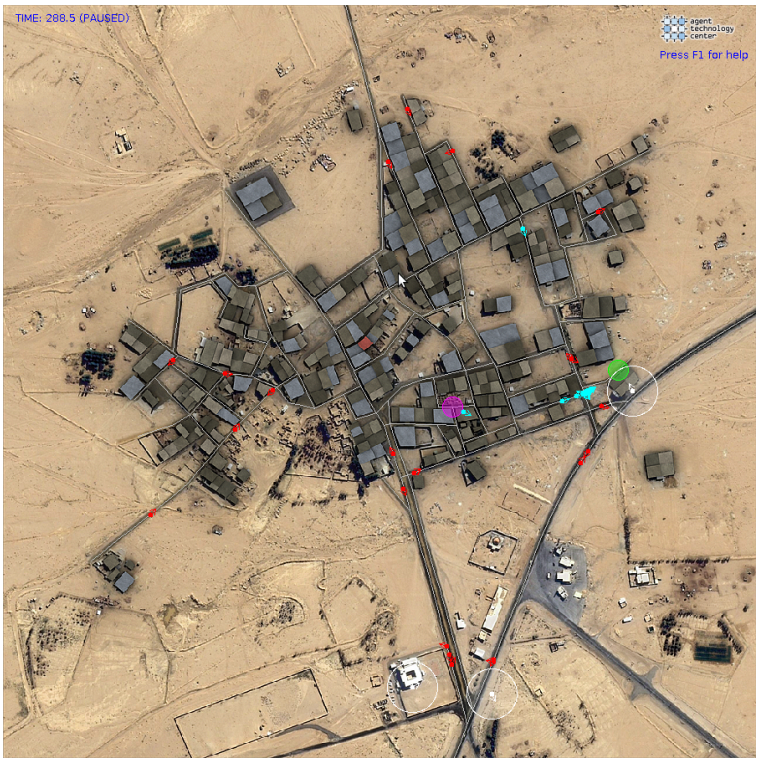

Fig. 4.8 In the last phase of the mission, the blue force disengages the operation around the safe house and heads towards the meeting point with the lift-off (depicted as the green circle). The troops are independently moving to the meeting point, while the UAVs leave the theatre and head towards an airport. The blue force asks for a surveillance support above the meeting point provided by a single UAV.

4.2.3 Mission specification

The above describe mission is just a single instance of a possible mission implementation. In fact, one of the objectives of this workpackage was to develop a technology allowing us to specify and subsequently execute a large range of mission scenarios composed of a number of basic building blocks, such as *exploration*, *surveillance*, *tracking* on the side of the UAVs, as well as behaviours including e.g., *securing an area*, *movement to a pre-defined*

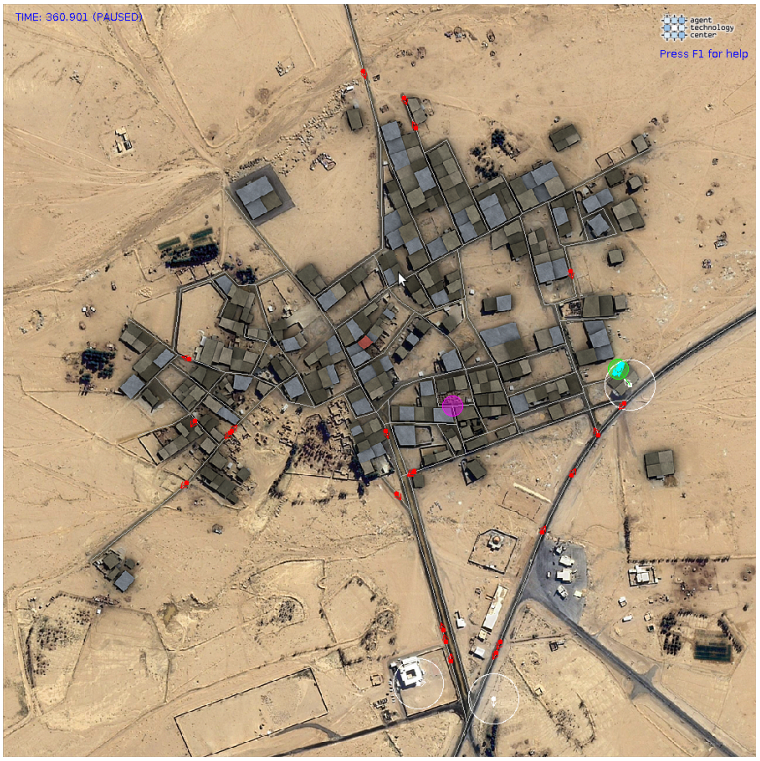

Fig. 4.9 The end of the mission. All the blue troops are in the collection point continuously guarded by a single UAV, while the remaining UAV team members left the theatre.

location, *tracking* or *evasion* on the side of ground forces, be it blue, or red troops.

While a thorough effort towards a general-purpose mission specification language is beyond the scope of this project, we designed and implemented a basic framework allowing us to encode basic sequential specifications of missions as sequences of higher-level behaviours/tasks the agents and agent teams should perform. In particular, the mission in our implementation is specified as a Prolog-style list of, possibly parametrized, achievement goal specifications, the agents are supposed to be pursue during the mission execution. In the scenario implementation, the goals are associated with a composable reactive behaviours implemented as a reactive planning policies (cf. the next subsection). The parameters of the goal symbols are bound to information stored in the agent's belief base.

Example 1. One of the most complex entities involved in the above described simulated mission are the blue force troops. The following is an example of the actual mission specification of a single blue team agent:

[

```
        wait_for_target,
        move_to(target_safe_house),
        cover_position,
        final([move_to(collection_point), cover_position])
]
```

The behaviour of blue force team agent is thus specified to first wait until the target location of the safe house is broadcast by the team of supporting UAVs, subsequently move to the safe house, secure the target location and when the mission turns into the final phase move to the meeting point and secure the final target location.

Similarly, the mission specification of red insurgents reads either

```
[ settle ]
```

or alternatively

```
[ hang_around ]
```

The first mission specification is for the insurgents located in the immediate vicinity of the safe house, and the latter to the agents located randomly on the map.

The individual behaviours associated with the goals are implemented as *Jazzyk* reactive policies described later in this chapter. The goal parameters, such as the position of the collection_point, or target_safe_house in the case of the blue force agent are bound to an information stored in the agent's belief base. The information about the latter is only broadcasted by the supporting team of UAVs, so the agent learns the position in run-time.

The above is an example of a possible mission execution. In fact, the mission specification for each simulated entity, be it a ground agent, or a UAV can be a sequential composition of a selection from the range of goals the agent in question is able to achieve. Below we describe how the agents deal with behaviours which are seen as interruptions to the specified goals, such as red insurgent pursuit by a blue troop, or fleeing by a red insurgent. These are implemented as implicit behaviours of the individual agents and do not need to be specified in the mission description.

4.2.4 Reactive planning as a simulated mission execution framework

The simple mission specification language described above provides an input to the simulated agents which in run-time attempt to execute the specified mission taking into account their own, inherent, behaviour specifications. As an example of such, take a blue agent which should opportunistically, during any mission phase, start a pursuit of an encountered red insurgent.

Taking into consideration our past experience with agent-oriented programming, we choose the state-of-the-art techniques from the field of reactive planning and agent-oriented programming languages as the starting

point for our implementation of the flexible mission specification and execution framework. In particular, our design of the fully configurable multi-agent mission simulation framework is based on the *Belief-Desire-Intention* (BDI) agent architecture. The BDI architecture dictates a decomposition of agent's knowledge bases into several components storing information the agent has about its environment, itself and its team peers (beliefs), the information about its goals (in this case, basically the mission specification together with the agent's internally adopted goals) and a representation of the agents plans, or programs, execution of which ensures establishment of the agent's goals, i.e., the agent's beliefs match its goals in that the agent believes that it had already achieved the goals eventually in the future.

Concretely, our approach to the design of the flexible mission execution framework is based on the framework of *Behavioural State Machines* by Novák **[7, 10]** and its associated agent-oriented programming language *Jazzyk*. Using this framework, we implemented a modular BDI architecture, in the was similar to that described by Novák and Dix in [8]. In the following, we first briefly introduce the framework of *Behavioural State Machines* and subsequently describe out adaptation and extension of the original *Jazzyk* interpreter, which we developed for the purposed of this project.

Behavioural State Machines

In [7], Novák introduced the framework of *Behavioural State Machines* (*BSM*). *BSM* framework is a reactive-planning approach to programming cognitive agents based on the *Belief-Desire-Intention* hybrid architecture. The *BSM* framework draws a clear distinction between the *knowledge representation* and *behavioural* layers within an agent. It thus provides a programming system that clearly separates the programming concerns of *how to represent an agent's knowledge* about, for example, its environment and *how to encode its behaviours*. In the core of the framework is a *generic reactive computational model* inspired by Gurevich's *Abstract State Machines* [1], enabling for efficient structuring of the program code. This section briefly introduces the *BSM* framework. Below, we introduce an extension of the *BSM* framework which we developed for the purposes of this project. For the complete formal description of the *BSM* framework, see [7].

Syntax

BSM agents are collections of one or more so-called *knowledge representation modules* (KR modules), typically denoted by $\mathcal{M}$, each representing a part of the agent's knowledge base. KR modules may be used to represent and maintain various mental attitudes of an agent, such as knowledge about its environment, or its goals, intentions, obligations, etc. Transitions between

states of a *BSM* result from applying so-called *mental state transformers* (*mst*), typically denoted by τ. Various types of mst's determine the behaviour that an agent can generate. A *BSM agent* consists of a set of KR modules $\mathcal{M}_1, \ldots, \mathcal{M}_n$ and a mental state transformer $\mathcal{P}$, i.e. $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$; the mst $\mathcal{P}$ is also called an *agent program*.

The notion of a KR module is an abstraction of a partial knowledge base of an agent. In turn, its states are to be treated as theories (i.e. sets of sentences) expressed in the KR language of the module. Formally, a KR module $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i)$ is characterized by a knowledge representation language $\mathcal{L}_i$, a set of states $\mathcal{S}_i \subseteq 2^{\mathcal{L}_i}$, a set of query operators $\mathcal{Q}_i$ and a set of update operators $\mathcal{U}_i$. A query operator $\models \in \mathcal{Q}_i$ is a mapping $\models : \mathcal{S}_i \times \mathcal{L}_i \rightarrow \{\top, \bot\}$. Similarly an update operator $\oplus \in \mathcal{U}_i$ is a mapping $\oplus : \mathcal{S}_i \times \mathcal{L}_i \rightarrow \mathcal{S}_i$.

Queries, typically denoted by φ, can be seen as operators of type $\models : \mathcal{S}_i \rightarrow \{\top, \bot\}$. A primitive query $\varphi = (\models\phi)$ consists of a query operator $\models \in \mathcal{Q}_i$ and a formula $\phi \in \mathcal{L}_i$ of the same KR module $\mathcal{M}_i$. Complex queries can be composed by means of conjunction $\wedge$, disjunction $\vee$ and negation $\neg$.

Mental state transformers enable transitions from one state to another. A primitive mst $\oslash\psi$, typically denoted by ρ and constructed from an update operator $\oslash \in \mathcal{U}_i$ and a formula $\psi \in \mathcal{L}_i$, refers to an update on the state of the corresponding KR module. Conditional mst's are of the form $\varphi \longrightarrow \tau$, where φ is a query and τ is a mst. Such a conditional mst makes the application of τ depend on the evaluation of φ. Syntactic constructs for combining mst's are: non-deterministic choice $|$ and sequence $\circ$.

Definition 1 (mental state transformer). Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be KR modules of the form $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i)$. The set of *mental state transformers* is defined as below:

- **skip** is a *primitive* mst,
  - if $\oslash \in \mathcal{U}_i$ and $\psi \in \mathcal{L}_i$, then $\oslash\psi$ is a *primitive* mst,
  - if φ is a query, and τ is a mst, then $\varphi \longrightarrow \tau$ is a *conditional* mst,
  - if τ and τ' are mst's, then $\tau|\tau'$ and $\tau \circ \tau'$ are mst's (*choice*, and *sequence* respectively).

Even though it is a vital feature of the *BSM* theoretical framework, for simplicity we omit the treatment of variables in the definitions of query and update formulae above. For a full fledged description of the BSM framework consult [7].

Semantics

The *yields* calculus, summarised below after [7], specifies an update associated with executing a mental state transformer in a single step of the language interpreter. It formally defines the meaning of the state transformation induced by executing an mst in a state, i.e. a mental state transition.

Formally, a *mental state* σ of a *BSM* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \tau)$ is a tuple $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ of KR module states $\sigma_1 \in \mathcal{S}_1, \ldots, \sigma_n \in \mathcal{S}_n$, corresponding to $\mathcal{M}_1, \ldots, \mathcal{M}_n$ respectively. $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ denotes the space of all mental states over $\mathcal{A}$. A mental state can be modified by applying primitive mst's on it and query formulae can be evaluated against it. The semantic notion of truth of a query is defined through the satisfaction relation $\models$. A primitive query $\models\phi$ holds in a mental state $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ (written $\sigma \models (\models\phi)$) iff $\models(\phi, \sigma_i)$, otherwise we have $\sigma \not\models (\models\phi)$. Given the usual meaning of Boolean operators, it is straightforward to extend the query evaluation to compound query formulae. Note that evaluation of a query does not change the mental state σ.

For an mst $\oslash\psi$, we use $(\oslash, \psi)$ to denote its semantic counterpart, i.e., the corresponding *update* (state transformation). Sequential application of updates is denoted by $\bullet$, i.e. $\rho_1 \bullet \rho_2$ is an update resulting from applying ρ_1 first and then applying ρ_2. The application of an update to a mental state is defined formally below.

Definition 2 (applying an update). The result of applying an update $\rho = (\oslash, \psi)$ to a state $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ of a *BSM* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$, denoted by $s \bigoplus \rho$, is a new state $\sigma' = \langle \sigma_1, \ldots, \sigma'_i, \ldots, \sigma_n \rangle$, where $\sigma'_i = \sigma_i \oslash \psi$ and σ_i, $\oslash$, and ψ correspond to one and the same $\mathcal{M}_i$ of $\mathcal{A}$. Applying the empty update **skip** on the state σ does not change the state, i.e. $\sigma \bigoplus \mathbf{skip} = \sigma$.

Inductively, the result of applying a sequence of updates $\rho_1 \bullet \rho_2$ is a new state $\sigma'' = \sigma' \bigoplus \rho_2$, where $\sigma' = \sigma \bigoplus \rho_1$. $\sigma \overset{\rho_1 \bullet \rho_2}{\rightarrow} \sigma'' = \sigma \overset{\rho_1}{\rightarrow} \sigma' \overset{\rho_2}{\rightarrow} \sigma''$ denotes the corresponding compound transition.

The meaning of a mental state transformer in state σ, formally defined by the *yields* predicate below, is the update set it yields in that mental state.

Definition 3 (yields calculus). A mental state transformer τ yields an *update* ρ in a state σ, iff $yields(\tau, \sigma, \rho)$ is derivable in the following calculus:

$$\frac{\top}{yields(\mathbf{skip}, \sigma, \mathbf{skip})} \qquad \frac{\top}{yields(\oslash\psi, \sigma, (\oslash, \psi))} \quad \text{(primitive)}$$

$$\frac{yields(\tau, \sigma, \rho), \sigma \models \phi}{yields(\phi \longrightarrow \tau, \sigma, \rho)} \qquad \frac{yields(\tau, \sigma, \rho), \sigma \not\models \phi}{yields(\phi \longrightarrow \tau, \sigma, \mathbf{skip})} \quad \text{(conditional)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1),\ yields(\tau_2, \sigma, \rho_2)}{yields(\tau_1 | \tau_2, \sigma, \rho_1),\ yields(\tau_1 | \tau_2, \sigma, \rho_2)} \quad \text{(choice)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1),\ yields(\tau_2, \sigma \bigoplus \rho_1, \rho_2)}{yields(\tau_1 \circ \tau_2, \sigma, \rho_1 \bullet \rho_2)} \quad \text{(sequence)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1),\ \rho_2 \neq \mathbf{skip}}{yields(\tau_1 / \tau_2, \sigma, \rho_1)} \quad \text{(chain preference)}$$

$$\frac{\forall \rho_1 \colon yields(\tau_1, \sigma, \rho_1) \wedge \rho_1 = \mathbf{skip},\ yields(\tau_1, \sigma, \rho_2)}{yields(\tau_1 / \tau_2, \sigma, \rho_2)} \quad \text{(chain preference)}$$

We say that τ yields an *update set* ν in a state σ iff $\nu = \{\rho | yields(\tau, \sigma, \rho)\}$.

The mst **skip** yields the update **skip**. Similarly, a primitive update mst $\oslash\psi$ yields the corresponding update $(\oslash, \psi)$. In the case the condition ϕ of a conditional mst $\phi \longrightarrow \tau$ is satisfied in the current mental state, the calculus yields one of the updates corresponding to the right hand side mst τ, otherwise the no-operation **skip** update is yielded. A non-deterministic choice mst yields an update corresponding to either of its members and a sequential mst yields a sequence of updates corresponding to the first mst of the sequence and an update yielded by the second member of the sequence in a state resulting from application of the first update to the current mental state. Finally, not appearing in the original *BSM* framework by Novák, we added the chain preference operator / which executes only the first, non-empty (**skip**) update yielded in the sequence of mst's.

The following definition articulates the denotational semantics of the notion of mental state transformer as an encoding of a function mapping mental states of a *BSM* to updates, i.e. transitions between them.

Definition 4 (mst functional semantics). Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be KR modules. A mental state transformer τ encodes a function $\mathfrak{f}_\tau : \sigma \mapsto \{\rho | yields(\tau, \sigma, \rho)\}$ over the space of mental states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle \in S_1 \times \cdots S_n$.

Subsequently, the semantics of a *BSM* agent is defined as a set of traces in the induced transition system enabled by the *BSM* agent program.

Definition 5 (BSM semantics). A *BSM* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ can make a step from state σ to a state σ', iff $\sigma' = \sigma \bigoplus \rho$, s.t. $\rho \in \mathfrak{f}_\mathcal{P}(\sigma)$. We also say, that $\mathcal{A}$ induces a (possibly compound) transition $\sigma \xrightarrow{\rho} \sigma'$.

A possibly infinite sequence of states $\sigma_1, \ldots, \sigma_i, \ldots$ is a run of BSM $\mathcal{A}$, iff for each $i \geq 1$, $\mathcal{A}$ induces a transition $\sigma_i \rightarrow \sigma_{i+1}$.

The semantics of an agent system characterized by a BSM $\mathcal{A}$, is a set of all runs of $\mathcal{A}$.

Additionally, we require the non-deterministic choice of a *BSM* interpreter to fulfil the *weak fairness condition*, similar to that in [5], for all the induced runs.

Condition 1 (weak fairness condition) *A computation run is weakly fair iff it is not the case that an update is always yielded from some point in time on but is never selected for execution.*

Jazzyk

Jazzyk is a programming language implementing the computational model of the *BSM* framework. Originally introduced in [7], the language comes with a standalone interpreter which can transparently incorporate a number of heterogeneous knowledge representation modules ranging from logic-programming-based, object-oriented programming language interpreters to

interfaces to robotic simulators. However, since the *A-Globe/AgentFly* technological infrastructure is based on the *Java Virtual Machine* run-time environment, the original language interpreter was not a suitable option for use in the context of this project. For the purposes of this project, we ported the *Jazzyk* programming language to the *Java* platform and developed a new incarnation of the *Jazzyk* interpreter capable of running in the *Java Virtual Machine*.The implemented full-fledged language interpreter allowed us to implement the behaviours of agents in our MAS simulation in a highly modular and flexible manner. The heterogeneity of the *BSM/Jazzyk* suite allowed us to exploit the strengths of declarative technologies, such as *Prolog* together with object-oriented technologies, i.e., *Java* language as a natural components of a BDI-based architecture for behaviour-rich , heavily deliberating, yet responsive and reactive agents in our simulated missions.

Language

The original syntax of the *Jazzyk* language is an instantiation of the abstract mathematical syntax of the *BSM* theoretical framework. `when then` construct encodes a conditional $mst\ \phi \longrightarrow \tau$. Symbols ; and , stand for choice | and sequence $\circ$ *BSM* operators respectively. To facilitate operator precedence, mental state transformers can be grouped into compound structures, blocks, using curly braces {...}. Since the original *BSM* framework did not include the *unless* operator /, we additionally implemented it in our extension.

Interpreter

Our implementation of the Jazzyk programming language, called *JazzykJVM*, was developed in a modern functional-object-oriented programming language *Scala*[1] which compiles into *Java* byte code, in result featuring a tight and transparent integration with the *Java* infrastructure.

Together with the *Jazzyk* programming language interpreter, we implemented two general-purpose knowledge representation modules: *Prolog* module and *Java* module. As the basis for the *Prolog* module accessible from the *Java* platform, we used the *tuProlog* v2.1.1[2] developed by *Department of Electronics, Informatics and Systems* of *Alma Mater Studiorum-Università di Bologna*, Italy. The *Prolog* module allowed us to exploit the conciseness and declarative language strengths of logic-programming. The *Java* module facilitating object-oriented is based on a *Java* scripting language interpreter

[1] `http://www.scala-lang.org/`

[2] `http://tuprolog.alice.unibo.it/`

BeanShell[3]. We describe the concrete usage of the modules within our implementation later in this chapter.

To better support source code modularity and re-usability, *Jazzyk* interpreter integrates a *Java* implementation of the popular *GNU M4*[4], a state-of-the-art macro preprocessor. The concrete *M4 Java* port we used is the *MNI Macro Processor (MMP)* package developed by Burkhardt Renz at *University of Applied Sciences*, Gießen-Friedberg, Germany.

Macros are a powerful tool for structuring and modularizing and encapsulating the source code and writing code templates. Before feeding the *Jazzyk* agent program to the language interpreter, first all the M4 macros are expanded and only afterwards the plain *Jazzyk* program is fed to the interpreter for execution.

4.2.5 Mission specification execution

As already indicated in the previous subsections above, the framework facilitating flexible mission specification and execution is heavily based on exploitation of the strengths of the framework of *Behavioural State Machines* and the associated programming language *Jazzyk*. In the following, we briefly describe the agent architecture developed for the purposes of the project and the details of mission-specific agent behaviour implementation. For illustration purposes, we only provide a description of the implementation of ground blue-force agents whose implementation features the richest range of behaviours.

Agent architecture

Above, we already noted that the architectural decomposition of the simulated entities is heavily inspired by the BDI architectural scheme. In particular, the agents' belief base is further decomposed into two sub-modules: a *Prolog*-based component storing the agent's information about its environment and itself, and a *Java*. The former stores primarily persistent information such as the agent's belief about being (in-)secure, or static information about the meeting point location, safe house location once received from the UAV team, etc., together with a logic-based inference engine allowing to draw deductive conclusions from this information. The second sub-component facilitates mainly manipulation with information about the topology of the simulated environment, i.e., the map of the urban area of the operations theatre and the related reasoning mechanisms, such as e.g., path planning on

[3] `http://www.beanshell.org/`

[4] `http://www.gnu.org/software/m4/`

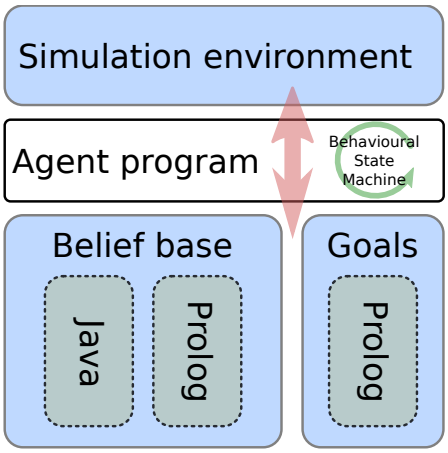

Fig. 4.10 Architecture of ground agents in the *Tactical AgentFly* simulation.

the map graph, etc. The goal base of the agent is implemented by a single *Prolog*-based knowledge representation module besides storing the mission specification and the agent's implicit goals, such as e.g., the temporary goal to flee in the face of a blue team, if also facilitates more complex inferences over the goals. In particular, the mechanism can be used to implement integrity constraints disallowing two mutually inconsistent goals to be derivable at the same time. We however did not need such a mechanism in the actually implemented simulation. The Figure 4.10 provides a schematic overview of the developed agent architecture.

Agent behaviours & the main program

Agents' overall behaviours are implemented as a set of self-encapsulated composable behaviours encoded as reactive plans, policies, in the programming language *Jazzyk*. The mission specification contains a sequence of high-level goals, which are associated with a single, or more concrete behaviours in the agent program. An example of an implementation of such a goal-specific behaviour is listed in Figure 4.11. The code includes a section for goal deliberation, i.e, decision mechanism for recognition that the goal was already achieved, and the behaviour itself, i.e., the behaviour initialization and a call to a macro implementing movement along a path between two adjacent nodes on the street graph.

Additionally, agents often feature implicit behaviours, i.e., such which were not provided in the mission specification, but the agent adopts their associated goals because of its own decision in run-time. The Figure 4.12 lists an example of such a behaviour implementing the tracking of a red insurgent.

```
define('MOVE_TO_DESTINATION', '
    {
        /* goal deliberation */
        when query goals (Dest) [{ current_goal(move_to(Dest)). }] and
            query beliefs (Dest, DNode) [{ location(Dest, DNode). }] and
            sense (DNode) [{ self.atNode(DNode) }]
        then {
            /* drop the goal and move on the next mission goal */
            update goals [{ shift_mission. }]
        } ;

        /* the behaviour itself */
        when query goals (Dest) [{ current_goal(move_to(Dest)). }] then {
            when query beliefs (Dest, DNode) [{ location(Dest, DNode). }] and
                sense (DNode, ONode) [{ ONode = self.nodeAt()}]
            then {
                /* behaviour initialization */
                when query beliefs (ONode, DNode, Path) [{
                        Path=self.planPath(ONode, DNode)
                    }]
                then update beliefs (Path) [{ switch_path(Path). }]
            } else MOVE_ALONG_PATH
        }
    }
')
```

Fig. 4.11 An example of the implementation of a mission-specific behaviour - a transport towards a given destination.

```
define('TRACK', '
    {
        when query goals (RedBoy) [{ current_goal(tracking(RedBoy)). }] then {
            when sense (RedBoy) [{ self.isAround(RedBoy) }] and
                    sense (RedBoy, TNode) [{
                    TNode = self.closestNodeToAgent(RedBoy)
            }] then {
                act [{ BlueForceHQ.askForAirSupport() }] ,
                update beliefs (TNode) [{ going_to([TNode]). }] ,
                MOVE_ALONG_PATH
            }
        }
    }
')
```

Fig. 4.12 An example of a *Jazzyk* code implementing an implicit tracking behaviour of a member of the blue team.

Finally, the behaviours are composed into an agent program. Using the *BSM* composition operators, the agent program basically defines the interrelationships among the behaviours of the agent. In the case of our implementation, listed in Figure 4.13, the behaviours are divided into two main groups composed by a chain preference operator. In result, the agent prefers to execute movement and tracking deliberation over the standard mission-specific behaviours, such as tracking, initial waiting for safe house discovery, securing a given location and tracking.

```
/* high-priority un-interruptible behaviours */
{
    MOVE_TO_DESTINATION ;
    CHECK_TRACKING_NEEDED
}
    /* chain preference - the behaviours below are executed UNLESS the above are performed */
    /

/* non-deterministic choice composition of goal-associated behaviours */
{
    HOMING ;
    WAIT_FOR_TARGET
    COVER_LOCATION ;
    TRACK
}
```

Fig. 4.13 An example of the implementation of a mission-specific behaviour - a transport towards a given destination.

4.3 Evaluation and experiments

The main efforts invested into this workpackage went into prototyping activities providing us with proof-of-concepts for various technologies and approaches described above. As a result, the main contributions of the prototyping efforts we achieved was preparation of technological and testing infrastructure for deeper investigation and subsequent evaluation of techniques for multi-agent planning and plan repair in mission-centric information collection tasks involving heterogeneous agents in the context of the the related continuation project Tactical AgentScout.

Besides the results of the experiments with the new implementation of the exploration information collection task, as the main results of the prototype evaluation, we provide the technology demonstrators included as a deliverable of the project together with this report described later in the Appendix A.

4.3.1 Multi-UAV area exploration

The main result of our evaluation of the two implemented exploration methods for exploration information collection task is the following claim:

Claim. In structured areas with apriori known structure and possibly irregularly distributed points of interests, the DVRP-based exploration method is significantly more efficient than naive implementations such as the zig-zag algorithm.

The above claim hinges on our experiments described below. Important in this context is the observation that naive methods, such as the zig-zag algorithm, uninformed about the topological structure of the target area spend significant time exploring areas without much information, such as open spaces

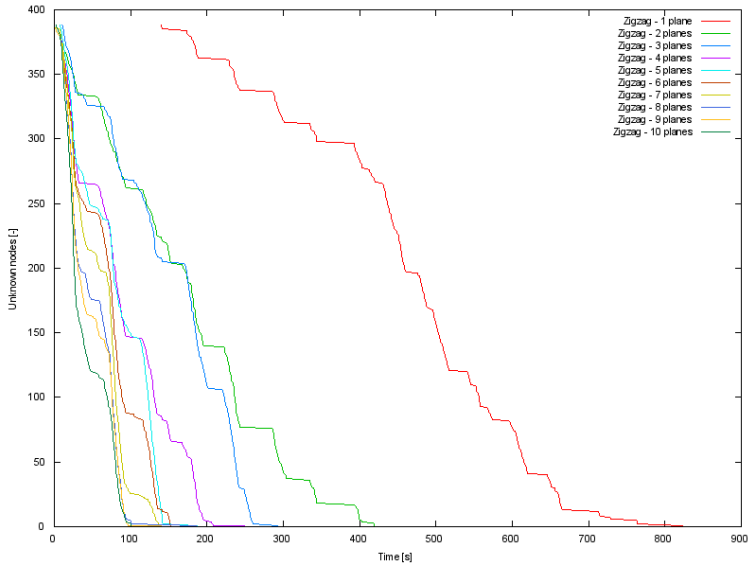

Fig. 4.14 Area exploration – unexplored points of interests in the time for variable number of plains using zig-zag exploration algorithm.

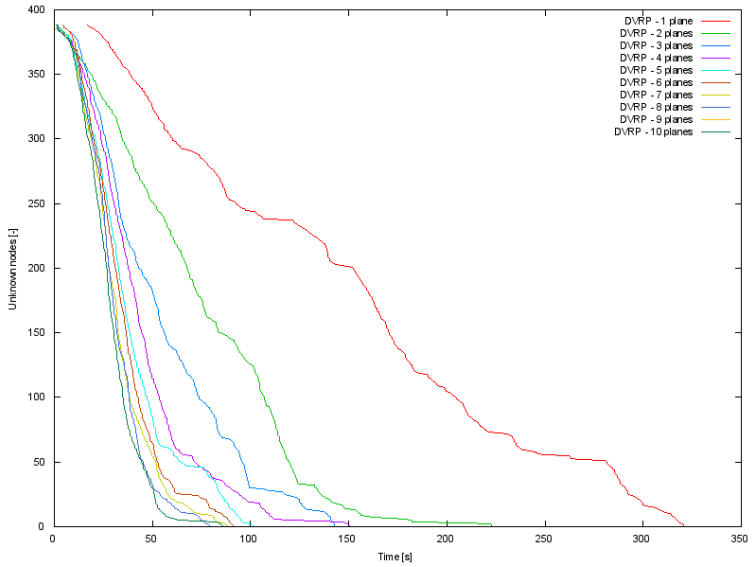

Fig. 4.15 Area exploration – unexplored points of interests in the time for variable number of plains using DVRP-based exploration algorithm.

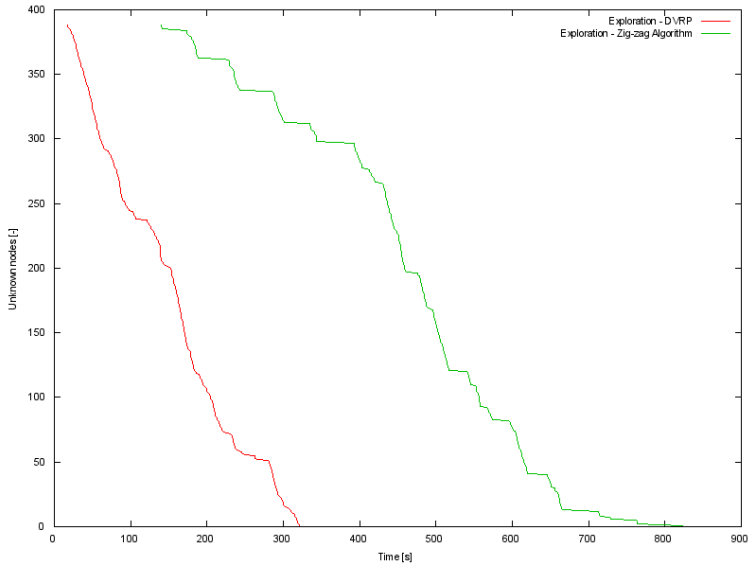

Fig. 4.16 Area exploration – unexplored points of interests in the time for 1 plane using zig-zag or DVRP-based exploration algorithm.

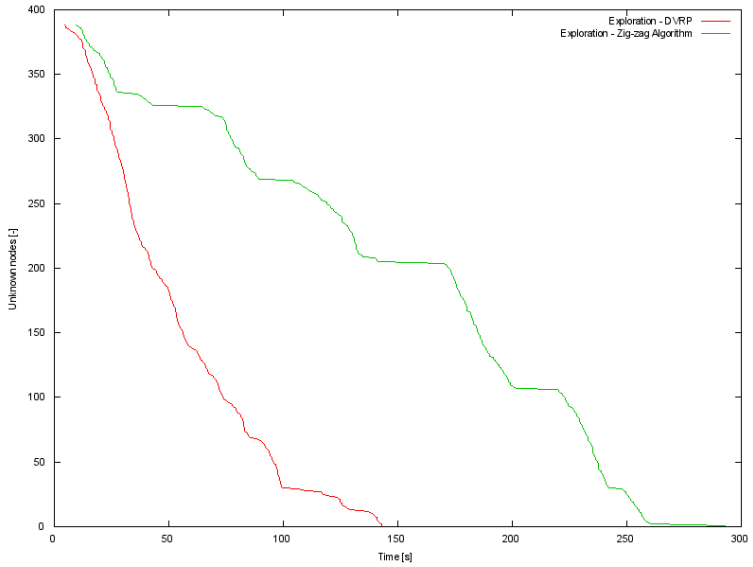

Fig. 4.17 Area exploration – unexplored points of interests in the time for 3 plane using zig-zag or DVRP-based exploration algorithm.

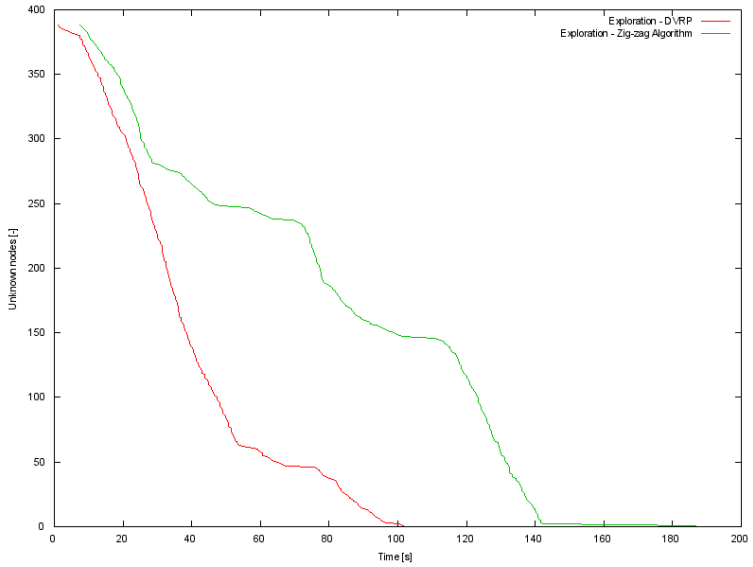

Fig. 4.18 Area exploration – unexplored points of interests in the time for 5 plane using zig-zag or DVRP-based exploration algorithm.

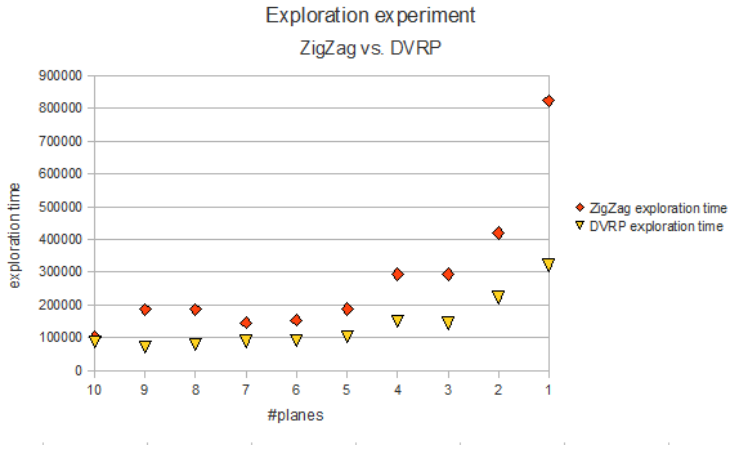

Fig. 4.19 Area exploration – time needed for exploration of all points of interests by varying number of plains for zig-zag and DVRP-based exploration algorithm.

(e.g., desert in our example scenario). Observe also, that the two compared methods should perform equally in situations when the topology of the target area is *apriori* unknown. In such a situation, it makes only sense to feed the DVRP-based method with odes of interest uniformly distributed on a grid over the target area.

The performance and scalability to the number of plains for both algorithms are shown on Figures 4.14 and 4.15. In both methods, the area is explored efficiently with about 6 plains. DVRP-based exploration is scientifically better in all configurations. Figures 4.16, 4.17, and 4.18 shows the comparison of both method for 1, 3, and 5 plains.

The performance of DVRP-based exploration proofs to be better than zig-zag in our scenario. Figure 4.19 shows the DVRP-based exploration method is about 100% faster than zig-zag exploration method. Experiment results show notable steps in the time, where only minimal number of nodes are explored. It corresponds to the situation, where the plains are flying over desert areas or making turns on the upper or lover part of the area. These blind steps are not significant in the results of DVRP-based method, because plains are optimize their routes across the points of interests. The only exception is a situation when the route planning heuristics fails to provide non-repeating route across points of interest (i.e. plain is passing already explored nodes) or there is need for longer flight over non interesting area (can bee seen for 1 plane in Figure 4.15).

4.4 Towards mission planning and multi-agent plan repair

As already indicated above, our main achievement along the research tracks leading to multi-agent mission planning and plan repair was the survey of the state-of-the-art results in the multi-agent planning research and first steps towards a theoretical analysis and a framework for multi-agent plan repair. The prototyping and development work was shifted into the corresponding workpackages of the related continuation project Tactical AgentScout, where these became one of the major research foci of the project. In the following, we include excerpts from the Tactical AgentScout *M3* report as delivered in July 2010.

4.4.1 Multi-agent mission planing: brief survey of the state of the art

Since the rise of interest in autonomous agents and multi-agent systems during the last decade of the last century, the problem of *multi-agent planning* and *coordination* in groups of cooperative self-interested agents stands in the focus of the AI scientific research community. In the past, the sheer computational complexity of the problem on one hand and its inherently decentralized character on the other led to abandoning of the concept of centralized planning for teams of agents. The first reason is a consequence of the observation

that multi-agent planning subsumes the problem of planning for a single agent, which in itself is computationally one of the most difficult problems in the field of artificial intelligence. The latter point stems from the observation that in not fully cooperative settings self-interested team members should prefer to disclose only as little information as possible about their internal plans. This in turn discourages applications of classical centralized planning and leads to just-in-time coordination techniques, such as application of Contract-Net Protocols [11], i.e., coordinating only when a real need arises. While in the meantime a number of more, or less rigorous approaches based on techniques such as plan merging, plan negotiations, task delegation (cf. e.g., [3]), or distributed continuous plan monitoring and repair [4], etc., arose, only recently we witness emergence of more compact fundamental theoretical results and rigorous approaches in the field. One of the most interesting instances are the theoretical multi-agent planning complexity results by [2]. One of the factors enabling reconsideration of centralized multi-agent planning for coordination in the AI planning community was the tremendous increase and ubiquity of widely available computational power which in turn enables application of state-of-the-art centralized planning algorithms in this domain.

4.4.2 Multi-agent re-planning and plan repair

Plan repairing is a process of partial adaptation of a plan during its execution according to new conditions in the environment. The plan has not to be altered as a whole, but only its local part which is inconsistent with the new conditions, has to be changed. On the contrary, *re-planning* is a process of restarted planning during execution of a plan. The new planning process starts from the current state under the current context. Re-planning do not reuse any parts of the old (inconsistent) plan.

The research challenge for the frame of plan repairing and re-planning include finding answers for questions as: *"For what problem types we need plan repairing and for what re-planning?"* The hypothesis supporting plan repairing approach requires local and relatively rare unpredictable effects. The re-planning approach do not care the amount or impacts of the unpredictable effects, as the planning process has to be always run from the current state to the goal state. On the other hand, re-planning requires larger amounts of information to be shared among the planners (agents) in order to reconstruct the global state and context of other agents.

Additionally, we hypothesize two other important types of unpredictable world effects from the perspective of the deliberative/reactive planning systems and approaches. The first one is a *stolid effect* and the other one is a *opportunistic effect*. A stolid effect causes only a specious plan inconsistency (e.g. solvable by simple ignoring of the effect) where such effects can be effec-

tively tackled by the plan repairing approaches. An opportunistic effect can even improve the whole plan provided that it is properly exploited. The opportunistic effects should be probably more re-planning friendly, as the plan repairing is a local process and can not simply consider the global state and the neighboring agents' context.

For an experimental validation and evaluation of the outlined hypotheses we have designed a testing scenario in two levels of abstraction. The next section summarizes the key principles and elements of the scenario.

Plan Repairing Formalization

The research focuses mainly on a direction from centralized simple plan repairing algorithms towards decentralized localized techniques based on multi-agent principles of negotiation and various kinds of commitments.

The design considers two main blocks of the planner and repairer. The input of the planner is a planning problem including possible actions (see below) and the output is a plan for multi-agent system (aka MAS plan, see below). The repairer requires a MAS plan as an input and returns another repaired MAS plan. The complete execution process takes a planning problem Π as an input, output is *True* if the goal state S_{Goal} was achieved and *Fail* if the goal state cannot be achieved. As a side effect of the algorithm the mission is planned, executed and prospectively a number of times repaired.

The planning function is denoted as $plan(...)$, the plan repairing function is denoted as $repair(...)$, and a plan simulating function returning a simulated state after k steps using a plan $\mathcal{P}$ is denoted as $simulate(...)$. An action execution function is defined as:

$$\begin{aligned} &exec : S \times S \times S \to S, \\ &\quad (\sigma, A_{del}, A_{add}) \mapsto (\sigma \cup A_{del}) \setminus A_{add}, \end{aligned}$$

where other definitions can be also considered in the future (e.g., $(\sigma, A_{del}, A_{add}) \mapsto \sigma$). The state S and action effects A_{add}, A_{del} will be precisely defined in their respective sections.

Planner

A planning problem Π is defined using a set of agents Ag (defined only using their actions a, consisting of preconditions pre, add effects, and delete effects del), states S described using propositional variables of language $\mathcal{L}_{WFF}$, initial state S_{Init}, and goal state(s) S_{Goal}. A planning problem definition Π follows:

$$
\begin{aligned}
\Pi &= (Ag, S, S_{Init}, S_{Goal}),\\
Ag &= (A_1, ..., A_n),\\
A_i &= \{\{pre\}a\{add\}\{del\}|\\
&\quad a \in Labels, pre \in \mathcal{L}_{WFF}, del \in \mathcal{L}_{WFF}, add \in \mathcal{L}_{WFF} \cup \{\epsilon\}\},\\
\epsilon &= \{\emptyset\}e\{\emptyset\}\{\emptyset\}, e \in Labels,\\
S &\subseteq 2^{\mathcal{L}_{WFF}},\\
s_{Init} &\in S,\\
S_{Goal} &\subseteq S.
\end{aligned}
$$

Input of the planner is a problem Π including the actions. Output of the planner is a MAS plan. MAS plan is defined as follows:

$$
\begin{aligned}
\mathcal{P} = (Pl_1, ..., Pl_n) : \exists m > 0 :\\
(Pl_i = a_1, ..., a_m : a_j \in A_i) \wedge\\
\wedge (\exists s_0, ..., s_m : s_k \in S : s_{k+1} = s_k \oplus \langle Pl_1(k), ..., Pl_m(k)\rangle) \wedge\\
\wedge \left(\forall k : 1 \leq k \leq m : \left\{\bigcup_{j=1}^{n} add(Pl_j(k))\right\} \cap \left\{\bigcup_{j=1}^{n} del(Pl_j(k))\right\} = \emptyset\right) \wedge\\
\wedge (s_0 = s_{Init}) \wedge (S_{Goal} \subseteq s_m),
\end{aligned}
$$

$$
\begin{aligned}
s' = s \oplus \langle a_1, ..., a_k\rangle \equiv (\forall 1 \leq j \leq k : pre(a_j) \subseteq s) \implies\\
\implies \left(s' = s \setminus \left\{\bigcup_{j=1}^{k} del(a_j)\right\} \cup \left\{\bigcup_{j=1}^{k} add(a_j)\right\}\right).
\end{aligned}
$$

A MAS plan is a global set of linear personal plans Pl. All the personal plans has the same length m (the empty points of the plan are filled with ϵ). A personal plan consists of actions a of appropriate agent A. The global states s of the system evolution are induced by the MAS plan actions of all agents$\langle Pl_1(k), ..., Pl_m(k)\rangle$ for each step k, if so the plan is considered *sound*. In each step k the actions of all agents must be *consistent*, i.e. *add* and *del* sets have to be exclusive. The initial and goal states have to be taken into account. Update of a state removes all *del* effects and adds all *add* effects.

Repairer

A plan repairing problem $\mathcal{R}$ is defined using a planning problem Π, resulting MAS plan $\mathcal{P}$, a fail (unanticipated) state s_F, and a fail step k:

$$\mathcal{R} = (\Pi, \mathcal{P}, s_F, k)\,,$$
$$s_F \subseteq S : s_F \neq s_k,$$
$$k \in 1..m.$$

An input of a repairer (and a repairing algorithm) is a problem $\mathcal{R}$ and it is defined as:

$$\mathcal{R}, \exists i : pre(Pl_i(k)) \subseteq s_F.$$

An output is a repaired sound and consistent MAS plan $\mathcal{P}'$ defined as:

$$\begin{aligned}\mathcal{P}' : \exists m > 0 : \exists s'_0, ..., s'_m : \\ \forall i < k : s'_i = s_i \wedge \\ \wedge s_k = s_F; \\ s_k \cong s'_k = S_F.\end{aligned}$$

Failures

We define two plan *failure* reasons:

1. effects of an action failed to modify the state $\implies s_{curr} \neq s_{sim}$.
2. $s_{curr} \neq s_{sim} \implies$ precondition of an action is not satisfied

A *weak failure* of a MAS plan $\mathcal{P}$ execution at the step k with current state s_{curr} is defined as follows:

$$\begin{aligned}\exists i \;:\; add(Pl_i(k-1)) \not\subseteq s_{curr} \wedge \\ \wedge \forall j > k \forall j : pre(Pl_j) \not\supseteq FailResidue, \\ FailResidue = add(Pl_i(k-1)) \setminus del(Pl_i(k-1)) \cap s_{curr}.\end{aligned}$$

A weak failure means the failure can be a caused by a stolid world effect or an opportunistic world effect, i.e. It can be prospectively fixed without decreasing quality (length) of the plan or even exploiting it, the quality of the plan can increase (the plan can be shortened). If a failure is not weak, it is *strong*.

Naive Repairing Algorithm

The simplest algorithm designed uses iterative re-planning of the plan from the point of the failure. The complexity of the Naive Repairing Algorithm respects the inner planner complexity. The worst case complexity is *PSPACE-*

complete (since a general planning fits the *PSPACE-complete* complexity class).

Blind Repairing Algorithm

The Blind Repairing Algorithm tries to solve a failure by adopting an alternative action(s) solving the failed effects. Firstly it finds agent(s) which caused the problem (their effects does not meets the simulated state). A after the agents are found, for each agent an alternative action is tried to be found (aka the relaxation de-commitment rule). If the personal alternative cannot be found a wide alternative is tried (another agent can adopt an action which solves the failure – aka the delegation de-commitment rule). Finally, even if the wide alternative cannot be found, the problem is ignored to be solved in the next steps prospectively. If a failure is ignored the repairing algorithm is called again in the next step by the main simulation algorithm.

The algorithm is not sound in the pure form as it can miss a solution of the repairing problem. Its modification towards the soundness can be managed simply by adding a fail-safe call of the Naive Repairing Algorithm if the algorithm iteratively fails at the goal state.

The non-iterative form of the algorithm has a linear complexity (iteration over the number of the agents). The iterative extension has a minimal quadratic complexity (number of the agents times the length of the plan). Maximal complexity reflects the complexity of the Naive Repairing Algorithm as it can be used as a fail-safe process.

Chapter 5
Discussion and conclusion

5.1 Modelling and integration of Vertical Take-off and Landing Assets

We provided a path (trajectory) planner for VTOLs augmented with speed limit constraints taking into account a simple model of (VTOL) dynamics. Simply said, the planner is providing plans the particular VTOL is able to fly through. The plan execution, e.g., the simulation of VTOL flying, is done with respect to its stabilization model with regulators that make the VTOL movement much more realistic. Even though our model does not take into account other environmental aspects such as wind, the simulation can give us insights of VTOLs behavior when fulfilling the mission tasks (such as surveillance). The experimental evaluation we made brought us quite positive results even though the method we introduced should be investigated more thoroughly (also from a theoretical point of view) to provide more general and brighter claims.

5.2 Planning in Dynamic and Resource Constrained Environments

We have developed multi-agent solver based on task allocation for domains of tracking, surveillance, and exploration taking into account high dynamism of the environment and heterogeneous UAVs teams. The performance and features of this approach has been evaluated in various scenarios and proofs its applicability in presented domains.

Further improvements can go towards better incorporation of the dynamics and constraints of the resources (e.g. UAVs and VTOLS) in the solver heuristics. The local agents planners should take into account situational conditions of the resource, it's dynamics, and more detailed model of capabilities. The

conjunction of the multi-agent solver described in Chapter 3 and the VTOL trajectory planning with full dynamics modeling described in Chapter 2 may provide additional improvement of the presented methods.

The high degree of dynamism in presented scenarios provides the need of frequent planning and allocation requests. Another further improvement may focus to flexible planning horizon to reduce the computational complexity of the local path planner. Preliminary experiments promise low impact to algorithm efficiency when limiting the number of tasks taken into account by local planners. The definition and implementation of flexible planning horizon with respect to the properties of tasks should provide reduction of computational needs.

5.3 Integrated coordination for mixed information collection activities

We have developed a novel technique for area exploration based on distributed multi-agent task allocation and evaluated it against a naive uninformed algorithm. Furthermore, we have developed an integrated mission scenario involving multiple phases which demonstrates various types of information collection tasks, such as exploration, surveillance and on-demand tracking. The fully configurable scenario will allow us in the future to test interaction of various information collection tasks supporting ground mission.

The inspirations for future work in the context of this workpackage include

1. research towards further extensions and advances on the configurable mission framework, with a special focus to extend the mission specification language and execution technology to enable partially ordered mission plans and contingency plans
2. study of interactions among various mission-level goals w.r.t. the tasking of the individual team members. This line of research ultimately leads to deeper investigation of problems involving multi-agent mission planning and plan repair. We intend to expand on this topic in the on-going research project Tactical AgentScout.
3. one could investigate coordination multi-agent techniques in conditions of failing communication. In the here reported project, we assumed perfect communication links between the agents.

References

1. E. Börger and R. F. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003.
2. R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In J. Rintanen, B. Nebel, J. C. Beck, and E. A. Hansen, editors, *ICAPS*, pages 28–35. AAAI, 2008.
3. M. de Weerdt and B. Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5(4):345–355, 2009.
4. M. desJardins, E. H. Durfee, C. L. O. Jr., and M. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.
5. Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.
6. P. Novák. Behavioural State Machines: programming modular agents. In *AAAI 2008 Spring Symposium: Architectures for Intelligent Theory-Based Agents, AITA'08*, March 26-28 2008.
7. P. Novák. Jazzyk: A programming language for hybrid agents with heterogeneous knowledge representations. In *Proceedings of the Sixth International Workshop on Programming Multi-Agent Systems, ProMAS'08*, volume 5442 of *LNAI*, pages 72–87, May 2008.
8. P. Novák and J. Dix. Modular BDI architecture. In H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, editors, *AAMAS*, pages 1009–1015. ACM, 2006.
9. P. Novák and W. Jamroga. Code patterns for agent-oriented programming. In *Proceedings of The Eighth International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS*, 2009.
10. P. Novák and M. Köster. Designing goal-oriented reactive behaviours. In *Proceedings of the 6th International Cognitive Robotics Workshop, CogRob 2008, ECCAI co-located workshop, July 21-22 in Patras, Greece*, pages 24–31, July 2008.
11. R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Computers*, 29(12):1104–1113, 1980.

Appendix A
Demonstators

VTOL Demo

In the first mentioned demonstration, the integration of the newly designed and implemented VTOL model and hex-grid planner is presented. The demo is based on the Tactical AgentFly I scenario, where the CTOL UAVs provided a surveillance mission. Since the VTOL algorithms were implemented into the AgentFly system, the possible tasks of the VTOLs copy the tasks available for the CTOL UAVs. They are rectangular zig-zag surveillance with decentralized partitioning of the area and tracking of a group of ground units. The demonstration also shows an integration of heterogeneous assets. In one group, the CTOL UAVs operates and in the other the VTOLs are used and tasked from the Common Operational Picture interface.

The model of the VTOL uses the presented dynamics simulation. It is visible especially in the turns where the trajectory controllers tries to hold the helicopter in the pre-planned trajectory. The planning process of the VTOLs can be visualized for each helicopter. Besides the trajectory and flight corridor, also the hex cells expanded in a state-space can be visualized. The blue hex cells represent elements in the OPEN list and the green hex cells represent the planned hex-grid plan.

MxN Surveillance Demos

The three MxN Demos present the proposed three algorithms for the area surveillance (a) Randomized Target Selection Algorithm, (b) Greedy Solver with Collective Allocation, and (c) Dynamic Vehicle-Routing Problem (DVRP) Multi-Agent Solver. The demos show various combinations of the number of the assets providing the surveillance and the ground targets. In the demos, the mobility model of the ground targets uses a random choice of next direction in each node of the street graph. It implies, the targets move randomly through the village staying always on the streets. The UAVs uses a conical

sensor with the occlusion simulation. Exclusive of the Randomized Algorithm, the remaining two methods uses an intra-agent communication channel for the exchange of the information needed by the algorithms.

Mission-centric Demo

The mission of the Mission-centric Demo begins with a group of hostiles (Red Forces) and a group of allies (Blue Forces). The hostiles are already in the village around a safe-house (the safe-house can be randomly placed by the scenario configuration). Additionally, there are other hostiles randomly spread over the village (small groups of 1-3 persons). The allies initially come at an assembly point from an edge of the world. An arbitrary number of UAVs is also supporting the allies.

The task of the Blue Forces is to cross the village and set an perimeter around the safe-house. If the allies run into a group of hostiles, they have to split into two groups (the smaller group must outnumber the hostiles of 1). The smaller group start to pursuit the hostiles. The hostiles will randomly flee through the village. The resting group will continue in their mission towards the safe-house. The fleeing hostiles can be captured by the allies. Allies without pursued hostiles will try to return to the main group of the forces. If a pursuing group cannot be further split, it picks only one (sub-)group of fleeing hostiles and continues the pursuit. The number of the allies in the main group (securing the perimeter) must not decrease under a constant. If there are no hostiles around the safe-house, the allies go to an extraction point. This behavior defines the context for the UAVs supporting the allies.

The mission has five main phases (waiting for the Blue Forces, ingress with pursuit, setting of the perimeter, regress, final transport). All these parts are covered by the UAVs with various behavior. In the first phase, all the UAVs are required to do a area search over the complete area, until the safe-house is found. When the safe-house is found, the UAV start to do complete area surveillance (using the Zig-zag algorithm). Then the first group is designated (all the allies) and the second phase begins. Always, if a new (sub-)group of the allies is formed (by the splitting), one UAVs is tasked to start to track the group to provide a tight surveillance of the area around the (sub-)group. Always, if two groups merges, the UAVs can continue in the complete area surveillance. In the third phase, the UAVs are tasked to do a area surveillance in a wider area around the safe-house. In the fourth phase, one of the UAVs starts to implicitly do a tracking of the only group left (all the allies) supporting them during the return. In the last fifth phase, all the UASs are tasked to do the complete surveillance again. The UAVs use the DVRP-based MxN tracking algorithm if there is more blue ground groups than the number of the UASs in the area.

The behavior of the units is done by a reactive rule-base system and implemented in the Jazzyk language.

Exploration Demos

The last two demos presents the difference between the Zig-zag surveillance algorithm proposed in the first phase of the Tactical AgentFly project and the surveillance based on the DVRP Multi-Agent Solver. The scenarios were used for the experiments comparing the two proposed and implemented methods in the two phases of the project. The environment setting is based on the MxN Surveillance Demos, but the ground targets are omitted and the metrics is based on the absolute number of observed street graph nodes, i.e. the crossings. The sensor also is conical with the occlusion simulation.

Appendix B
Technology overview

The architecture of the system is based on the Java programming language (additionally, the Jazzyk language is used together with Prolog in the mission-centric rule definitions). The multi-agent platform A-globe is used as a communication layer and agent handling container for the agents. The A-globeX Simulation is based on the A-globe Platform and simulates the entities behaving in the environment (see the FigureB.1).

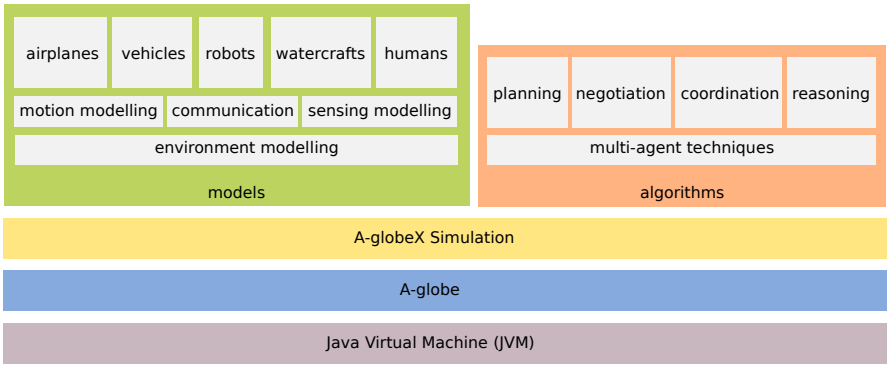

Fig. B.1 The architecture of the A-globeX Simulation Platform

The three surveillance algorithms together with the Mission-centric scenario were implemented in a lightweight wrapper of the A-globeX Simulation. The wrapper enables simpler entity description and definition of the agent logic. Additionally, the wrapper acts as an experimental platform. The experiments can be configured and run directly in programming language (in this case Java) and can be simply chained into more complex experiment batches. The course of the simulation is optionally controlled from the Time Provider Agent of the A-globeX Simulation. The agent behavior in the wrapper calls using the A-globe Topics sub-system the Tactical Module in the

AgentFly system. The plane behavior computes the positions of the planes and helicopters in the space and reports it back to the agent behaviors in the wrapper.

Visualization of the scenarios is based on a 3D Visio tool integrated with the entity behaviors of the A-globeX Simulation. As a 2D output, it is used a lightweight visualization package of the wrapper using Java2D extension with accelerated drawing.