2014

Dynamic Task-Allocation for Unmanned Aircraft Systems

Tim Bakker
Virginia Commonwealth University

DYNAMIC TASK-ALLOCATION FOR UNMANNED AIRCRAFT SYSTEMS

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

TIM BAKKER

Master of Science, Virginia Commonwealth University, 2010

Director: Dr. Robert H. Klenke,

Associate Professor, Department of Electrical & Computer Engineering

Virginia Commonwewalth University

Richmond, Virginia

May, 2014

Acknowledgement

This work would not have been possible without the support of many people helping me throughout my academic career. First I would like to express my deepest gratitude to my advisor, Dr. Robert H. Klenke, for his support, guidance and belief in me. I would also like to thank the members of my advisory board: Dr. Ashok Iyer, Dr. Wei Zhang, Dr. Paul Brooks, and Dr. Yongjia Song for their feedback and insightful thoughts.

I would like to thank the Virginia Commonwealth University School of Engineering for supporting and funding my academic research in the UAV lab. As well, I would like to thank my fellow graduate students, Matthew T. Leccadito, Thomas W. Carnes, Garrett L. Ward, Joel D. Elmore and in particular Siva T. Patibandla. Lastly, I would like to thank my family, parents-in-law, my beautiful wife and daughter for supporting me, believing in me and being the joys of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

DYNAMIC TASK-ALLOCATION FOR UNMANNED AIRCRAFT SYSTEMS

By Tim Bakker

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2014.

Director: Dr. Robert H. Klenke,

Associate Professor, Department of Electrical & Computer Engineering

This dissertation addresses improvements to a consensus based task allocation algorithms for improving the Quality of Service in multi-task and multi- agent environments. Research in the past has led to many centralized task allocation algorithms where a central computation unit is calculating the global optimum task allocation solution. The centralized algorithms are plagued by creating a single point of failure and the bandwidth needed for creating consistent and accurate situational awareness off all agents.

This work will extend upon a widely researched decentralized task assignment algorithm based on the consensus principle. Although many extensions have led to improvements of the original algorithm, there is still much opportunity for improvement in providing sufficient and reliable task assignments in real-world dynamic conditions and changing environments. This research addresses practical changes made to the consensus based task allocation algorithms for improving the Quality of Service in multi-task and multi- agent environments.

CHAPTER 1

INTRODUCTION

This dissertation presents the research work done in the field of autonomous Unmanned Aircraft Systems (UAS) and, in particular, collaboration amongst a set of UAS. With the recent publishing of the first annual UAS road map for integrating unmanned systems into the United States national airspace, it has become more-clear how future utilization and exploitation of Unmanned Aircraft Systems can be exercised for civil applications. A heterogeneous collaborative set of UAS, in this dissertation also referred to as agents, could potentially perform an endless set of civil duties, from search, rescue, and surveillance to law enforcement.

The UAS currently flown by the military, such as the Predator, need two operators at all times [1]. For future deployment in urban environments, the dependency on adequate pilots for flying each UAS is not viable. Human interaction will shift from flying the UAS to monitoring and releasing tasks to the UAS population. Another significant challenge is the highly dynamic environment when UAS are applied in the national airspace and the pilots' ability to respond and act on these changing conditions.

The aforementioned challenges call for an increasing level of decision making ability and autonomy for each individual UAS, and a certain synchronism amongst the fleet. This autonomy must be executed at real-time, adapting to changes in Situational Awareness (SA). As the number of UAS grows, most task allocation and path planning algorithms are considered to be in the category of NP-hard or NP-complete and this presents a challenge. Collaborative research within the UAS environment already poses many obstacles, from the limitations of the wireless link, to the size and weight constraints of the hardware and the implications of this on processing power and power consumption.

Task allocation algorithms can be divided into centralized, decentralized, and auction algorithms. Centralized algorithms have been extensively researched in the field of controls and transpiration over the last century and include algorithms for solving problems like the Traveling Salesman Problem or the Knapsack problem [1, 2, 3, 4]. Some of these algorithms are very powerful and can find a near optimum solution in reasonable time.

These centralized planners are either implemented at a local server, calculating the solution for the entire fleet, or deployed decentrally, where each agent is calculating the same solution. Some drawbacks of the central planning algorithms are the limited mission-range and the solution being dependent on the integrity of Situational Awareness. The centralized planner can heavily congest the communication network by relaying huge amounts of SA data to a central processing unit (or units) to prevent allocation conflicts and increase the performance of the solution.

Wireless network technology has made substantial progress in the last decade with the emergence of the Wireless Local Area Network standards 802.11 g/n. Where bandwidth of WLANs has been ever increasing, the usage of these links should be somewhat conservative, through optimizing algorithms and limiting data communication amongst nodes in the network. This restriction is even more important when the numbers of agents is increasing and packet collision becomes a prominent factor in wireless transport.

The chance for packet collisions could increase even more when the system is implemented to handle incorrect or corrupted data by performing retries. Furthermore when the complexity of the mission increases the processing power needs to adequately scale in order to produce an adequate solution or to find a solution in real-time. Lastly, the central planning algorithm deployed centrally introduces a Single Point Of Failure (SPOF) when the either the processing unit fails or the network link gets interrupted.

CHAPTER 2

PROBLEM DEFINITION

The general problem of task allocation for a cooperating set of UAS can be stated as follows: given a set of heterogeneous tasks $K = \{1, ..., N^t\}$, and a set of heterogeneous agents $J = \{1, ..., N^k\}$ find a solution to the tasks and allocate the tasks amongst the agents in the fleet while optimizing the Quality of Service (QoS) described below. The problem researched herein can be best described as a heterogeneous multiple depot traveling salesman problem, without the requirement to return to depot.

Tasks can be as simple as traversing a set of given waypoints to search or survey a given area. For each task, a solution must be calculated based on the SA and the description of the task. The task could also involve multiple steps, requiring multiple agents with different capabilities. The solution to these tasks requires the agents to form teams intelligently in order to provide a solution to the task.

A simple example is a search task where the objective is to find a target in a search area and, once this target is found by an infra-red camera, a high resolution image must be taken of the surrounding area. This task would require an agent with an on-board infra-red camera to find the target and another agent to take a high resolution image of its surrounding. Although many mission scenarios can be envisioned, this research will focus on surveillance and search missions.

To optimize the task allocation problem based on QoS, it is of utmost importance to fully describe what QoS entails in the context of this research. QoS is defined by latency, the time between releasing a task to the set of agents and completion of the given task. Finishing tasks early will increase the performance and efficiency of the fleet of agents.

Furthermore, all tasks need to be assigned whenever possible to at least one agent. Tasks that are able to be executed within reasonable constraints should be allocated to an agent and be executed. Finally, the task allocation process must reliable and stable. This can be heavily influenced by changing environments, SA, number of agents in the fleet, and connectivity of the communications network.

In a heavily changing environment, and thus changing SA, the outcome of a task allocation can alter significantly over time. A scenario where task allocation outcome could change is when new agents are added to the fleet or old agents are removed from the fleet due to health factors. The task allocation process should be stable and continuously result in optimum solutions in this type of changing environment.

The reliability of the wireless network is a major influence on the task allocation algorithm and thus has direct influence on the Quality of Service (QoS) provided by the agents in the fleet. In a dynamic wireless network with multiple agents, connectivity between agents is changing continuously and the tasks allocation process should provide a reliable and optimized allocation of tasks in spite of the changing communications network.

As mentioned before, environment and the health of the vehicle can greatly influence the outcome of a mission, and should therefore be key factors in generating solutions and allocating tasks among the fleet [5]. An agent running low on fuel should only be allocated tasks that it can entirely execute before landing safely.

The fleet needs to be fully autonomous and should require minimal human interaction for solving mission scenarios. The sole purpose of the human operator is to dispatch tasks and monitor the health of each UAS, intervening when a potentially dangerous situation arises. This requires the agent to be equipped with Artificial Intelligence (AI) and algorithms to provide the robustness and QoS outlined above.

This dissertation introduces the Asynchronous Polling Consensus-Based Bundle Algorithm (APCBBA) for addressing the aforementioned features and provide the outlined metrics for QoS. The algorithm is an extension of the Consensus-Based Bundle Algorithm [6], but provides a more robust task allocation solution while increasing the quality of the task allocation solution. The remainder of this dissertation includes an extended review of related research into task allocation algorithms in Chapter 3. Chapter 4 provides an in depth description of the APCBBA algorithm and how the changes impact the QoS metrics. The Mission Control System is the software onboard of the UAS enabling collobarative operation; an overview of the architecture of the software and implementation is given in Chaper 5. A brief explanation of the RAMS simulator can be found in Chapter 6 followed by a description of the hardware platform in Chapter 7. A Mixed-Integer Linear Programming (MILP) reference solution is defined in Chapter 8 with validation results comparing it to solutions given by an exhaustive search algorithm. Chapter 9 provides simulation and real-world results of APCBBA for several QoS metrics, including results for optimality of the solution, response time, and team formation.

CHAPTER 3

RELATED WORK

Central planning algorithms, as mentioned in the introduction, have been widely applied to this problem and can be executed either centrally at a single location, or decentrally at each agent [7, 8], where each agent is calculating a total solution for the entire fleet. This decentralized solution is heavily dependent on the consistency of the SA across the fleet. Small differences in the SA between agents could yield entirely different solutions to the task allocation problem and, therefore, result in task allocation conflicts. On-going research has realized consensus strategies to resolve inconsistencies in the SA [9, 10, 11]. Although these methods have shown to be effective for finding a near-optimum solution, they require a significant amount of bandwidth to communicate and converge on a globally consistent SA. Although the consensus ensures that the SA is consistent, it does not have to represent the true SA. The discrepancy between the true SA and the converged SA is directly related to the ability to find an optimum solution. In [10], consensus is not only applied to the SA, but also to the task assignment. This introduces robustness by resolving task conflicts and does not require a perfect convergence of the SA.

The Hungarian method for task allocation was initially described in 1955 [12]. Thereafter, this method has been widely used in finding the global optimal solution in task allocation problems. Although originally the Hungarian method was a centralized algorithm for solving a task allocation problem, in [13] the authors have extended the algorithm to be used in a decentralized allocation environment.

An interesting "swap-stick" algorithm is applied in [14] where agents can swap tasks with neighboring agents or stick with their current allocated tasks. The algorithms does require an initial allocation of all the tasks among all agents, which then will apply the “swap-stick” methodology to come to a feasible and better solution. The proposed algorithm is fully decentralized and guarantees to converge within finite-time.

The third method for allocating tasks utilizes auction algorithms [15, 16, 17, 18]. These algorithms can provide conflict free task assignments with near-optimum solutions. Each agent will bid for tasks it can accomplish and the agent with the winning bid will receive the assignment. Since the auction algorithm will ensure no conflicts exist, convergence on the SA is unnecessary. Each agent can prepare bids for assignments based solely on its own interpretation of the environment. Most auction algorithms employ an auctioneer, which could be an agent or a centralized server that receives all the bids and determines the winner of the assignment. This applies restrictions to the network topology, as agents need to be connected to the auctioneer in order to be part of the task allocation process.

The Consensus-Based Auction Algorithm (CBAA) and the Consensus-Based Bundle Logarithm (CBBA) [6, 19], are auction algorithms where consensus is applied to the winning bid list. This takes away the requisite for a specific appointed auctioneer and provides robustness in different network topologies. CBAA is an algorithm designed to handle one task at a time; if multiple tasks are needing to be allocated, multiple iterations of the CBAA are required. CBBA, in contrast, can handle multiple task assignments by bundling tasks together and placing bids on a bundle of tasks. For multiple tasks, the bundling of tasks [6, 20] has shown to converge faster than sequential auction algorithms [6, 21].

3.1 Auction and market-based task allocation algorithms

In auction algorithms, artificial market-based principles are used to solve the problem of allocating tasks among a fleet of agents. In the simplest approach, there is one auctioneer, as in [22], that introduces new tasks and oversees the bidding process. Newly created tasks are assigned a price or granted a reward when the task has been successfully executed. This reward, together with the task description, is broadcast to the agents, where each agent has to determine a cost for the task. This cost can be based on the time it takes for agent k, where $k \in K = \{1, ..., N^k\}$, to execute task $j \in T = \{1, ..., N^t\}$ or it could be based on the distance the agent must travel to execute the task. Each agent that is able to produce a cost for task j will communicate this to the auctioneer, which in turn will assign the task to the agent with the most profit (reward cost). Since the task gets assigned to the agent with the largest profit at that time, the algorithm can be seen as a greedy algorithm. Greedy algorithms are not always able to find the global optimum solution.
In the previous example it would only take one iteration through the auction algorithm to assign a given task j. In contrast, in the method presented in [15], each agent wants to maximize its profits, so it begins by communicating its highest price and throughout iterations of the auction algorithm the price will be lowered until only one winner is left with the lowest price bid for task j. This is also known as an English auction; the same paper also researches the Dutch auction for task allocation. In this auction algorithm, the auctioneer initially sets a price for a block of tasks and the auctioneer will sell when agents are willing to buy. Eventually the price will be lowered on the tasks that were not sold, and other agents will start to purchase the discounted tasks. In [15] Hart and Craig-Hart conclude that the Dutch auction algorithm will mostly thrive in environments with a strong heterogeneity amongst tasks and agents. In

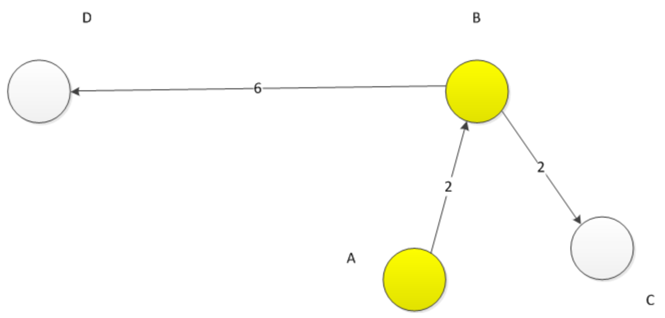


Fig. 1. The marginal return for adding task C to the already existing tasks A and B is higher than adding task D.

contrast, the English auction would be more efficient when few types of agent and or tasks exist. As previously mentioned, the determination of a cost for task j, can be based on distance and/or time required for agent k to complete task j. The cost can also be calculated based on the marginal return, when task j is added to the already existing task list for agent k.

Although the prize for task C in Fig. 1 might be lower, the marginal return is higher for adding task C, instead of task D, to the already existing tasks A and B for agent i. The cost of flying to task D is higher than flying to task C. This method is known as the cheapest insertion heuristic [23] and the incremental cost will be the cost that the auctioneer receives from agent i for task C. The auctioneer will reward task C to the agent with the lowest incremental cost.

For simplicity, a single auctioneer was implied in the above reasoning. In [21], this limitation is taken away so that every agent can also act as an auctioneer. Conflicts can occur when multiple agents decide to auction the same task; consequently, this paper implements a validation step that resolves the problem when two auctioneers are auctioning the same task. The benefit of a decentralized auctioneer is that it will not introduce a Single Point of Failure, but limited research exists on this topic.

A comparative study was done in [24], where a centralized, market-based, and behavioral method was used for task allocation in a multi-robot environment. In this paper, it is again shown that centralized planners can yield an optimal solution in almost all situations, but computation time significantly increases when adding active agents. The behavioral method described is more of a dispatch approach, where each agent is calculating a solution based solely on its own SA. The global cost for executing the tasks increases with the addition of agents in the fleet. No communication is present between the agents regarding task allocation, resulting in more allocation conflicts with more agents driving up the global cost. The market based approach utilized in this paper is said to outperform the central method for computation time for a fleet of more than five agents. Furthermore, the market-based approach has near optimal results when compared to the centralized planner.

An interesting question is analyzed in [17]- why use a competitive behavior, like an auction algorithm, to implement a cooperative mechanism? A simple answer is that agents are not programmed to be selfish, although this cannot be said of all the implementations of the auction algorithm. As explained in the English auction [15], the prices are incremented until each agent reaches the threshold where it either cannot make any revenue or has won the task. This is clearly a selfish approach, but still yields a solution to the task allocation algorithm. Another factor concerning auction algorithms is starvation of tasks or agents. These are situations where not all tasks get allocated to agents, or where not all agents get allocated tasks. In the understanding that the solution found is near optimum, the latter situation is of not much interest, but the first must be prevented. This could be done by increasing the price over time so older, unassigned tasks become more attractive [22].

Use of auctioneers alongside the application of the Partially Observable Markov Decision Process to all available tasks in an decentralized framework is shown to be an interesting approach [25]. The task solving process is fully solved based on local SA using the Markov properties, although a global optimized solution cannot be guaranteed. A similar approach is taken in [26], where the authors use a Markov chain search process along with simulated annealing to produce a neari optimized task allocation solution. A repeated greedy on-line auction task allocation algorithm is analyzed in [27] and compared to the optimal off-line solution (competitive ratio). The authors conclude that a competitive ratio of 1 can be found when the payoff for assigning new tasks is close to uniform. Under the above stated conditions, the on-line repeated greedy algorithm can find a close-to optimum solution. Furthermore, in [27] the authors prove the existence of a generally accepted lower-bound competitive ratio of 1/3 for greedy auction algorithms under certain conditions [28]. The authors in [29] use sequential and parallel single-item auctioning and repeat the auctioning process for the tasks (items) until completion of the task. This introduces the ability of re-auctioning tasks that due to circumstances cannot be completed or allocating tasks to better suited agents and thus increasing the overall task allocation solution. In [30] the authors develop a probability model to determine the preference for a task and an auctioning algorithm is to assign task and resolve task conflicts.

The general auction algorithm is used in [31, 32] for allocating tasks with precedence constraints and deadlines. The use precedence- and deadline-constraints to split the set of tasks in disjoint groups where the agents are only allowed to bid for one task out of one particular group. The authors show through simulation that this method gives almost optimal solutions for varying values of the minimum price increase.

3.2 Consensus algorithm for Situational Awareness

Centralized planners are very sensitive to the consistency of the SA across the fleet. When the SA is not uniform, each agent will create different solutions to the set of tasks, T. This can cause allocation conflicts, where several agents will target the same task. To converge to a consistent and true SA among a fleet of multiple agents, K, in a real-world scenario with limited communication is not feasible. The SA awareness of each agent is updated too frequently to enable relaying to each agent in the fleet each time the environment changes. Consensus algorithms are used to get a level of consistency in the state information, , of each agent in the fleet, although this consensus state might still be inconsistent with the true SA of each individual agent. The consensus algorithm only requires network communication between local neighbors to reach consistent state information across the fleet of agents. The state information for vehicle k, θ^k, consists of data that is relevant for the task and allocation and execution process. For instance, the state information will hold data regarding GPS position, velocity, and health (including fuel state) for vehicle k [11]. The general consensus equation is formulated as

$$\dot{\theta}^k = \sum_{l=1}^{N^k} \sigma_{kl} G_{kl} (\theta^l - \theta^k) \tag{3.1}$$

where G is a matrix representing the communication network, having a value of one if there is direct communication possible from agents $k \in K$ to $l \in K$. σ is a weighing matrix, determining the confidence agent k has in the state information of agent l. A discrete version of the consensus algorithm can be implemented as

$$\theta^k(t+1) = \theta(t) + \sum_{l=1}^{N^k} \sigma^k l G_{kl} (\theta^l(t) - \theta^k(t)) \tag{3.2}$$

Each time an agent receives state information from a neighboring agent, it will execute the consensus algorithm and update its local state information. The final consensus value is a weighted average of the initial state information of all the agents [11]. In [9] and [33], researchers demonstrate that a global consensus can be achieved asymptotically without communication noise and when a spanning tree can be found in directed graph G. Even with existing noise in the communication of the state information, the consensus algorithm can achieve a certain level of consistency, ϵ. The level of consistency is important within the cooperative control algorithms for determining the quality of the solution and avoiding task allocation conflicts.

3.3 Consensus algorithm for task allocation

Consensus-Based Auction Algorithm (CBAA) and
Consensus-Based Bundle Algorithm (CBBA) [6] are two methods proposed by a research team from Massachusetts Institute of Technology (MIT). Where consensus before was built on the SA, these methods build consensus on a winning bid list created by an auction algorithm. Creating consensus on the winning bid list takes away the necessity of having any auctioneers managing the auction process. The task-allocation problem is described in [6] as the following equation,

$$\max \sum_{k=1}^{N^k} \left(\sum_{j=1}^{N^t} C_{kj}(X^k, p^k) x_{kj} \right) \tag{3.3}$$

with constraints making sure that all tasks are being assigned and no task-conflict exists. The variable X^k represents the task list for agent k, and the vector p represents an ordered sequence of the assigned tasks, creating the flight path. The decision variable x_{kj} is 1 when agent k has been assigned task j, otherwise, x_{kj} is equal 0. The summation between the parentheses is the local reward for agent k, depending on the scoring function c_{kj}, which in turn depends on the tasks assigned x^k and the flight path p. CBAA is a single-task assignment algorithm, which can only assign one task at a time. When multiple tasks need to be allocated, this must be done in sequential fashion. CBBA, on the other hand, can handle multiple tasks at a time and will bundle tasks that together create a bigger reward. The CBAA and CBBA algorithms are comprised of two phases, which will be further described.

Phase 1 is the auction process, where agents bid on tasks. First, a valid task list h_{ij} is created by comparing the highest bid value y_{kj} for each task j with the cost/reward function c_{kj}, where $h_{kj} \in \{0, 1\}$, and h_{kj} will be 1 if task j for agent k has a bigger reward c_{kj} than the winning bid list y_{kj}. Once the valid tasks have been determined, the algorithm will choose the task with the maximum reward and update the winning bid list y_{kj} and the local task list x^k. An iteration of the CBAA follows with phase 2 of the algorithm; CBBA on the other hand keeps bundling tasks.

Phase 2 is the consensus part of the algorithm, where agents try to converge to a global consistent winning bid list y^k. At each iteration of phase 2, agent k receives the winning bid list y^k from its neighbors. Agent k will scan this received list and update its local y^k with the highest bid value y_{kj} found in the received list. If it finds itself being outbid by another agent, it will remove the task from its local task list x^k. An adjacency matrix g_{lk} again represents a direct network connection between agents k and l in the fleet. Consensus will only be applied for the agents where g_{kl} is equal to 1. Determining the point of consensus is based upon the bandwidth used for communicating the winning bid list among agents, when communication comes to a stop convergence has been reached, so assumed. But in a wireless imperfect communication network this might not at all time be sufficient and reliable. Futhermore significant network traffic is created to resolve all inconsistencies in the winning bid list for a short period of time increasing with number of agents and tasks. This spike in network traffic can further degrade task allocation reliability.

The score for a task is determined by a time-discounted reward $S_j^{P_j}$ given by Eq. 3.4 and a cost based on the distance to travel for task j.

$$S_j^{P_k} = \sum \lambda_j^{\tau_k^j(P_k)} c_j \tag{3.4}$$

Where $\lambda_j < 1$ is the discounting factor, τ_k is the estimated time agent k will arrive at task j, and c_j is the static reward for task j. The time-discounted reward is penalizing tasks for being executed later where the reward diminishes with time. But depending on λ the cost for task j could quickly outweigh the reward S_j far before the deadline of task j and allocation of task j becomes economically not feasible.

The CBBA algorithm incorporates some major changes in the auction process. Where CBAA could only handle a single task at a time, CBBA can construct a bundle consisting of multiple tasks and send this over to the consensus-phase. Although the CBBA results in conflict-free task allocation, it does not account for obstacles within the path p^k, possible influencing the marginal and total score for agent k. In [19], the obstacle avoidance algorithm is implemented locally at each agent. When the final assignments for each agent is known, the paper proposes to check for collisions and, using Dijkstras shortest-path algorithm, add waypoints/tasks to the path list p^k, staying clear of any obstacles and no-fly zones.

Churning is described as when agent k is not able to decide which task of a set of tasks should be added to the bundle due to sensor noise. The agent will jump between adding task A to the bundle and, in the next iteration, task A will be abandoned and task B will be added to the bundle. This can cause degradation of the algorithm, and in [19] is mitigated by adding a bonus if the same task is assigned to the same agent, resulting in a higher marginal reward than adding another task and having to remove the already existing tasks in the list.

In [34], the above explained CBBA algorithm is extended to incorporate the ability of handling pop-up targets and limited network communication. An extra phase is added to the original CBBA algorithm for dealing with new tasks, which invalidates all the values occurring after the insertion of the new task in the bundle. Furthermore, where CBBA applies consensus on all tasks in the bundle, ECBBA limits this to achieving consensus per task in the bundle. Although consensus is now achieved per task, just as in the CBAA algorithm, the bundling of tasks still achieves better overall optimization. The network communication range for applying the ECBBA algorithm is limited to only the agents near the tasks. The authors in [34] have observed that only agents near the task engage in the bidding process and thus the network for applying the ECBBA algorithm can be limited in range. From simulation they obtain a loss of 2% in optimality with limited communication compared to a full communication network.

The functionality of the CBBA algorithm is extended in [35] by adding a reward for executing tasks within a time window, fuel cost, and agent capability. The scoring function for each task is also updated updated with additional factors for fuel consumption, and adding penalties for executing tasks beyond their valid time window. These factors must comply with the Diminishing Marginal Gain (DMG) property in order to ensure convergence. Additionally, constraints are added to Eq. 3.3 to prevent tasks from being executed by incapable agents. Furthermore, the paper discusses several real-time re-planning scenarios under varying network connectivity and their effects on the optimality of the solution.

Whereas the bundle phase of the CBBA algorithm can be fully executed asynchronously on each agents with respect to the other agents in the fleet, the consensus phase cannot. The consensus phase, due to the limitations of the de-confliction rules in table 1 of [6], must be synchronous amongst the agents because of the necessity of applying consensus on the latest and most up-to-date information. In [36, 37] a fully asynchronous CBBA algorithm (ACBBA) is described, where the de-confliction table is updated and extended to provide asynchronous consensus on the winning bid list. Whereas the original table only described how to update the winning bid list y and time stamp vector s, the newly proposed table also specifies which messages to re-broadcast and when. The ACBBA algorithm achieves similar optimality compared to the original CBBA algorithm, with faster convergence and fewer messages passed between agents. Since the ACBBA de-confliction table determines when and how messages are being communicated among agents in the fleet, the authors in [37] propose to determine convergence of the consensus phase based on network traffic.

The Team Consensus-Based Bundle Algorithm (TCBBA) is introduced in [38]. This algorithm is designed to be used among static teams of agents, reducing the number of human operators to one per team. TCBBA initially allocates tasks within a team using CBBA but then applies task sharing between teams. Two versions of the TCBBA algorithm are discussed. In the first version, all of the teams share the tasks that are unassigned after consensus has been reached within the team of assigned tasks. What follows is another round of the CBBA algorithm on the complete list of unassigned tasks. Tasks being allocated in the outer-loop of the CBBA algorithm are inserted in the original solution. The second version of the TCBBA algorithm performs the outer-loop CBBA algorithm on unassigned tasks as in the first version, but instead of using the original solution it will create a completely new solution.

The original CBAA algorithm proposes allocating all tasks before executing the assigned tasks. Das et al. [39] describe executing the assigned tasks in parallel with running their version of the CBAA algorithm. This effectively handles dynamic changes in the environment, causing differing optimal assignments of the tasks among the fleet of agents.

In [40] the authors focus on extending the CBBA algorithm to incorporate priority tasks. A third phase is added to the original two phases of the CBBA algorithm to resize the initial overloaded bundle of tasks to the capacity of the agent, while maintaining the allocation of priority tasks. Furthermore, the authors increase the score of priority tasks by weights and biases so they outbid non-priority tasks.

CHAPTER 4

ASYNCHRONOUS POLLING CONSENSUS-BASED BUNDLE ALGORITHM

In order to address the issues regarding determining the point of consensus, the cost outweighing the reward, the network bandwidth spikes, and statically forming teams as discusssed above, the Asynchronous Polling Consensus Based Bundle Algorithm (APCBBA) has been developed. Furthermore the APCBBA algorithm has several updates regarding the problem definition outlined in Chapter 2. These improvements include an updated score function, a polling strategy to request the winning bid list from neighboring agents and dynamic team forming. The changes to the scoring function are designed to increase the percentage of allocated tasks among the agents, prevent starvation of tasks, and increase the quality of the solution. Requesting the winning bid list from neighboring agents provides a means to easily determine consensus among the set of agents $K = \{1, ...N^k\}$ and introduces new features for determining consensus. Team forming of agents creates independently working units able to accomplish otherwise not possible tasks or to more efficiently execute tasks. Imperative to the task allocation problem is the centralized objective function in Eq. 4.1, which strives to maximize the score $S_j^k(P^k)$ over all the agents K and for all the tasks $T = \{1, ...N^t\}$.

$$\max \sum_k \sum_j S_j^k(P^k) \qquad j \in T, \quad k \in K \tag{4.1}$$

$$\tag{4.2}$$

Where P^k is an ordered set of tasks starting based on the scheduled starting time $t_j^k(P^k)$.

4.1 Scoring function

One important measure of QoS is the response time from the time the task is released to the agents to the time when the task is scheduled, in order to reduce this response time, a time-discounted reward is applied to each task. This means the reward for a given task will be reduced when a task is scheduled later in time. A potential problem with the time-discounted reward function is starvation of tasks caused by the reward being reduced to a value below the task's cost far before the deadline of the task. As in [6], the time-discounted reward is solely dependent on the release time of the task and applies an exponential decline of the task reward. A more subtle approach to applying time-discounted reward is penalizing the reward more heavily when it is getting closer to the deadline of the task. In this case, tasks are still penalized for being executed later but the task will retain the bulk of it's initial reward for longer, outweighing the cost. This effectively increases the number of tasks being allocated. The proposed reward function used in APCBBA is:

$$R_j^k(P^k) = r_j(1 - e^{\lambda_j(t_j^k(P^k) - \rho_j)}) \tag{4.3}$$

Where $R_j^k(P^k)$ is the dynamic reward and r_j is the static reward for task j, and $t_j^k(P^k)$ and ρ_j are the scheduled start time of task j and the deadline of task j, respectively. This reward function will shift the emphasis, in comparison to the reward function used in [6], from the release time of the task to the deadline of the task.

Monte-Carlo simulation results in Fig. 2 outline the aforementioned feature of the proposed reward function. The agents and tasks are randomly placed on a 4 Km^2 2-D space; λ_j is a gaussian distribution in the range of 0.1 to 1 s-1, release times and durations are randomly chosen in a set interval. Fig. 2 shows that the proposed reward function has a higher percentage of allocated tasks than the original reward function from [6], increasing the number of tasks being allocated.

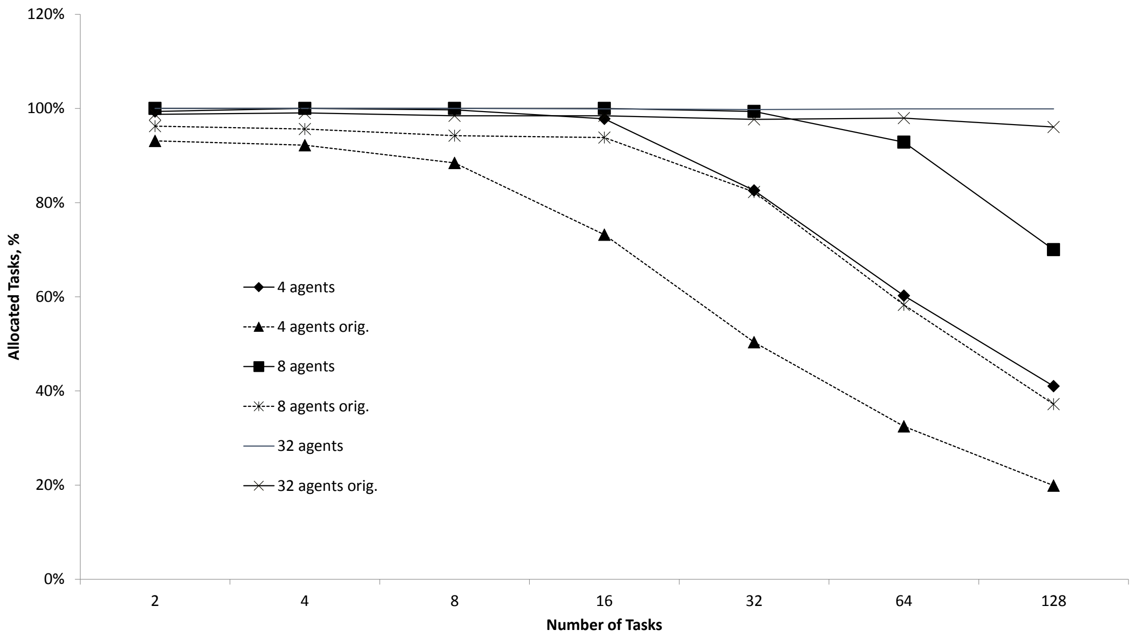


Fig. 2. Average percentage of allocated tasks vs. number of tasks for differing numbers of agents, using the reward function from Eq. 4.3 and the original time-discounted reward function applied in [6]

.

The cost function $C_j^k(P^k)$ is based on the total distance $d_j^k(P^k)$ traversed for all the previous tasks in the ordered set of tasks P^k, up to and including the newly inserted task j.

$$d_j^k(P^k) = d_{\text{start}}^k(P^k) + \sum_i d_{i \to i+1}^k(P^k)) \qquad i \forall P^k, i \in T \tag{4.4}$$

Where f^k is a fuel penalty per distance unit, $d_{\text{start}}^k(P^k)$ is the distance from the depot (i.e., the starting point) to the first task in P^k, $d_{\text{end}}^k(P^k)$ is the distance from the last task in P^k to the depot or starting location of agent k, and $d_{i \to i+1}^k(P^k)$ is the distance from task i in the path P^k to the next task in P^k. The above equation holds true to the Diminishing Marginal Gain property [6], since the score for task j cannot increase as other tasks are added before task j in P^k. The Diminishing Marginal Gain property is required to guarantee the overall task allocation solution converges to a better solution with every iteration through the algorithm. The cost function $C_j^k(P^k)$ is based on the total distance $d_j^k(P^k)^k$ traveled for task j multiplied by a fuel penalty f^k.

$$C_j^k(P^k) = f^k \cdot d_j^k(P^k) \tag{4.5}$$

The final score $S_j^k(P^k)$ for task j at agent k is given by Eq. 4.6 and has several additional parameters to it to produce feasible solutions and influence the solution based on health, and suitability of agent k to task j. $U_j^k(P^k)$ prevents the assignment of tasks j before the release time φ_j and V_j^k makes sure that task j is given ample time for execution before the deadline of task ρ_j. Z_j^k is the suitability of agent k for performing task j and is implemented as a capability matrix.

$$S_j^k(P^k) = \left(R_j^k(P^k) - C_j^k(P^k)\right) \cdot U_j^k(P^k) \cdot V_j^k(P^k) \cdot \\ H_j^k(P^k) \cdot Z_j^k \tag{4.6}$$

Where,

$$U_j^k(P^k) = \begin{cases} 1 & \varphi_j \leq t_j^k(P^k) \qquad j \in T \\ 0 & \text{otherwise} \end{cases}$$

$$V_j^k(P^k) = \begin{cases} 1 & t_j^k(P^k) + \tau_j \leq \rho_j \qquad j \in T \\ 0 & \text{otherwise} \end{cases}$$

$$H_j^k(P^k) = \begin{cases} 1 & d_{\text{rem}}^k \geq d_j^k(P^k) \qquad j \in T \\ 0 & \text{otherwise} \end{cases}$$

$$Z_j^k(P^k) = \begin{cases} 1 & \text{task } j \text{ is suitable for agent } k \qquad j \in T \\ 0 & \text{otherwise} \end{cases}$$

One important property the scoring function needs to adhere to, as Eq. 4.6 does, is the Diminishing Marginal Gain (DMG). This property requires the score of task j not to increase when other tasks are added before it. The property merely ensure that during the decentralized task allocation process, when task j is assigned to a different agent then agent k, the score for agent k does not decrease more than the marginal score of task j and thus the total score for all agents in K always increases.

4.2 Bundling algorithm

The optimality of the decentralized task allocation solution is also dependent on the outcome of the bundling algorithm running locally on each agent. The bundle algorithm is by no means an exhaustive search of the total solution space, and thus, will not find the optimal path for the given task set T in all circumstances. The following algorithm gives an overview of how tasks are bundled to give a local path.

For every task in T loop through the unassigned tasks in T and add a task j to path P^k, at all the possbile positions, and determine the position and task which is yielding the hightest total score for path P^k.

The bundling algorithm takes more computation than the algorithm explained in [6] because the score of task j is dependent on the distance traveled to all the previous tasks in P^k. This makes the APCBBA bundling algorithm of complexity $\mathrm{O}\big((N^t)^4\big)$, whereas the original algorithm is of complexity $\mathrm{O}\big((N^t)^3\big)$.

4.3 Asynchronous Polling

The original strategy for converging on a consistent task assignment proposed in [6] exchanges a set of vectors amongst neighboring agents by actively sending these vectors to other agents when a significant update in the local list has occurred. Since these algorithms are expected to run synchronously over all agents, this could cause a significant increase in message passing and bandwidth usage for a short period when a new task is being released. The bandwidth used will be an exponential function of the number of agents and number of tasks released. Fig. 3 gives an excerpt of the network bandwidth used during a mission where five sets of ten tasks are released to four agents. The communication between agents is simulated using a mesh-type network in the aforementioned ns-3 simulator. An asynchronous non-polling implementation of the CBBA task allocation algorithm best described in [37] is used to create the traffic seen in Fig. 3. At every peak a set of ten tasks is released and agents actively send their winning bid lists to neighboring nodes. These communication peaks can cause for an increase in the Packet Loss Ratio (PLR) in the network which can, in turn, degrade the task allocation solution and increase the convergence times. The baseline bandwidth usage in Fig. 3 represents the normal sending of status packets from the Flight Control System in each vehicle to the Ground Control System.

To increase stability of the task allocation process, reduce convergence times, and easily handles changes in the SA, APCBBA uses a polling strategy to request and receive the winning bid list from direct neighboring agents. This strategy introduces several new concepts and features to reduce convergence times, reduce the network communication spikes as seen in Fig. 3, and increase task allocation stability. Figure 4 shows the network bandwidth usage in a similar mission scenario as Fig. 3, only the network communication spikes are reduced due to the polling strategy of the APCBBA algorithm. The spikes have been significantly reduced with a slight increase of the baseline bandwidth usage due to continuous communication of the winning bid list.

Each agent will communicate several vectors which include two vectors for identifying the task within the lists, $n^k \in \mathbb{R}^{N^t}$ which is a unique identifier for a single or group of tasks and $m^k \in \mathbb{R}^{N^u_j}$ which identifies sub-tasks within a group of task. The other vectors are the winning bid list $y^k \in \mathbb{R}^{N^t}$, the winning agent list $x^k \in \mathbb{R}^{N^t}$, a vector $s^k \in \mathbb{R}^{N^t}$ for the state of task j and a time stamp vector $\phi^k \in \mathbb{R}^{N^t}$ indicating the last update to a task j in s^k.

Task j will exist in these lists up to to the point that the winning agent has *completed* task j or when task j has been *aborted* or *deleted*, which will be communicated through vector s^k. The general reduction of the bandwidth used throughout the overall mission shown in Fig. 4 when a new set of tasks is released is due to the fact that the vectors to be communicated shrink in size when tasks are being executed and completed. Keeping the tasks unexecuted in the list provides the ability for any agent to keep bidding for tasks that previously might not have been of interest but through changes in the task allocation, SA, or environment, at a later stage become more interesting to it.

Table 1 provides the decision rules for updating agents local set of vectors y^k, x^k, and s^k when the vectors from a neighboring agent l have been received.

Each task in an agent's list can be in one of the following states: *idle/reset, auction, assigned to agent, executing, completed, or deleted.* Only when neighboring agents indicate the state of task j is *executing, completed, or deleted* will the local agent, upon receiving this information, update it's own state for task j. Only when the state of task j for both the receiving and local agent is in *auction*, or *assigned* status can the bidding and consensus steps be applied. A conflict free assignment is assumed for a task during the transition from *assigned* to agent, to *executing*, meaning only one agent will have been assigned to the task and will be able to set the state to executing. When conflicting task assignments are encountered, for example when agent k thinks task j is assigned to agent l and agent l thinks task j is assigned to agent k, task j will be *reset* for both agents and the bidding process starts again for task j.
Consensus is determined for each task in the path P^k, only when all neighboring agents with direct communication agree on the assignment of task j to agent k and all agents before task j in path P^k has been assigned to agent k, will the status of task j at agent k transition from *auction* to *assigned.* Consensus on the assignment of task j to agent k is achieved when every agent with a direct connection to agent k at that time, agrees that task j should be assigned to agent k. This consensus is given by equation Eq. 4.7.

$$\gamma_j^k = A_j^{kl} \wedge G_l^k(t) \qquad \forall k, j \tag{4.7}$$

Table 1. Decision rules for agent k (receiver) upon receiving vectors y^l and x^l from agent l (sender)

x_j^l is	x_j^k is	Receiver's action
l	k	If $y_j^l > y_j^k \rightarrow$ update y_j^k & $A_j^{kl} = 0$ for task j If $y_j^l = y_j^k$ and $h^l > h^k \rightarrow$ update y_j^k and $A_j^{kl} = 0$ for task j If $y_j^l < y_j^k \rightarrow A_j^{kl} = 0$ for j
	l	If $y_j^k > y_j^l \rightarrow$ update y_j^k and $A_j^{kl} = 0$ for task j If $y_j^l < y_j^k \rightarrow s_j^k = reset$ and $A_j^{kl} = 0$ for task j
	$m \notin k, l$	If $y_j^l > y_j^k \rightarrow$ update y_j^k and $A_j^{kl} = 0$ for task j If $y_j^l = y_j^k$ and $h^l > h^k \rightarrow$ update y_j^k and $A_j^{kl} = 0$ for task j
	none	update y_j^k and $A_j^{kl} = 0$ for task j
k	k	$A_j^{kl} = 1$ for task j
	l	$s_j = reset$ and $A_j^{kl} = 0$ task j
	$m \notin k, l$	$A_j^{kl} = 0$ for task j
	none	$A_j^{kl} = 0$ for task j
$m \notin k, l$	k	If $y_j^l > y_j^k \rightarrow$ update y_j^k and $A_j^{kl} = 0$ for task j If $y_j^l = y_j^k$ & $h^l > h^k \rightarrow$ update y_j^k and $A_j^{kl} = 0$ for j
	l	update y_j^k & $A_j^{kl} = 0$ for task j
	m	$A_j^{kl} = 0$ for task j
	$n \notin k, l, m$	If $y_j^l > y_j^k \rightarrow$ update y_j^k and $A_j^{kl} = 0$ for task j If $y_j^l = y_j^k$ and $h^l > h^k \rightarrow$ update y_j^k and $A_j^{kl} = 0$ for task j
	none	update y_j^k and $A_j^{kl} = 0$ for task j
none	k	$A_j^{kl} = 0$ for task j
	l	Reset task and $A_j^{kl} = 0$ for task j
	$m \notin k, l$	$A_j^{kl} = 0$ for task j
	none	$A_j^{kl} = 0$ for task j

Where $A_j^{kl} \in \mathbb{R}^{N^k}$ is a boolean variable where the value is determined by table 1 and the l^{th} element is equal to "1" when both agent k and agent l agree on task j to be assigned to agent k. $G_l^k(t)$ is a directed graph for agent k indicating direct communication between neighboring nodes in the communication network at time t. The logical conjunction $\wedge$ operator applies a logical 'and' operation to both the directed graph $G_l^k(t)$ and A_j^{kl}. During the period when tasks are in the *assigned* status, the bidding and consensus steps are still applied giving the opportunity to other agents to still put in a higher bid for task j due to changes in SA. Only when the status of the task j, by agent k, is updated to *executing* will this prevent any other agents from bidding on task j. Once the executing agent has completed it's assigned task j the status will be changed to *completed.* In order to delete task j out of the communication vectors y_j^k, x_j^k, and s_j^k, the task has to acquire the state *completed or deleted* and all neighboring agents will have to transition to the same state through communication, before the task can be deleted from the local vectors.

4.4 Team formation

Team forming can be instantiated in situations where an individual agent could not perform the available tasks due to time, resource, or capability constraints. In these mission scenarios, additional agents can be added to the task in order to mitigate the constraints holding back execution of task j. The process of forming teams for execution of task j is a multi-step process which can be applied to any task in need of assistance from other agents. Whenever a task $j \in T$ can not be solely executed by agent k, the task can be split in a set of sub-tasks N_j^u. The agent is free to determine the cardinality of N_j^u in order to maximize the score of j. A preliminary score for the base-task j will be calculated based on the current SA of agent $l \in K$ and its knowledge of neighboring agents. Each agent will still compete for base-task

j and through the market-based and consensus principles of APCBBA the winning agent will be assigned the base-task. At this moment the winning agent will release all sub-tasks N_j^u to neighboring agents and where each sub-task $n \in N_j^u$ is given an appropriate static reward r_u, release time ς_u, duration τ_u, and deadline ρ_u. In order to prevent fragmentation of the base-tasks, only agent k holding the base-task j is allowed to divide task j; i.e. any agents winning any of the sub-tasks N_j^u are not allowed to subsequently divide sub-task $u \in N_j^u$.

Algorithm 1 Multi-agent team forming for agent k

```
1: Agent $k \rightarrow$ solve task $j$ and create sub-tasks $N_j^u$
2: Agent $k$ bids for task $j$
3: if $\gamma_j^k = 1$ then
4:     if $\left|N_j^u\right| \geq 1$ then
5:         Release tasks $N_j^u$ to all agents $l \in K$
6:         if $\gamma_u^l \; \forall u \in N_j^u, k \neq 1$ within timeout period then
7:             Release and reset base-task $j$ to all agents $l \in K$
8:         else
9:             if $\gamma_u^k \; \forall u \geq 1$ then
10:                Execute all tasks $u \in N_j^u$ assigned to agent $k$
11:            end if
12:        end if
13:    else
14:        Execute task $j$ assigned to agent $k$
15:    end if
16: end if
```

Once the sub-tasks N_j^u are released any agent can start bidding for these sub-tasks, and agent k who has won base-task j will wait for all sub-tasks to be assigned; i.e. $\gamma_u^l \forall u, j$ should be equal to one. Once all tasks are assigned, agent k can claim the reward for base-task j minus the cost, which will be the static rewards r_u assigned to the sub-tasks $u \in N_j^u$. At this point agent k will assume all sub-tasks N_j^u will be executed and completed in the near future by their assigned agents, and thus can proceed with with normal operation including executing any task $u \in N_j^u$ assigned to agent k.

Although the above algorithm was chosen for supporting team formation with the group of agents, a different strategy would be to have each agent release its sub-tasks even before it acquired the base-tasks. Each agent would release all the sub-tasks it calculated as the solution and wait for these sub-tasks to be assigned before applying a bid to the base-task. This would take away the concern of when agent j is winning the base-tasks but all of its released sub-tasks are never assigned, as could be the case in the first algorithm. Some concerns with the latter algorithm is the support of another level of sub-tasks or sub-tasks needing to have unique IDs so they can be distinguished per agent. Furthermore each agent will not only be calculating and releasing the sub-tasks for its solution of the base-task, it would also have to apply bids for sub-tasks of the same base-task but released by other agents creating conflicts of interest. This would take a considerable amount of extra computation per agent and would make the team formation procedure more complicated, and the task alloction algorithm would be flooded with sets of sub-tasks for each agent per base-task.

```
1: procedure FindMax($T$, $P^k$)
2:     $MaxScore \leftarrow 0$
3:     for all tasks in $T$ do
4:         for all task $j$ in $T$ not assigned do
5:             for all Indexes $m$ in $P^k$ do
6:                 Insert task $j$ in $P^k$ at index $m$
7:                 for all tasks $i$ in $P^k$ do
8:                     Calculate score $S_i^k$ for task $i$
9:                     $TotalScore \leftarrow TotalScore + S_j^k$
10:                end for
11:                if $TotalScore > MaxScore$ then
12:                    Store task $j$ and index $m$
13:                    $MaxScore \leftarrow TotalScore$
14:                end if
15:            end for
16:        end for
17:        Add task $j$ with highest score to bundle $B^k$
18:        Insert task $j$ into path $P^k$ at position $m$
19:    end for
20: end procedure
```

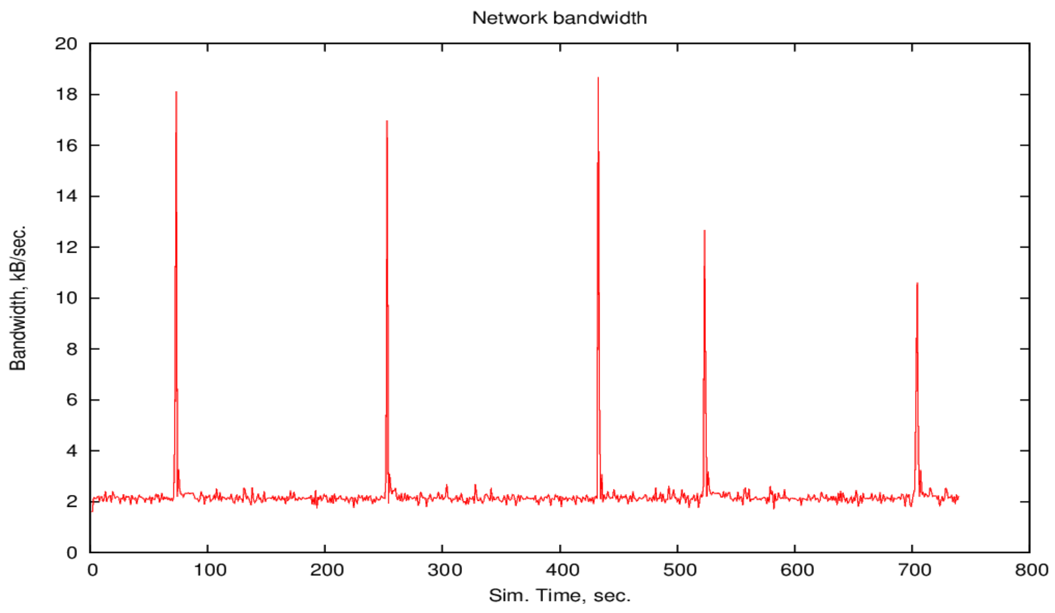


Fig. 3. Typical network bandwidth usage during a mission with four agents and ten tasks using an asynchronous task allocation algorithm.

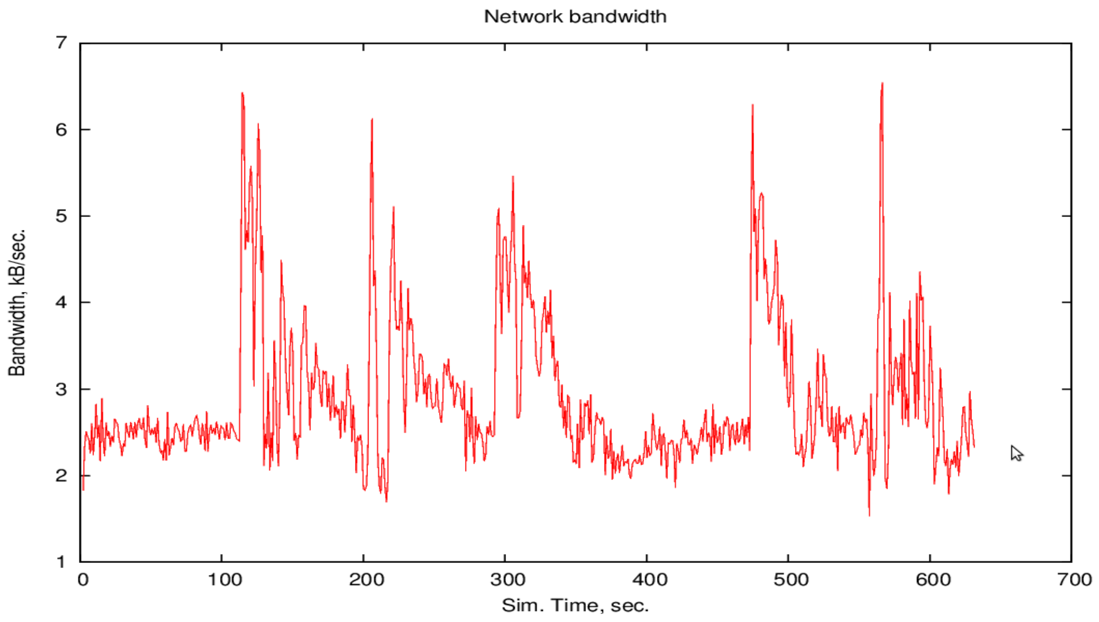


Fig. 4. Typical network bandwidth usage during a mission with four agents and ten tasks using a polling asynchronous task allocation strategy.

CHAPTER 5

MISSION CONTROL SYSTEM

The execution of the task allocation algorithms is performed by the Mission Control System (MCS). This integrated piece of software is responsible for handling tasks, calculating the score of tasks, creating bundles of tasks, executing tasks, running the APCBBA algorithm and acting as a pass-through communication hub for communications between the Flight Control System (FCS) and Ground Control System (GCS). This FCS to GCS communication data could represent waypoint command changes or commands for changing altitude, but also includes reporting information like current position, altitude, or orientation. All this communication data is been captured by the MCS to produce an updated Situational Awareness (SA), which is periodically broadcast to neighboring agents. Agents receiving SA data from other agents can use this information to detect possible mid-air collisions or utilize this information to solve a task where team formation is required, as is described in section 4.4.

Figure 5 illustrates the different on-board and ground-based components and their communication architecture which together form the intelligent UAS for executing collaborative task assignments. The on-board communication between modem and MCS is established through a 10 Mbit Ethernet connection and a serial connection is used as communication between MCS and FCS. The MCS is able to fully control the FCS through this serial connection, just as a human operator could utilizing the GCS. The wireless mesh-communication between the UAS and, in particular, the modems are described in section 7.2.

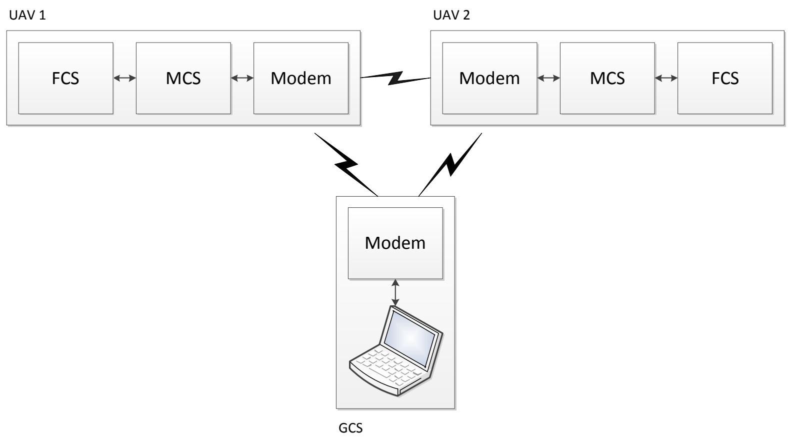


Fig. 5. Communication architecture between two UAVs and the GCS.

The MCS software is executed on a Gumstix Verdex Pro XL6P Small Board Computer (SBC) containing a processor running at 600 MHz, 128 MB of RAM, 32 MB of flash storage, and a Micro-SD slot [41]. Additional modules add capabilities for serial communication and Ethernet. An embedded version of Linux is running on the Gumstix as the Operating System (OS), providing the MCS software with a Portable Operating System Interface. This interface provides functionalities for interfacing to the file system, serial ports, and Ethernet stack, and the capability to execute threads. The Gumstix SBC can provide the necessary raw processor power for calculating and solving centralized and local task allocation problems in real-time, for sets where N^t is smaller than 12, limited by the frequency loop of the main thread.

The MCS software is fully written in C++ and uses, wherever possible, Object Oriented Programming (OOP) to utilize its power of abstraction, polymorphism, and inheritance. An architectural overview of the MCS software is given in Fig. 6. After an initialization routine, three threads are spawned, the main thread and two listener threads for the Ethernet and serial sockets. The main thread, which runs at 10 Hz, takes care of processing any VACS packets received, including processing any newly received tasks. The other responsibilities of the main thread are applying the collision detection algorithm, updating and broadcasting any SA information, and applying the task allocation algorithm. Tasks assigned to the local agent will be executed in the main thread and throughout each iteration of the loop; statistical data can be collected which at the end of the loop will be written to a file.

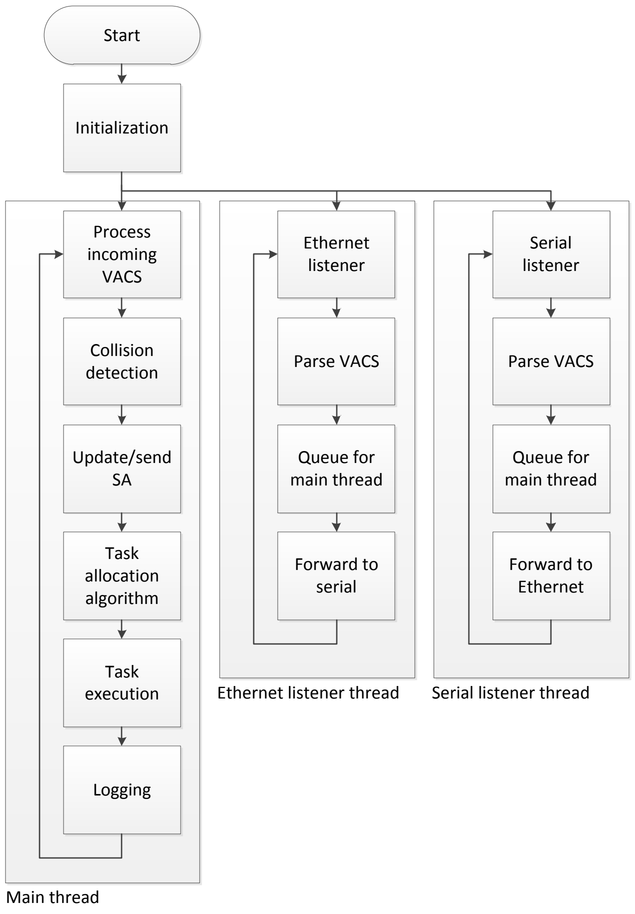


Fig. 6. MCS software architecture.

The following sections describe in more detail some responsibilities and aspects of the MCS software.

5.1 Task processing

The MCS software handles all collaborative responsibilities which include receiving and processing any tasks released by the GCS. For every received task from the GCS, a derived task-object is created depending on the type of task. The parent class is common for all derived classes and implements a set of interfaces and functionalities common to all tasks. This includes holding intrinsic parameters to the task, like the unique identification numbers, the static reward, and the release and deadline time of the task. A comprehensive set of functions provide safe and easy access to the stored parameters and functionality for solving the task allocation problem. Each class derived from the parent class implements a type of task and provides functionalities for calculating the score given the current path P^k and executing the task within its computed solution. Figure 7 shows the relationship between the base- and derived-class for different types of tasks.

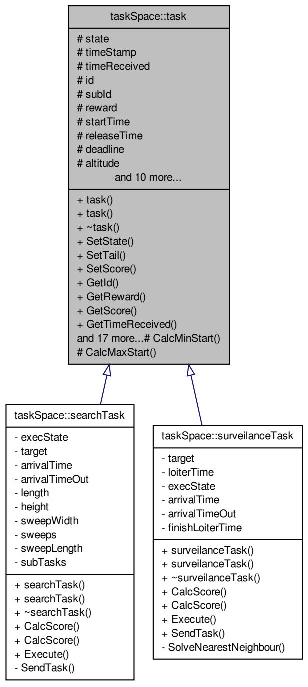


Fig. 7. Inheritance diagram for tasks.

As of now, the only supported types of tasks are search and surveillance; a new type of task would only require the derived-class to be implemented, providing functionality for calculating the score and logic for executing the task. Each time a new task object is instantiated, a partial solution for the task is computed using parameters that are not affected by changing SA or environment dynamics. Only when a score is requested by the task allocation algorithm, i.e. by calling the *CalcScore())* function, is the complete solution for the given task calculated, producing the latest and most up-to-date score. In case of a search task, at the moment of calling *CalcScore()* the search area will be divided in regions based on the agent's current knowledge of the neighboring agents, creating a solution optimized for score. Once a solution is created for the base search-task, the status of the task will transition from *idle* to *auction* and the search-task will be entered into the APCBBA task allocation algorithm. Only when the agent wins the search task and the status of the task turns into *assigned* will the agent release the sub-tasks to neighboring agents, and the APCBBA will subsequently take care of allocating these sub-tasks. After releasing the sub-tasks, all sub-tasks u_j should be assigned to neighboring agents within a timeout window. If for some reason not all sub-tasks are assigned the base-task will be set to *reset*, thus re-introducing the base-task to the task allocation algorithm for all agents, and all the released sub-tasks will be *deleted*. The different stages of a surveillance- and search-task are outlined in Fig. 8.

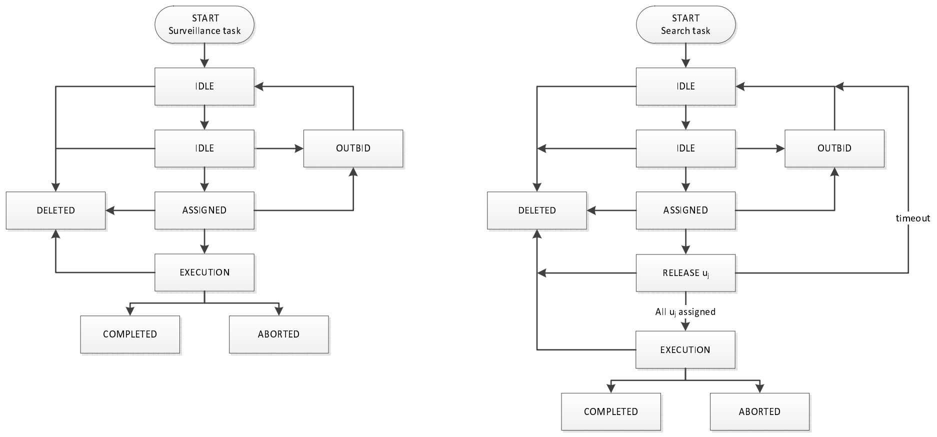


Fig. 8. Different stages of the surveillance- and search- task.

A surveillance task is introduced by the GCS as a base-task with single or multiple waypoints, where each waypoint is considered to be a point of interest and thus for each waypoint a sub-task is created. Every agent receiving this base surveillance task will split up the task in the same manner, therefore making all the sub-tasks for the given base-task identical across all agents. Once all sub-tasks are created, the agents will start to bundle their tasks, finding an optimum route/path for the given sub-tasks, where each sub-task also receives a score based on the positioning within the path. When the bundle is created, and a score has been calculated for each sub-task, the agents will enter the sub-tasks into the *auction* stage where the APCBBA algorithm will try to achieve an optimum allocation of the (sub-)tasks.

5.2 Task allocation algorithms

The implementation for the task allocation algorithms is heavily dependent on the parent class named 'algorithm', as shown in Fig. 10. This parent class provides several supporting functions for its derived classes, which consist of different types of task allocation algorithms. Some of these generic functions include; creating and storing a bundle B^k, and path P^k, and handling communications regarding the task allocation algorithm with other agents.

The algorithm for creating an optimized bundle and local path is based on iterating over the set of remaining tasks, where with each iteration, a feasible task j with the highest score is added to B^k and inserted in P^k. The aforementioned bundle algorithm in Section 4.2 is of $O\left({N^t}^4\right)$ complexity. Figure 9 shows the average run-times for solving, a single agent, centralized task allocation problem with varying number of tasks N^t, while being executed on the Gumstix SBC mentioned above.

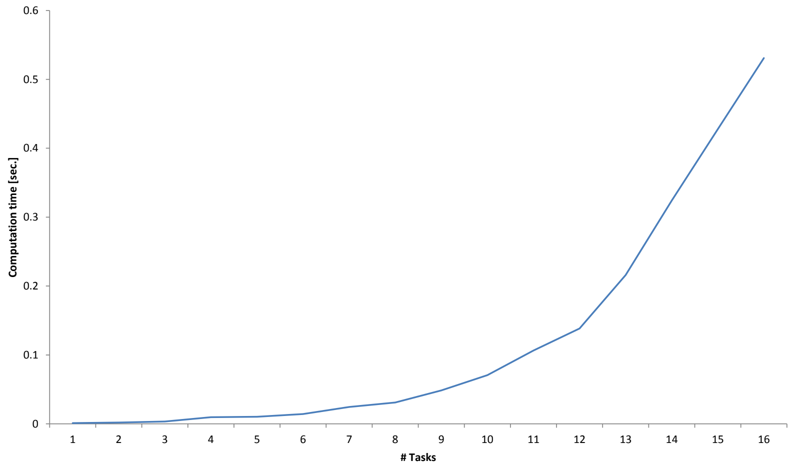


Fig. 9. Average run-time in seconds for varying number of tasks N^t.

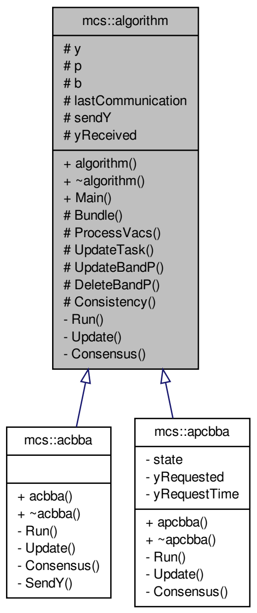


Fig. 10. Inheritance diagram for task allocation algorithms.

The derived instances of the parent class 'algorithm' implement different types of task allocation algorithms and currently implement a version of ACBBA described in [37] as well as APCBBA as described herein. The derived class implements the logic for updating the different communication vectors according to the decision table, which for APCBBA are given by Table 1, but also determines when consensus has been achieved.

The polling strategy explained in Chapter 4 is implemented in the derived class as a state machine running through states for requesting the communication vectors from neighboring agents, updating the local communication vectors and determining consensus. If consensus has been reached on an individual or a set of tasks, the determination is handled in the derived class because this differs for APCBBA vs. ACBBA and other algorithms. APCBBA will determine consensus on individual tasks thus speeding up the convergence time and executing tasks that are possible, whereas ACBBA will determine the point of consensus on the whole bundle of tasks.

5.3 Communication

Communication between agents and the GCS follows the internal VCU Aerial Communications Standard (VACS). The generic VACS packet format is shown in Table 2.

Table 2. VACS packet format

Byte	Name	Purpose
1	Sync 1	First synchronization byte
2	Sync 2	Second synchronization byte
3	Destination	Destination address of packet (tail number)
4	Source	Source address of packet (tail number)
5	Msg. ID H	Unique message ID (high byte)
6	Msg. ID L	Unique message ID (low byte)
7	Data length H	Length (N) of data field (high byte)
8	Data length L	Length (N) of data field (low byte)
9 ... 9 + (N-1)	Data field	Payload of message
9 + N	Checksum 1	First checksum (Fletcher's)
10 + N	Checksum 2	Second checksum (Fletcher's)

The source and destination addresses are determined by the tail number of the UAS where the GCS is dictated to be address '0'. Broadcasting of packets to all UAS and GCS will require the destination address of the VACS packet to hold a value of '255'.

A specific range of unique message IDs from 300 to 399 are reserved to support the collaborative framework, a subset of this range from 300 to 349 is used for general task handling, and anything above 349 and up to 399 can be used by the task allocation algorithms. The following messages are currently implemented int the Mission Control System:

Table 3. Collaborative messages

Section	ID	Purpose
Task general	300	SA reporting
	308	Task status reporting
	310	Surveillance task
	311	Search task
Algorithm	350	Communication vectors

Basic communication functions for sending and receiving VACS packets are implemented in the ’commIo’ parent class. Derived instances can support serial or Ethernet communication using UDP packets. The MCS utilizes two communication sockets, one for connecting to the wireless modem using Ethernet, and the other for a serial connection to the FCS. Both sockets run asynchronously and separately from the main MCS thread, to ensure a fast and responsive link at all times between GCS and FCS, independent on the running time of the main thread of the MCS. Any incoming VACS packets of interest to the MCS on either socket are parsed, decoupled, and queued for processing within the MCS main thread.

5.4 Ground Control Station

The Ground Control Station developed over the last ten years within the Virginia Commonwealth University UAV lab has proven to be of tremendous quality and has been extended to support this collaborative research. Such an extension encompasses

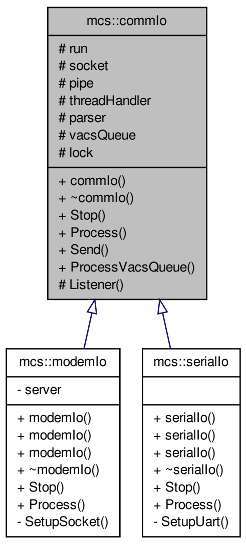
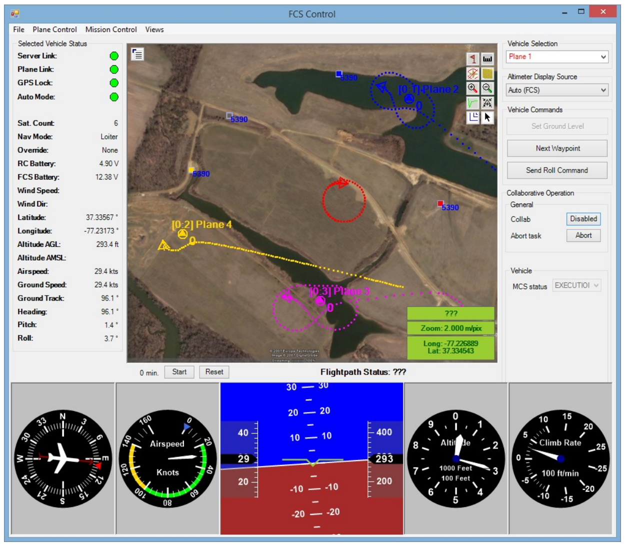


Fig. 11. Inheritance diagram for communication.

making the visual map not only display the current position of the UAV's and their waypoints but also highlighting the collaborative tasks released by the operator and the current task being executed by each UAS, as can be seen in Fig. 12.



Selected Vehicle Status

Server Link:	🟢
Plane Link:	🟢
GPS Lock:	🟢
Auto Mode:	🟢
Sat. Count:	6
Nav Mode:	Loiter
Override:	None
RC Battery:	4.90 V
FCS Battery:	12.38 V
Wind Speed:	
Wind Dir:	
Latitude:	37.33567 °
Longitude:	-77.23173 °
Altitude AGL:	293.4 ft
Altitude AMSL:	
Airspeed:	29.4 kts
Ground Speed:	29.4 kts
Ground Track:	96.1 °
Heading:	96.1 °
Pitch:	1.4 °
Roll:	3.7 °

0 min.

Flightpath Status: ???

Vehicle Selection: Plane 1

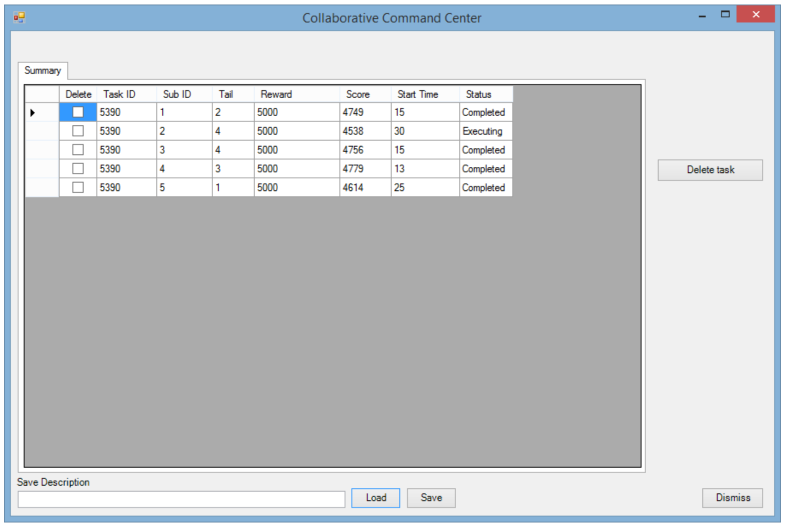
Altimeter Display Source: Auto (FCS)

Collaborative Operation — General: Collab: Disabled; Abort task: Abort

Vehicle — MCS status: EXECUTIOI

Fig. 12. Five collaborative tasks are being executed by four agents.

Furthermore, the GCS embeds a pop-up screen named the 'Mission Control Screen', shown in Fig. 13. This screen contains all the relevant information about the collaborative tasks including assignment of task to agent (tail number), starting time of task, final score achieved, and status of the task. The screen also provides the operator the functionality to abort or delete a task before it has been completed.



	Delete	Task ID	Sub ID	Tail	Reward	Score	Start Time	Status
▶	☐	5390	1	2	5000	4749	15	Completed
	☐	5390	2	4	5000	4538	30	Executing
	☐	5390	3	4	5000	4756	15	Completed
	☐	5390	4	3	5000	4779	13	Completed
	☐	5390	5	1	5000	4614	25	Completed

Fig. 13. Mission Control screen giving an overview of all the tasks completed, deleted or presently being executed.

Additional changes to the GCS include separate logging files for collaborative events, to minimize the data that is required to be parsed when comparing and analyzing the outcome of the task allocation process. Standard logging files for a typical mission scenario can easily grow to a size well over 10 MB, including all the data that is received from the UAV's stored in XML format. Parsing these huge files takes a tremendous amount of time, and thus separate logging files have been created for capturing only collaborative related events, cutting parsing time for post-mission analysis down tenfold.

5.5 Collision detection

In a crowded airspace with multiple autonomous flying UAS, a robust collision detection system is of high importance to protect the public and the aircrafts. A simple but highly effective collision detection system has been designed based on the positional knowledge the agent acquires of the other UAS through communicating and broadcasting their SA. The collision detection system should warn of any imminent mid-air collisions and take appropriate evasive maneuvers to prevent a collision.

A three tier system has been designed to indicate different levels of collision detection. The first stage is the basic *no collision* stage, which indicates that no collisions are foreseen within the near future based on the k current heading, altitude and airspeed of agent k and it's knowledge of the positions of any local UAS. Whenever two agents are within a certain distance. the warning range, and hold a similar altitude, the collision detection stage will be *collision warning*, Fig. 14.B. The last stage is *collision alarm*, the stage where an evading maneuver will be engaged. Two conditions can initiate a transition from the *collision warning* to *collision alarm* stage, either when the heading of agent k intersects with the position of agent l given a margin the heading sweep, as in Fig. 14.C, or when the two agents are within the alarm range of each other irrespective of the heading of the agents as in Fig. 10.D. The warning range (outer circle in Fig. 14) and alarm range (inner circle in Fig. 14) are dependent on the airspeed of agent k, where the radius of the warning range is greater than or equal to the alarm range and both increase with airspeed. The algorithm for collision detection is outlined in Alg. 2.

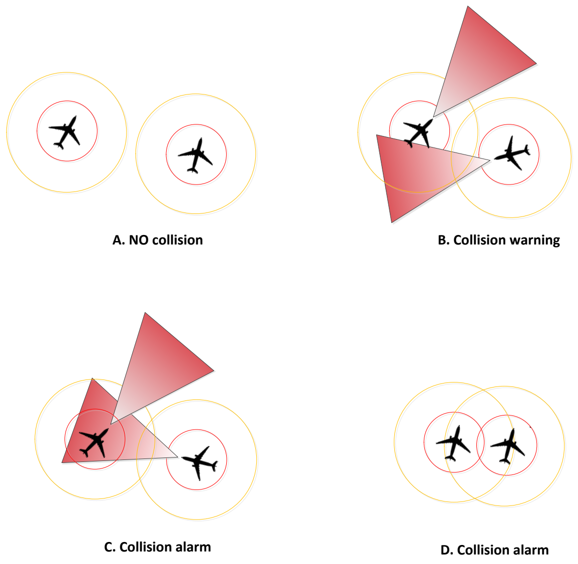


Fig. 14. Collision detections stages.

Algorithm 2 Collision detection algorithm for agent k detecting a possible collision with agent l $\forall N^k$

```
1: if distance from $k$ to $l$ < COLLISION WARNING RANGE then
2:     if |heading of $k$ - bearing to $l$| < HEADING SWEEP then
3:         $collision \leftarrow TRUE$
4:     else if distance from $k$ to $l$ < COLLISION ALARM RANGE then
5:         $collision \leftarrow TRUE$
6:     else
7:         $collision \leftarrow FALSE$
8:     end if
9: else
10:    $collision \leftarrow FALSE$
11: end if
```

The basic evasive maneuver for mid-air collisions is to command a change in altitude. Whenever an agent detects an impending collision, depending on it's current altitude relative to the other agent, it will either increase or decrease it's altitude and airspeed. This maneuver is depicted in Fig. 15, where 'UAV1' detects a potential collision and flies at a lower altitude, thus decreasing airspeed and altitude, and where 'UAV2' will do the opposite. The SA communication mentioned before keeps every agent up-to-date on positional information of neighboring agents including altitude, and through this information relative altitude between UAS can be established.

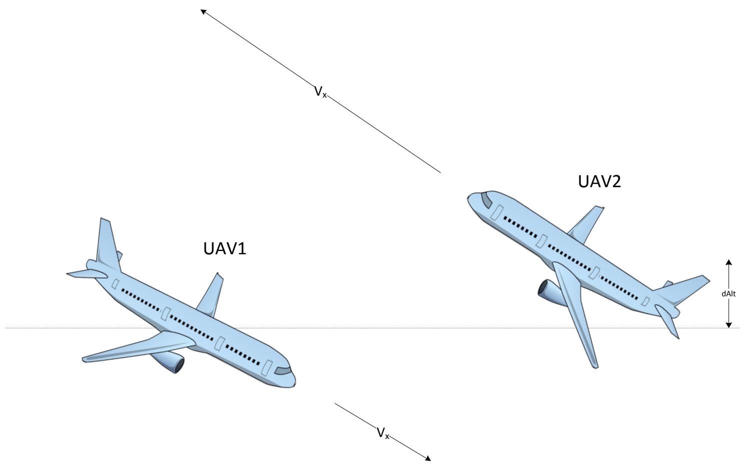


Fig. 15. Collision evading maneuver, UAV1 decreases airspeed and altitude and UAV2 increases airspeed and altitude.

CHAPTER 6

RAMS SIMULATOR

The RAMS simulator is a multiple agent, low-fidelity discrete-event simulator, designed and implemented by the VCU UAV research lab to perform simulations in the field of collaborating agents [42]. The simulator implements the modular architecture as is shown in Fig. 16.

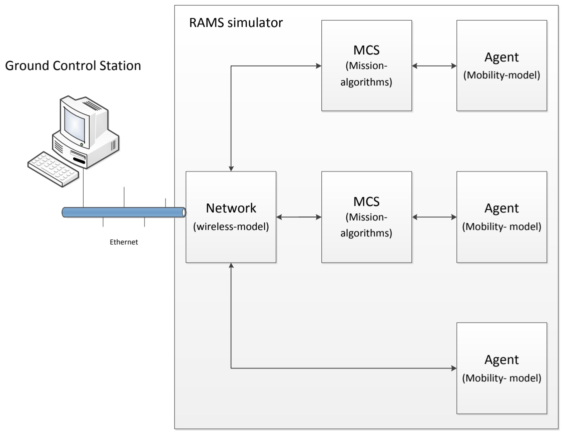


Fig. 16. Modular architecture of RAMS simulator.

The different modules in the RAMS simulator are the network, the MCS, and the agent module. The network module is a wireless network model able to run a low-fidelity mesh network, or, when connected to ns-3, capable of providing a high-fidelity wifi or mesh network [43]. Ns-3 is a widely used academic discrete-network simulator and is seamlessly integrated with the RAMS simulator. However, when running the RAMS simulator with ns-3 the overall simulation speed is limited to real-time, whereas in low-fidelity network simulation the simulation speed can be significantly increased. The mobility of the UAS is modeled by the agent module, which can utilize a low-fidelity aerodynamics model of an aircraft or a high-fidelity model when connected to the open-source simulator called FlightGear. The low-fidelity model enables significant speed-up in the simulation speed, which is very useful for test and evaluation of collaborative algorithms and their implementations.

The MCS module depicted in Fig. 16 is a module specifically dedicated to running the MCS software described in Chapter 5. This module functions like a wrapper around the MCS software, which is loaded as a dynamic library, mimicking every external interface (e.g., serial and Ethernet communication) that is also used on the hardware platform. The wrapper provides the ability to test and debug the MCS software using the RAMS simulator and then transfer the tested software to the actual hardware platform with no modifications done to the base code. With the ability to also run a high-fidelity network and aircraft simulation, the RAMS simulator is very well suited for testing the researched algorithms in a more realistic environment, with changing communication dynamics and aircraft behavior.

Every module is self-contained and thus works independently of one another, eliminating the need of an event scheduler. Each module requests from the RAMS controller a period to sleep before waking itself up and running through another iteration of the module. This sleep- and run- time of the module together creates the frequency that the module is expected to run. To eliminate some of this deterministic behavior of every module running at the exact specified frequency, a Gaussian distributed sleep offset is induced into the sleep-time of each module. This drift in module frequency can create anomalies and change the behavior of the entire system, including the task allocation algorithms which is useful for testing.

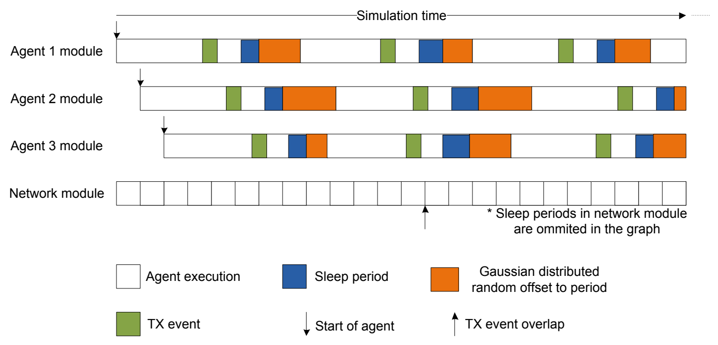


Fig. 17. Example of a timing diagram for different agent modules and the Gaussian distributed offset.

As mentioned above, without the high-fidelity simulation enabled for the network and agent module, the RAMS simulator is able to speed up simulation significantly. The following table shows the maximum speed-up possible with varying numbers of agents, where a speed-up of 1 is real-time. These results were obtained on a dedicated regular workstation with an Intel Core Duo clocked at 2.4 GHz, 4 MB of cache and 4 GB of RAM memory, with a Linux Ubuntu distribution with kernel version 3.2.0-25.

Table 4. Maximum speed-up possible of the RAMS simulator with varying numbers of agents.

#agents	speed-up
1	190
2	170
3	110
4	90
8	50
16	30

The RAMS simulator can be used as a framework in a wide array of research fields for examining strategic decision algorithms, robot teaming, and robot learning. The simulator can effectively simulate the effects that a real-world wireless network can have on the performance of decision algorithms and is the main simulator used to support this research.

CHAPTER 7

HARDWARE PLATFORM

The hardware platform for field testing the algorithms developed in this research, is done on a set of three Multiplex Easy Gliders Pro, with an approximate flight time of 20 minutes depending on wind and weather. The gliders were initially developed and built as part of a VCU master's thesis targeted towards collaborative UAS operations [44], but since have undergone further development, including the installation of a wireless modem supporting mesh-technology [45] and the MCS software, including APCBBA.

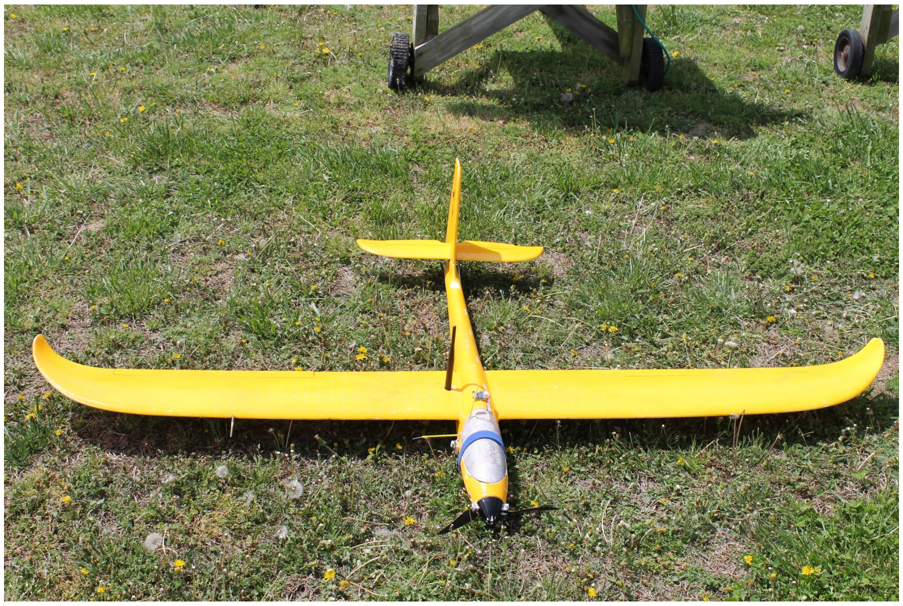


Fig. 18. One of the three gliders used during this research.

.

Each glider is able to be controlled autonomously through waypoints given by the GCS or MCS, or manually where a safety-pilot is in Radio Control of the glider. Having the ability to switch to manual control at any moment is vital to the safe utilization of any autonomous UAS used for research purposes.

The three main components carried on-board the gliders that enable collaborative operation include, a Flight Control System, a Mission Control System, and a wireless modem. An architectural overview of the major and minor components and their interconnections is given by Fig. 19.

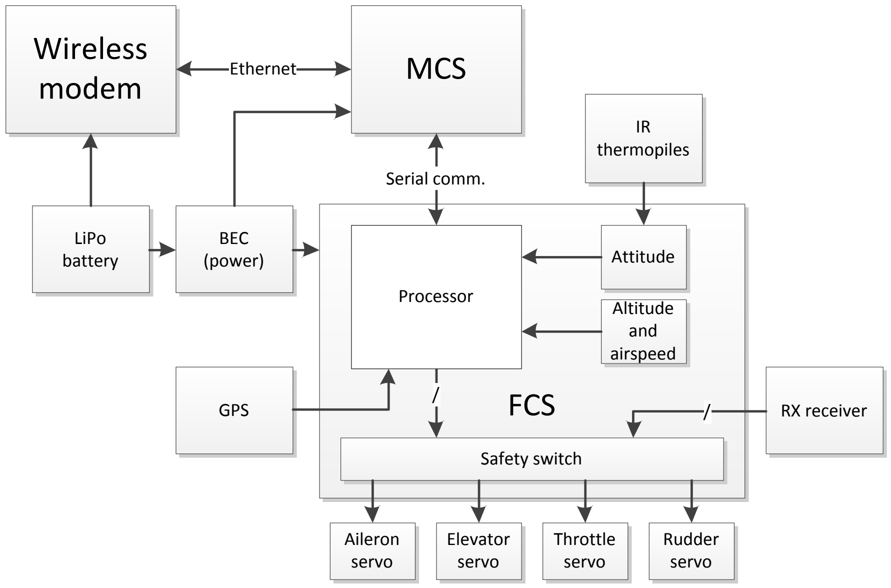


Fig. 19. System architectural overview of a UAS used in this research.

.

It is important to note that in this implementation, all of the computation necessary to implement APCBBA is executed in the MCS carried on-board each aircraft. The Mission Control System software and hardware were described in Chapter 5; the following sections give a brief overview of the other main components.

7.1 Flight Control System

The Flight Control System used in the hardware platform is designed and built by a graduate student in the Virginia Commonwealth University UAV lab as part of a master's thesis [46]. The FCS is designed around an Atmel AVR 32-bit microcontroller that runs the flight control and navigational algorithms. The design purpose of the FCS was to be low cost and small enough to fit in the glider. It uses an array of sensors to determine altitude, airspeed, and attitude of the UAS. Furthermore, an NMEA-enabled GPS can be connected to provide the navigational algorithms with up-to-date positional information. The FCS board also houses the safety-switch circuit, which guarantees that the control of the airplane can be switched from manual (safety-pilot) to autonomous and back, independent of the main processor.

Several different navigation and flight modes can be chosen to control the UAS, including a standard waypoint mode that will fly the UAS directly from waypoint to waypoint, loiter mode that will place the UAS in a stable orbit around a waypoint, and cross-track mode that will also have the UAS try to fly from waypoint to waypoint, but will have the UAS track the rhumb line between the two waypoints. The loiter mode is used during collaborative operation when the UAS is idle and no tasks are currently allocated or when the surveillance task needs surveillance for a certain amount of time. Normal waypoint mode is applied between tasks to travel from task to task. Lastly, during the search task, cross-track mode is enabled to follow the lawn-mower waypoint pattern created to cover the full search area.
Additionally, the FCS Printed Circuit Board (PCB) can also house a Digi Xbee wireless modem for cheap and easy communication to the GCS, but this is not used for collaborative operation. Instead an 5.2 GHz enabled wireless modem is used

to provide communication between UAS and GCS, which is explained in the next section.

7.2 Wireless communication

Essential to the quality of the full collaborative system including the task allocation algorithms, is the importance of having a reliable and robust wireless communication link. A wireless communication system was developed by a graduate student at the Virginia Commonwealth University UAV lab to provide the reliable link with plenty of bandwidth and the ability to use an ad-hoc mesh-topology [45]. This mesh-topology provides several advantages over the more classic infrastructure-type network where a central access point routes all the traffic, as is standard in a WIFI network. In a mesh network every modem in the network can relay data, meaning that each UAS can talk directly to one another without having the data relayed through an access-point. Furthermore, a transmission path can be created between UAS that could include another UAS when a direct transmission link is not available. Using a mesh-topology not only creates efficient data communication but also increases the operational communication range of the UAS fleet.

The developed communication system described above is based on the Bullet 5 modem, developed and sold by Ubiquiti Networks, Incorporated. The Bullet is sold off-the-shelf with firmware incapable of supporting mesh network technology and thus the original firmware was replaced with an embedded Linux distribution called OpenWrt. This distribution provides several extra functionalities not included in the standard firmware, including support for the 802.11s standard that describes mesh network technology. In addition to running a different firmware on the Bullet modems, a user-space application provides automatic detection and identification of the different agents/UAS in the network based on tail numbers. This application broadcasts a discovery packet twice a second to all UAS, signifying the presence of the UAS within the network. This discovery packet includes a tail number, a MAC address, and a time-stamp to synchronize the modems across the network. The modified Bullet software architecture is shown in Fig. 20.

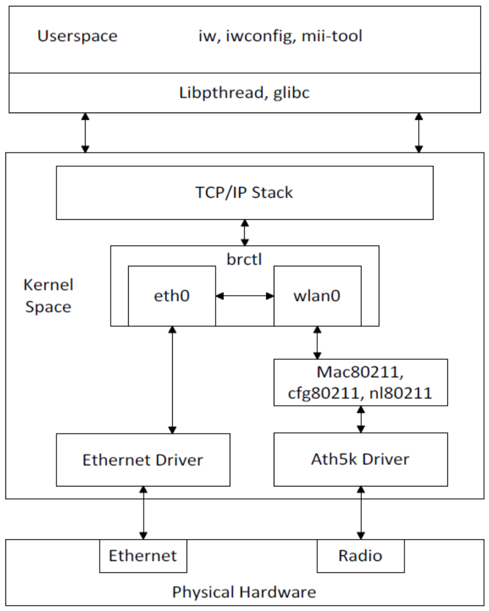


Fig. 20. Wireless modem software architecture.

.

CHAPTER 8

MIXED-INTEGER LINEAR PROGRAMMING

To provide a reference solution to the task allocation problem, a Mixed-Integer Linear Programming model was developed. This solution produces an optimum allocation of the tasks that subsequently can be compared to the solutions generated by APCBBA. The following section presents the Mixed-Integer Linear Programming model to the task allocation problem described in 2.

8.1 Model

The MILP model is formulated to schedule a task j after task i and determine the start time t_j^k for each task j and agent k so that the global score Eq. 8.1 for all tasks and agents is maximized. The following notation is used for the model:

Sets:

$T \rightarrow$ Set of all tasks $\{1, ...N^t\}$

$K \rightarrow$ Set of all agents $\{1, ...N^k\}$

Indices:

$i \rightarrow$ task $i \in T$

$j \rightarrow$ task $j \in T$

$k \rightarrow$ agent $k \in K$

Parameters:

$r_j \rightarrow$ Is the static reward for task j

$\varphi_j \rightarrow$ Is the release time for task j

$\rho_j \rightarrow$ Is the deadline for task j

$\tau_i \rightarrow$ Is the time it takes to execute task i

$\psi_{ij}^k \rightarrow$ Is the traveling time from task i to task j

$c_{ij} \rightarrow$ Is the cost of scheduling task j after task i

$M \rightarrow$ Big M ($>$ than max. scheduling time)

Variables:

$$x_{ij}^k = \begin{cases} 1, \text{ if task } j \text{ is scheduled after task } i. \\ 0, \text{ otherwise} \end{cases}$$

$t_j^k \rightarrow$ Is the scheduled time of task j for agent k

$t_i^k \rightarrow$ Is the scheduled time of task i for agent k

$\alpha_j \rightarrow$ linear time-discount factor

The objective function, Eq. 8.1, maximizes the global score for all agents $k \in K$

and tasks $j \in T$ by minimizing the time-discounted reward $\alpha_j^k(t_j^k - \varphi_j)$ and the cost for c_{ij}^k. The static reward r_j is determined by the user when the task is created.

$$\max \sum_i \sum_j \sum_k (r_j x_{ij}^k - \alpha_j(t_j^k - \varphi_j) - c_{ij}^k x_{ij}^k) \tag{8.1}$$

Subject to:

$$\sum_k \sum_i x_{ij}^k \leq 1 \qquad \forall j \tag{8.2}$$

$$\sum_k \sum_{j=N^k} x_{ij}^k \leq 1 \qquad \forall i \tag{8.3}$$

$$x_{ij}^k \leq \sum_{h \in T} x_{hi}^k \qquad \begin{matrix} \forall k, j, i \\ h \in T \end{matrix} \tag{8.4}$$

$$t_j^k \geq \varphi_j x_{ij}^k \qquad \forall k, j, i \tag{8.5}$$

$$t_j^k + \tau_j x_{ij}^k \leq \rho_j x_{ij}^k \qquad \forall k, j, i \tag{8.6}$$

$$t_j^k \geq t_i^k + \tau_i x_{ij}^k + \psi_{ij}^k x_{ij}^k - M(1 - x_{ij}^k) \qquad \forall k, j, i \tag{8.7}$$

The following section briefly explains the purpose of each constraint. The constraint given by Eq. 8.2 limits task j to be only assigned once for all tasks in T. Constraint 8.3, is similar but has an exception; hence the summation does not start at $j = 0$. This exception is caused because dummy tasks are added to very beginning of the set of tasks T for every agent. These tasks have a release and deadline time of 0 and are scheduled as the very first task for each agent by the solver. The dummy tasks are scheduled after themselves, implying that task i equals task j, and hence if the dummy task needs to be scheduled in front of a regular task, this dummy task must be scheduled twice as task i. Constraint 8.3 limits the regular tasks, non dummy tasks to be scheduled only once as task i before task j. Constraint 8.4 guarantees that task i must be scheduled before task j can be scheduled for agent k; i.e. task j for agent k should only be scheduled after task i, if and only if task i has already been scheduled for agent k. Tasks can only be scheduled after their respective release time φ_j, ensured by constraint 8.5, and before their deadline ρ_j minus the duration τ_j of task j, constraint 8.6. The final inequality constraint in Eq. 8.7 limits the solution space of task j to only being scheduled after the completion of task i plus the traveling time ψ_{ij}^k from task i to j.

8.2 Gurobi solver

The aforementioned model was implemented for use in Gurobi [47]. Gurobi is an optimization solver for mathematical programming and is able to solve a various set of problems, including Linear Programming, Quadratic Programming, and Mixed-Integer Linear Programming problems. It provides several Application Programming Interfaces (API) for different programming languages, including Matlab, C, C++, Java, Python, and more. The implementation of the model above was done in C++, in line with the programming language used for the MCS software.

The variables x_{ij}^k and t_j^k in the described model are considered to be decision variables and are of the binary and double type. Gurobi uses a strict matrix definition for its dimensions, where x_{ij}^k is a 3D matrix with rows being agent k, columns being task j, and depth being task i. Similarly, a 2D decision variable matrix was created for t_j^k, where rows are agent k and columns are task j. The parameter variables used in the model are normal matrices (arrays), with multiple dimensions when required, but still adhere to the strict dimensioning definition.

Gurobi, through its various techniques and solvers, will find values for the decision variables to maximize the objective function in Eq. 8.1, while producing a feasible solution adhering to the constraints of the model. The stopping conditions for the

Gurobi solver are; when the optimal solution has been found through observing the optimality gap and when a hard-coded running time limit of 10 minutes is reached.

8.3 Validation

The following results show how the MILP solution compares to an exhaustive search algorithm for up to four agents and up to eight tasks. This maximum number of tasks and agents is imposed by the exhaustive search algorithm, which can only reasonably compute the solution of a task allocation problem where the complexity is limited to four agents and eight tasks. The results were produced by running Monte-Carlo simulations with random placement of a set of tasks within 4 Km^2 and a random start, duration and deadline for the set.

Table 5. Average score difference in percentage between MILP and the exhaustive search solutions.

tasks	1 agent	2 agents	3 agents	4 agents
1	0.00	0.00	0.00	0.00
2	0.00	0.03	0.00	0.00
3	0.01	0.06	0.09	0.05
4	0.04	0.38	0.03	0.23
5	0.01	0.01	0.09	0.33
6	0.09	0.01	0.17	0.19
7	0.05	0.09	0.34	0.31
8	0.09	0.28	0.18	0.65

Table 5 gives the average score differences in percentages over a total of more than 900 tasks. In total 91.1% of MILP solutions are identical to solutions generated by the exhaustive search algorithm.

Differences between the solutions can be explained by the difference in determining the score. The cost for the MILP solution is calculated by the distance between tasks i and j, whereas in APCBBA the cost for task j is the total distance to travel to task j. The score function for the decentralized algorithm needs to adhere to the DMG property and thus cannot be based on the distance between task i and j. The other way around would be changing the MILP model where the cost would be based on the total distance to task j but this would render the model to be non-linear and more difficult to solve. Furthermore, where the discounted-reward for APCBBA is exponential towards the deadline of task j, the MILP model uses a linear time-discounted reward again.

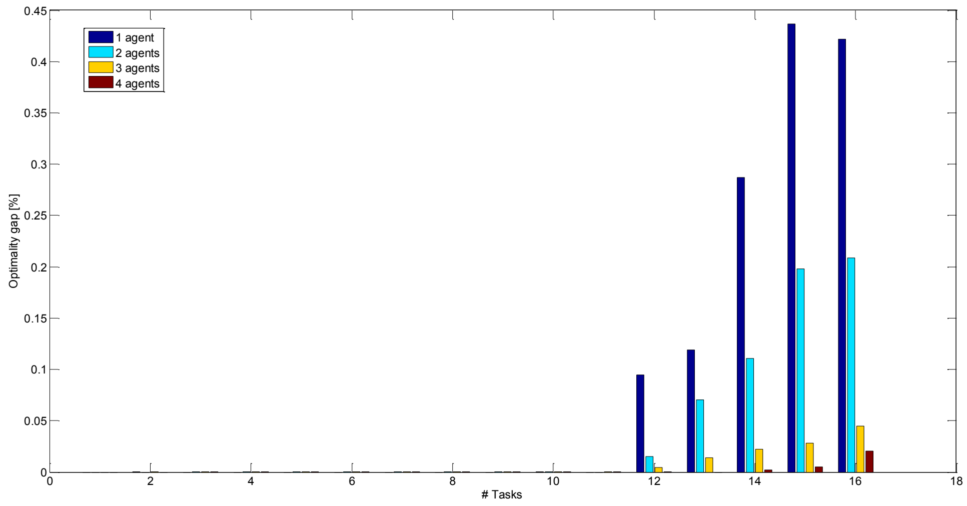


Fig. 21. Optimality gap between returned solution and the solution generated when root relaxation is applied.

The above figure shows the optimality gap between the calculated optimal solution and the best known bound. For the higher number of tasks the optimality gaps increases indicating that Gurobi might be producing sub-par solutions to the problem. This gap is partly due to the time-limit applied to finding the optimal solution, and it has been observed that when running the MILP solution algorithm on task sets with more than 12 tasks the solver might take hours to find the optimal solution. Furthermore, from Fig. 21 it can be observed that with the a higher number of agents the optimality gap reduces. A reasonable explanation would be that with a higher number of agents the quest for finding feasible solutions, where all tasks are allocated, increases, and thus better solutions are found quicker.

Appendix B lists the source code of the Gurobi MILP implementation, in which a small problem is solved with two agents and four tasks.

CHAPTER 9

RESULTS

The following results compromise data generated from simulation and real-world flight testing.

9.1 Simulation results

The following results are created with different simulators and implementations of the task allocation algorithms and are provided to the reader in steps to verify and validate each intermediate solution. The RAMS simulator, described in Chapter 6, is the main simulator providing the most realistic simulation of several UAS, with each running a decentralized task allocation algorithm. In addition, an implementation of CBBA is provided by the Aerospace Controls Lab of MIT [48] in Matlab, which does not simulate any form of communication, but has the full CBBA algorithm implemented. The aforementioned CBBA Matlab implementation was modified to implement the proposed APCBBA scoring function and allow direct comparison with the original discounted scoring function described in [6]. As previously described, an exhaustive search algorithm was implemented to provide a guaranteed optimal solution by 'brute force', iterating over all the possible solutions and determining their score. This algorithm is heavily limited by the number of tasks and agents before the computation of the solution becomes intractable; for reasonable run-times, the limit is set to 4 agents and 8 tasks. The total number of unconstrained solutions for the multi-agent and multi-task allocation problem can be computed by applying Eq. 9.1, where Fig. 22 is showing the total number of solutions for different numbers of

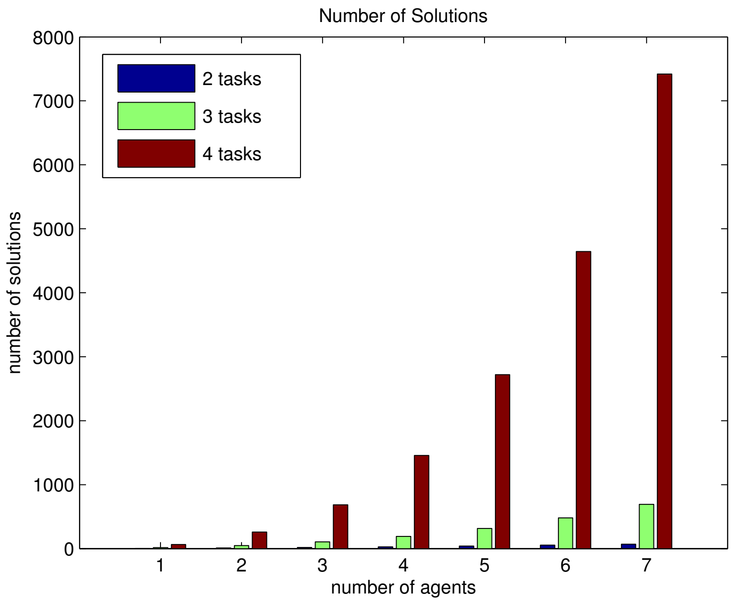


Fig. 22. Number of solutions per agent with differing number of tasks.

tasks and agents. The MILP solution previously described must be used to provide optimal, or near optimal, solutions for problems with a higher number of tasks and/or agents because of the limitations of the exhaustive search algorithm.

$$\sum_{j=1}^{N^t} \left(\frac{N^t!}{(N^t - j)!} \cdot \binom{j + N^k - 1}{j} \right) \tag{9.1}$$

All task allocation solution parameters, including the order of the tasks, the start time of each tasks, and to whom the tasks are assigned, are evaluated by the same standards, creating an equal scoring metric for all the different algorithms and simulations. The overall scoring metric for a set of tasks J is calculated using Eq. 4.1.

The simulation results are obtained by randomly placing the agents and tasks in

a 4 Km^2 square 3-D area with a minimum altitude of 30 meters and a maximum of 300 meters. The target airspeed of the agents for every task is set to be at 15 m/s and $\lambda = 0.1s^{-1}$. The static reward r_j, release time ς_j, duration τ_j, and deadline ρ_j for task j are all given random values in an appropriate range.
The optimality of the solutions produced by APCBBA are compared with solutions from running the same task allocation problem through the aforementioned CBBA and APCBBA Matlab implementations, the exhaustive search algorithm, and the Mixed-Integer Linear Programming (MILP) algorithm in the remainder of this chapter.

In the remainder of this chapter, the following naming convention will be used: an (M) placed behind the algorithm's name signifies the algorithm being run in Matlab, and an (R) placed behind the algorithm's name signifies that the algorithm is being run in the RAMS simulator.

In summary, the results show that the APCBBA algorithm produces better optimum results when compared to CBBA, allocating more tasks with a better overall score and efficiency. Furthermore, some high-fidelity simulations with the RAMS simulator show better, more robust, and conflict free allocations of tasks with APCBBA over CBBA.

9.1.1 Optimality

The total score traveled by the agents is compared for the Matlab implementations of APCBBA, CBBA, and the exhaustive algorithm, and the MILP algorithm in Gurobi. Figure 23 compares the performance of the 4 solutions where the number of agents N^k is 4 and with different values for the number of tasks N^t. Through its updated scoring function, APCBBA scores better than CBBA over the whole range of N^t, but does not outperform the MILP or the exhaustive search algorithm solutions for the same problem, as would be expected. Ordering and optimizing the local path based on the full distance for the agent to travel to each task in APCBBA reduces some of the greedy effects the CBAA scoring function applies to creating its local path.

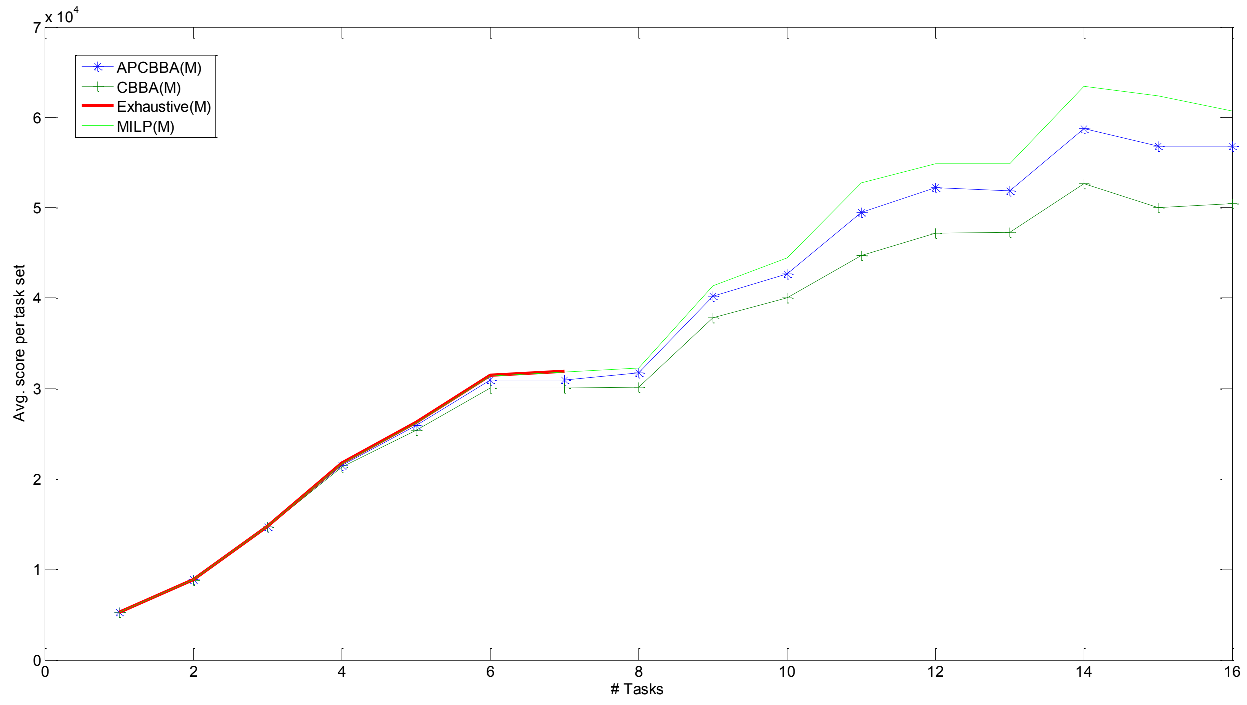


Fig. 23. Average total score per task set for 4 agents and 30 task sets for each N^t.

The average difference between the total score of APCBBA and MILP, and CBBA and MILP for the same set of tasks and agents as in Fig. 23 is given in Fig. 24.

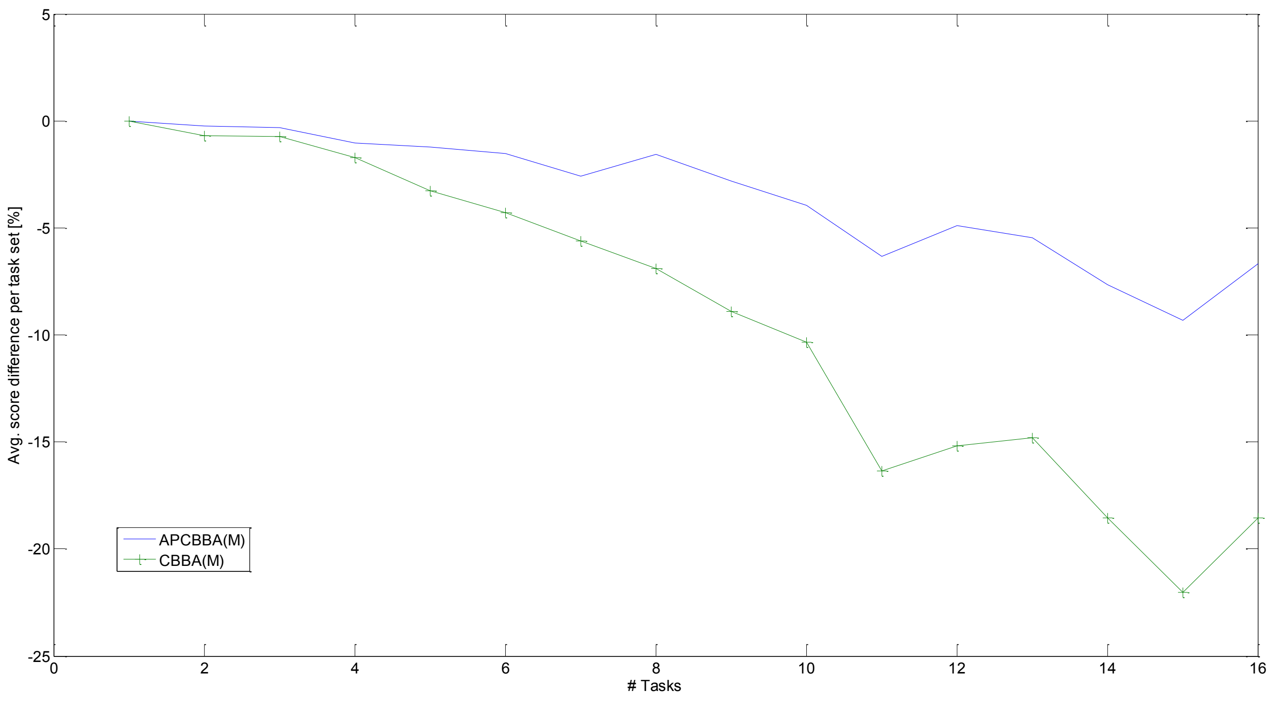


Fig. 24. Average percentage difference in score per task set when compared with the MILP solution.

The average percentage increase of APCBBA compared to CBBA over the full set of tasks for 4 agents is 5.4%. When comparing the full 1800 task sets and their solutions, with agents ranging from 1 to 4 and sets of tasks ranging from 1 to 16, the APCBBA algorithm produces solutions that are equal to the exhaustive and MILP solution 49.8% of the time and CBBA generates equal solutions 38.4% of the time. It seems that for higher number of tasks the optimality of the APCBBA and CBBA solutions diverge from the MILP solution, but APCBBA continues to generate better solutions than CBBA. The above results show how the APCBBA and CBBA Matlab implementations fare against the MILP and exhaustive search algorithms. The following results present the comparison of APCBBA, running on the RAMS simulator, and the solutions produced by the exhaustive search and MILP algorithm.

The optimality of the decentralized task allocation solution is also dependent on the outcome of the bundling algorithm running locally on each agent and explained in Appendix A. The bundle algorithm is by no means an exhaustive search of the total solution space, and thus, will not find the optimal path for the given task set T in all circumstances. The following data compares the APCBBA algorithm running on the RAMS simulator with the MILP and exhaustive search solution for a single agent. Since a single agent does not engage in the auction process, it will give a good estimate of the optimality of the solution produced by the local bundling algorithm. Figure 25 shows the total score for a single agent and over 400 task sets, ranging from 1 to 16 tasks. The APCBBA algorithm run on the RAMS simulator produces better results than the CBBA implementation in Matlab over the full range by an average of 6.4%.

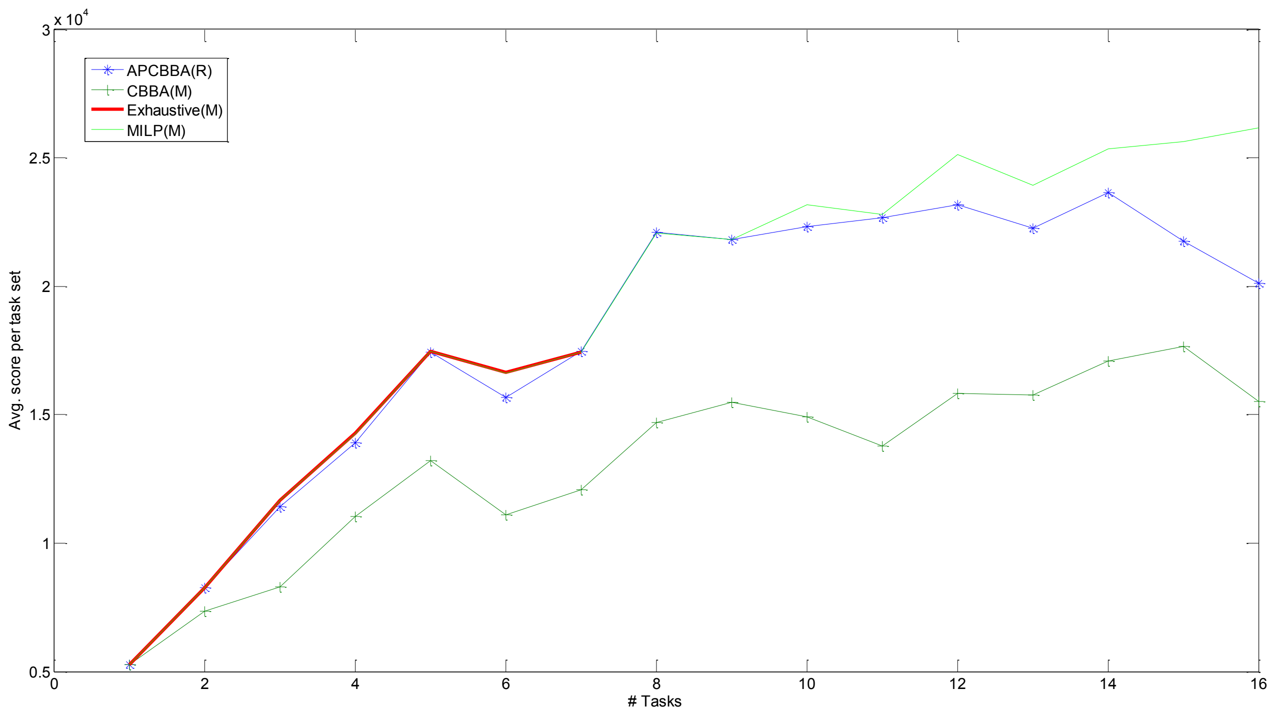


Fig. 25. Average score per task set for a single agent and 30 task sets for each N^t.

Just as in Fig. 24 the optimality of the solution for higher number of tasks seems to diverge from the MILP solution, indicating a reduced optimal solution for higher number of tasks. However, as in the results presented in 24, APCBBA continues to outperform CBBA. Furthermore, from Fig. 25 it can be concluded that the bundling algorithm used in CBBA has shortcomings for generating optimal path solutions. This is in part caused by the 'Greedy' nature of bundling algorithm used in CBBA.

Comparing the cost between the APCBBA and MILP solutions, gives insight into the order of the tasks and which tasks are being allocated. Over the full set of solutions with a single agent and a task set ranging from having 1 to 16 tasks, 44% of the solutions were identical between MILP and APCBBA running on the RAMS simulator, compared to 26% for CBBA.

The following data shows the results from the RAMS simulator when 4 agents are applied to the task allocation problem for up to 16 tasks.

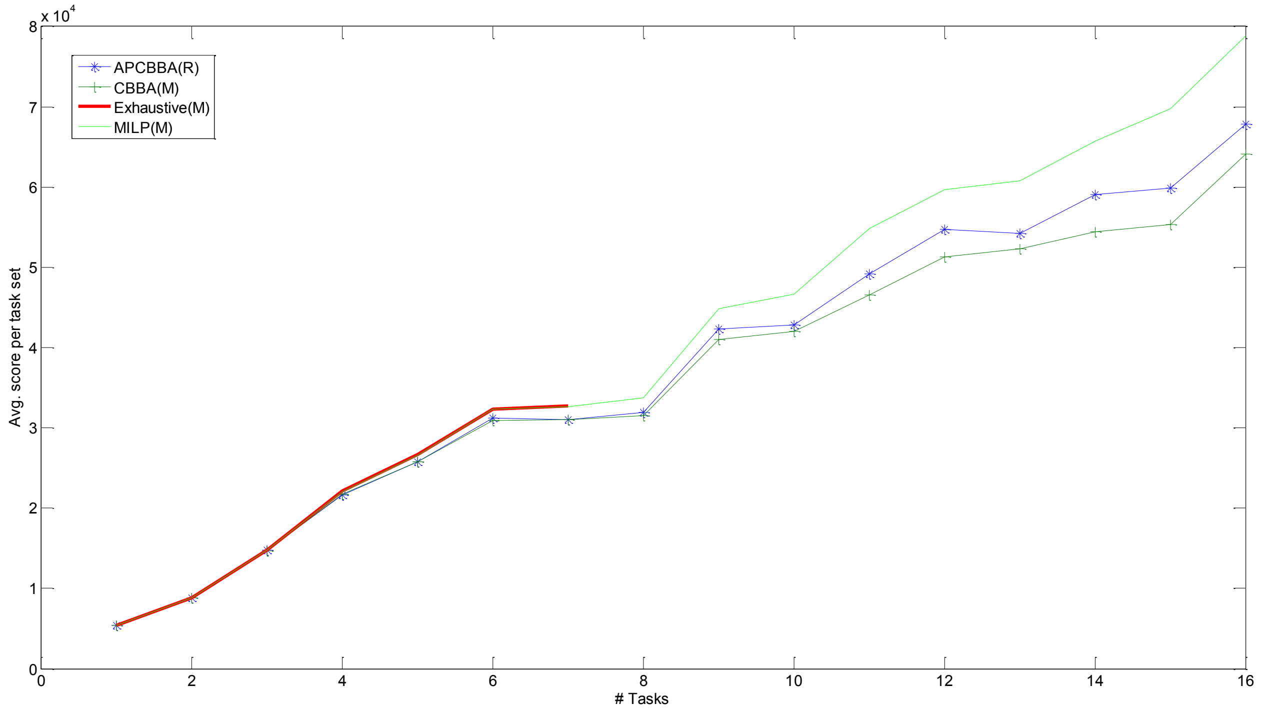


Fig. 26. Average score per task set for 4 agents and 30 task sets for each N^t.

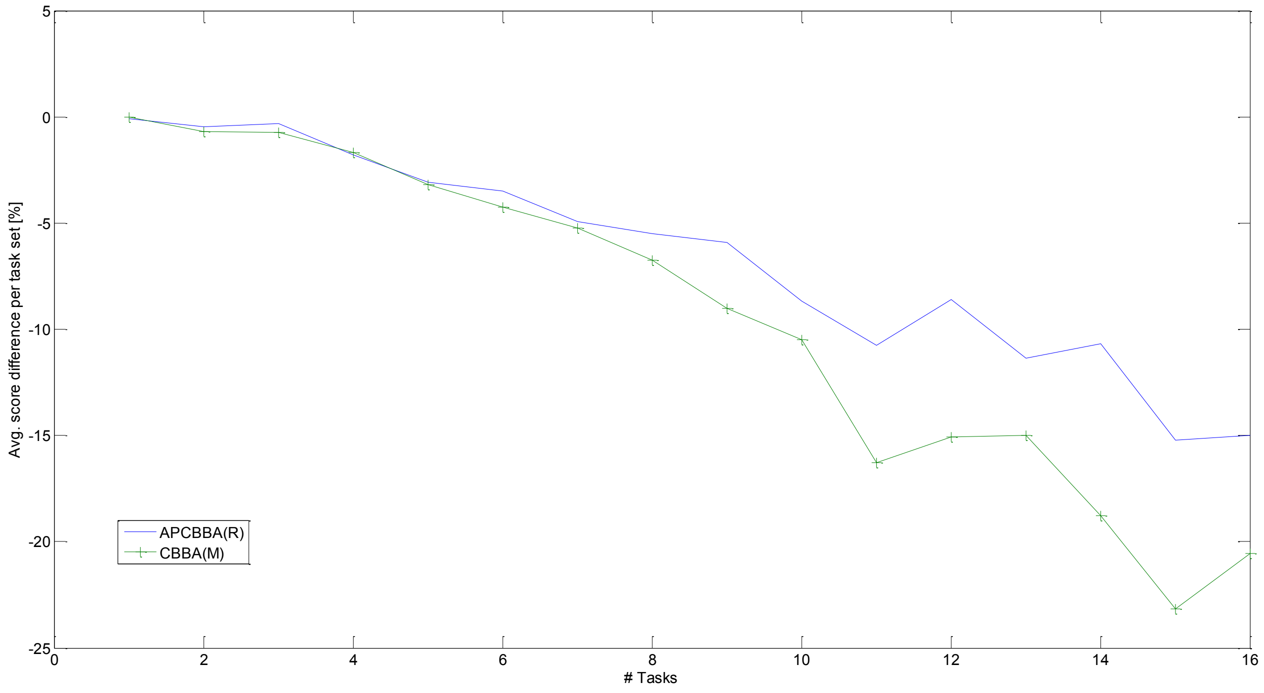


Fig. 27. Average score score difference per task set for 4 agents and 30 tasks set for each N^t.

Compared to Fig. 25, CBBA does produce better solutions for 4 agents compared to the CBBA results for a single agent assignment. When applying a decentralized auction algorithm the overall performance of the solution increases, while the gap between APCBBA and CBBA becomes smaller. Still APCBBA outperforms CBBA by an average of 2.8% over the full range of tasks set with 4 agents.

9.1.2 Response time

The time it takes to reach convergence on the set of tasks is of importance for reducing the response time to the tasks and is a Quality of Service metric defined in Chapter 2. APCBBA can determine consensus per task, reducing the time from the release of the task set T to execution of the first task j of the path P^k, and possibly all subsequent tasks. The consensus time per task is the time measured from having task j enter the task allocation algorithm to the assignment of task j to an agent. Figure 28 gives an average of the consensus time for APCBBA and CBBA for different task set sizes. The data presented in Fig 28 was produced using the RAMS simulator in combination with ns-3, creating more realistic network responses to the communication produced by the two algorithms APCBBA and CBBA.

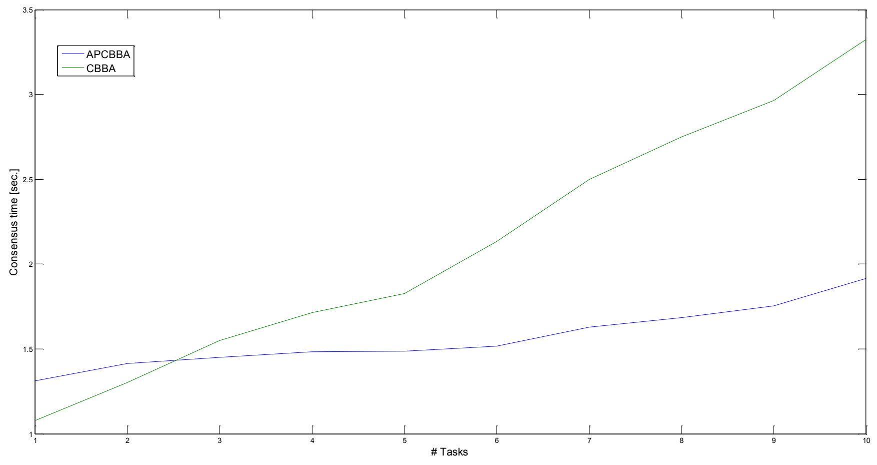


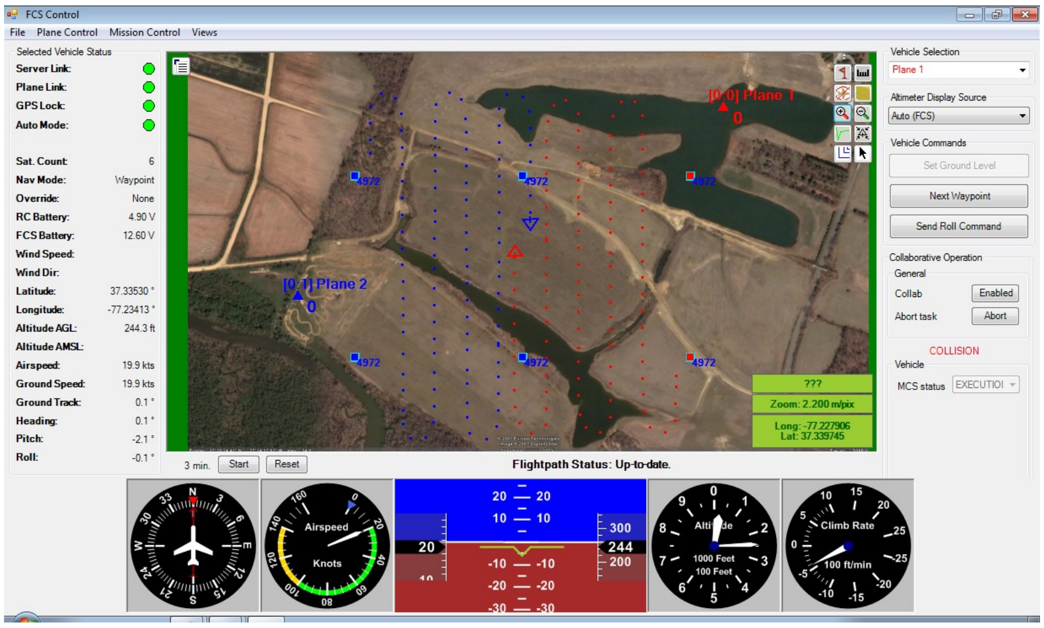
Fig. 28. Time to reach consensus among a set of N^t tasks and 4 agents.

CBBA determines consensus on a bundle of tasks and thus all the tasks in the bundle need conflict-free allocation before consensus is reached. The metric for determining consensus with CBBA is based on agents not actively sending out any updated winning bid lists to neighboring agents for a certain period of time. For the results in Fig. 28 this grace period was set to be 2 seconds, which is conservative according to the authors in [37]. The actual CBBA data presented in Fig. 28 does not include this 2 second grace period and gives the time between tasks entering the auction algorithm and the time radio silence is obtained and packets are no longer sent.

The results in Fig. 28 show that although CBBA is very quick for a small set of tasks, the consensus time increases rapidly with increasing size of the task set. The average consensus time per task for APCBBA does not increase as much as with CBBA. Thus APCBBA achieves much better response times over CBBA for task sets with a higher number of tasks N^t.

9.1.3 Team formation

Forming teams within APCBBA is a two-step process. This process will be illustrated using an example of performing area search as a base-task. The base-task, including the complete search area, is solved by each agent and a score is determined. The agent with the highest score for the base-task will be able to release the sub-tasks it has created within its solution. At this moment, the full auction process starts again on the sub-tasks, where all agents can bid for one or more sub-tasks. The solution from the base-task is based on the agent's knowledge of neighboring agents, including the neighboring agent's position, altitude, and capabilities.



Selected Vehicle Status

Server Link:	●
Plane Link:	●
GPS Lock:	●
Auto Mode:	●
Sat. Count:	6
Nav Mode:	Waypoint
Override:	None
RC Battery:	4.90 V
FCS Battery:	12.60 V
Wind Speed:	
Wind Dir:	
Latitude:	37.33530 °
Longitude:	-77.23413 °
Altitude AGL:	244.3 ft
Altitude AMSL:	
Airspeed:	19.9 kts
Ground Speed:	19.9 kts
Ground Track:	0.1 °
Heading:	0.1 °
Pitch:	-2.1 °
Roll:	-0.1 °

Vehicle Selection: Plane 1

Altimeter Display Source: Auto (FCS)

Collaborative Operation

General

Collab: Enabled

Abort task

COLLISION

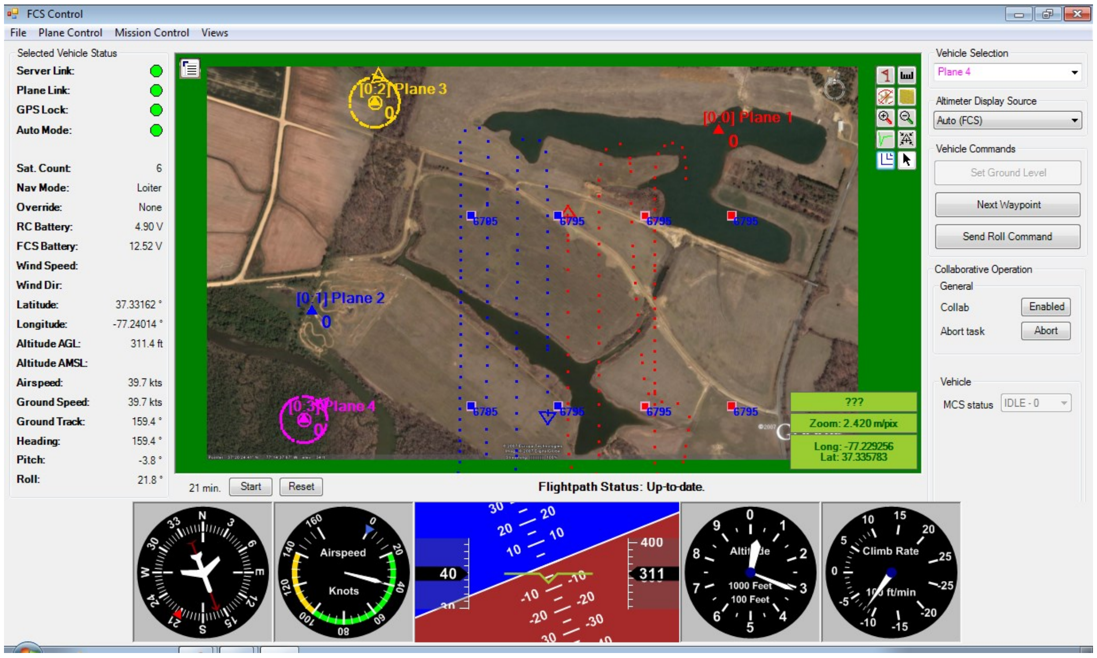
Vehicle

MCS status: EXECUTIOI

Fig. 29. Team formation of 2 agents given a tasks to search an area.

Figure 29 gives a snapshot of 2 agents searching an area. The whole search area was split into two sections, by the winning agent of the base task, where each agent was deemed the winner of one of the two sections (sub-tasks). Each agent, depending on its height, will determine the internal waypoints for effectively searching the dedicated area using a lawn-mower pattern. The waypoints extend the search area in order to give the agents sufficient space and area to turn around and get back onto the rhumb line. To reduce the number of sweeps of the lawn-mower pattern and increase efficiency, the agent will orient the pattern parallel to the longest boundary of the area, as can be seen in Fig. 29 where the height (vertical) of the individual search area is clearly greater than the width (horizontal).

Notice the head-on collision with the red and blue agents, and the 'COLLISION' message displayed to the right on the screen, detecting the imminent danger. The original search height was 300 feet,but due to the collision detection the red agent has descended to an altitude of 244 feet while decreasing its airspeed to 20 knots, and the blue has done the opposite, increasing its altitude and airspeed.



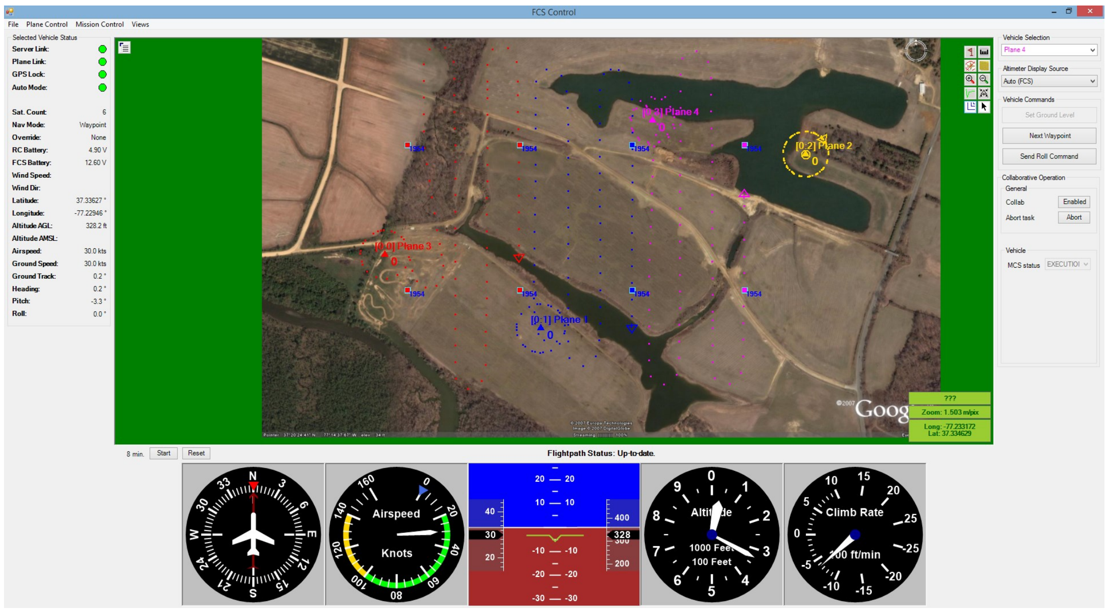
Selected Vehicle Status

Server Link:	●
Plane Link:	●
GPS Lock:	●
Auto Mode:	●
Sat. Count:	6
Nav Mode:	Loiter
Override:	None
RC Battery:	4.90 V
FCS Battery:	12.52 V
Wind Speed:	
Wind Dir:	
Latitude:	37.33162 °
Longitude:	-77.24014 °
Altitude AGL:	311.4 ft
Altitude AMSL:	
Airspeed:	39.7 kts
Ground Speed:	39.7 kts
Ground Track:	159.4 °
Heading:	159.4 °
Pitch:	-3.8 °
Roll:	21.8 °

Vehicle Selection: Plane 4
Altimeter Display Source: Auto (FCS)
Collab: Enabled
Abort task: Abort
Vehicle
MCS status: IDLE - 0
Zoom: 2.420 m/pix
Long: -77.229256
Lat: 37.335783
21 min.
Flightpath Status: Up-to-date.

Fig. 30. Two out of 4 agents forming a team to complete a search mission.

In Fig. 30, 4 agents were active and a search task was issued to take low-resolution pictures. The winning agent of the base-task has divided the search area into 3 smaller regions. Agents red, blue, and pink are able to perform the released search task, but yellow has capability constraints and is equipped with only an IR sensor. Through the task allocation algorithm, 2 of the 3 search areas were assigned to the red agent and the last area was assigned to blue. The pink agent, although able to perform the search mission, was not able to obtain a sub-task by outbidding red or blue for one of its assigned areas. The following figure shows all 3 capable agents being engaged in the search mission while the yellow agent is still incapable of executing the required task.



Selected Vehicle Status

Server Link:	●
Plane Link:	●
GPS Lock:	●
Auto Mode:	●
Sat. Count:	6
Nav Mode:	Waypoint
Override:	None
RC Battery:	4.90 V
FCS Battery:	12.60 V
Wind Speed:	
Wind Dir:	
Latitude:	37.33627 °
Longitude:	-77.22946 °
Altitude AGL:	328.2 ft
Altitude AMSL:	
Airspeed:	30.0 kts
Ground Speed:	30.0 kts
Ground Track:	0.2 °
Heading:	0.2 °
Pitch:	-3.3 °
Roll:	0.0 °

Vehicle Selection: Plane 4

Altimeter Display Source: Auto (FCS)

Collaborative Operation — General: Collab: Enabled; Abort task: Abort

Vehicle — MCS status: EXECUTIOI

Zoom: 1.503 m/pix

Long: -77.233172 Lat: 37.334629

8 min.

Flightpath Status: Up-to-date.

Fig. 31. Three out of 4 agents forming a team to complete a search mission.

The team formation algorithm demonstrates stable allocation of sub-tasks, forming teams on the fly to complete missions in need of agent cooperation. Although the implementation and capabilities of the algorithm and underlying support framework is still limited, the base algorithm for forming teams is promising and is suitable for supporting more advanced mission scenarios.

9.2 Real-world results

In the previous chapter, APCBBA was extensively simulated and the results were compared to other algorithms producing optimal solutions. This chapter shows some real-world results, flying the gliders discussed in Chapter 7. Having only 2 safety-pilots limits the number of gliders in the air simultaneously to two. None the less, the results show that APCBBA fares well in a real-world environment and creates conflict free assignments.

9.2.1 Task allocation

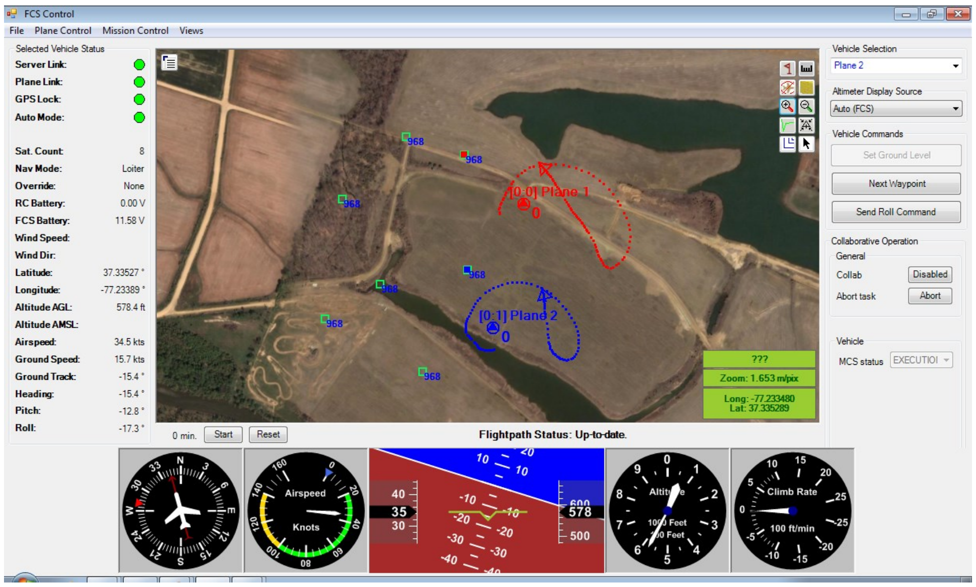
In total, four sets of surveillance tasks were issued and each set had a different number of tasks. Table 6 gives an overview of the assignment of the tasks per UAS in order of execution and the average time it took to reach consensus for that task set.

Table 6. Assignment of tasks to UAS (in order of execution) and the average consensus time

ID	# tasks	Algorithm	UAS 1	UAS2	Avg. consensus [sec.]
5637	9	APCBBA	1, 2, 3, 4	5, 6, 7, 8, 9	0.28
		APCBBA(M)	1, 2, 3, 4	5, 6, 7, 8, 9	
		MILP(M)	1, 2, 3, 4, 9	5, 6, 7, 8	
4780	10	APCBBA	1, 10, 7, 6, 9, 8	5, 2, 3, 4	0.52
		APCBBA(M)	1, 10, 7, 6, 9, 8	5, 2, 3, 4	
		MILP(M)	1, 2, 3, 4, 6	5, 7, 10, 8, 9	
7349	7	APCBBA	1, 2, 5	6, 7, 3, 4	1.7
		APCBBA(M)	1, 2, 4	5, 6, 7, 3, 5	
		MILP(M)	2, 3, 7, 4	5, 6, 1	
968	8	APCBBA	7, 1, 2	8, 6, 5, 3, 4	0.58
		APCBBA(M)	7, 1, 2, 4	8, 6, 5, 3	
		MILP(M)	7, 1, 2	8, 6, 5, 3, 4	

The consensus time for task set 7349 is extended because of a communication timeout occurring where both UAS could not communicate for a short period. At the time of the timeout the UAS had not allocated all tasks without conflict. In the case of running CBBA, this would have been detected as reaching consensus and the UAS would start executing the task, but with existence of conflicts in the allocation. APCBBA, handles the communication error by waiting for communication to be re-established and resolving the conflicts. The mere ability of detecting communication problems makes the APCBBA algorithm more robust in real-world environments where communication errors are likely. Figure 32 shows task set 968 being executed by UAS 1 (plane 1) and UAS 2 (plane 2) after allocation of the task set has been completed. The same set of tasks is also run through the APCBBA and MILP implementation in Matlab. Three out of 4 tasks sets are identical for the 2 APCBBA algorithms and 2 out of 4 tasks sets yield similar results as the MILP solution.

Testing of the APCBBA algorithm in real world conditions is far from complete and needs many more scenarios to generate data to analyze the algorithm's response in all conditions and circumstances. Further test flying is needed to analyze and determine the optimality and robustness of APCBBA in real-world conditions.



Selected Vehicle Status

Server Link:	🟢
Plane Link:	🟢
GPS Lock:	🟢
Auto Mode:	🟢
Sat. Count:	8
Nav Mode:	Loiter
Override:	None
RC Battery:	0.00 V
FCS Battery:	11.58 V
Wind Speed:	
Wind Dir:	
Latitude:	37.33527 °
Longitude:	-77.23389 °
Altitude AGL:	578.4 ft
Altitude AMSL:	
Airspeed:	34.5 kts
Ground Speed:	15.7 kts
Ground Track:	-15.4 °
Heading:	-15.4 °
Pitch:	-12.8 °
Roll:	-17.3 °

Vehicle Selection: Plane 2
Altimeter Display Source: Auto (FCS)
Collaborative Operation — General: Collab: Disabled
MCS status: EXECUTIOI
Zoom: 1.653 m/pix
Long: -77.233480 Lat: 37.335289
0 min.
Flightpath Status: Up-to-date.

Fig. 32. Two UAVS on a collaborative surveilance mission

9.2.2 Collision detection

During test flying of the gliders, it is the safety pilot's responsibility to track the UAS and prevent any damage to aircraft, personnel and equipment, whenever possible. None the less, the gliders are equipped with a collision detection system described in 5.5. During test flying, several evasive maneuvers were engaged by the UAS to eliminate the possibility of a collision. One of these near collisions is shown in Fig. 33 where two UAS are on a collision course. Each data point in the figure is a GPS coordinate and the order of GPS coordinates is shown by the blue and red arrows.

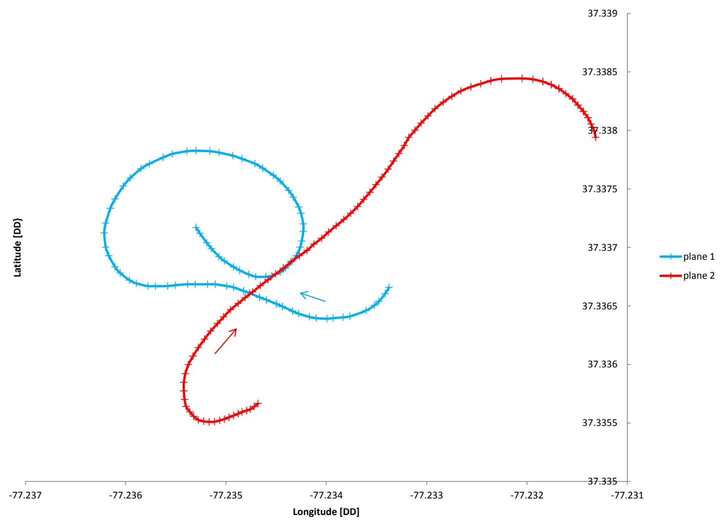


Fig. 33. GPS position coordinates for two UAS on collision course

The accompanying evasive maneuver for the near collision is shown in 34, where the altitude change is indicated by the solid red and blue lines. The moment of detecting a possible collision is shown in the same figure by the red and blue dashed lines. Once the UAS have separated enough in distance, the collision warning is turned off and the UAS will return to its normal altitude.

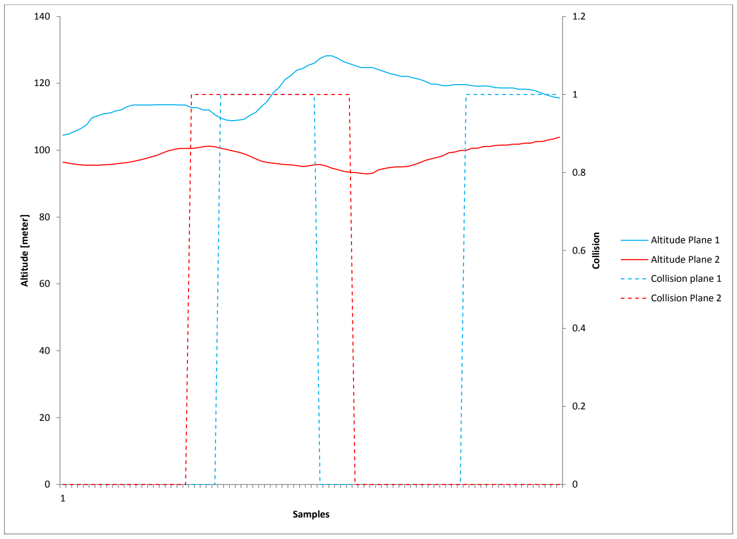


Fig. 34. Evasive maneuver for same UAS, as in Figure. 33, changing altitude to avoid collision.

CHAPTER 10

CONCLUSION

In this dissertation, the APCBBA algorithm has been introduced as an extension to the CBBA [6] and ACBBA [37] algorithms. The APCBBA algorithm has shown to improve the original algorithm by improving the task allocation solution by an average of at least 3.1% over CBBA, improving the robustness of the algorithm in dynamic real-world environments with communication failures or limited communication capabilities, and introducing dynamic team forming. The APCBBA algorithm can furthermore deterministically detect consensus within the fleet of agents by requesting the winning bid list from neighboring agents and determine consensus per task. The latter reduces the response time for the first and possible subsequent tasks from the time that the task was released. Although limited data is available, the real-world flight testing has shown the APCBBA algorithm to be robust and produce conflict free task allocation solutions.

An MILP model to the task allocation problem was developed and implemented to provide a basis of comparison for the results achieved by the APCBBA and CBBA algorithms. The MILP model gives an identical solution to an exhaustive search algorithm 91% of the time. For the other 9% the solution was within 0.8% of the exhaustive search algorithm. A major advantage of the MILP solution is the ability to handle bigger task sets much better than the exhaustive search algorithm developed for this research, but even the MILP solver will take several hours, if not preempted, for tasks sets containing 14 or more tasks.

10.1 Future work

Although the APCBBA has shown real promise for future use in collaborative UAS applications, more and extensive testing is necessary with dynamic conditions and environments. Although the APCBBA algorithm has been stress tested through real-world flight testing, it has not been tested in enough situations where conditions, and in particular the wireless network, have been put to the test. The RAMS simulator, although capable of simulating a high-fidelity network in combination with ns-3, does not have the ability to run scenarios where a certain percentage of the wireless transmissions will fail. This will be necessary for further testing and better determination of the robustness and optimality of APCBBA.
Currently, the GCS is an integral part of the system to release collaborative tasks in simulation and during real-world flight testing. The storage of the released tasks and the received solutions to the task allocation problem on the GCS is done using an xml format and that is highly inefficient. In the futurei, task allocation algorithm simulations, should not be dependent on the GCS to release the tasks. An additional collaborative controller within the MCS module of the RAMS simulator should take over the responsibilities of releasing tasks, collecting allocation information, and storing this for further processing and analysis in Matlab. The whole collaborative simulation framework should be independent of the GCS and should be fully automated. Additional statistics can be tracked within this collaborative controller to determine the time between release of a task set and the assignment or execution of the set.
Although the MILP model was created for solving a multi-task multi-agent task allocation problem, it supports single-agent problems as well. The MILP model would make a great candidate for replacing the bundle algorithm explained in A but does need to adhere to the DMG property. Making the MILP model compatible would

require the cost function to be dependent on the previous task in the path, and would result in the model becoming non-linear. Further research will involve how to handle this non-linearity and make the MILP model suitable for execution within the existing MCS framework.

REFERENCES

[1] John Bellingham et al. "Multi-Task Allocation And Path Planning For Cooperating Uavs". In: *Proceedings of Conference of Cooperative Control and Optimization.* Citeseer, 2001.

[2] Christos G. Cassandras and Wei Li. "A receding horizon approach for solving some cooperative control problems". In: *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on.* Vol. 4. IEEE, 2002, pp. 3760–3765.

[3] Jonathan How, Ellis King, and Yoshiaki Kuwata. "Flight demonstrations of cooperative control for UAV teams". In: *AIAA 3rd" Unmanned Unlimited" Technical Conference, Workshop and Exhibit.* Vol. 2012. 2004, p. 9.

[4] Kendall E. Nygard, Phillip R. Chandler, and Meir Pachter. "Dynamic network flow optimization models for air vehicle resource allocation". In: *American Control Conference, 2001. Proceedings of the 2001.* Vol. 3. IEEE, 2001, pp. 1853–1858.

[5] Brett Bethke, Jonathan P. How, and John Vian. "Group health management of UAV teams with applications to persistent surveillance". In: *American Control Conference, 2008.* IEEE, 2008, pp. 3145–3150.

[6] Han-Lim Choi, Luc Brunet, and Jonathan P. How. "Consensus-based decentralized auctions for robust task allocation". In: *Robotics, IEEE Transactions on* 25.4 (2009), pp. 912–926.

[7] Timothy W. McLain and Randal W. Beard. "Coordination variables, coordination functions, and cooperative timing missions". In: *Journal of Guidance, Control, and Dynamics* 28.1 (2005), pp. 150–161.

[8] Tal Shima, Steven J. Rasmussen, and Phillip Chandler. “UAV team decision and control using efficient collaborative estimation”. In: *American Control Conference, 2005. Proceedings of the 2005*. IEEE, 2005, pp. 4107–4112.

[9] Randal W. Beard et al. “Decentralized cooperative aerial surveillance using fixed-wing miniature UAVs”. In: *Proceedings of the IEEE* 94.7 (2006), pp. 1306–1324.

[10] Mehdi Alighanbari and Jonathan P. How. “Decentralized task assignment for unmanned aerial vehicles”. In: *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC’05. 44th IEEE Conference on*. IEEE, 2005, pp. 5668–5673.

[11] Wei Ren and Yongcan Cao. “Simulation and experimental study of consensus algorithms for multiple mobile robots with information feedback”. In: *Intelligent Automation and Soft Computing* 14.1 (2008), pp. 73–87.

[12] HW Kuhn. “The Hungarian method for the assignment problem”. In: *Naval Research Logistics (NRL)* 52.1 (2005), pp. 7–21.

[13] Stefano Giordani, Marin Lujak, and Francesco Martinelli. “A distributed algorithm for the multi-robot task allocation problem”. In: Trends in Applied Intelligent Systems. Springer, 2010, pp. 721–730.

[14] Usman A. Khan and Soummya Kar. “A decentralized algorithm for the preferred assignment problem in multi-agent systems”. In: *2013 European Control Conference (ECC)*. Zurich, Switzerland, 2013, p. 766.

[15] Douglas M. Hart and Patricia A. Craig-Hart. “Reducing swarming theory to practice for UAV control”. In: *Aerospace Conference, 2004. Proceedings. 2004 IEEE*. Vol. 5. IEEE, 2004, pp. 3050–3063.

[16] Allison Ryan et al. “Decentralized control of unmanned aerial vehicle collaborative sensing missions”. In: *American Control Conference, 2007. ACC’07.* IEEE, 2007, pp. 4672–4677.

[17] Brian P. Gerkey and Maja J. Mataric. “Sold!: Auction methods for multi-robot coordination”. In: *Robotics and Automation, IEEE Transactions on* 18.5 (2002), pp. 758–768.

[18] William E. Walsh and Michael P. Wellman. “A market protocol for decentralized task allocation”. In: *Multi Agent Systems, 1998. Proceedings. International Conference on.* IEEE, 1998, pp. 325–332.

[19] Luca F. Bertuccelli et al. “Real-time multi-UAV task assignment in dynamic and uncertain environments”. In: *presentado al AIAA Guidance, Navigation, and Control Conference, Chicago, Illinois.* 2009.

[20] David C. Parkes and Lyle H. Ungar. “Iterative combinatorial auctions: Theory and practice”. In: *Proceedings of the national conference on artificial intelligence.* Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000, pp. 74–81.

[21] PB Sujit and Randy Beard. “Distributed sequential auctions for multiple UAV task allocation”. In: *American Control Conference, 2007. ACC’07.* IEEE, 2007, pp. 3955–3960.

[22] Amir Ajorlou et al. “Market-Based Coordination of UAVs for Time-Constrained Remote Data Collection and Relay”. In: ().

[23] Charles E. Pippin and Henrik Christensen. “A Bayesian formulation for auction-based task allocation in heterogeneous multi-agent teams”. In: *SPIE Defense,*

Security, and Sensing. International Society for Optics and Photonics, 2011, pp. 804710–804710–11.

[24] M. Bernadine Dias and Anthony Stentz. “A comparative study between centralized, market-based, and behavioral multirobot coordination approaches”. In: *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. Vol. 3. IEEE, 2003, pp. 2279–2284.

[25] Matthijs TJ Spaan, Nelson Gonalves, and Joao Sequeira. “Multirobot coordination by auctioning POMDPs”. In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1446–1451.

[26] Kai Zhang, Emmanuel G. Collins Jr, and Dongqing Shi. “Centralized and distributed task allocation in multi-robot teams via a stochastic clustering auction”. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 7.2 (2012), p. 21.

[27] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. “Competitive analysis of repeated greedy auction algorithm for online multi-robot task assignment”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4792–4799.

[28] Brian P. Gerkey and Maja J. Mataric. “A formal analysis and taxonomy of task allocation in multi-robot systems”. In: *The International Journal of Robotics Research* 23.9 (2004), pp. 939–954.

[29] Maitreyi Nanjanath and Maria Gini. “Repeated auctions for robust task execution by a robot team”. In: *Robotics and Autonomous Systems* 58.7 (2010), pp. 900–909.

[30] Min-Hyuk Kim, Hyeoncheol Baik, and Seokcheon Lee. “Response Threshold Model Based UAV Search Planning and Task Allocation”. In: *Journal of Intelligent and Robotic Systems* (2013), pp. 1–16.

[31] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. “Distributed algorithm design for multi-robot task assignment with deadlines for tasks”. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on.* IEEE, 2013, pp. 3007–3013.

[32] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. “Multi-robot assignment algorithm for tasks with set precedence constraints”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011, pp. 2526–2533.

[33] Wei Ren, Randal W. Beard, and Timothy W. McLain. “Coordination variables and consensus building in multiple vehicle systems”. In: Cooperative Control. Springer, 2005, pp. 171–188.

[34] Travis Mercker et al. “An extension of consensus-based auction algorithms for decentralized, time-constrained task assignment”. In: *American Control Conference (ACC), 2010.* IEEE, 2010, pp. 6324–6329.

[35] Sameera Ponda et al. “Decentralized planning for complex missions with dynamic communication constraints”. In: *American Control Conference (ACC), 2010.* IEEE, 2010, pp. 3998–4003.

[36] Luke B. Johnson et al. “Improving the efficiency of a decentralized tasking algorithm for UAV teams with asynchronous communications”. In: *AIAA Guidance, Navigation, and Control Conference (GNC).* 2010.

[37] Luke B. Johnson et al. “Asynchronous Decentralized Task Allocation for Dynamic Environments”. In: *Proceedings of the AIAA Infotech@ Aerospace Conference, St. Louis, MO.* 2011.

[38] Matthew Argyle, David W. Casbeer, and Randy Beard. “A Multi-Team Extension of the Consensus-Based Bundle Algorithm”. In: *American Control Conference (ACC), 2011.* IEEE, 2011, pp. 5376–5381.

[39] GP Das et al. “A fast distributed auction and consensus process using parallel task allocation and execution”. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on.* IEEE, 2011, pp. 4716–4721.

[40] Giulio Binetti, David Naso, and Biagio Turchiano. “Decentralized task allocation for surveillance systems with critical tasks”. In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1653–1664.

[41] Gumstix, Inc. 2014. URL: `https://www.gumstix.com`.

[42] Tim Bakker et al. “RAMS: A Fast, Low-Fidelity, Multiple Agent Discrete-Event Simulator”. In: *SCS Summersim 2013.* Toronto, Canada, 2013.

[43] Tim Bakker, Siva T. Patibandla, and Robert H. Klenke. “A Framework for Integration of ns-3 with RAMS simulator”. In: *American Institude for Aeronautics and Astronautics, SciTech 2014, National Harbor, MD.* 2014.

[44] Lloyd B. Mize. “Development of a Multiple Vehicle Collaborative Unmanned Aerial System”. MA thesis. Virginia Commonwealth University, School of Engineering, 2011.

[45] Siva Teja Patibandla. “Development of Mobile Ad-Hoc Network for Collaborative Unmanned Aerial Vehicles”. MA thesis. Virginia Commonwealth University, School of Engineering, 2013.

[46] Jose Ortiz. "Development of a Low Cost Autopilot System for Unmanned Aerial Vehicles". MA thesis. Virginia Commonwealth University, School of Engineering, 2010.

[47] Gurobi Optimization, Inc. *Gurobi Optimizer Reference Manual*. 2014. URL: `http://www.gurobi.com`.

[48] MIT - ACL. *CBBA with Time-Windows (Matlab)*. URL: `http://acl.mit.edu/projects/cbba.html#Variants`.

Appendix A

MCS BUNDLE ALGORITHM

A.1 Algorithm

The following algorithm creates an optimized local path (it does not in all instances create the best path) for agent k given a set of tasks T.

Algorithm 3 Bundle algorithm for N^t tasks

```
1:  $MaxScore \leftarrow 0$
2:  for all tasks in $T$ do
3:      for all task $j$ in $T$ not assigned do
4:          for all Indexes $m$ in $P^k$ do
5:              Insert task $j$ in $P^k$ at index $m$
6:              for all tasks $i$ in $P^k$ do
7:                  Calculate score $S_i^k$ for task $i$
8:                  $TotalScore \leftarrow TotalScore + S_j^k$
9:              end for
10:             if $TotalScore > MaxScore$ then
11:                 Store task $j$ and index $m$
12:                 $MaxScore \leftarrow TotalScore$
13:             end if
14:         end for
15:     end for
16:     Add task $j$ with highest score to bundle $B^k$ and insert into path $P^k$ at position
    $m$
17: end for
```

A.2 Complexity

The bundle algorithm has a complexity of $\mathrm{O}\big((N^t)^4\big)$, which theoretical proof is given below:

The algorithm starts with 4 for loops, the first loop runs for exactly N^t times and with every iteration a task is added to the bundle and path. The second loop runs for the number of tasks not assigned yet in N^t which is getting smaller every run through the outer loop. The third loops run for for every task in path P^k after inserting the new task, and includes the last loop recalculating for every task in P^k the new score after task j has been inserted. The final loop has a complexity of BigO1. The total number of iterations depending on N^t is then:

$$= \sum_{n=1}^{N^t} (N^t - (n-1)) * n^2$$

$$= \sum_{n=1}^{-} n^3 + n^2(N^t + 1)$$

$$= \sum_{n=1}^{-} n^3 + N^t n^2 + n^2$$

Using summation identities:

$$= \frac{N^{t2}(N^t+1)^2}{4} + \frac{N^{t2}(N^t+1)(2N^t+1)}{6} + \frac{N^t(N^t+1)(2N^t+1)}{6}$$

Some simplication:

$$= \frac{N^{t4} + 4N^{t3} + 5N^{t2} + 2N^t}{12}$$

Only considering the higher order polynomial gives us $\mathrm{O}\big(N^{t4}\big)$.

Appendix B

MILP SOURCE CODE

Below is source code for a simple example of two agents and four tasks. The first thirty lines of code are part of setting up the task problem in matrix form, from then on the code is generating the MILP model using the Gurobi C++ API.

```
#include <stdlib.h>
#include <stdio.h>
#include <sstream>
#include "gurobi_c++.h"

#define K 2        //!< Number of agents
#define T 4
#define I T + K        //!< Number of tasks
#define J I
#define SIM_TIME 1000    //!< Maximum simulation time [s]
#define M SIM_TIME
#define ALPHA 0.9    //!< Time discount

/**
General note: x-axis are agents, y-axis are tasks j, z-axis are i
*/
```

```
int main(int argc, char *argv[]){

    GRBVar ***dvX;       // Task assignment decision variable
    GRBVar **dvT;        // Task scheduling time

    //! \var Reward parameter for doing task j (since start of simulation
    double cR[J] = { 0, 0, 5000, 6000, 7000, 2000};

    //! \var Release time of task j [sec.] (absolute time from beginning
    double cS[J] = { 0, 0, 4, 22, 5, 10};

    //! \var Deadline time of task j [sec.] (absolute time from beginning
    double cD[J] = { 0, 0, 10, 28, 12, 19};

    //! \var Duration parameter for doing task j
    double cTau[J] = { 0, 0, 2, 3, 2, 5};

    //! \var Cost parameter for doing task j after task i
    double cC[J][I] = { {0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 2400, 1200, 1400},
                        {0, 0, 1000, 0, 2300, 1500},
                        {0, 0, 1200, 2500, 0, 3000},
                        {0, 0, 1400, 2600, 1000, 0} };
```

```
//! \var Traveling time from task i to task j [sec]
double cPsi[J][I] = { {0, 0, 0, 0, 0, 0},
                      {0, 0, 0, 0, 0, 0},
                      {0, 0, 0, 4, 2, 2},
                      {0, 0, 2, 0, 4, 3},
                      {0, 0, 2, 5, 0, 6},
                      {0, 0, 2, 4, 2, 0} };

//! \var Variables k = agents, j = tasks after i, i = tasks before j
int k, j, i;

try{

    GRBEnv env = GRBEnv("gurobiLog.txt");

    GRBModel model = GRBModel(env);
    model.set(GRB_StringAttr_ModelName, "Task assignment");

    /*** Creating 2D array for scheduling times of tasks (rows are ag
    dvT = new GRBVar*[K];
    for(k = 0; k < K; k++){

        dvT[k] = model.addVars(J);
        model.update();
        for(int j = 0; j < J; j++){
            // Set variable to be semi-continuous
            dvT[k][j].set(GRB_CharAttr_VType, 'S');

            // UB is the deadline, since task cannot be scheduled pas
            dvT[k][j].set(GRB_DoubleAttr_UB, cD[j] - cTau[j]);

            // LB is the release time, since task cannot be scheduled
            dvT[k][j].set(GRB_DoubleAttr_LB, cS[j]);

            // Set variable name
            std::ostringstream vname;
            vname << "t[" << k << "][" << j << "]";
            dvT[k][j].set(GRB_StringAttr_VarName, vname.str());
        }
    }
    model.update();

    /*** Creating 3D array for binary decision variable X (rows are a
    dvX = (GRBVar ***)new GRBVar*[K*J];

    for(k = 0; k < K; k++){
        GRBVar **temp = new GRBVar*[J];

        for(j = 0; j < J; j++){
            temp[j] = model.addVars(I);
            model.update();
        }
        dvX[k] = temp;
    }
    model.update();

    // Set specification for dvX
    for(k = 0; k < K; k++){
        for(j = 0; j < J; j++){
            for(i = 0; i < I; i++){
                // Set variable to be binary
                dvX[k][j][i].set(GRB_CharAttr_VType, 'B');

                // Set upper boud to be 1 (not sure if necessary)
                dvX[k][j][i].set(GRB_DoubleAttr_UB, 1);

                // Set variable name
                std::ostringstream vname;
                vname << "x[" << k << "][" << j << "][" << i << "]";
                dvX[k][j][i].set(GRB_StringAttr_VarName, vname.str())
            }
        }
    }
```

```
// Create linear objective expression
GRBLinExpr expr = 0;
for(k = 0; k < K; k++){
    for(j = 0; j < J; j++){
        for(i = 0; i < I; i++){
            expr += cR[j] * dvX[k][j][i] - (ALPHA * (dvT[k][j]-cS
        }
    }
}

// Set Objective function for model
model.setObjective(expr, GRB_MAXIMIZE);
model.update();

// Add bound constraints to X, constraint 1
for(i = 0; i < I; i++){
    GRBLinExpr expr = 0;
    for(k = 0; k < K; k++){
        for(j = K; j < J; j++){
            expr += dvX[k][j][i];
        }
    }
    // Set constraint name
    std::ostringstream vname;
    vname << "c2[" << k << "][" << i << "]";
```

```
        model.addConstr(expr <= 1.0, vname.str());
    }

    // Add bound constraints to X, contraint 2
    for(j = 0; j < J; j++){
        GRBLinExpr expr = 0;
        for(k = 0; k < K; k++){
            for(i = 0; i < I; i++){
                expr += dvX[k][j][i];
            }
        }
        // Set constraint name
        std::ostringstream vname;
        vname << "c3[" << j << "]";
        model.addConstr(expr <= 1.0, vname.str());
    }

    // Add constraint 3
    for(k = 0; k < K; k++){
        for(j = 0; j < J; j++){
            for(i = 0; i < I; i++){
                GRBLinExpr expr = dvX[k][j][i];
                for(int h = 0; h < I; h++)
                    expr += -dvX[k][i][h];
                // Set constraint name
                std::ostringstream vname;
                vname << "c4[" << k << "][" << j << "][" << i << "]";
                model.addConstr(expr <= 0, vname.str());
            }
        }
    }

    // Add bound constraints to T, contraint 4
    for(k = 0; k < K; k++){
        for(j = 0; j < J; j++){
            for(i = 0; i < I; i++){
                GRBLinExpr expr = dvT[k][j] - dvT[k][i] - dvX[k][j][i
                std::ostringstream vname;
                vname << "c5[" << k << "]" << "[" << j << "]" << "["
                model.addConstr(expr >= -M, vname.str());
            }
        }
    }

    // Add constraint 5, scheduled time should be past the release ti
    for(k = 0; k < K; k++){
        for(j = 0; j < J; j++){
            for(i = 0; i < I; i++){
                GRBLinExpr expr = dvT[k][j] - cS[j] * dvX[k][j][i];
                std::ostringstream vname;
                vname << "c6[" << k << "]" << "[" << j << "]" << "["
                model.addConstr(expr >= 0, vname.str());

            }
        }
    }

    // Add constraint 6, time scheduled should be before the deadline
    for(k = 0; k < K; k++){
        for(j = 0; j < J; j++){
            for(i = 0; i < I; i++){
                GRBLinExpr expr = dvT[k][j] - M + (dvX[k][j][i] * (cT
                std::ostringstream vname;
                vname << "c7[" << k << "]" << "[" << j << "]" << "["
                model.addConstr(expr <= 0, vname.str());
            }
        }
    }

    // Make sure dummy tasks are scheduled
    for(k = 0; k < K; k++){
        GRBLinExpr expr = dvX[k][k][k];
        std::ostringstream vname;
        vname << "c8[" << k << "]" << "[" << j << "]" << "[" << i <<
```

```
            model.addConstr(expr == 1, vname.str());
        }

        // Find solution
        model.optimize();
        // Write problem model
        model.write("main.lp");
        // Write solution
        model.write("main.sol");

        // Clean up
        for(k = 0; k < K; k++){
            delete [] dvT[k];
            for(j = 0; j < J; j++)
                delete[] dvX[k][j];
            delete[] dvX[k];
        }

        delete[] dvX;
        delete[] dvT;

    }catch(GRBException e){
        std::cout << "Exception during optimization!!" << std::endl;
    }
}
```