Bachelor's Thesis

Czech
Technical
University
in Prague

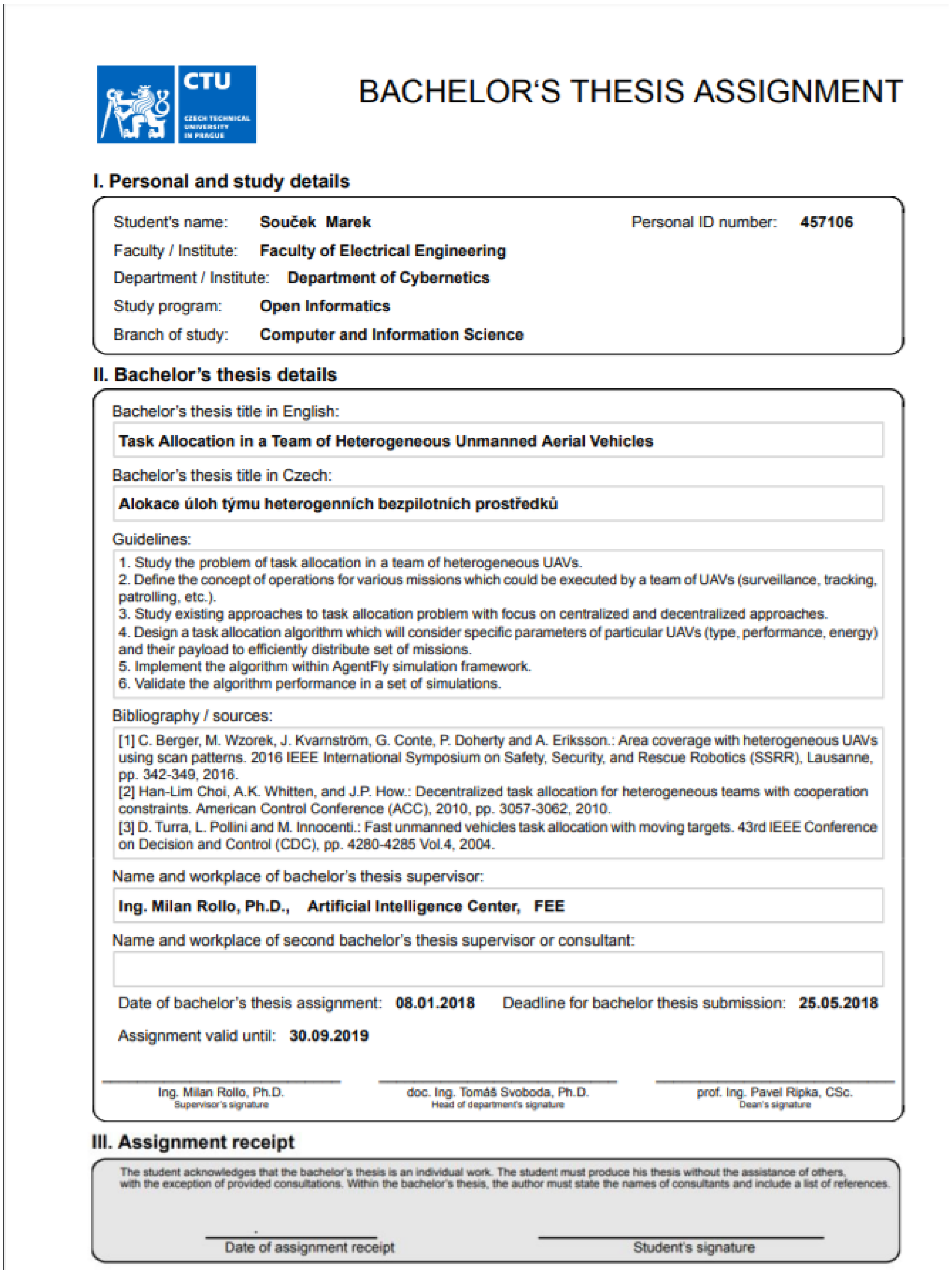
F3

Faculty of Electrical Engineering
Department of Cybernetics

Task Allocation in a Team of Heterogeneous Unmanned Aerial Vehicles

Marek Souček

May 2018
Supervisor: Ing. Milan Rollo, Ph.D.



BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Souček Marek**	Personal ID number:	**457106**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor's thesis details

Bachelor's thesis title in English:

Task Allocation in a Team of Heterogeneous Unmanned Aerial Vehicles

Bachelor's thesis title in Czech:

Alokace úloh týmu heterogenních bezpilotních prostředků

Guidelines:

1. Study the problem of task allocation in a team of heterogeneous UAVs.
2. Define the concept of operations for various missions which could be executed by a team of UAVs (surveillance, tracking, patrolling, etc.).
3. Study existing approaches to task allocation problem with focus on centralized and decentralized approaches.
4. Design a task allocation algorithm which will consider specific parameters of particular UAVs (type, performance, energy) and their payload to efficiently distribute set of missions.
5. Implement the algorithm within AgentFly simulation framework.
6. Validate the algorithm performance in a set of simulations.

Bibliography / sources:

[1] C. Berger, M. Wzorek, J. Kvarnström, G. Conte, P. Doherty and A. Eriksson.: Area coverage with heterogeneous UAVs using scan patterns. 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Lausanne, pp. 342-349, 2016.
[2] Han-Lim Choi, A.K. Whitten, and J.P. How.: Decentralized task allocation for heterogeneous teams with cooperation constraints. American Control Conference (ACC), 2010, pp. 3057-3062, 2010.
[3] D. Turra, L. Pollini and M. Innocenti.: Fast unmanned vehicles task allocation with moving targets. 43rd IEEE Conference on Decision and Control (CDC), pp. 4280-4285 Vol.4, 2004.

Name and workplace of bachelor's thesis supervisor:

Ing. Milan Rollo, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **08.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Milan Rollo, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature

/ Declaration

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

Prague, date 25. 5. 2018

....

Abstrakt / Abstract

Tato práce se zabývá problematikou přidělování úloh v týmu heterogenních bezpilotních prostředků (UAV). Definuje koncept operací pro širokou škálu misí, které mohou být prováděny týmem UAV. Koncept operací zahrnuje mise jako mapování, hlídky, sledování cíle, vyhledávání a další. Práce pak zavádí formalizaci těchto misí, UAV, prostředí a základních úloh, na něž jsou mise rozkládány. Při řešení problematiky alokace úloh na UAV se, ve formě automatizovaného plánovacího jazyka PDDL, využívají nedávné pokroky v oblasti obecného plánování. Součástí práce je také přehled a srovnání dostupných PDDL plánovačů. Práce navrhuje architekturu systému pro plánování a alokaci úloh na UAV, která využívá PDDL plánovač a je schopna zvládnout širokou škálu misí v dynamickém světě, kde operátor může kdykoliv přidat nový úkol nebo ukončit existující. Navrhnutá architektura je implementována a integrována do AgentFly simulátoru. Na závěr je provedeno několik simulací s různými scénáři pro otestování navrhnutého systému.

Klíčová slova: alokace úloh, plánování, bezpilotní prostředky, UAV, PDDL

This thesis addresses problem of task allocation in a heterogeneous team of Unmanned Aerial Vehicles (UAVs). The concept of operations is defined for a wide range of missions which can be executed by a team of UAVs. The concept of operations includes missions like mapping, patrol, tracking, search and other. Thesis then introduces a formalization for these missions, UAVs, environment and elementary tasks into which missions are decomposed. Recent advances in the field of general planning are used in form of an automated planning language PDDL to solve problem of tasks allocation. Overview and comparison of available PDDL planners in presented. Thesis proposes a system architecture for planning and task allocation that utilizes PDDL planner and is capable to handle a wide range of missions in a dynamic world where the operator can add a new mission or terminate existing one at any time. Proposed architecture is implemented and integrated within AgentFly simulator framework. Finally, several simulations with multiple scenarios are conducted to test capabilities of proposed system.

Keywords: task allocation, planning, unmanned aerial vehicles, UAV, PDDL

Contents

Tables / Figures

Chapter 1
Introduction

In recent years, the UAVs (Unmanned Aerial Vehicles) have quickly expanded and started to be used in many industries thanks to improvements of sensor equipment, increasing range and descending prices. Lower prices also allow deploying of entire UAV teams.

Cooperative UAV teams open opportunity for new types of missions. Search for moving target can serve as an example. If more UAVs cooperates, search pattern can be used to prevent undetected target move to already searched area. Cooperative UAV teams can also significantly decrease mission execution time if that mission is parallelizable.

Several military applications are a driving force for research and progress in field of control and cooperation of UAV team. UAVs allow military to swiftly cover large or hard to access areas with only a small number of operators. UAVs can also operate in dangerous zones without the risk of human loss. Among frequently used military applications belong border patrols in difficult terrain or search and destroy missions. But many civilian applications also exist. UAVs are already used in search for missing person or search and rescue missions after disasters. Another application can be found in modern precise agriculture where UAVs are used to map the state of crops. Finally, applications as package delivery or air quality monitoring and many others in other domains.

This thesis proposes solution to situation where multiple missions are part of one large operation controlled by one operator and executed by a team of heterogeneous UAVs. Approach of decomposing large or complex missions into set of elementary tasks is used. Actual task allocation to UAVs is represented and solved as general planning problem in PDDL language.

This thesis is organized as follow. In Chapter 2 is concept of operation with an overview of missions and description of UAVs and environment. Chapter 3 introduces formalization of missions, UAVs, environment and elementary tasks into which missions are decomposed. Chapter 4 provides a study of existing approaches and state of the art algorithms. In Chapter 5 is an overview and comparison of available PDDL planners. Chapter 6 proposes a system architecture for planning and task allocation that utilizes PDDL planner and is capable to handle a wide range of missions in a dynamic world where the operator can add a new mission or terminate existing one at any time. Chapter 7 describes an implementation of proposed system architecture and its integration within AgenFly simulation framework. Results from several conducted simulations with different scenarios are presented in Chapter 8. Finally, in Chapter 9 is conclusion and future work.

Chapter 2
Concept of operations

One operator should be able to control large group of UAVs in operation area just by creating missions that need to be completed. The operator creates high-level missions (e.g. map this area) which represent a large amount of work and that may require use multiple UAVs for successful completion. The system decomposes these missions to sets of elementary tasks and distributes these tasks to UAVs. Tasks and UAVs are heterogeneous, so some UAVs are more competent to do certain tasks then others.

2.1 Scenarios

This section provides an overview of missions that are considered and implemented in the system. Brief motivation, a form of input required from the operator and description of intended behavior are presented for each mission.

2.1.1 Mapping

Mapping is one of the most frequently used UAV applications. Mapping has many useful applications even outside of obvious use in exploration and cartography. For example in [1] scenario in agriculture is presented where aerial photography of fields, respectively crops is used to determine the status of the crops. Gathered data are then used to determine when, where and in what amounts of fertilizers or pesticides should be used.

The operator defines the area of interest to be mapped and selects required resolution. The shape of the area is described as a flat polygon. Team of UAVs will take pictures of area inside that polygon in required or better resolution.

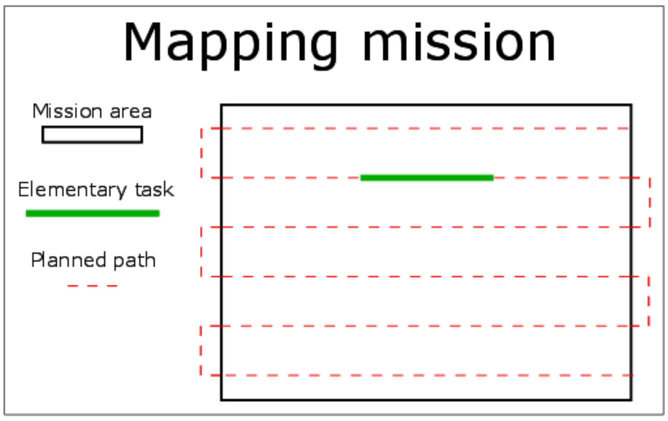


Figure 2.1. Mapping mission.

The main objective for UAVs is to create a map in required resolution in as short time as possible. Flight path consists of parallel lines in such distances and height to ensure required resolution and some minimal overlap. The flight path is divided into several segments (elementary tasks) with same length and assigned to UAVs. Any number of segments can be assigned to one UAV based on its speed, position and sensors on board to minimize makespan of the mission.

Data from each UAV will be downloaded and combined together to create final map after all UAVs complete their parts. Output is map of area in required resolution.

2.1.2 Search

Thanks to increasing range and improvements of sensor technology team of UAVs are becoming a very interesting choice for solution of search and rescue missions such as rescue operations after a natural disaster or searching for lost persons [2].

The operator defines the area of interest to be searched and selects required resolution. The shape of the area is described as a flat polygon. Team of UAVs will take pictures of area inside that polygon in required or better resolution.

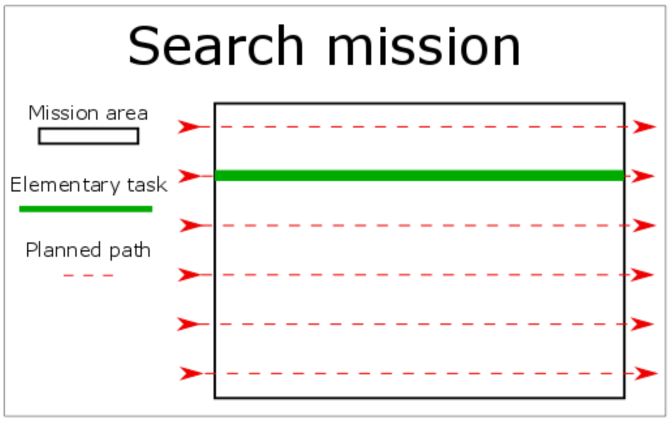


Figure 2.2. Search mission.

Search is similar to mapping, but the goal, in this case, is to find target instead map the whole area. Target will be mobile in most cases. Therefore team of UAVs should coordinate and fly in formation. Use of formation allows to search a larger area without gaps at one time. This prevents or at least minimizes the chance of target's unnoticed move to the already searched area.

2.1.3 Tracking

Tracking is task where UAVs are already successfully used. This type of mission is required in several domains. The best known applications are in security. Whether it's tracking of an intruder on private property, or military UAV keeping eye on target to provide real time intelligence. But tracking has also other more peaceful applications, such as monitoring the movement of herds of wildlife [1].

The operator creates a tracking request. This request includes information about target needed for efficient assignment of UAVs. Current position and estimation of minimal and maximal speed of target are required minimum. Other requirements on UAV capabilities such as sensor type can possibly be added in further to constrain UAV that can be assigned to this task.

The system then assigns this task to UAV. The assignment has to satisfy constraints on UAV capabilities and tries to minimize latency. Selected UAV then move to the target position and begins auto-tracking. UAV in auto-tracking mode does not require any other control from the system. The system just needs to be aware of this assignment and not to plan any other tasks for this UAV.

Real time feed of sensor data is provided during tracking task execution. Tracking has infinite duration and no termination condition. Tracking ends and is considered completed when operator sends termination request.

2.1.4 Patrolling

Another application of UAV squad is a patrol. Team of UAVs can supplement static surveillance cameras. The mobility of UAV allows covering of larger perimeter and dynamic changes if needed. Also if some intruder is detected UAV can go into the tracking mode as described above in tracking scenario. And provide real time, comprehensive picture of intruder's actions and movements.

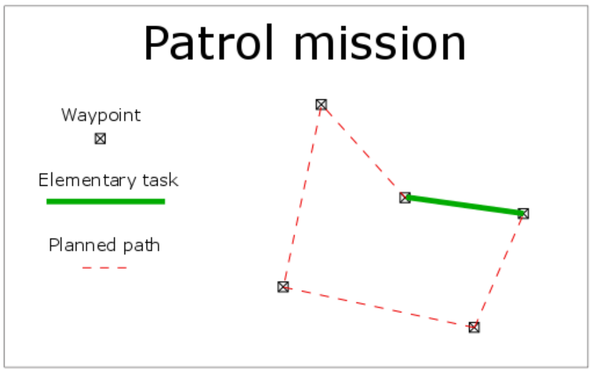


Figure 2.3. Patrol mission.

The operator defines patrol route and required period. Patrol route is described as a list of way-points. The main objective is to ensure that each segment (elementary task) of patrol route between two subsequent way-points is checked (fly throw) in each time period. Real time feed of sensor data is provided during patrol task execution.

2.1.5 Measurement

Measurement is task category where the goal is to obtain dataset consisting of several data points from different locations. An example can be air quality monitoring presented in [1] where a grid of measurements is required. Another example is a request for multiple photos of one object from different positions and angles which are then used for 3D modeling of that object.

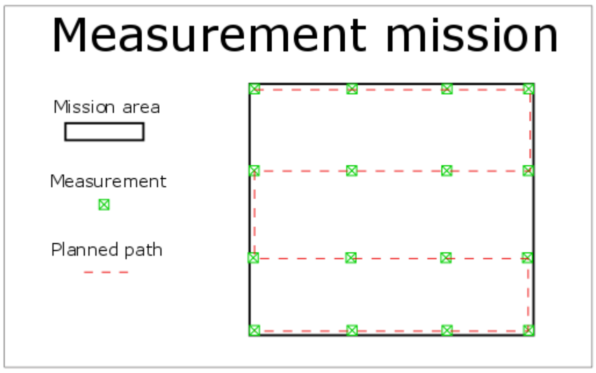


Figure 2.4. Measurement mission.

The air quality monitoring scenario is selected to represent this category of tasks in the system. The operator defines the area of interest, height and required density of data points. The shape of the area is described as a flat polygon. Also, duration of a single measurement has to be provided.

The system then creates set of individual measurements needed to cover the area of interest and distributes them to available UAVs. Data from UAVs will be downloaded and combined together to create the final dataset after all UAVs complete their parts.

2.2 UAV types

First and most obvious source of heterogeneity is UAVs construction type. Construction type of UAVs affects maximal/minimal flight speeds, turning rate, flight levels or weight and size of the payload. These flight characteristics are categorically different for vehicles from different classes as VTOL (Vertical Take-Off and Landing) or fix-wing but differ even within one class.

In addition to that, each vehicle can carry a different set of sensors and equipment on board. Each sensor can be described by set of values. For description of some equipment is one boolean value enough (has or has not sensor of atmospheric pressure). Other equipment needs one number to express its capabilities and limitations (radio range, amount of ammo). And for example camera like sensors may need several values like spectrum (e.g. normal, infrared), FOV (field of view), FPS (frames per second) and resolution.

Properties above together with fuel or energy level create state vector of given vehicle. And as shown in [3] not only that state vectors differ from vehicle to vehicle, but they also change in time due to fuel consumption and external environmental conditions.

2.3 Environment

All scenarios mentioned above consider outdoor operations which may spread over large area. A detailed description of an outdoor environment in such scale can be very complex. Some variables may even be only estimations or not completely reliable predictions of state in given time especially if the weather is taken to account. This unpredictable nature of outdoor environment makes effective and optimal planning unreasonably complex.

Since this thesis is focused more on task allocation and a little bit on trajectory planning. But do not deal with flight control in task execution level, we will consider simplified environment.

No weather will be considered in calculations. Extreme conditions (e.g. heavy storm, hail, fire etc.) may project to the overall planning of the operation. But the reaction to such conditions and plan adjustments or changes are left on the human operator. That may include temporary no-flight zones or termination of some missions in order to protect UAVs. It is assumed that UAVs themselves take care of reactions to other more standard conditions like a wind on flight control level.

Because flight altitude of most UAVs do not exceed 100 meters some obstacles like high buildings or mountains should be taken to account. Event thought our environment is a large open outdoor area where almost any obstacle can be fly over.

Chapter 3
Problem formulation

This chapter introduces the formalization used for UAVs, environment, missions and elementary tasks.

3.1 UAV

Each UAV is described by a set of properties. Type in sense of Fixed-wing or VTOL is not saved directly. Because construction type does not affect planning. But differences between Fix-wing or VTOL will project to properties like minimal speed and other.

Let V be a UAV. Then V is described by current position (x, y), minimal v_{min}, maximal v_{max} and optimal v_{opt} speeds in m/s. Where v_{opt} is optimal cruise speed used for planning. Also estimation of remaining flight time t_{rem} in s based on energy or fuel level.

We consider that only one sensor is installed on board of each UAV. Sensor is affecting capabilities of UAV and is actually part of it. For that reason are sensor properties included in UAV properties too, but their meaning is describe in 3.2.

$$V = [x, y, t_{rem}, v_{min}, v_{opt}, v_{max}, FOV_v, FOV_h, R_v, R_h, FPS]$$

3.2 Sensor

Properties of sensor are vertical and horizontal field of view in degrees donated as FOV_v and FOV_h respectively, resolution as number of pixels taken in vertical R_v and horizontal R_h direction and frame rate as FPS.

$$FOV_v, FOV_h \in < 0, 180 >$$

$$R_v, R_h \geq 0$$

The sensor is pointed towards the ground in such way that vertical direction is aligned with flight direction and horizontal is from left to right in compare to flight direction of UAV (Figure 3.1).

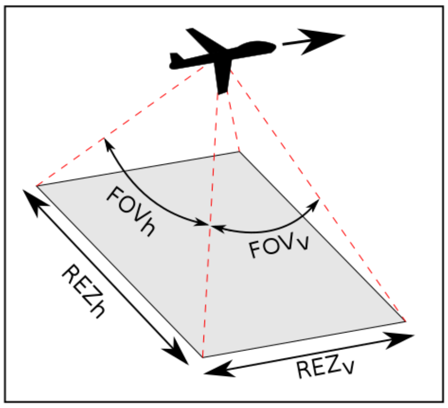


Figure 3.1. Sensor properties and orientation

3.3 Environment

Terrain in operation area is represented as a height map. And obstacles or no-fly zones can be placed in the operation area. Use of polygon to define no-fly zone is straightforward, but rectangular shape of zones is used for simplicity. The location of upper-left corner is given by (Z_{j1}, Z_{j2}), and lower-right corner by (Z_{j3}, Z_{j4}). Together, these two pairs make up the j^{th} row of matrix Z.

3.4 Missions

In this section, formalization for high-level missions is introduced. Missions formalization at this level defines the form of input that is used by the operator to input new missions to the system.

3.4.1 Mapping

Let M be a mapping mission. Rectangular bounding box aligned with axes is created around area of interest which is defined as polygon. Bounding box have lower-left corner given by (x_1, y_1), and upper-right corner by (x_2, x_2). Also required map resolution REZ is given in px/cm.

$$M = [x_1, y_1, x_2, x_2, REZ]$$

3.4.2 Search

Let S be a search mission. Rectangular bounding box aligned with axes is created around area of interest which is defined as polygon. Bounding box have lower-left corner given by (x_1, y_1), and upper-right corner by (x_2, x_2). Required map resolution REZ is given in px/cm. And search start time t_{start} is given as tiestamp in $[ms]$.

$$S = [x_1, y_1, x_2, x_2, REZ, t_{start}]$$

3.4.3 Tracking

Let T be a tracking mission. Then last known position of target is given as (x, y). Estimations of target's minimal and maximal speeds are given as v_{min} and v_{max} in $[m/s]$. And time $t_{deadline}$ until which can be target found on given position is given as tiestamp in $[ms]$ to help with prioritization of this task.

$$T = [x, y, v_{min}, v_{max}, t_{deadline}]$$

3.4.4 Patrolling

Let P be a patrolling mission. Patrol route is given as a list of way-points $wp_i = (x_i, y_i)$ and height h. Length of period is given as t_{period} in $[s]$.

$$P = [(wp_1, \ldots, wp_n), h, t_{period}]$$

Where $i \in 1 \ldots n$ and n is number of way-points.

3.4.5 Measurement

Let ME be a measurement mission. Rectangular bounding box aligned with axes is created around the area of interest which is defined as polygon. Bounding box have lower-left corner given by (x_1, y_1), and upper-right corner by (x_2, x_2). Spacing d between measured points in square grid and their height h are defined in $[m]$. Duration t of single measurement is given in $[s]$.

$$ME = [x_1, y_1, x_2, x_2, d, h, t]$$

3.5 Elementary tasks

In this section formalization of elementary tasks is introduced. Elementary tasks serve as units of work. These tasks can be assigned to particular UAV. Each high-level task mentioned in the section above is decomposed to set of these elementary tasks.

3.5.1 Flight

Flight in the straight line from point A to point B is a simple but useful task. Most bigger high-level missions described in scenarios 2.1 as mapping, patrol and others can be broken apart to set of elementary flight tasks.

Elementary flight task E_f consists of information about two points given as $A = (A_x, A_y)$ and $B = (B_x, B_y)$, flight height h in m and time window when it can be executed given as $t_{delay} \geq 0$ and $t_{deadline} \geq 0$ in s where $t_{deadline} = 0$ means no deadline. Boolean value $reverse$ determine whether E_f can be completed also in reverse direction from point B to point A. E_f can be completed in reverse direction only if $reverse = true$.

$$E_f = [A_x, A_y, B_x, B_y, h, t_{delay}, t_{deadline}, reverse]$$

E_t is considered as successfully completed if in time window bounded by t_{delay} and $t_{deadline}$ some UAV fly in given height h from A to B or from B to A if $reverse = true$.

3.5.2 Visit

Elementary visit task E_v is defined as coordinates (x, y), height h, duration $t_{duration}$ and time window when it can be executed given as $t_{delay} \geq 0$ and $t_{deadline} \geq 0$ in s where $t_{deadline} = 0$ means no deadline.

$$E_v = [x, y, h, t_{duration}, t_{delay}, t_{deadline}]$$

E_v is considered as successfully completed if in time window bounded by t_{delay} and $t_{deadline}$ some UAV loiters at given coordinates for $t_{duration}$. Loiter behavior is not defined and do not constrain the type of UAV assigned to this task. Loiter can be implemented as hovering for UAV with VTOL capabilities and circling for others.

3.5.3 Tracking

Tracking from task assignment point of view is a basic allocation of UAV which is capable to get to last known location of the target in reasonable time (before deadline). Task definition also includes information about target min/max speeds estimates and only UAVs that can handle such speed range can be assigned to the task. So tracking task is used as a test case for implementation of constraints on UAVs. Used implementation is general and can be used for many other constraints on different tasks (e.g. minimal resolution of camera).

Elementary tracking task E_t is defined as coordinates (x, y), minimal v_{min} and maximal v_{max} speeds of target in m/s and deadline $t_{deadline} \geq 0$ in s.

$$E_t = [x, y, v_{min}, v_{max}, t_{deadline}]$$

E_t is considered successfully completed if some UAV V which satisfy constraints

$$V.v_{min} \leq E_t.v_{min}$$

$$V.v_{max} \geq E_t.v_{max}$$

is at given coordinates before $t_{deadline}$ and can start autonomous tracking. That means that this UAV does not have scheduled any task after this one.

Chapter 4
State of the art

Many studies have been conducted in the field of mission planning and control. But most of them are closely focused on single mission type like mapping [2], target tracking [4] or patrol and surveillance [5]. And many publications are focused on the military use of UAVs with scenario covering the sequence of search, target identification, attack and hit validation [6–8]. But situation is worse with general approaches or systems that support a wider range of missions.

General situation with not closely specified tasks is considered in [9]. But more or less brute-force approach for centralized solving of goal assignment, resource allocation, and trajectory optimization is presented. [9] proposes only partial recalculation for faster reactions to dynamic changes. That leads to less optimal solutions but for small changes in big problems usually still good enough.

In [10] a receding horizon control scheme is proposed for dynamic vehicle trajectories. Receding horizon is approach used to solve control problems for which feedback is hard or impossible to obtain. The based idea of receding horizon is in rapid re-solving of a smaller optimization problem for a short-term plan. Length of plan vary during execution and depend on the situation. Mainly distances between vehicle and targets are used in the calculation of length to the next plan horizon. This possibly real-time approach shows solutions surprisingly close or near-identical to solutions obtained by brute-force approach similar to [9].

In-depth study of UAV health is presented in [3]. Dynamic changes in health vector which effecting UAVs capabilities are considered. Presented framework is also capable of handle wide range of tasks including tracking, search, payload drop-off or communication relay. Markov Decision Processes are used in high-level planning and execution level is implemented with use of learning-capable adaptive controllers. The close feedback loop is created between these two levels to enable anticipation of failures at the planning level.

Some decentralized algorithms for task allocation have been developed in effort to increase the mission range and remove single point of failure. Which are two main downsides of centralized approaches. In decentralized algorithms, situational awareness has to be same among all agents and any inconsistencies in the situational awareness might cause conflicting assignments. Thus, decentralized algorithms generally make use of consensus algorithms to converge on a consistent situational awareness before performing the assignment. But convergence to situational awareness may take a significant amount of time especially in networks with lower bandwidth.

To deal with this problem CBBA (Consensus-Based Bundle Algorithm) algorithm is presented in [11]. CBBA produce conflict-free solutions independent of inconsistencies in situational awareness. This is achieved by a combination of consensus-based and auction algorithms. Reasonable optimality is mathematically proven and fast convergence validated by experiments for CBBA algorithm in [11].

Later several modified versions of CBBA were proposed. In [8] algorithm based on CBBA that can handle heterogeneous UAVs is presented. It uses scoring matrix, which reflects the heterogeneity between the UAVs and targets. The scoring matrix is then optimized by episodic parameter optimization that utilizes reinforcement learning.

Another extension of CBBA that address problem of cooperation is proposed in [12]. Cooperation has proven to be a very challenging problem. Only variant for the cooperation of two UAV is considered in two variants soft-constrained cooperation preference and hard-constrained cooperation requirement. Task decomposition with associated scoring changes and task elimination protocols where introduced to deal with cooperation.

Another option is to use some general planning approaches. In recent years, automatic planning languages have been extended to support numerical expressions and temporal planning problems. Options offered by PDDL, as one of such languages, for planning in UAV domain are explored in [13–14] and this thesis uses a similar approach.

Chapter 5
PDDL planners

We decided to create PDDL domain and use some of the already existing planners for actual assigning of elementary tasks to UAVs and planning of the whole operation.

But selection of suitable planner is quite challenging. Our domain requires some advanced PDDL features except basic STRIPS and typing which are supported by almost all of available planners.

The key features are fluents for numeric calculations required to describe speeds and distances. Durative actions (introduced in PDDL 2.1 [15]) are another obvious need since flying around take some time and this is a simple way to describe it and also efficient to solve [1]. Finally, our domain also requires timely initiated predicates (introduced in PDDL 2.2 [16]) because of time constraints in some missions like patrol that needs to be executed in some time window.

Unfortunately, high requirements are not the only problem. The first step is to find some implemented and ready to use planners. But download and run procedure is not always straightforward. I came across few not compilable source codes and even one planner that runs but is not able to parse input files. Fortunately, GitHub project itSimple[2] provides a quite extensive list of planners and also include compiled and ready to use executables. Next problem is documentation of available planners that is weak if any. Often there is no other chance to find out capabilities of planner then download and try.

After testing on several domains with different levels of requirements each planner is classified as a basic, numeric or temporal planner.

Based on test results and more detail comparison of planners that fulfill all requirements the OPTIC planner is selected and used in this thesis. Overview of test results and comparison of suitable planners are below.

5.1 Basic planners

Basic planners are able to solve basic PDDL problems which require STRIPS and typing. These planners are usually older, simple and robust but are not even close to our requirements. In this category are following planners BlackBox, FF, MaxPlan, Plan-A and yahsp.

5.2 Numeric planners

Numeric planners add different levels of support for fluents and some of them also basic support for durative actions. Almost all of them also provides full support for `:adl` which is rare and limited feature between temporal planners mentioned below. Overview of planners and their capabilities is presented in Table 5.1.

[1] In the first experiments and attempts to model domain with this behavior I actually used only basic actions and fluents, but it was extremely slow event on really small problems with one or two UAVs.

[2] `https://github.com/tvaquero/itsimple/blob/master/README`

Planner	:stripes	:typing	:adl	:fluents	:durative-actions
Metric-FF	OK	OK	OK	OK	X
hsp-p	OK	OK	OK	OK	OK
lprpg	OK	OK	X	PARTIAL	X
Mips-xxl	OK	OK	OK	OK	OK
sgplan6	OK	OK	OK	OK	OK

Table 5.1. PDDL features supported by numeric planners.

5.3 Temporal planners

Temporal planners focus on durative actions and other time related features like continuous numeric effects, time initial predicates and other. Comparison of supported features is in Table 5.2. Because differences in supported PDDL features are minimal last two rows of Table 5.2 are added with additional information about the behavior of planner.

Only LPG planner is not capable to solve problems in our domain, because it does not support timed initial literals. Other planners in this category can handle all requirements of our domain.

Feature	LPG	LPG-TD	POPF	OPTIC
:fluents	OK	OK	OK	OK
:adl	P+	P+	P	P
:negative-preconditions	OK	OK	X	X
:durative-actions	OK	OK	OK	OK
:timed-initial-literals	X	OK	OK	OK
:duration-inequalities	OK	OK	OK	OK
Continuously provides best yet found solution	YES	YES	NO	YES
Is deterministic	NO	NO	YES	YES

Table 5.2. PDDL features supported by temporal planners. P stands for partial and + for better.

5.3.1 LPG

The official description on web[1] states: ”LPG (Local search for Planning Graphs) is a planner based on local search and planning graphs that handles PDDL2.1 domains involving numerical quantities and durations.”

LPG was first of more capable planners which was used in some initial domain experiments. Unfortunately, it does not support time initiated literals so when they were added LPG could not be used anymore.

Even though LPG do not fulfill all requirements it has several nice features. First is fairly wide support of `:adl` including `:negative-preconditions` which is not common between temporal planners. And next one is a continuous output of best yet found solution.

[1] `http://lpg.unibs.it/lpg/`

5.3.2 LPG-td

Short description on top of official web[1] says: ”LPG-td is a new version of LPG that extends and improves version 1.2, and that took part in the 4th International Planning Competition, 2004. LPG-td is an extension of LPG to handle the new features of the standard planning domain description languages PDDL2.2: `Timed initial literals` and `derived predicates`.”

Unlike OPTIC or POPF planners below this planner can handle negative preconditions and has better `:adl` support overall. It is true that in most cases negative preconditions can be easily removed by introducing a new predicate, but use of simple negation when writing a domain for LPG-td is a nice benefit.

LPG-td also print the best plan yet found during the planning process. Which is another benefit especially if planning time is constrained. Because it is still possible to retrieve some less optimal plan event if planning is interrupted prematurely suppose that some plan was found before the interruption.

On the other hand, LPG-td is not deterministic. Each time a different solution in different quality and search time is returned. These quality and time differences can be large and assumption that longer planning means better quality does not apply in this case. One run can return a reasonable solution in few seconds and the second run take minute and return solution twice that long. So extreme differences are rare but occurred several times.

From reliability point of view, LPG-td is not the best. Some Italian error message followed by segmentation fault occurred few times when solving bigger problems. And some unsolvable problems lead to an empty solution or infinite loop of messages

```
found worse quasi-solution. Restart using stored quasi-solution
```

5.3.3 POPF

”POPF supports a substantial portion of PDDL 2.1 level 5, including actions with (linear) continuous numeric effects and effects dependent on the durations of the actions. Its support for ADL, though, is limited: in the general case, it cannot handle negative preconditions, disjunctive preconditions, conditional effects etc. As a compromise, it will attempt to preprocess ADL preconditions and effects into STRIPS-like conjuncts, through grounding existential quantifiers, and using some simple elimination rules for static facts), which allows it to support a few useful PDDL constructs.”[2]

From behavior stand of view, POPF is the complete opposite to LPG-td. It returns only the final solution but is deterministic.

Solutions provided by POPF for some small problems are not optimal but it generally finds solutions with better quality then LPG-td on bigger ones.

In comparison to LPG-td is POPF more reliable and stable especially on bigger problems. No crashes or loops and if no solution is found it just simply print `Problem unsolvable!`. On the other side waiting for the only final solution of bigger problems can cause problems in some time sensitive applications.

5.3.4 OPTIC

OPTIC (Optimising Preferences and Time-Dependent Costs) is planner developed by the same group as POPF above and according to one of the authors Dr. Amanda

[1] http://lpg.unibs.it/lpg/
[2] https://nms.kcl.ac.uk/planning/software/popf.html

Coles in [1] ”OPTIC, Is a planner capable of reasoning with the full range of PDDL 3 preferences, including temporal preferences.”

OPTIC is built on extension of POPF and inherit some of characteristics including limited `:adl` support and deterministic behavior while adding support for new PDDL 3 features. But PDDL 3 features have not been tested because they are not required in our domain.

OPTIC is much more chatty in output than other planners. In addition to standard syntax errors, it also prints warnings about missing requirements in the domain and some problem analysis, heuristic and pruning info. This additional info can be very useful when debugging domain or trying to find out why is problem unsolvable.

One important change to OPTIC behavior is that after the first solution is found it continues in search for better one similarly to LPG and LPG-TG. But OPTIC do not have any build in time or memory limit. This means that it needs to be terminated externally. Use of bash command ulimit is suggested on web[2] to solve this situation. Limitation to 30 minutes of CPU and 4GiB of memory is shown below.

```
ulimit -t 1800
ulimit -v 4000000
./optic-clp domain.pddl problem.pddl
```

In comparison to LPG-td and even POPF, it seems that OPTIC is more reliable and faster on bigger problems. I have encountered several problems where LPG-TD and POPF finished with some error or claimed that problem is unsolvable but OPTIC had no problem to find a valid solution.

Also quality of even the first solution is same or better as solution found by POPF and better than an average of several LPG-TD runs (which give different solutions every time). It should be noted that first solution from OPTIC is used because it does not have build in stop mechanism and also on our test problems next better solution was never found even after long period of time (the first solution in few seconds then over half hour of waiting). But in problems created in simulator OPTIC often managed to find two or three solutions in given time (10-20 seconds).

[1] https://nms.kcl.ac.uk/amanda.coles/software.html
[2] https://nms.kcl.ac.uk/planning/software/optic.html

Chapter 6
System architecture

The system allows dynamic additions and removals of missions to operation by the operator at any time. When such change occurs replanning is executed to reflect changes.

The system can be split into four module if AgentFly simulator which provides world simulation, visualization and simulation of user input is considered as one module. Model module holds all information about current state of UAVs and missions. Model is also responsible for all data needed for next replanning. Planning module is responsible for asynchronous replanning. The controller module is then responsible for handling all communication between system and outside world and decisions when to replan. The last module is AgentFly simulator which in our case replaces outside world. Figure 6.1 shows all modules and their relationships.

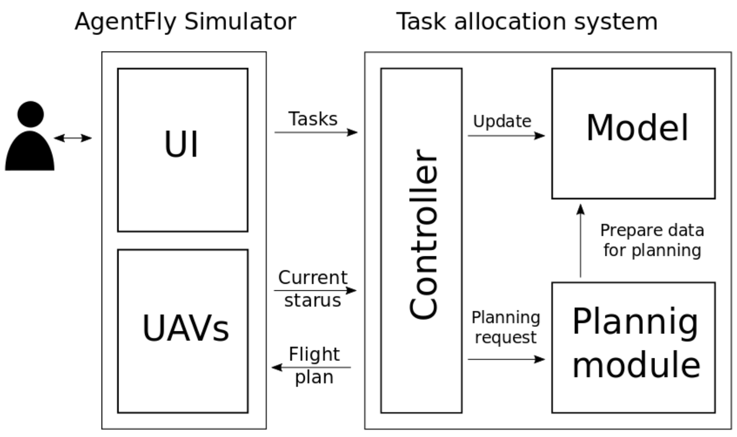


Figure 6.1. System architecture.

6.1 Model

Model and its data structure is a backbone of the whole system around which all other parts are built. Model holds and encapsulates lists of all UAVs and missions, provides access to elementary tasks and also some global variables and constants like current simulation time.

The model represents current state of operation. That means it keeps track of the progress of tasks as well as currently executed actions for UAVs. But includes also data needed for next replanning.

Data needed for replanning are for example delays and deadlines in elementary tasks stored as plan relative time variable which is covered in more detail in 7.3.2. Or estimations of UAV position and finished tasks at time which corresponds to the beginning of next plan. Model not only include such data but it is also responsible for their preparation and updates because it already has all necessary data needed for these computations in description of the current state.

6.2 Planning module

Planning module is responsible for the asynchronous execution of planning. Online planning when UAVs are already in the air is a complex problem and has three main challenges, task interruption, timeline differences and asynchronous execution. How are these challenges overcome is described in 7.3.

When replanning request from controller is received, planner reads data from model and creates PDDL problem. Then OPTIC planner is used to solve prepared problem and when a solution is parsed to a new plan this plan is returned through previously provided callback back to controller.

The plan is represented as a list of plan elements for each UAV stored in the map. Plan element can represent elementary task, move action or waiting.

6.3 Controller

The controller holds references to model, planner and current plan. The controller decides when to replan, sends replanning requests to planner, updates plan when replanning is finished and handle all communication with AgentFly simulation. Here is an overview of all actions with brief descriptions.

Registration of new UAV. Reference to the pilot entity in simulation is stored. This reference is later used for sending flight plan updates. Description of UAV is pulled from simulation, new vehicle model is created and added to operation model. Replanning is not started after new UAV is registered because simulation calls this method before actual spawn so the current position cannot be retrieved. Also, none of prepared scenarios add nor remove UAV during execution.

Controller forwards requests from the operator for adding or removing mission to model and initiates replanning after some mission is added or removed.

Controller updates flight plans of individual UAVs in simulation when replanning is finished and the new plan is returned by planner. How plan is transformed and send to UAV is described in 7.4.

Controller also listens to completion of way-points. Information about currently executed plan element is updated when UAV reach some way-point. If completed way-point is ending way-point of currently executed elementary task, then this task is marked as completed.

In some cases, UAV's flight plan can be altered in reaction to way-point completion. Waiting loop is extended if UAV currently do not have any plan assigned. This approach is inspired by the concept of layered planner in [17] where several planners are layered on each other based on their priority. For example, loiter planner create waiting loop while plan is not available and execution of planned task can be overridden by collision avoidance planner. Also if currently executed task is tracking new random way-point is added to simulate tracking behavior.

6.4 AgentFly simulator

AgentFly simulator replaces whole outside real world. That includes world simulation with UAVs and operators which are replaced by pre-prepared scenarios to simulate input of new missions. AgentFly simulator is also used for visualization.

In order to task allocation system work with UAV simulation, only minimal modifications in AgentFly simulator were required. Simulator has to notify controller about two events. First is creation of new UAV. This is done by one simple call of apropriate method from pilot entity initialization. And second is notification when UAV reaches a way-point. This is also one simple call from Behavior class where flight plan is actually executed.

Simulation of human input took a bit more effort. The ScenarioPlayer interface has to be implemented and XML configuration created and added to run configuration of the project for each scenario. But most of XML with simulation and world configuration can be reused. Only scenario configuration needs to be changed. Reference to correct ScenarioPlayer implementation needs to be edited for each scenario as well as configuration, starting position and initial flight plan for each UAV.

Chapter 7 Implementation

This chapter includes description of key parts of implementation.

7.1 Decomposition

High-level and complex missions can be decomposed to set of elementary tasks. Only small number of elementary tasks types can cover a wide range of missions. Types of elementary tasks used in our domain are defined in 3.5. Delays and deadlines in elementary tasks have to be in plan relative timeline as described in 7.3.2.

Decomposition of tracking mission is the easiest one. Tracking task is directly mapped to elementary tracking task and only deadline has to be converted to plan relative time. Decomposition of more complex task is described below.

7.1.1 Mapping

Decomposition of mapping mission M is done similarly to construction of flight path in [2]. The area is divided to set of paralleled straight strips aligned with the longer side of mapping area. Width of strip and height of flight is calculated from $M.REZ$ and sensor properties of UAV V. Suppose that sensors on all UAV are the same.

$$width = M.REZ * V.R_h$$

$$height = \frac{width}{2 * tan(\frac{V.FOV_h}{2})}$$

Each strip is divided into equally long parts with length as close to $FLIGHT\_LENGTH$ constant as possible. In test scenarios $FLIGHT\_LENGTH = 200$ m. Each strip part is then mapped to one elementary flight task which can be executed in both directions and has no time delay nor deadline.

7.1.2 Search

Decomposition of search mission S is almost same as mapping mission. Only difference is that UAV should fly in formation, to prevent target from undetected move to already searched area. Translated to elementary tasks area is divided to strips same way as mapping. But all stripes have to be searched in the same direction at same time. So whole strip is considered one elementary flight task which can be executed only in direction from A to B. Delay equals to $S.t_{start}$ converted to plan relative time and deadline is set to delay increased by execution time of the slowest UAV.

7.1.3 Patrol

Elementary flight task between every two subsequent way-points of patrol mission is created for each period. Elementary tasks can be executed in both directions and have delay and deadline equal to start and end of period converted to plan relative time. In mission decomposition and each subsequent preparing for replanning elementary tasks for periods which starts before t_{limit} are added.

$$t_{limit} = currentSimulationTime + PLAN\_AHEAD\_LIMIT$$

Both variable $currentSimulationTime$ and constant $PLAN\_AHEAD\_LIMIT$ are part of model.

7.1.4 Measurement

Rectangular area of interest of measurement mission ME is covered by individual measurement in square grid with origin at lower left corner $(ME.x_1, ME.y_1)$ and length of edge $ME.d$. Same height $ME.h$ and duration $ME.t$ is set for all measurements. Each individual measurement is then represented by one elementary visit task which has no time delay nor deadline.

7.2 Mapping to PDDL domain

Use of PDDL is composed of three parts domain, problem and solution. Domain defines object types and properties which can be used to describe particular state. Domain also includes definitions of actions which can be taken to change these properties. Problem file includes a list of object, description of initial and goal state and optionally metric used for solution comparison. Only expressions defined in the domain can be used in problem definition. Domain and problem definition are then given to PDDL planner. Planner search for a solution in form of sequence of actions defined in domain optimized by number of actions or provided metric.

PDDL domain for solving problem of elementary tasks allocation to UAVs is created. This domain has to be able to describe UAVs and each type of elementary task. It also has to know how to move UAVs and execute elementary tasks. This give baseline of how domain looks like.

This domain defines three object types vehicle, task and location. Set of predicates and function is then used to define object properties and relationships between these objects. And domain includes actions which describe UAV movement and each task type execution.

7.2.1 UAV and world

Figure 7.1 shows part of tasks allocation domain that provides UAV definition and allows UAV to move around. Location object is created for each point of interest in world and distance between each two locations has to be defined by `distance` function.

For UAV, object of vehicle type is created and its position is define by predicate `is-at`. Minimal, maximal and optimal speeds of given UAV are defined by `min-speed`, `max-speed` and `speed` functions respectively. Predicate `available` is used to forbid execution of more concurrent actions for one UAV. Predicate `available` with time initiated predicate feature also allow to define time which UAV need to finish the current task in non-interrupting replanning that is described in 7.3.

Move action is defined to allow UAV move from one location to another. Move is defined as durative action with duration calculated from distance and UAV's optimal speed. Purpose of `available` predicate is nicely shown here. Move action cannot start if UAV is not available and effect statement ensure that the UAV is labeled as unavailable during the entire action. Function `move-time` is a sum of durations of all move actions executed by all UAVs. Purpose of `move-time` is described later in 7.2.3.

Figure 7.2 shows part of problem that defines objects and the initial state. This problem contains two identical UAV and shows how `available` can be used to delay first planned action for one UAV while other can work.

```
(:predicates
        (available ?v - vehicle)
        (is-at ?v - vehicle ?l - location)
)
(:functions
        (distance ?a ?b -location)
        (speed ?v -vehicle)
        (max-speed ?v -vehicle)
        (min-speed ?v -vehicle)
        (move-time)
)
(:durative-action move
    :parameters (?v - vehicle ?from ?to - location)
    :duration   (= ?duration (/ (distance ?from ?to) (speed ?v)))
    :condition  (and
                    (at start (available ?v))
                    (at start (is-at ?v ?from)))
    :effect (and
                (at start (increase (move-time) ?duration))
                (at start (not (is-at ?v ?from)))
                (at start (not (available ?v)))
                (at end (is-at ?v ?to))
                (at end (available ?v))))
```

Figure 7.1. Part of domain with UAV and world description.

```
(:objects
    uav1 uav2 - vehicle
    a b - location)
(:init
    (is-at uav1 a)
    (available uav1)
    (= (speed uav1) 8)
    (= (min-speed uav1) 0)
    (= (max-speed uav1) 10)

    (is-at uav2 b)
    (at 30 (available uav2)) ;; 30s delay
    (= (speed uav2) 8)
    (= (min-speed uav2) 0)
    (= (max-speed uav2) 10)

;; distance in both directions need to be defined
    (= (distance a b) 100)
    (= (distance b a) 100)
)
```

Figure 7.2. Part of problem with UAV and location initialization.

7.2.2 Elementary tasks

Figure 7.3 shows part of tasks allocation domain that provides expressions for the definition of all elementary task and execution of elementary flight task type.

A task object is created for each elementary task. Type of task is then distinguished by set of predicates. Each type has its own predicate which establishes given task object as task of that type and also defines location or locations related to this task. For example predicate `(t-flight t a b)` defines task object `t` to be an elementary flight task from location `a` to location `b`. The reverse direction has to be defined by another separate statement if task `t` can be completed in both directions. Additional functions are used to define other task properties like target speed range for tracking task.

Another two predicates `can-start` and `completed` are common for all tasks. Need for `completed` in goal state description is obvious. On the other hand `can-start` is not that intuitive. It has been added later as a fix for some problems related to metric and its effect is explained in 7.2.3.

Each type of elementary tasks also has an action that defines task execution. Figure 7.3 shows an example of such action for flight task. Action `task-flight` is basically upgraded `move` action. Predicate `t-flight` is used to check that given task object and locations make together valid and existing task. Also `t-flight` needs to hold over the whole duration of action this, together with time initiated literals, is used to define required delay or deadline of a task.

Figure 7.4 shows how task is described in problem initial state and goal state. Examples of all combinations of time constraints on task are included.

```
(:objects t1 t2 t3 t4 - task)
(:init
;; no time constraint
    (can-start t1)
    (t-flight t1 a b)

;; time constraint delay
    (can-start t2)
    (at 7 (t-flight t2 a b))

;; time constraint deadline
    (can-start t3)
    (t-flight t3 a b)
    (at 20 (not (t-flight t3 a b)))

;; time constraints delay and delay
    (can-start t4)
    (at 7 (t-flight t4 a b))
    (at 20 (not (t-flight t4 a b)))
)
(:goal (and (completed t) (completed t2)
            (completed t3) (completed t4)))
```

Figure 7.4. Part of problem with task initialization and goal.

.

```
(:predicates
        (t-flight ?t -task ?a ?b - location)
        (t-visit ?t -task ?a - location)
        (t-track ?t -task ?a - location)
        (completed ?t - task)
        (can-start ?t - task)
)
(:functions
        (visit-duration ?t - task)
        (target-min-speed ?t - task)
        (target-max-speed ?t - task)
)
(:durative-action task-flight
    :parameters (?v - vehicle ?t - task ?from ?to - location)
    :duration   (= ?duration (/ (distance ?from ?to) (speed ?v)))
    :condition  (and
                (over all (t-flight ?t ?from ?to))
                (at start (can-start ?t))
                (at start (available ?v))
                    (at start (is-at ?v ?from)))
    :effect (and
                (at start (not (is-at ?v ?from)))
                (at start (not (available ?v)))
                (at start (not (can-start ?t)))
                (at end (is-at ?v ?to))
                (at end (available ?v))
                (at end (completed ?t))))
```

Figure 7.3. Part of domain with task description.

7.2.3 Metric

Few initial runs have shown that makespan metric is weak. Plans had unnecessary move actions even for simple mapping task and ideal number of UAVs (Figure 7.5). Unnecessary move actions in planed path for UAV are not removed when additional duration does not cause this path to be the longest one.

Move cost is introduced to deal with this problem. Cost of each move action is equal to its duration. And sum of all move costs is added to metric. This metric modification improves quality of found plans (Figure 7.6) but does not remove it completely even when experimented with different wights. Move cost is implemented in domain as `move-time` function.

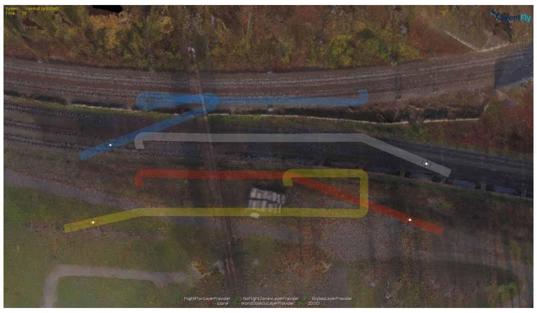


Figure 7.5. Plan without move cost.

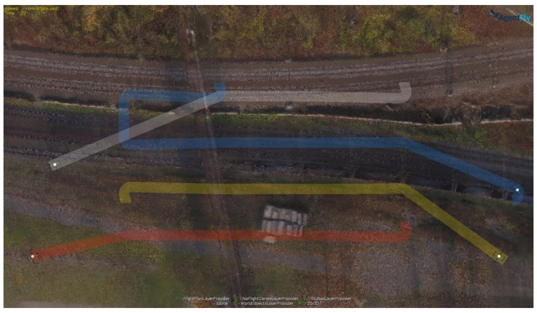


Figure 7.6. Plan with move cost.

Another problem arises when move cost is in metric. In 5 is shown that most of advanced PDDL temporal planners do not support negative preconditions. Condition in tasks that forbids executions of completed task was removed as redundant when task allocation domain was rewritten to remove negative preconditions. Keeping this condition means new predicate. But if planner search for the shortest plan it will not include same task twice. But move cost changed this logic. UAVs often used flight tasks instead of move actions to reduce move cost. Predicate `can-start` is added to fix this situation. Predicate `can-start` is essentially negation of `competed` added to action conditions. Only during actual execution of task both `can-start` and `competed` have value false. Because once execution of task starts it cannot be started again but it is not completed either. Use of `can-start` is presented in `task-flight` action (Figure 7.4).

Other costs ware considered when modifying metric. For example cost that would enforce some kind of ASAP (As Soon As Possible) or priority behavior. This would significantly improve way how tracking and patrol task are handled as described in 8. But unfortunately, this seems to be impossible within PDDL 2.2 expression capabilities. I did not find a way how to access time at which is action assigned from that action. So cost which grows over time until task is finished is not an option.

Some priority behavior in sense of task `T1` has to be started or finished before task `T2` is possible with the use of `:adl` expressions. But temporal PDDL planners usually provide only partial support of `:adl` expressions. So one needs to be cautious when using `:adl` in domains otherwise planner may not be capable of solving problems in this domain. And this kind of priority can cause other problems in situations with multiple UAVs.

Imagine a situation where one priority task is far from rest of UAVs and tasks. One UAV starts move to this task but until it gets there the rest of UAVs have to wait and cannot execute tasks even though they are ready on starting positions. This behavior may find some use in some special scenarios but is harmful in our task allocation domain.

7.3 Planning

There are three main implementation challenges connected with online planning when UAVs are already in the air.

7.3.1 Task interruption

First one is fact that UAVs already executing some actions when replanning occurs. Some UAVs currently executing action `waiting` or `move` between tasks. In these cases current action can be interrupted. And the execution of a new plan can start immediately.

But problem arises when currently executed action is some elementary task. In general, there are three options. First, new plan starts after current task will be completed. Second, current task is interrupted and new plan starts immediately. But the progress of current task is lost and the whole task has to be planned again. Example for this behavior is air quality measurement which ether finishes with result or not. In some cases there is a third option where current task is interrupted but its progress is not lost and only rest of it has to be planned again. Example of the third option is mapping where already mapped part can be saved and res can be can be completed later.

What is the optimal option for given situation depends on the priority of current task, the priority of new tasks that caused replanning, distances of this UAV and other UAVs to new tasks as well as UAV suitability for new tasks. In general, it is close to impossible to find the optimal option for each currently execution task before replanning. Which is needed because interrupted ones have to be included in that replanning.

The solution implemented in this thesis relies on parallel computation. Only first two options non-interrupting and interrupting with lost progress are considered. Two separate PDDL problems definitions are created. One that interrupts all currently executing tasks. That allows faster response of some UAV to tasks as tracking but force restart and replanning of all currently executing tasks. Second problem definition does not interrupt any currently executing elementary task. And two parallel processes with OPTIC are started to solve each problem. Fact that OPTIC planner is 32bit one thread executable that can be accessed and controlled through terminal and runs in its own process significantly simplifies parallelization. After outputs of both processes are parsed, new plans are compared and better one is chosen.

7.3.2 Timeline differences

The second implementation challenge is the difference between simulation and plan timeline.

Time did not seem to be a problem in initial part of implementation when only single planning situation wear considered and solved. But problem comes to light with controller which connected the system to simulation. Plan obtained from planner always starts at time zero and does not matter when, with respect to fixed simulation timeline, is replanning started. PDDL problem definition also has to include time values, like delays, in that floating timeline of plan. Now two independent timelines collide and original implementation was not ready for that.

Two types of time variable are introduced to deal with this situation. Absolute time is time on fixed simulation timeline and is represented as milliseconds stored in a long integer which is format used in AgentFly simulation. Plan relative time is time on plan timeline and is represented as seconds stored in double. Conversion between absolute and plan relative time is easy if simulation offset of given plan is known. Conversion formula is then

$$absolute = 1000 * relative + offset$$

Next part is to decide which of already existing time variables belong to which type and add simulation offset variables and conversions where needed.

Plan relative time is used in two places. All time variables in elementary tasks are plan relative. Because they are directly written to planning problem definition and are not read from anywhere else. Simulation offset is added to elementary tasks so it can convert variable to absolute form. But more importantly to be able to move the whole task to different plan timeline with different when next replanning occurs. Also all time variables in a plan are plan relative because they are not directly needed in absolute form and this simplifies parsing the output of planner. Simulation offset is added to plan too.

Time variables in tasks are absolute. They are an input of operator which is expected to work in simulation timeline and they are not accessed by planner during planning. Also, all time variables in the controller are absolute.

7.3.3 Asynchronous execution

The third implementation challenge is an asynchronous execution of planning.

Most operations in task allocation system like adding mission or handling of waypoint completion are short and are executed synchronously by calling simulation thread.

The only exception is planning which is long and computation heavy process so it has to be executed asynchronously.

PDDL planning problem definition cannot be created from the current situation. Because when replanning finishes part of the new plan would be already in past and therefore invalid. That is why the start of a new plan is moved to future to time t_{offset}. This t_{offset} also serves as a deadline for replanning process.

Now we need to estimate how the situation at t_{offset} looks like and create PDDL planning problem based on that. For such estimation data about all missions, elementary tasks, current plan, UAVs and their current states are needed. Model is responsible for estimation and preprocessing all data that are needed for replanning because it already has all data needed for this calculation.

Data preparation includes estimation of position and currently executing tasks at t_{offset} for each UAV based on the current plan. Mark currently not completed elementary tasks which will be completed before replanning finish. Also, some missions may want to change their set of elementary tasks (e.g. adding next period in patrol). And finally, move all time variables as delays and deadlines in elementary tasks from their current plan relative timline to new one which starts at t_{offset}.

Data preparation also has to reflect two types or replanning. Interrupting replanning is easier and all required actions are described above. Suppose that currently executing tasks at t_{offset} are included in plannig problem.

But a few changes are required for non-interrupting replanning. Currently executing tasks at t_{offset} are also marked as completed before replanning finish. Estimation of UAV position has to be set to end way-point of that task. And delay after which can be UAV considered as available in new plan is set based on t_{offset} and task end time.

Time limit for OPTIC planner has to be set so it finishes before t_{offset}. OPTIC continuously print best yet found solution to standard output. This is very important because it allows to retrieve solution after OPTIC is externally terminated. Simple bash script *solve.sh*, with a time limit in seconds and problem file as arguments, is used to wrap external termination and OPTIC as one command.

```
#!/bin/bash
ulimit -t \$1
../pddl/optic-clp ../pddl/domain.pddl \$2
```

7.4 UAV flight plan update

It is controller's responsibility to update flight plan of UAVs when new plan is created. Controller can access UAV entities in simulation via reference to *PilotAgent* in *Vehicle* class which represents UAV in model. From *PilotAgent* can be obtained *FlightPlan-Wrapper* which contain current flight plan and provide methods like remove, insert and append way-points.

New plan is converted to list of way-points in for cycle over all plan elements. Plan element can be an elementary task, move action or waiting. List of way-points cannot contain two subsequent way-points on same coordinates. So last way-point in list has to be checked before first way-point of plan elements is added. Converting flight task or move action to way-points is straightforward. Tracking task is simply one way-point corresponding to the target location when this way-point is completed the controller adds next random one to simulate auto-tracking behavior.

Unfortunately, AgentFly simulator is developed with focus on airplanes and does not support hovering or any type of wait behavior. So all waiting need to be replaced by additional way-points close to each other. And that is not much precise way to delay UAV at some way-point.

In order to work properly *FlightPlanWrapper* requires setting of plan position called unchangeable part. In other words, it needs to know what is current position and what part of the flight plan is in past and cannot be changed. It can be set anywhere and *FlightPlanWrapper* will work. But if position in past is set then *PilotAgent* refuse to accept this flight path as a new one.

This concept of unchangeable part elegantly solves situation where planner return early and beginning of new plan is in future. *FlightPlanWrapper::getPosition* with new plan beginning as argument is used to get correct unchangeable part. If new plan begins in past or *getPosition()* fails current plan position from *PilotAgent::getCurrentPlanPosition* is used as fallback.

After the unchangeable part is set. All way-points in changeable part of *FlightPlanWrapper* are then removed. And a list of new way-points appended. After all changes flight plan needs to be replanned and validated by call of *FlightPlanWrapper::plan()*. This actually execute flight path planning with respect to flight dynamics of UAV. After that *FlightPlanWrapper* can be passed to *PilotAgent* as new flight path.

Chapter 8 Experiments

Several scenarios is used to test all types of missions. The same simulation environment is used for all scenarios. World dimensions are

$$x \in < -1024, 1024 >$$

$$y \in < -643, 643 >$$

$$z \in < -0, 6000 >$$

where z axis is up direction. Some topography and texture data are loaded but do not effect simulation, no collisions are set. Also none no-flight zones are included in environment.

Three predefined UAV types are used in scenarios. Speeds are different for each type but used sensor is same for simplicity. Table 8.1 describe all predefined UAV types. Different number and type combination of UAVs as well as their starting positions are used in each scenario. All UAVs are created through XML configuration and no UAVs are added nor removed dynamically during scenarios. Entity initialization take some time so actual spawn of UAVs is execute with delay at $simulationTiem = 5$ s.

Type	v_{min}	v_{opt}	v_{max}	FOV_v	FOV_h	R_v	R_h
slow	0	7	10	70	70	640	480
normal	5	15	20	70	70	640	480
fast	10	30	35	70	70	640	480

Table 8.1. UAV configurations used in experiments.

When mission is added replanning with $t_{rep} = 15$ s time limit is started immediately. But because replanning requires access to the current position of UAVs all missions have to be added after UAVs are in simulation. This is the reason why all missions in all scenarios are created and added at $simulationTiem = 6$ s or later.

8.1 Mapping scenario

Mapping scenario shows mapping mission decomposition and tests how UAVs with different speeds are handled. Scenario includes one mapping mission and three UAVs with different speeds. Mapping area is big enough to be decomposed into more than three lines to avoid simple one to one assignment.

ID	type	x	y	color
UAV_1	fast	500	-200	white
UAV_2	normal	500	-280	red
UAV_3	normal	500	-360	yellow

Table 8.2. Mapping fleet definition.

All UAVs are spawned at $simulationTiem = 5$ s with initial wait loops. Mapping mission $M1$ is created, added and planning started at $simulationTiem = 6$ s.

$$M1 = [-450, -450, 250, 50, 0.2]$$

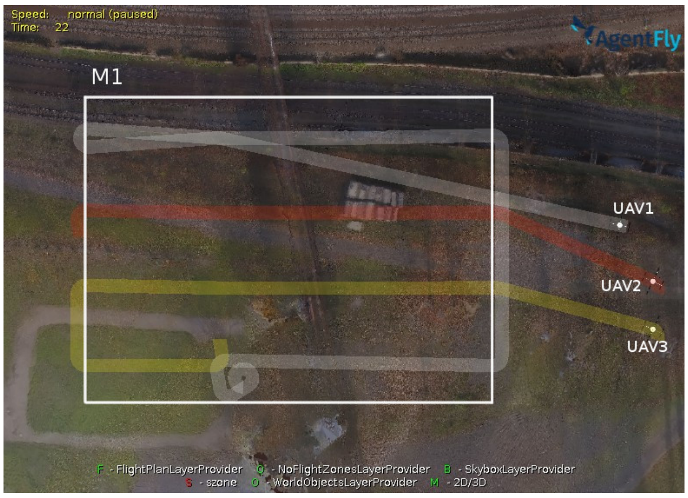

Speed: normal (paused)
Time: 22
AgentFly
M1
UAV1
UAV2
UAV3
F - FlightPlanLayerProvider Q - NoFlightZonesLayerProvider B - SkyboxLayerProvider
S - szone O - WorldObjectsLayerProvider M - 2D/3D

Figure 8.1. Mapping scenario plan.

Mapping mission $M1$ is decomposed to 12 elementary tasks in 4 lines. As expected, faster UAV_1 gets more (5 of 12) tasks assigned. Found plan (Figure8.1) looks reasonable because UAV_1 is twice as fast as other UAVs and also got more and longer move actions.

But found plan has $metric = 175$ and is definitely not an optimal one. Path of UAV_1 can be slightly improved even without changing task allocation. A simple change of flight direction for the first task of UAV_1 reduces metric by 1.4. And manually found plan has $metric = 105.5$. If decomposed lines were labeled 1-4 from the top this better plan assign lines 1-2 to UAV_1, line 3 to UAV_2 and line 4 to UAV_1.

8.2 Tracking scenario

Five UAVs of different types are present in this scenario. Constrained UAV selection is tested when two tracking missions are added. Not all UAVs are capable to complete given missions because of speed constraints.

ID	type	x	y	color
UAV_1	normal	-100	100	green
UAV_2	fast	100	100	blue
UAV_3	normal	-100	-100	yellow
UAV_4	fast	200	-100	red
UAV_5	slow	100	-150	white

Table 8.3. Tracking fleet definition.

All UAVs are spauned at $simulationTiem = 5$ s with initial wait loops. And two tracking missions $T1$ and $T2$ are created, added and planning started at $simulationTiem = 6$ s.

$$T1 = [-50, 200, 12, 18, 40000]$$

$$T2 = [100, -100, 8, 18, 40000]$$

Positions of UAVs and missions, combination of UAV types and parameters of missions are chosen to test most of constraints and optimality of assignment. $T1$ can be assigned to normal or fast UAVs and test optimality of assignment. UAV_1 is the closest UAV to $T1$, but UAV_2 can get there faster. $T2$ can be assigned only to the normal type of UAV and tests compliance with the allocation constraints. Both too fast UAV_4 and too slow UAV_5 can get to $T2$ faster then UAV_3 but can be assign.

The system handled this simple scenario without any problems. Figure 8.2 shows found plan which is optimal and do not violate any constraints.

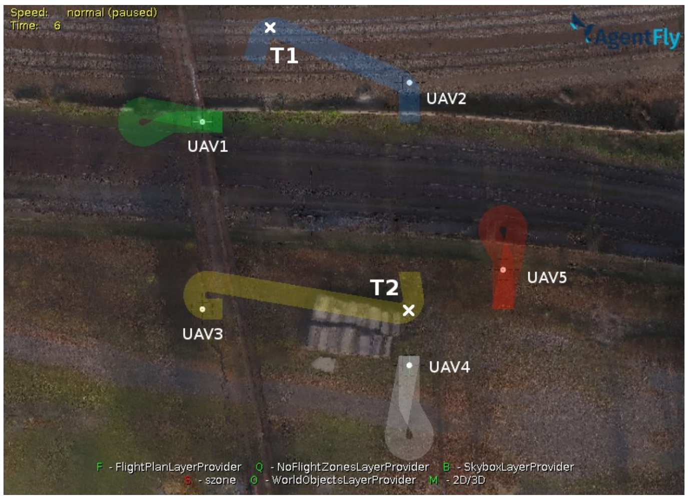


Figure 8.2. Tracking scenario plan.

This scenario has demonstrated that system is capable to deal with target tracking. But used formalization of tracking task is not ideal for real world applications. The main problem is concept of deadline. Deadline represents time until which target can be found close enough to given position and tracking can start. Only estimations of deadline are available and with each second danger of losing the target grows in the real world. This lead to need to start tracking as soon as possible. But 7.2.3 explain that system does not offer this kind of optimization because PDDL does not provide tools to achieve this behavior.

8.3 Search scenario

Search scenario contains one search mission and four UAVs spread around the whole map. Planning and timing for flight in formations are tested in this scenario when UAVs closer to task have to wait for other.

ID	type	x	y	color
UAV_1	normal	500	-100	yellow
UAV_2	normal	0	0	white
UAV_3	normal	-400	-350	blue
UAV_4	normal	-500	-50	read

Table 8.4. Search fleet definition.

All UAVs are spawned at $simulationTiem = 5$ s with initial wait loops. Search mission $S1$ is created, added and planning started at $simulationTiem = 6$ s.

$$S1 = [150, -500, 550, -100, 0.2, 70000]$$

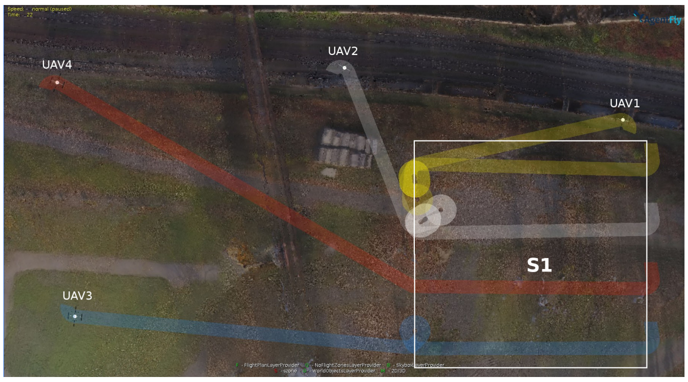


Figure 8.3. Search scenario plan.

This scenario demonstrates that system is capable to plan a flight in formation to cover the search area. Figure 8.4 shows timeline of found plan where all tasks are aligned. Shown timeline is plan relative so tasks do not begin at 70 s as defined in $S1$ but at 49 s. This corresponds to simulation offset of this plan relative timeline which is

$$S1_{added} + t_{rep} = 6 + 15 = 21$$

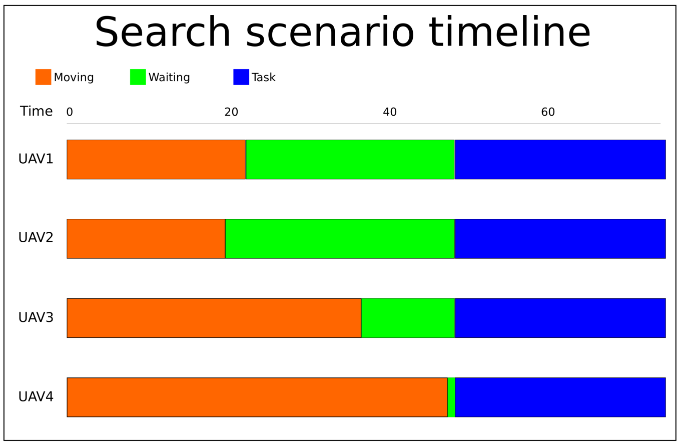


Figure 8.4. Search timeline. Displayed in plan relative time.

But usability of used approach in real world applications is even more limited than in case of tracking. Similarly, as in tracking case, search tasks are usually intended as priority tasks that begin as soon as possible. Another problem is set of elementary tasks types that is narrow and very general. This set allows simple PDDL domain with faster planning while covering a wide range of missions but causes situation that only tools to synchronize more elementary tasks are delay and deadline constraints. That leads to a group of tasks with fixed time positions which planner only haves to obey and cannot optimize. Also, a danger of unsolvable planning problem grows with task fixed in time because when the start time is set to early UAVs may not be able to get there in time.

8.4 Patrol scenario

Patrol scenario includes one patrol mission and two UAVs. This tests cooperation of UAVs and time constraints that come with periodical execution of tasks in patrol mission.

ID	type	x	y	color
UAV_1	fast	500	-200	white
UAV_2	fast	500	-280	red

Table 8.5. Patrol fleet definition.

All UAVs are spawned at $simulationTiem = 5$ s with initial wait loops. Patrol mission $P1$ is created, added and planning started at $simulationTiem = 6$ s.

$$P1 = [((350, 55)(350, -45)(750, -95)(750, 0)(350, 55)), 100, 50]$$

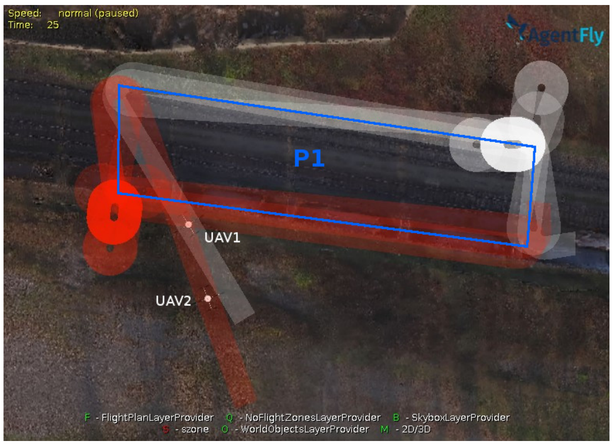

Speed: normal (paused)
Time: 25
AgentFly
P1
UAV1
UAV2
F - FlightPlanLayerProvider Q - NoFlightZonesLayerProvider B - SkyboxLayerProvider
S - szone O - WorldObjectsLayerProvider M - 2D/3D

Figure 8.5. Patrol scenario plan.

This scenario demonstrates that system is capable to handle patrol for multiple co-operating UAVs (Figure 8.5). Timeline of plan (Figure 8.6) shows that all elementary tasks are planned correctly within their periods but also reveals unexpected behavior of planner. Planner exploits a gap in patrol mission definition when optimizing the plan. It saves time by assigning the same segment of patrol route twice in row around the edge of each period. This behavior is valid within given definition of patrol because the first repetition is in ending pored and the second one starts in next period. But such behavior was not intended and I expect that in real world situations unwanted. The solution to this problem in future work can be redefinition of patrol task or cost modification to enforce execution of segment as soon as possible.

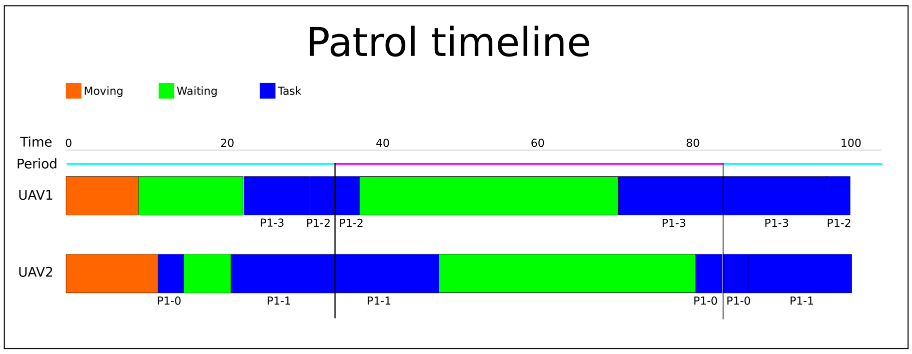


Figure 8.6. Patrol initial plan timeline. Displayed in plan relative time.

8.5 Measurement scenario

Measurement scenario consists of one measurement mission and three UAVs. Similarly to mapping scenario one UAV is faster to test handling of UAVs with different speeds. Measurement mission can be easily decomposed to a high number of elementary task by change of density. A higher number of elementary tasks in this scenario is used as a stress test for the PDDL planner.

ID	type	x	y	color
UAV_1	fast	0	-50	yellow
UAV_2	normal	-50	-175	white
UAV_3	normal	50	-175	red

Table 8.6. Measurement fleet definition.

All UAVs are spawned at $simulationTiem = 5$ s with initial wait loops. Measurement mission $ME1$ is created, added and planning started at $simulationTiem = 6$ s.

$$ME1 = [-400, -650, 400, 250, 200, 100, 8]$$

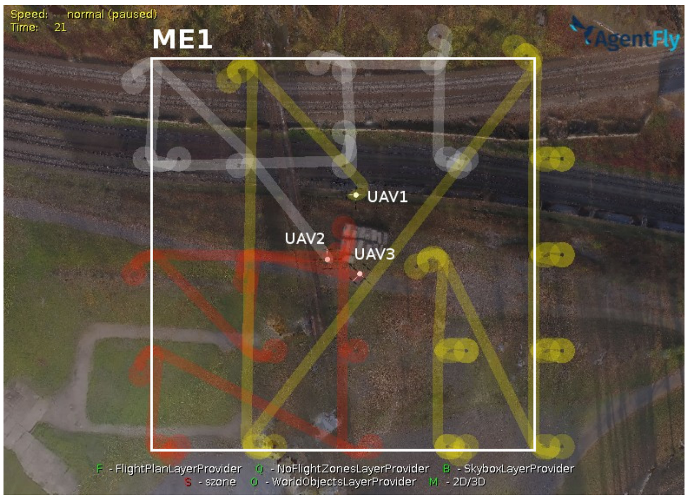


Figure 8.7. Measurement plane.

Nice division of an area to sectors can be observed in found plan (Figure 8.7). Each sector is then assigned to one UAV which is a nice result of travel time optimization. More work is assigned to faster $UAV1$ as expected but unfortunately a big part of time gain is wasted for long trips to other sectors instead of expansion of one sector.

Mission $ME1$ is decomposed to 25 elementary tasks and PDDL planer have no problem to provide a solution within given t_{rep} time limit. But experiments on modified versions with higher densities were simulated to stress test PDDL planner with larger sets of simple tasks without any time constraints. Table 8.7 show results of stress test.

Number of tasks	25	36	49	64	81
Planning time [s]	0.8	2.9	10.7	14.8	74.5

Table 8.7. Stres test results

8.6 Replanning scenario

Replanning scenario is extended variant of patrol scenario. This scenario includes one patrol mission and two UAVs same as in patrol scenario. But after while tracking mission is added.

ID	type	x	y	color
UAV_1	fast	500	-200	white
UAV_2	fast	500	-280	red

Table 8.8. Replanning fleet definition.

All UAVs are spawned at $simulationTiem = 5$ s with initial wait loops. Patrol mission $P1$ is created, added and planning started at $simulationTiem = 6$ s. And tracking mission $T1$ is added and replannig started at $simulationTiem = 60$ s.

$$P1 = [((350, 55)(350, -45)(750, -95)(750, 0)(350, 55)), 100, 50]$$

$$T1 = [300, -200, 15, 15, 0.2, 100000]$$

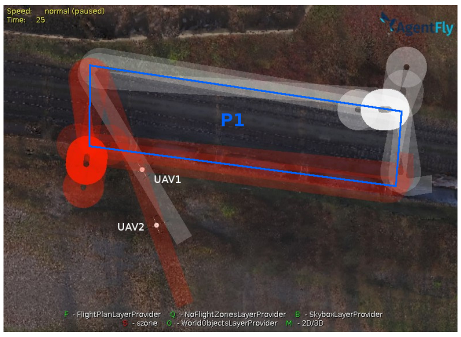

Speed: normal (paused)
Time: 25
AgentFly
P1
UAV1
UAV2
F - FlightPlanLayerProvider Q - NoFlightZonesLayerProvider B - SkyboxLayerProvider
S - szone O - WorldObjectsLayerProvider M - 2D/3D

Figure 8.8. Replanning scenario initial plan.

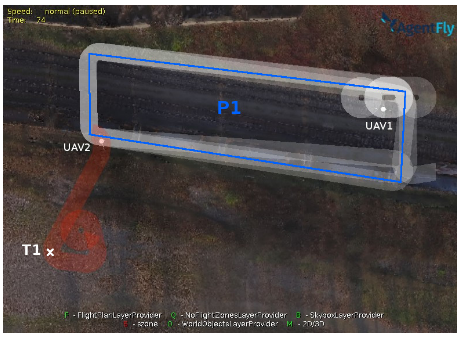

Speed: normal (paused)
Time: 74
AgentFly
P1
UAV1
UAV2
T1 x
F - FlightPlanLayerProvider Q - NoFlightZonesLayerProvider B - SkyboxLayerProvider
S - szone O - WorldObjectsLayerProvider M - 2D/3D

Figure 8.9. Replanning scenario after replanning with tracking.

There are two reasons why to extend patrol scenario. Patrol mission is one of more complex mission that includes delays and deadlines. Second one is that this scenario can be expected in real world where patrolling UAV detects intruder and new tracking mission is created.

This scenario demonstrates how system can react to dynamic change and replan already executing tasks. UAV_2 which is closer to $T1$ is assigned to it while $UAV1$ take over whole $P1$ after replanning.

Chapter 9
Conclusion and future work

This thesis presented system that can handle task allocation in a team of heterogeneous UAVs. Presented system uses decomposition of a wide range of high-level missions to only three types of elementary tasks. Task allocation problem which includes elementary tasks and UAVs is defined and solved as PDDL problem. The resulting plan obtained from PDDL planner is then transformed to list of way-points and passed to UAV where flight path is calculated. The system is implemented to work with AgentFly simulator.

Very flexible PDDL domain is proposed to deal with a team of heterogeneous UAVs. Only simple variant of domain that handles different speeds of UAVs is implemented. Tracking task serves as a test case for constraints that limit assignment of this task only to UAVs with certain parameters. More specifically constraints on minimal and maximal speed of UAV are introduced for tracking task. These constraints were successfully tested and their general and relatively simple implementation allows easy introduction of many other constraints as required resolution or payload capacity for other tasks in future.

Experiments with several scenarios in the simulation demonstrated that system can plan for a wide range of tasks including mapping, search, patrol, tracking and other. The system is also capable of handling scenarios with several different types of tasks at once.

Few limitations of PDDL have been encountered especially in expression capabilities for a metric that is used for plan optimization. For example, cost increasing with latency cannot be modeled. That leads to the cumbersome uses of time constraints as delays and deadlines in tasks like tracking where low latency is required. Exploration of PDDL successor languages or extended variants of PDDL like PDDL+, NDDL or other is suggested for future work to address this problem.

Also, search task revealed some limitations of flight in formation and cooperation caused mainly by a narrow set of too generalized elementary task. Consideration of a new type of elementary task or modification of existing ones is suggested for future work to improve support for cooperation and time synchronization between multiple UAVs.

Except improvements mentioned above, future work should be focused on support of obstacles and no-flight zones. As well as improvements in decomposition process which should be also able to reflect the heterogeneity of UAVs same as task allocation and planning.

References

[1] V. Maršálek, M. Selecký, and M. Rollo. Akční scénaře. 2018,

[2] C. Berger, M. Wzorek, J. Kvarnström, G. Conte, P. Doherty, and A. Eriksson. Area coverage with heterogeneous UAVs using scan patterns. *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2016,

[3] N.K. Ure, G. Chowdhary, J.P. How, M.A. Vavrina, and J. Vian. Health Aware Planning under uncertainty for UAV missions with heterogeneous teams. *2013 European Control Conference (ECC)*. 2013,

[4] D. Turra, L. Pollini, and M. Innocenti. Fast unmanned vehicles task allocation with moving targets. *43rd IEEE Conference on Decision and Control (CDC)*. 2004, 4280-4285.

[5] Stephen Buerger, Wendy A. Amai, Timothy J. Blada, Charles Q. Little, Jason C. Neely, and Joshua Love. The Sandia Architecture for Heterogeneous Unmanned System Control (SAHUC). 2015,

[6] Yan Jin, A. A. Minai, and M. M. Polycarpou. *Cooperative real-time search and task allocation in UAV teams.* In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*. 2003. 7-12 Vol.1.

[7] l. Xu, and U. Ozguner. *Battle management for unmanned aerial vehicles.* In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*. 2003. 3585-3590 vol.4.

[8] Dong Jun Kwak, Sungwon Moon, Suseong Kim, and H. Jin Kim. Optimization of Decentralized Task Assignment for Heterogeneous UAVs. *IFAC Proceedings Volumes*. 2013, 46 (11), 251 - 256. DOI https://doi.org/10.3182/20130703-3-FR-4038.00072. 11th IFAC Workshop on Adaptation and Learning in Control and Signal Processing.

[9] John Bellingham, Michael Tillerson, Arthur Richards, and Jonathan P. How. *Multi-Task Allocation and Path Planning for Cooperating UAVs.* In: *Cooperative Control: Models, Applications and Algorithms*. Boston, MA: Springer US, 2003. 23–41. ISBN 978-1-4757-3758-5.
`https://doi.org/10.1007/978-1-4757-3758-5_2`.

[10] C. G. Cassandras, and Wei Li. *A receding horizon approach for solving some cooperative control problems.* In: *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*. 2002. 3760-3765 vol.4.

[11] H. L. Choi, L. Brunet, and J. P. How. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Transactions on Robotics*. 2009, 25 (4), 912-926. DOI 10.1109/TRO.2009.2022423.

[12] Han-Lim Choi, A.K. Whitten, and J.P. How. Decentralized task allocation for heterogeneous teams with cooperation constraints. *American Control Conference (ACC)*. 2010, 3057-3062.

[13] Davide Dell'Anna. *Numerical and temporal planning for a multi-agent team acting in the real world*.

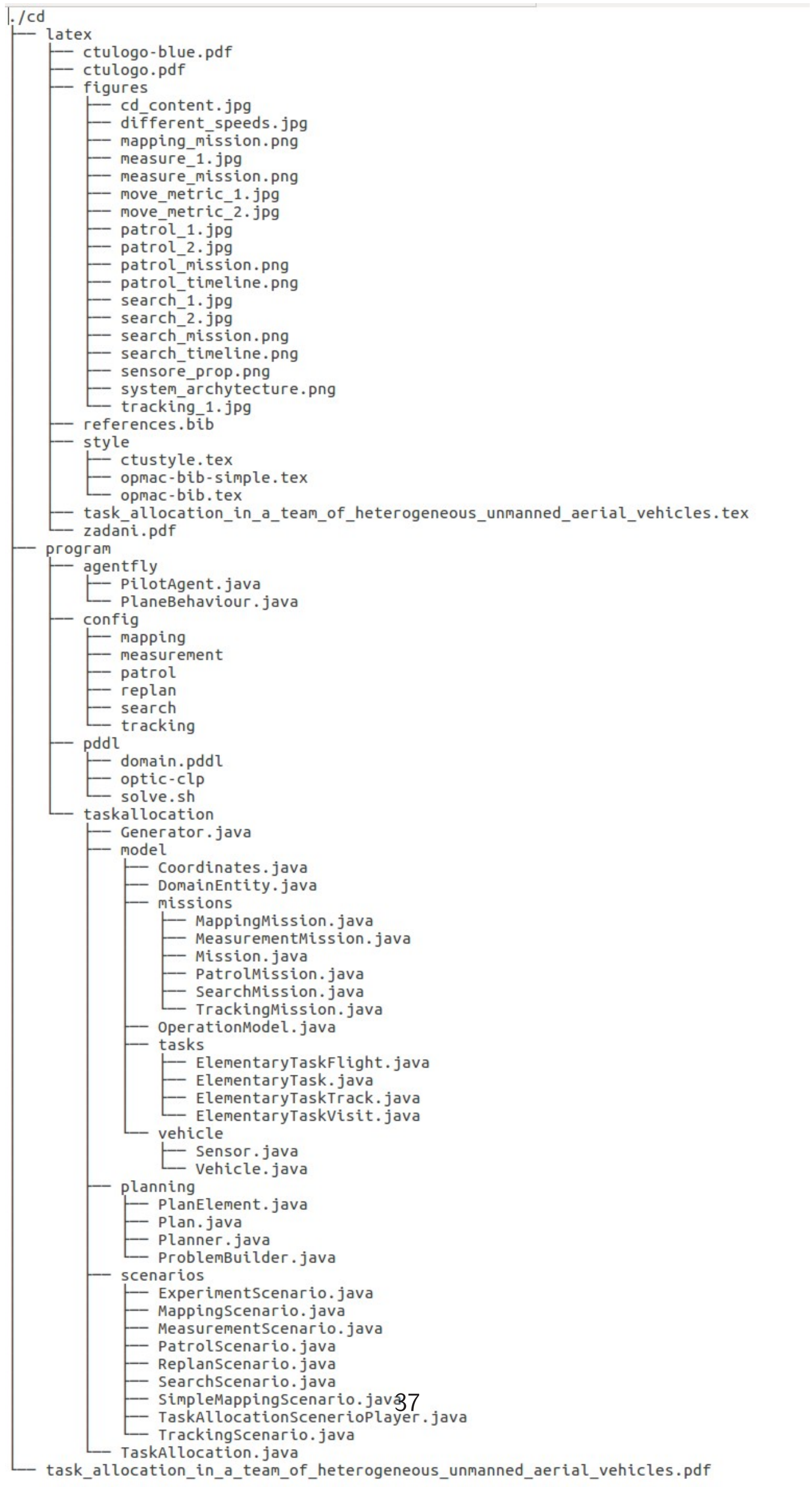
[14] Johann Uhrmann, Ruben Strenzke, and Axel Schulte. Task-based guidance of multiple detached unmanned sensor platforms in military helicopter operations. *COGIS, Crawley*. 2010,

[15] Maria Fox, and Derek Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Int. Res.*. 2003, 20 (1), 61–124.

[16] Stefan Edelkamp. PDDL2. 2: The language for the classical part of the 4th international planning competition. 2004,

[17] M. Selecký, and M. Rollo. Distributed control of heterogeneous team of autonomous uavs. *2016 Xponential an auvs experience*. 2016,

Appendix A
Content of enclosed CD



```
./cd
├── latex
│   ├── ctulogo-blue.pdf
│   ├── ctulogo.pdf
│   ├── figures
│   │   ├── cd_content.jpg
│   │   ├── different_speeds.jpg
│   │   ├── mapping_mission.png
│   │   ├── measure_1.jpg
│   │   ├── measure_mission.png
│   │   ├── move_metric_1.jpg
│   │   ├── move_metric_2.jpg
│   │   ├── patrol_1.jpg
│   │   ├── patrol_2.jpg
│   │   ├── patrol_mission.png
│   │   ├── patrol_timeline.png
│   │   ├── search_1.jpg
│   │   ├── search_2.jpg
│   │   ├── search_mission.png
│   │   ├── search_timeline.png
│   │   ├── sensore_prop.png
│   │   ├── system_archytecture.png
│   │   └── tracking_1.jpg
│   ├── references.bib
│   ├── style
│   │   ├── ctustyle.tex
│   │   ├── opmac-bib-simple.tex
│   │   └── opmac-bib.tex
│   ├── task_allocation_in_a_team_of_heterogeneous_unmanned_aerial_vehicles.tex
│   └── zadani.pdf
├── program
│   ├── agentfly
│   │   ├── PilotAgent.java
│   │   └── PlaneBehaviour.java
│   ├── config
│   │   ├── mapping
│   │   ├── measurement
│   │   ├── patrol
│   │   ├── replan
│   │   ├── search
│   │   └── tracking
│   ├── pddl
│   │   ├── domain.pddl
│   │   ├── optic-clp
│   │   └── solve.sh
│   └── taskallocation
│       ├── Generator.java
│       ├── model
│       │   ├── Coordinates.java
│       │   ├── DomainEntity.java
│       │   ├── missions
│       │   │   ├── MappingMission.java
│       │   │   ├── MeasurementMission.java
│       │   │   ├── Mission.java
│       │   │   ├── PatrolMission.java
│       │   │   ├── SearchMission.java
│       │   │   └── TrackingMission.java
│       │   ├── OperationModel.java
│       │   ├── tasks
│       │   │   ├── ElementaryTaskFlight.java
│       │   │   ├── ElementaryTask.java
│       │   │   ├── ElementaryTaskTrack.java
│       │   │   └── ElementaryTaskVisit.java
│       │   └── vehicle
│       │       ├── Sensor.java
│       │       └── Vehicle.java
│       ├── planning
│       │   ├── PlanElement.java
│       │   ├── Plan.java
│       │   ├── Planner.java
│       │   └── ProblemBuilder.java
│       ├── scenarios
│       │   ├── ExperimentScenario.java
│       │   ├── MappingScenario.java
│       │   ├── MeasurementScenario.java
│       │   ├── PatrolScenario.java
│       │   ├── ReplanScenario.java
│       │   ├── SearchScenario.java
│       │   ├── SimpleMappingScenario.java
│       │   ├── TaskAllocationScenerioPlayer.java
│       │   └── TrackingScenario.java
│       └── TaskAllocation.java
└── task_allocation_in_a_team_of_heterogeneous_unmanned_aerial_vehicles.pdf
```