

Technical report 20-020

The Distributed Bayesian Algorithm: Simulation and Experimental Results for a Cooperative Multi UAV Search Use-Case*

J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter

To cite this work, please refer to the published version:

J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter, “The distributed Bayesian algorithm: Simulation and experimental results for a cooperative multi UAV search use-case,” *Proceedings of the 11th International Workshop and Optimization and Learning in Multiagent Systems (OptLearnMAS 2020)*, Virtual conference, May 2020. doi:-

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
phone: +31-15-278.24.73 (secretary)
URL: <https://www.dcsc.tudelft.nl>

* This report can also be downloaded via <https://dpub.eu/20-020>

The Distributed Bayesian algorithm: simulation and experimental results for a cooperative multi UAV search use-case

Jeroen Fransman
Delft Center for Systems and Control
(DCSC), Delft University of
Technology
j.e.fransman@tudelft.nl

Joris Sijs
Netherlands Organisation for Applied
Scientific Research (TNO)

Henry Dol
Netherlands Organisation for Applied
Scientific Research (TNO)

Erik Theunissen
Netherlands Defence Academy
(NLDA)

Bart De Schutter
Delft Center for Systems and Control
(DCSC), Delft University of
Technology

ABSTRACT

In this work, the Distributed Bayesian (D-Bay) algorithm is applied to an autonomous search use case. Within the use case multiple unmanned aerial vehicles equipped with cameras cooperatively search an area and minimize the required time. The use case is modeled within the continuous Distributed Constraint Optimization Problem (DCOP) framework. This framework extends the (discrete) DCOP framework by allowing variables with continuous domains. Compared to similar DCOP solvers, the characteristics of the D-Bay algorithm are well suited for the use case and allow for the implementation on autonomous vehicles with limited resources (computational power, memory, and communication bandwidth). Experimental results are given and these results are used to validate a simulation environment. Within the simulation environment various scenarios are implemented. The D-Bay algorithm was able to find solutions within 3.5 % of the optimal solution with a limited amount of samples per agent.

KEYWORDS

DCOP; Bayesian optimization; UAV; Cooperative search

1 INTRODUCTION

Autonomous vehicles are used for various tasks in order to increase situational awareness. Examples include surveillance, search, patrolling, observing, and pursuit-evasion. These tasks are often represented as a mobile sensor coordination problem [27], a cooperative search problem [2], or a patrolling problem [21].

In the literature, numerous types of autonomous vehicles have been used to perform cooperative search. The reader is referred to the work of Veres et al. [23] for a systematic overview of methodologies used for the main classes of autonomous vehicles. For outdoor environments Unmanned Aerial Vehicles (UAVs) are generally used thanks to their ability to traverse a large number of types of terrain. This work focuses on the cooperative search problem in an outdoor environment performed by UAVs. In typical real-world use cases, several UAVs need to search a predefined region in as little time as possible to find particular objects or victims. In these use cases, a single operator should be able to control all vehicles in an easy and straightforward manner. Therefore, a UAV should be able to optimize its own path with respect to the paths of the other UAVs.

For this reason, search problems have been modeled within various frameworks, such as the task allocation framework, Markov decision processes, and game theory. The reader is referred to the work of Robin and Lacroix [20] for a unifying taxonomy of search related problems and a comprehensive survey of the application of a wide range of problem frameworks. The usage of UAVs introduces an implementation problem for the algorithms used to optimize the individual paths as the complications arise from limitations in communication, computation, and/or memory. For UAVs these complications become apparent as the required hardware needs to be small and lightweight to reduce negative effects on the endurance. Additionally, due to the distance between the UAVs during real-world applications, the communication capabilities to a central system are often limited. In other words, UAVs are able to communicate with nearby UAVs but not with a central system. This makes it impractical to use a centralized approach.

The DCOP framework has been applied to a wide range of problems. Some notable examples include the works of Meisels [13], Modi et al. [14], Petcu and Faltings [17], Gershman et al. [9], and Yeoh and Yokoo [25]. A problem modeled as a DCOP is based on an objective function that quantifies the collaborative goal of agents. Thanks to the focus on the distributed nature of the underlying problems, the solvers for DCOP are focused on local computation and explicit communication strategies. This makes the DCOP framework ideal for modeling and solving the distributed cooperative search task.

It will be shown that the characteristics of the D-Bay algorithm make it suitable for the application to UAVs with limited resources. Secondly, experimental results are given for a cooperative search use case with multiple UAVs to illustrate the application of the D-Bay algorithm in a real-world scenario. In the use case a predefined area needs to be searched in as little time as possible. Additional simulation results are given in order to evaluate the performance of the D-Bay algorithm for a heterogeneous group of UAVs.

Section 3 introduces the D-Bay algorithm and evaluates its characteristics. In Section 3.2 the overview of the hardware and the software of the UAVs used in the experiments are presented. Afterwards, experimental evaluation of the D-Bay algorithm with multiple UAVs is included in Section 4. Finally, Section 5 summarizes the results and defines future work.

2 PROBLEM DEFINITION

We consider a cooperative search use case with multiple autonomous quad-rotor vehicles. These vehicles are selected thanks to their versatility and high maneuverability. As the autonomous vehicles will distributively optimize their trajectories, they are referred to as (autonomous) agents. Typically, these search operations are performed by autonomous vehicles equipped with one or more cameras. While surveying, the images from the cameras are evaluated by image-processing software to detect and identify objects of interest. The solution to the cooperative search problem involves an optimal division of the search region between all agents. The optimal division is defined as the division that will require the least amount of time for all vehicles to complete the entire search and to return to their initial position.

More specifically, we consider a (rectangular) search region defined by its width and height as $R = (R_w, R_h)$, where $R_w \geq R_h$. It is assumed that there are no obstacles and that the size of the region is known by all agents. Every agent has a single variable x_i it can assign, where i is the agent index. The variables of the agents are related to the size of their individual segments $R_i = (R_{i,w}, R_h)$ and $R_w = \sum_{i=1}^N R_{i,w}$.

The search problem is considered to be two-dimensional, as the altitude for all UAVs is kept constant during the search. The altitude results from a trade-off between the required resolution for successful detection of the images and the size of the area imaged by the camera. Higher resolution images will therefore enable a larger observed area.

In addition to the time spent searching, the travel times of the UAVs towards the start of their segments ($p_{i,s}$), and from the end of the segment ($p_{i,f}$) back to their initial positions (p_i) are taken into account. The UAVs will traverse a series of equidistance parallel legs within their search segments, where a *leg* indicates a straight line during which scanning is performed. This search pattern is often referred to as a lawnmower pattern. As shown by Ablavsky and Snorrason [1], the lawnmower pattern is optimal for a rectangular search area and UAVs of which the sweep width is larger than the turn radius. For quad-rotor vehicles this holds for any sweep width as these platforms are holonomic. The sweep width is based on the width of the camera image ($l_{i,w}$) on the ground. In order to ensure complete coverage, the distance between the legs is based on the sweep width such that consecutive tracks interleave.

An additional distance (l_t) is added before the start of the leg to ensure that all oscillations (caused by the cornering) are eliminated to ensure that the image quality is constant while searching. The velocity of the UAVs during scanning and transit is denoted by v_s , and v_t , respectively. An overview of the search segment of a single agent can be seen in Figure 1.

For agent a_i the number of legs of its lawnmower pattern is defined by

$$N_{i,l}(x_{i-1}, x_i) = \left\lceil \frac{R_{i,w}}{2l_{i,w}} \right\rceil = \left\lceil \frac{x_i - x_{i-1}}{2l_{i,w}} \right\rceil,$$

where $\lceil \cdot \rceil$ is the ceiling operator. The start and finish positions of the pattern of agent a_i can be defined according to its neighboring

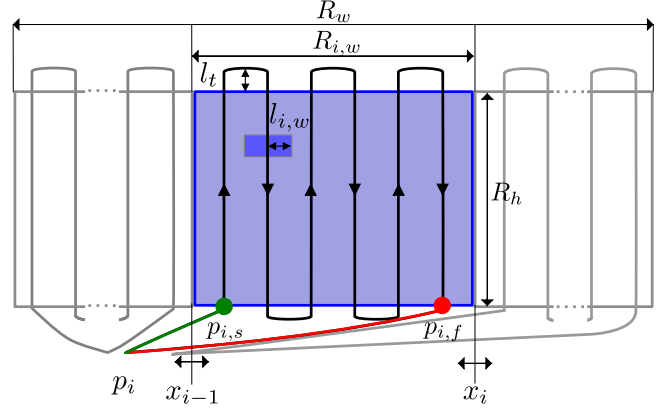


Figure 1: Overview of the search segment of agent a_i .

agent a_{i-1} as

$$p_{i,s}(x_{i-1}) = (x_{i-1} + l_{i,w}, 0),$$

$$p_{i,f}(x_i) = \begin{cases} (x_i - l_{i,w}, 0) & \text{if } N_{i,l} \text{ is even,} \\ (x_i - l_{i,w}, R_h) & \text{otherwise.} \end{cases}$$

Based on these values and the properties of the search area, the required time to scan a segment (including transit) for a_i is calculated as

$$f_i(x_{i-1}, x_i) = T_{i,s}(x_{i-1}) + T_{i,sc}(x_{i-1}, x_i) + T_{i,f}(x_i), \quad (1)$$

where the transit times ($T_{i,s}(\cdot)$ and $T_{i,f}(\cdot)$) and the time spent scanning ($T_{i,sc}(\cdot)$) are defined as

$$T_{i,s}(x_{i-1}) = \frac{|p_i - p_{i,s}(x_{i-1})|^2}{v_t},$$

$$T_{i,f}(x_i) = \frac{|p_i - p_{i,f}(x_i)|^2}{v_t},$$

$$T_{i,sc}(x_{i-1}, x_i) = T_{i,c} + T_{i,l}.$$

The time spent scanning depends on both the time required for traversing the legs, defined as

$$T_{i,l} = \frac{N_{i,l}(x_{i-1}, x_i) R_h}{v_s},$$

and on cornering between the legs, defined as

$$T_{i,c} = \frac{(N_{i,l}(x_{i-1}, x_i) - 1) 2(l_t + l_{i,w})}{v_s} + 2N_{i,l}(x_{i-1}, x_i) \tau,$$

where τ is the time required by the UAV to make a turn. In the next section the search problem will be modeled within the DCOP framework.

2.1 Problem formulated as a DCOP

A Distributed Constraint Optimization Problem (DCOP) is a problem framework that is based on a global objective function that needs to be optimized in a distributed manner. The global objective function is defined as the aggregate of utility functions. Typically, the summation operator ($\sum(\cdot)$) or the maximum operator ($\max(\cdot)$) are used as the aggregation operator. The agents within the DCOP are defined based on variables. Each agent is able to assign a value to its variables. Instead of the global objective function, an agent only

knows two properties of the problem. (1) the local utility functions of which its variables are used as argument, (2) its neighboring agents, which are defined as agents that *share* a utility function, i.e. if there exists a utility function of which its arguments include variables of both agents. This local view on the problem creates the need for the agents to cooperate to optimize the assignment of their variables in terms of the global objective function.

The value assignments are restricted by the domains of the variables. This feature of DCOP makes it suitable for problems with bounded inputs such as many real-world problems. However, conventionally the domains are defined as finite discrete sets, while many real-world problems (including cooperative search) are best described using finite continuous sets. The reason for the discrete set definition is based on the origin of the DCOP framework: DCOP originates from the Constraint Satisfaction Problem (CSP) framework [22], which is mainly used to model problems such as graph coloring problems and meeting scheduling problems. These problems inherently have finite and discrete domains. The DCOP framework emerged from extending the CSP framework to agent-based distributed optimization by Yokoo et al. [26] and generalization to utility functions instead of constraint satisfaction checking. Within this process of extending and distributing, the domains have not been updated to include continuous values.

To address this issue, in this paper, a continuous version of DCOP is used to model the cooperative search problem. Following the notation of Fioretto et al. [7], a continuous DCOP is defined by $\mathcal{D} = \langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha, \eta \rangle$ where:

- $\mathbf{A} = \{a_1, \dots, a_M\}$ is the set of agents, where M is the number of agents.
- $\mathbf{X} = \{x_1, \dots, x_N\}$ is the set of variables, where $N \geq M$ is the number of variables.
- $\mathbf{D} = \{\mathbf{D}_1, \dots, \mathbf{D}_N\}$ is the set of domains of all variables, where $\mathbf{D}_i \subseteq \mathbb{R}$ is the (continuous) domain associated with variable x_i .
An assignment denotes the projection of variables onto their domain as $\sigma : \mathbf{X} \rightarrow \Sigma$. In other words, for all $x_i \in \mathbf{X}$ if $\sigma(x_i)$ is defined, then $\sigma(x_i) \in \mathbf{D}_i$. An assignment of a subset of variables is denoted by $\sigma_V = \{\sigma(x_i) : x_i \in \mathbf{V}\}$.
- $\mathbf{F} = \{f_1, \dots, f_K\}$ is the set of utility functions, where K is the number of utility functions.
- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is a mapping from variables to agents. The agent to which variable x_i is allocated is denoted as $\alpha(x_i)$.
- η is an operator that combines all utility functions into the objective function.

The global objective of a DCOP is to minimize the objective function, defined by $G(\sigma) = \eta_{f_n \in \mathbf{F}} (f_n(\sigma_{V_n}))$.

Based on this definition the cooperative search problem can be cast into a DCOP as,

- $\mathbf{A} = \{a_1, \dots, a_N\}$, where N is the number of UAVs,
- $\mathbf{X} = \{x_1, \dots, x_N\}$, since every agent has a single variable ($M = N$),
- $\mathbf{D} = \{\mathbf{D}_1, \dots, \mathbf{D}_N\}$, where $\mathbf{D}_i = [0, R_w]$ is related to the search region,
- $\mathbf{F} = \{f_i(x_{i-1}, x_i) : i \in \{1, \dots, N\}\}$, where $f_i(x_{i-1}, x_i)$ is defined by Equation 1 with $x_0 = 0$,

- $\alpha = \{x_i \rightarrow a_i : i \in \{1, \dots, N\}\}$, where every agent is assigned a single variable,
- $\eta = \max(\cdot)$, in order to minimize the maximum time required for all agents.

Apart from the definition of the continuous DCOP as given above, a DCOP is typically represented in the form of a graph. Two frequently used forms are the (undirected) constraint graph and the (directed) pseudo-tree [8]. In both graphs, the agents are shown as nodes and neighboring agents are connected through an edge. Note that a constraint graph can be converted into a pseudo-tree by means of various procedures, such as depth-first-search [3]. A benefit of the pseudo-tree over the constraint graph is that the pseudo-tree introduces hierarchy to the variables and thereby divides the problem into subproblems. The hierarchy creates an implicit communication structure that is used to send messages between the agents without requiring all-to-all communication. The subproblems can be exploited by algorithms in order to efficiently solve the DCOP.

3 ALGORITHM OVERVIEW

Even for near-optimal solvers the complexity increase is linear based on the cardinality of the largest domain.

The main reason is that the majority of the DCOP solvers are created for DCOPs with discrete domains. Therefore, in order to apply these solvers, the continuous domains need to be discretized. A straightforward approach is to use equidistant discretization for all variables. This allows for the creation of domains with a cardinality of arbitrary size. As for most problems, the quality of the solution depends on the resolution of the variables, where a high resolution allow for better solutions. This creates a trade-off between solution quality and computational and memory requirements.

3.1 Description of D-Bay

The D-Bay algorithm involves four sequential phases:

- (1) **Pseudo-tree construction** The agents create a pseudo-tree from the constraint graph of the DCOP, by performing a depth-first search traversal.
- (2) **Allocation of utility functions** Similar to the allocation of variables, all utility functions are exclusively allocated to the agents.
- (3) **Sample propagation** In this phase, every agent optimizes its local variables through the Bayesian optimization method and the exchange of **sample** and **utility** messages. This phase is initiated by a **sample** message from the root agent. The phase finishes when a termination criterion is reached by the root agent. The samples are selected through the optimization of an acquisition function. A graphical overview of the sample phase is shown in Figure 2.
- (4) **Assignment propagation** The final phase is the assignment propagation phase, in which the root agent sends the final assignment of all its variables to its children as a **final** message. Based on these assignments the children can assign their own variables to the value corresponding to the optimal utility value.

Based on the taxonomy introduced by Yeoh et al. [24], the D-Bay algorithm can be classified as a sample-based solver. This class

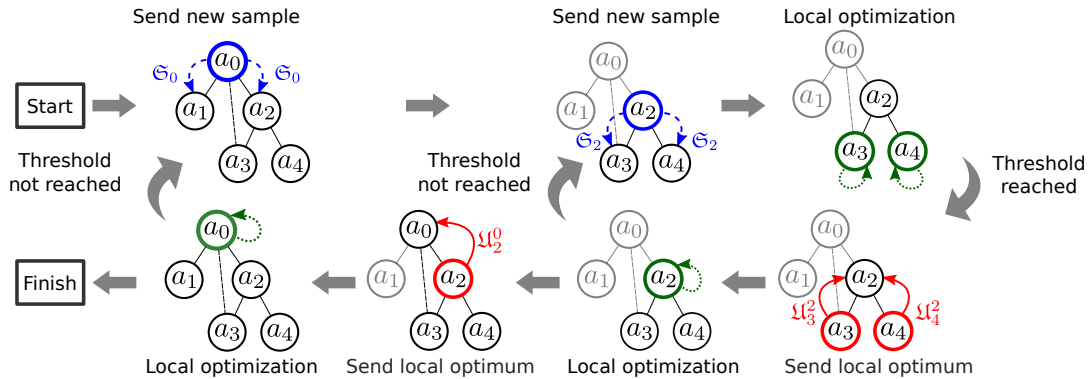


Figure 2: Graphical overview of the sample phase of D-Bay. Agents are indicated by circles labeled with an agent index, and utility functions are shown as black lines. Starting from the root a_0 (top-left), a sample message \mathfrak{E}_0 is sent to its children (a_1, a_2). After iterating between its children and calculating its local utility, agent a_2 combines all local utilities and sends a utility message \mathfrak{U}_2^0 to its parent.

of solvers uses probabilistic measures to coordinate the sampling of the global search space. The probabilistic measures are used to balance exploration and exploitation of the search space. The two main differences with existing sample-based solvers (DUCT [16] and Distributed Gibbs [15]) is the sample selection and their iterative approach. Firstly, within the D-Bay algorithm, the Bayesian optimization method is used for the selection of the samples. The Bayesian optimization method consists of two elements: a *probabilistic model* to approximate an unknown (utility) function, and an *acquisition function* to optimally select a new sample. This method is also referred to as an active learning approach as the acquisition function makes use of previously sampled values and the probabilistic model in order to *learn* as much about the function in a sample-efficient manner. To significantly reduce the computational load, a Markovian class kernel [5] is used within the D-Bay algorithm. A Markovian class kernel possess the property that the corresponding covariance matrices can be inverted analytically. Secondly, instead of iterating over the entire pseudo-tree, agents iterate between parent and children only. While this typically requires more messages than DUCT and Distributed Gibbs, it ensures determinism with respect to the utility value of a sample. In other words, a sample will always return the same utility value.

3.2 Characteristics of D-Bay

The optimization of the local variables is restarted every time a new sample message from the parent is received. An agent only needs to store the utility of the values based on the current (local) iteration in order to send the *best* utility value back to its parent, thereby restricting the memory requirement per agent to $\mathcal{O}(l)$. The message size $\mathcal{O}(t)$, as the size of the utility messages is fixed at $\mathcal{O}(1)$ while the size of the sample messages is proportional to the maximal depth of the tree t . The number of messages scales with $\mathcal{O}(cl^t)$, where c denotes the largest number of children. In this scenario the problem structure cannot be exploited to split the problem into subproblems.

Here d denotes the largest cardinality of the domains and η denotes the largest number of neighboring agents. This highlights

the fact that by discretizing the continuous domains, the runtime complexity will significantly increase. Furthermore, for DUCT there are large memory requirements $\mathcal{O}(d^l)$ as it needs to store all the best costs for all messages. Distributed Gibbs is much more memory efficient and merely requires $\mathcal{O}(l)$. With respect to the message characteristics, DUCT sends less messages than Distributed Gibbs, but they are larger in size. A comparison of the D-Bay algorithm with its closest related solvers for these key characteristics is shown in Table 1.

Table 1: Comparison of algorithm characteristics (based on Table 4 of Fioretto et al. [7]).

Algorithm	Runtime		Message	
	Complexity	Memory	#	Size
DUCT	$\mathcal{O}(l\eta d)$	$\mathcal{O}(d^l)$	$\mathcal{O}(lN)$	$\mathcal{O}(t)$
D-Gibbs	$\mathcal{O}(l\eta d)$	$\mathcal{O}(l)$	$\mathcal{O}(lN\eta)$	$\mathcal{O}(1)$
D-Bay	$\mathcal{O}(l)$	$\mathcal{O}(l)$	$\mathcal{O}(cl^t)$	$\mathcal{O}(t)$

In this section an overview of the Unmanned Aerial Vehicles (UAVs) used for the experiments is given. The hardware, software architecture, and the simulation environment are discussed.

3.3 Hardware overview

In the experiments, quad-rotor UAVs (3DR-X4) are equipped with a downward facing camera (GoPro Hero5 Session [10]) that image a small section of the search area while traveling over it while taking snapshots. This camera was selected as it is low-weight and eliminates the vibrations induced by the UAV by software-based image stabilization. The camera is connected to an onboard computer (Raspberry Pi 3B+ [19]), which also runs the D-Bay algorithm and handles the communication between the agents. The size, weight, and accessibility of this computer made it ideal for the application. Communication between the agents is done through low-power radios (XBee Pro [4]). This radio module is able to create a meshed network using the ZigBee protocol between all modules for up

to 750 m. Note that this holds for an outdoor environment with line-of-sight between the modules. Within the network messages up to 84 bytes can be sent. During flight, the UAV is regulated by a flight controller (3DR Pixhawk 1 [12]), which uses measurements from an inertial measurement unit for linear acceleration and angular speed, barometric pressure measurements for height, and GPS measurements for latitude and longitude. This flight controller runs the PX4 autopilot software [6]. An overview of the UAV and its components is given in Figure 3.

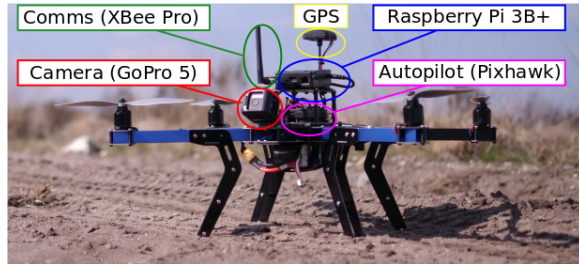


Figure 3: Overview of the 3DR-X4 UAV used during the experiments. The key components required for autonomous flight are highlighted.

3.4 Simulation environment

The configuration of the UAVs is modeled in a realistic simulation environment in order to validate the autonomous actions of the UAVs. In this work, the Gazebo simulator [11] is used as it offers a high level of flexibility in modeling the environment and the UAVs. There is a large open-source community that actively supports and extends the simulation environment, and creates high-fidelity virtual models for various (autonomous) vehicles. These models require a high level of expert knowledge to construct and validate. Therefore, within our simulations, the community model of the 3DR Iris is used. The 3DR Iris is the commercial version of the 3DR-X4 and is only different in appearance due to the enclosure. An example of the UAV model in the simulation environment is shown in Figure 4.

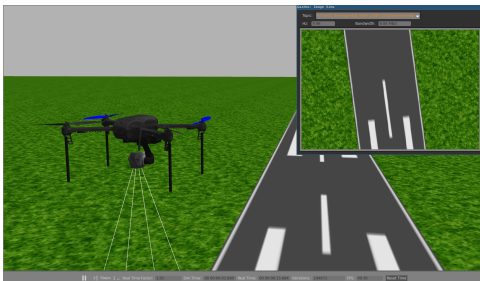


Figure 4: Example of the simulation environment (Gazebo) of the UAV (3DR Iris) and its simulated view of the downward facing camera.

3.5 Software overview

An additional benefit of the Gazebo simulation environment is its extensive interface with Robotic Operating System (ROS) [18]. This allowed the seamless transition between simulation and experimental execution of the D-Bay algorithm and testing of the cooperative search use case. Apart from an interface to Gazebo and the autopilot, the ROS middleware is also used to connect all other software components within the computer, such as the communication module, the D-Bay algorithm, and the camera interface. A schematic overview of the software architecture is shown in Figure 5.

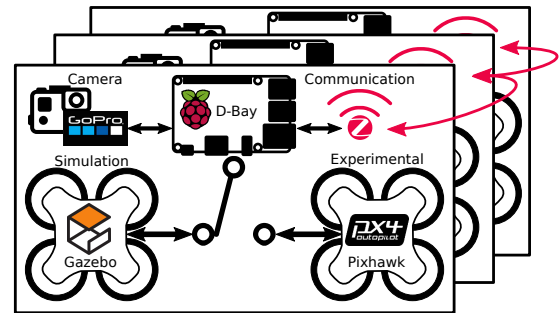


Figure 5: Schematic overview of the software architecture.

4 EXPERIMENTAL EVALUATION

At this altitude the scan width was 5 m, therefore $l_{i,w} = l_w = 5$ m. A velocity of 3 m/s was found to produce a stable flight without large oscillations in speed and position. Postprocessing of the camera images showed the velocity of the UAVs did not negatively influence the image quality. For this reason, both the velocity during scanning and transition are equally set to $v_t = v_s = 3$ m/s. By investigating the data collected during the cornering allowed for the determination of the turn length (l_t) and time required per turn (τ). While there were some minor differences in the amount of oscillations after a turn, all were damped out after 5 m. Therefore, the turn length was set as $l_t = 5$ m. The average time required per turn was found to be 1.5 s, however when the UAV was greatly affected by wind, this could increase to as much as 7.5 s. Even though there were no strong winds, all experiments were located at an old airfield which offered no protection from the wind. As these gusts of wind occurred only sporadically, the time required per turn was set to $\tau = 1.5$ s. Additionally, a search region of $R = (R_w, R_h) = (200 \text{ m}, 50 \text{ m})$ was defined to ensure all UAVs would have enough battery power to complete the entire search. The lower left corner of the search region is used as reference position (0, 0). The UAVs were placed near the reference position with (approximately) 5 m intervals to ensure enough spacing between the UAVs for safe takeoff and landing conditions. The required time for the reference experiment to finish was approximately 240 s. The times required by the UAVs to finish scanning varied considerably. This was to be expected as the UAVs had a similar sized area to scan, but greatly different travel times.

The trajectories of the UAVs for 18 samples are shown in Figure 6.

Table 2: Experimental results compared to reference.

# samples	Result [s]	Difference	
		Percentage [%]	Absolute [s]
6	221	7.9	19
12	218	9.2	22
18	208	13.3	32

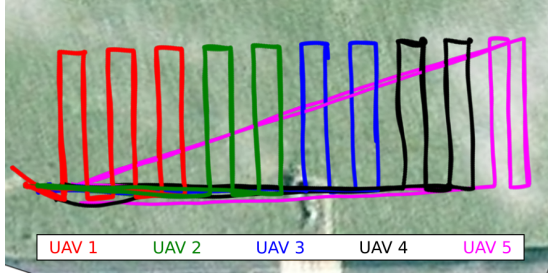


Figure 6: Experimental results of the color coded GPS tracks of the UAVs. The initial locations of the UAVs are in the bottom left corner.

4.1 Validation of simulations

During cornering the UAVs were affected most as the waypoints of the trajectory are located closely together. A gust of wind can push an UAV off-course, thus needing additional time to recover. Note that not all UAVs were affected to the same extent.

4.2 Evaluation of communication

For this reason, it is appropriate to evaluate the communication between the agents.

Despite these issues, we can conclude that the XBee network was capable of sending the messages for the D-Bay algorithm. Thereby showing the D-Bay algorithm can be successfully implemented on a low-bandwidth communication network.

4.3 Additional simulations

Within the validated simulation environment additional simulations were performed. Five scenarios were constructed by changing the properties of some UAVs in order to evaluate the performance of the D-Bay algorithm for a group of heterogeneous UAVs. Scenarios 1-3 alter the scan width of a subset of UAVs in order to resemble UAVs with cameras of different quality. Scenario 4 varies the scanning velocity of three UAVs. In scenario 5 one UAV has a higher scanning velocity, but a smaller scan width. The (altered) parameters for the scenarios are defined as follows:

Scenario 1: $l_{1,w} = 10$ m,

Scenario 2: $l_{1,w} = 10$ m, $l_{5,w} = 20$ m,

Scenario 3: $l_{1,w} = 10$ m, $l_{3,w} = 2.5$ m, $l_{5,w} = 15$ m,

Scenario 4: $v_{1,s} = 6$ m/s, $v_{3,s} = 1.5$ m/s, $v_{5,s} = 4.5$ m/s,

Scenario 5: $v_{1,s} = 6$ m/s, $l_{1,w} = 2.5$ m.

In Table 3, an overview of the results of the simulations for 15 samples per agent are compared with the (optimal) results of a brute-force method. In Figure 7 the simulation results for scenario

Table 3: Simulation results compared to optimum.

Scenario	Result [s]	Difference	
		Percentage [%]	Absolute [s]
1	179.0	3.2	5.5
2	160.0	0.4	0.7
3	179.9	2.4	4.3
4	181.8	3.5	6.2
5	188.5	2.8	5.2

3 are shown. The simulation results show a close approximation to

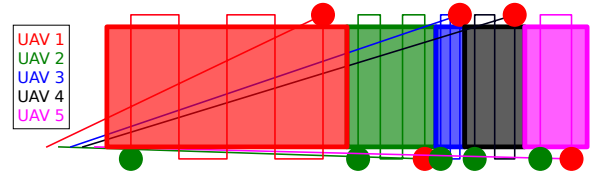


Figure 7: Simulation results of the trajectories of the UAVs for scenario 3. The trajectories are color coded by UAV.

the optimum for all scenarios. Similar results were obtained when only 10 samples per agent were used. The results for all scenarios (except scenario 1) did not differ more than 2% from the results with 15 samples. This is an indication that the D-Bay algorithm converges relatively fast even with a very limited amount of samples per agent.

5 CONCLUSIONS

In this work, experimental results for a cooperative search use case with multiple Unmanned Aerial Vehicles (UAVs) have been presented. This problem was modeled within the continuous Distributed Constraint Optimization Problem (DCOP) framework and solved with the Distributed Bayesian (D-Bay) algorithm. The D-Bay algorithm is specifically designed for continuous DCOPs and operates directly on the continuous domains. This makes the D-Bay algorithm excellently suitable for the application of real-world problems such as use cases with autonomous vehicles with limited resources (computational power, memory, and communication bandwidth).

6 ACKNOWLEDGMENT

The authors would like to thank Mathijs Lomme from TNO for his help during the experiments.

REFERENCES

- [1] Vitaly Ablavsky and M Snorrason. 2000. Optimal search for a moving target: A geometric approach. In *AIAA Guidance, Navigation and Control Conference*. Denver, Colorado.
- [2] Jose Joaquin Acevedo, Begoña C. Arrue, Ivan Maza, and Anibal Ollero. 2013. Cooperative large area surveillance with a team of aerial mobile robots for long endurance missions. *Journal of Intelligent and Robotic Systems: Theory and Applications* 70, 1-4 (2013), 329–345. <https://doi.org/10.1007/s10846-012-9716-3>
- [3] Baruch Awerbuch. 1985. A new distributed depth-first-search algorithm. *Information Processing Letters* 20, 3 (1985), 147–150.
- [4] Digi International. 2014. XBee Pro RF Module. (2014). <https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf>
- [5] Liang Ding and Xiaowei Zhang. 2018. Scalable stochastic kriging with Markovian covariances. *arXiv arXiv:1803.02575* (2018).
- [6] Dronecode Project, Inc. 2019. PX4 autopilot software. (2019). <https://px4.io/>
- [7] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. 2018. Distributed Constraint Optimization Problems and Applications: A Survey. *Journal of Artificial Intelligence Research* 61 (mar 2018), 623–698. <https://doi.org/10.1613/jair.5565>
- [8] Eugene C Freuder and Michael J Quinn. 1985. Taking advantage of stable sets of variables in constraint satisfaction problems. In *International Joint Conference on Artificial Intelligence*. Los Angeles, California, 1076–1078.
- [9] Amir Gershman, Amnon Meisels, and Roie Zivan. 2009. Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research* 34 (2009), 61–88. <https://doi.org/10.1613/jair.2591>
- [10] GoPro. 2019. GoPro Session 5. (2019). <https://gopro.com/en/us/yourhero5/session>
- [11] Nathan Koenig and Andrew Howard. 2004. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sendai, Japan, 2149–2154.
- [12] Manufactured by 3DR. 2014. Pixhawk 1 Flight Controller. (2014). <https://pixhawk.org/>
- [13] Amnon Meisels. 2007. *Distributed Search by Constrained Agents: Algorithms, Performance, Communication*. Springer Science, London, UK.
- [14] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. 2005. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161, 1-2 (jan 2005), 149–180. <https://doi.org/10.1016/j.artint.2004.09.003>
- [15] Duc Thien Nguyen, William Yeoh, Hoong Chuin Lau, and Roie Zivan. 2019. Distributed Gibbs: A Linear-Space Sampling-Based DCOP Algorithm. *Journal of Artificial Intelligence Research* 64 (mar 2019), 705–748. <https://doi.org/10.1613/jair.1.11400>
- [16] Brammert Ottens, Christos Dimitrakakis, and Boi Faltings. 2018. DUCT: an upper confidence bound approach to distributed constraint optimization problems. *Transactions on Intelligent Systems and Technology* 8 (2018), 1–27.
- [17] Adrian Petcu and Boi Faltings. 2005. A scalable method for multiagent constraint optimization. In *International Joint Conference on Artificial Intelligence*. Edinburgh, UK, 1413–1420.
- [18] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*. Kobe, Japan.
- [19] Raspberry Pi Foundation. 2019. Raspberry Pi 3B+. (2019). <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>
- [20] Cyril Robin and Simon Lacroix. 2016. Multi-robot target detection and tracking: taxonomy and survey. *Autonomous Robots* 40, 4 (2016), 729–760. <https://doi.org/10.1007/s10514-015-9491-7>
- [21] R. Stranders, E. Munoz De Cote, A. Rogers, and N. R. Jennings. 2013. Near-optimal continuous patrolling with teams of mobile information gathering agents. *Artificial Intelligence* 195 (2013), 63–105.
- [22] E Tsang. 1993. *Foundations of Constraint Satisfaction*. Elsevier, London. <https://doi.org/10.1016/C2013-0-07627-X>
- [23] Sandor M. Veres, Levente Molnar, Nicholas K. Lincoln, and Colin P. Morice. 2011. Autonomous vehicle control systems – a review of decision making. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 225, 2 (2011), 155–195. <http://pii.sagepub.com/content/225/2/155.short>
- [24] William Yeoh, Ariel Feiner, and Sven Koenig. 2010. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research* 38 (2010), 85–133.
- [25] William Yeoh and Makoto Yokoo. 2012. Distributed problem solving. *AI Magazine* 33 (2012), 53–65.
- [26] M Yokoo, E H Durfee, T Ishida, and K Kuwabara. 1998. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 10, 5 (1998), 673–685. <https://doi.org/10.1109/69.729707>
- [27] Roie Zivan, Tomer Parash, and Yarden Naveh. 2015. Applying max-sum to asymmetric distributed constraint optimization. In *International Joint Conference on Artificial Intelligence (IJCAI)*. Buenos Aires, Argentina, 432–439.