ABSTRACT

BIEBER, CHAD MICHAEL. Central Command Architecture for High Order Autonomous Unmanned Systems. (Under the direction of Larry M. Silverberg.)

This dissertation describes a High-Order Central Command (HOCC) architecture and presents a flight demonstration where a single user coordinates 4 unmanned fixed-wing aircraft. HOCC decouples the user from control of individual vehicles, eliminating human limits on the size of the system, and uses a non-iterative sequence of algorithms that permit easy estimation of how computational complexity scales. The Hungarian algorithm used to solve a min-sum assignment with a one-task planning horizon becomes the limiting complexity, scaling at $O(x^3)$ where $x$ is the larger number of vehicles or tasks in the assignment. This method is shown to have a unique property of creating non-intersecting routes which is used to drastically reduce the computational cost of deconflicting planned routes. Results from several demonstration flights are presented where a single user commands a system of 4 fixed-wing aircraft. The results confirm that autonomous flight of a large number of UAVs is a bona fide engineering sub-discipline, which is expected to be of interest to engineers who will find its utility in the aviation industry and in other emerging markets.
Central Command Architecture for High Order Autonomous Unmanned Systems

by
Chad Michael Bieber

A dissertation submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

Aerospace Engineering

Raleigh, North Carolina 2014

APPROVED BY:

Gregory Buckner

Thom Hodgson

Andre Mazzoleni

Larry M. Silverberg
Chair of Advisory Committee
DEDICATION

For my sister, Annie.
BIOGRAPHY

The author is always a little nervous at the questions, “Where are you from?” Born in San Francisco, CA, he made his first move at 7 days old. He claims Colorado as home, but has lived in 9 states, including east and west coasts and the two largest states, and has put tires or feet in another 40. At the tender age of 7 his mom unwittingly took him to an air show, after which he insisted he be taken to an Air Force Recruiter to find out how to become a pilot. After successfully implementing a 15-year Master Plan, he became an instructor pilot in T-1 and C-5 aircraft for the Air Force. Realizing that the future involved planes that didn't need humans to fly them, he changed careers and decided to leverage that instructor time to figure out how to teach airplanes to fly themselves. He's probably on the list of people time travelers will come back in time to kill.
ACKNOWLEDGEMENTS

Many people helped to push me down the path that brought me here, and I could never thank them all. First, chronologically and in importance, I would like to thank my parents. They let me explore, and fail, and figure out how to do things on my own. They supported my interests, drove me to practices and meetings, taught me to drive myself to them, and then opened their arms to let me go, and still had them open when I came back. They tried hard to teach me that the answer, and the facts, are not always the most important part of the problem, although it took more than two people to convince me of that. To my advisor, Larry Silverberg: thank you for the ideas, the conversations, and the freedom to run with things on my own. Colin, I owe you many thanks, and much blame, for coffee consumed and time wasted discussing the important things in life, none of which had anything to do with either of our research projects, and for helping me retrieve wayward airplanes. Sally, you reminded me I was interested in learning and that I could still do the math. Fred, you are a beast. A huge, lumbering, wonderful beast who has rocked me to sleep and populated my dreams and nightmares. I will not miss you, but I will, occasionally, remember you fondly. Thank you for showing me how well, and how poorly, pieces come together to make a whole, and always reminding me that it takes some magic to get a pile of metal airborne. Dr. Buckner, your class, the first in my graduate career, convinced me I was on the right path. Dr. Ramasubramanian, you showed me how to be Dr. Frankenstein to a little robot monster. I don't know if that was your intent, but I will never forget the elation I felt the first time my creation jumped up and moved around on its own. Lastly, my wonderful wife inadvertently showed me that a PhD was something a person could do, and then helped me to embark on this path and supported me all the way.
# TABLE OF CONTENTS

LIST OF FIGURES ........................................................................ vi

CHAPTER 1 Introduction ................................................................. 1

CHAPTER 2 Method ................................................................. 5
  2.1 Architecture ........................................................................ 5
  2.1.1 Staging .......................................................................... 7
  2.1.2 Routing .......................................................................... 7
  2.1.3 Vehicle Assignment .......................................................... 8
  2.1.4 Deconfliction ................................................................. 9
  2.1.5 Autonomous Flight Controls ........................................... 11
  2.2 Algorithm Complexity ......................................................... 12
  2.3 Hardware .......................................................................... 13

CHAPTER 3 Results ............................................................ 16
  3.1 Multiple Point Tasks ............................................................ 17
  3.2 Area Task ........................................................................ 18
  3.3 Dynamic Tasking ............................................................... 18
  3.4 Obstacle .......................................................................... 20

CHAPTER 4 Conclusion .......................................................... 22

BIBLIOGRAPHY ........................................................................... 24

APPENDIX ............................................................................... 26
  Appendix A Non-Intersecting Theorem ........................................ 27
LIST OF FIGURES

Figure 2.1  System Architecture diagram [3] ................................................. 6
Figure 2.2  Visibility graph with shortest path shown. ................................. 8
Figure 2.3  Shared Vertex Example ............................................................... 10
Figure 2.4  Task under shared route ............................................................... 10
Figure 2.5  Demonstration system at Lake Wheeler Agriculture Fields ... 14

Figure 3.1  Aerial view of flight test area at Lake Wheeler Road Field Laboratory. [25] .... 16
Figure 3.2  Multi-task initial assignment ......................................................... 17
Figure 3.3  Assignment following task completion .......................................... 17
Figure 3.4  Assignment to swaths ................................................................. 18
Figure 3.5  Completing swaths .............................................................. 18
Figure 3.6  Dynamic assignments Begin ......................................................... 19
Figure 3.7  Adding swaths ................................................................. 19
Figure 3.8  Beginning to complete swaths ......................................................... 19
Figure 3.9  Adding more swaths ................................................................. 19
Figure 3.10  Vehicle 2 assigned to a swath ...................................................... 20
Figure 3.11  Vehicles 1 and 4 assigned to the new swaths ................................ 20
Figure 3.12  Routing around an obstacle ......................................................... 21

Figure A.1  Non-intersecting connecting line segments are shown in black ................. 28
Figure A.2  Non-Intersecting assignment example ........................................... 29
Unmanned Aerial Vehicle (UAV) system development is booming in anticipation of new market opportunities. Many systems will employ multiple UAVs simultaneously, particularly those that involve covering large areas (geospatial imagery), require areal coverage over an extended period of time (persistent surveillance), or need fault- or damage-tolerant systems (communications networks). The human operator is the limit on number of user-controlled vehicles in multi-vehicle systems. This human limit depends on where in the control loop the operator is located. Cummings et al. surveyed multiple studies and showed the limit ranges from from 1 vehicle at the flight-control level to as many as 12 at the mission control level [1]. A human as the limiting factor is not unique to controlling UAVs. In 1956, Urwick described a manager’s span of control to be about 4 to 8 other people, depending on the level of interaction required with and amongst the group [2]. No multi-vehicle system of UAVs will be able to functionally exceed this limit without removing the human as manager of individual vehicles. One new method that decouples the human from system management is based on a High-Order Central Command (HOCC) architecture developed for multi-vehicle systems [3]. This dissertation presents a flight demonstration of a multi-vehicle system using the HOCC architecture.

The requirements of an architecture under which high-order Autonomous Unmanned Aerial Systems (AUAS) become feasible is the subject of ongoing research. Bellingham et al. [4] separated
their architecture into a) routing, b) vehicle assignment of tasks, and c) deconfliction, after which the vehicles receive their directions. Cummings et al. [1] separated their architecture into a) central mission and payload management, b) navigation, c) vehicle autopilots, and d) flight controls. Shima and Rassmussen, [5] describe a pantheon of architectures across the three axes of Command, Information Availability and Performance. Concerning user load, Adams et al. [6] studied the problems that arise with human attention when one or more users give commands and monitor data. Donmez et al. [7] showed that wait time increases with the number of vehicles in AUAS that do not have full autonomous planning. These studies suggest that full autonomy is a necessary requirement in high-order central command. About computational requirements, routing and task assignment are combinatorial problems that have prohibitive computational requirements for high-order AUAS, necessitating heuristics. Gardiner et al. [8] reviewed real-time collision avoidance algorithms for multiple UAVs and illustrated the computational complexities that limit the number of vehicles and flight time. How et al. [9] demonstrated a simultaneous task assignment and deconfliction approach with 8 vehicles. These studies suggest the use of heuristics and non-iterative algorithms to reduce computational complexity. Pertaining to the question of the division of labor between remote and on-board processing, it is natural to perform the flight controls on-board to reduce communication throughput requirements to remote transmission of off-board command decisions at perhaps 1 Hz or less. Edwards and Silverberg, 2010 [10] exploited this division of labor in an autonomous soaring flight demonstration. Concerning the choice between acquiring on-board states from synthesized relative measurements (using situational awareness) or from inertial measurements (like IMU and GPS), Levedahl and Silverberg, 2005 [11] showed that acquiring on-board states from inertial measurements is particularly well-suited to the central command problem because of the sensitivity issues that arise in synthesized relative measurements in high-order AUAS.

Two fundamental requirements of HOCC are vehicles capable of both autonomous navigation and task execution as well as a Ground Control Station (GCS) that monitors vehicle states, takes inputs from the user, and assigns tasks as needed to complete the input goals. Many research endeavors have focused on decentralization, using processing power on the vehicles themselves to plan paths, bid for tasks, and otherwise choose how to accomplish the desired goals. While small groups of fully autonomous vehicles cooperatively solving problems are certainly useful, we can look to human organizations, comprised of many autonomous agents (people), to see that large, complex systems usually have some kind of management that provides goals and general guidance for the organization (centralized control) and people that autonomously respond to those goals in a dynamic environment (decentralized execution). Failing to let subordinates operate with sufficient autonomy is referred to derisively as micro-managing. In this dissertation, the central GCS interfaces with the user and sends task assignments with basic routing to the individual vehicle, and then
the vehicles execute the commands. In effect, this is the classic Air Force doctrine of Centralized Command – Decentralized Execution [12].

A critical part of operating multiple vehicles is deconfliction, a method of creating collision-free paths. Deconfliction operates in the future, planning paths that will happen, which separates it from collision avoidance, or the act of sensing and avoiding threats in the present. Collision avoidance at the vehicle level involves sensing an environment and altering the planned path to avoid sensed obstacles. This will be an integral part of future UAS operations, but is insufficient for route planning of multiple vehicles. Current aircraft operations use both complementary methods - collision avoidance at the aircraft level (See/Sense-And-Avoid or Traffic Collision Advisory Systems (TCAS) equipment) and routes assigned by Air Traffic Control that are created to be deconflicted. This dissertation demonstrates a unique method of deconfliction resulting from the chosen algorithms, rather than imposed on the routing after the fact. In particular, a minimum-sum assignment method results in routes that are non-intersecting. More than simple deconfliction, non-intersecting routes remain conflict free even if the vehicle speed is changed.

At its core, the problem involves multi-vehicle path-planning and assignment, both combinatorial problems, as well as deconfliction, which is usually solved by checking the solution, adjusting for conflicts, and iterating over previous steps until a suitable result is obtained. Simply enumerating the possible combinations of assignments of vehicles to tasks requires counting $n!$ possibilities. Individual parts of this problem have well-established heuristics; however, combining the parts into a system capable of real-time response to a dynamic environment continues to be a challenge. Mixed Integer Linear Programming (MILP) can be used to solve the complete problem, but the computational costs are prohibitive. In 2011, a team in Auburn used MILP to plan a 3 vehicle flight with fixed obstacles. The 160 second flight took 800 seconds of computation [8]. Even with continued advances in processing speed, scaling up a complete real-time MILP solution for many vehicles will continue to be unfeasible. Receding Horizon reduces how far ahead in time an algorithm plans, improving computational speed of many techniques, MILP included [8, 9]. While a shortened planning horizon means some optimal solutions may be missed, in a dynamic environment changes occur before completion of the current plan. Well chosen horizons can offer an excellent trade off of optimality for efficiency. Changing where decisions are being made also impacts speed and optimality. Decentralized control puts decision making authority on the vehicles. Using a cooperative technique, like a market-based approach, vehicles work together to create a joint solution [13]. Placing decision-making on board an aircraft means that available processing power scales with the number of vehicles, but requires that vehicles maintain situational awareness and limits the value of their inputs to their area of awareness. In 2011, Kuwata and How simulated 5 vehicles using a Receding Horizon MILP method with computation times ranging from .2 to 2 seconds.
for decentralized planning and .5 to 3 seconds for centralized planning [14]. While certainly an improvement over solving the complete problem with MILP, it suggests that even decentralized RH-MILP would have a hard time solving high-order systems of vehicles in a dynamic environment.

The HOCC system architecture used here decomposes the complete problem in a manner similar to Bellingham et al. [4], then uses well-established algorithms at each step in a non-iterative sequence such that no result at any step requires the re-accomplishment of a previous step. This sequential implementation allows the computational complexity of the complete set of algorithms to be easily estimated and permits the system to scale computationally in understandable ways. Decoupling the human user from individual vehicles removes the limitation caused by human span-of-control leaving the upper bound of number of vehicles to be determined by algorithm and hardware.
2.1 Architecture

The HOCC architecture is described by the block diagram shown in Fig. 2.1. Drawing from the work of Bellingham et al. [4], the HOCC architecture is divided into central command, autonomous planning, and autonomous flight controls. The central command block directs the execution of objectives. It contains a staging process that converts the high-level objective into assignable tasks. The autonomous planning block is an efficient, non-iterative sequence of three processes called Routing (R), Vehicle assignment (V), and Deconfliction (D). A unique feature of the architecture introduced in this paper, as shown below, is that while the processes have internal iterations, they never require going back to a previous process (external iterations). The complexity of the individual processes used, namely the Visibility graph method[15], the A* method[16], and the Hungarian method[17], are well known. The elimination of external iterations, as described below, leads to a robust, computationally efficient algorithm. Throughout this paper, we only consider vehicles that are assignable to every task (homogeneity).
Figure 2.1 System Architecture diagram [3].
2.1.1 Staging

As shown in Fig. 2.1, a user is provided information from an operational area and other field parameters, and data \( C_j \) \((j = 1, 2, \ldots)\) generated by AUAV payload \( P_j \) \((j = 1, 2, \ldots)\). Using this and other information, the user commands an objective. In absence of fully autonomous planning algorithms that remove the vehicle from the calculus, user load would be, at the very least, proportional to the number of vehicles, which is prohibitive in high-order systems. Thus a critical feature of the HOCC architecture introduced in this paper is the decoupling of the objective from the vehicles and, instead, the expression of the commands in terms of a relatively small number of parameters. Operator commanded dynamic re-tasking of individual vehicles no longer applies. The staging process converts an objective into a set of tasks. This paper considers spatial staging, such as passing over a point (for imagery or delivery), a line scan (following a line segment) which could be part of covering a wider region, and loitering (orbiting a point) which could arise when a vehicle is waiting for further instructions or to monitor a point of interest. The spatial tasks have starting points, so the conversion of an objective into a set of tasks determines a set of spatial points that vehicles need to reach. During the staging process the candidate routes are unknown and the assignment of vehicles to tasks is unknown.

2.1.2 Routing

Autonomous planning is the second block of the HOCC architecture and routing is its first process. Support tasks, such as launching or recovering and refueling, are conducted by the administrator (A) who operates in the background. The routing process itself is performed in two parts, mapping the environment and choosing a path [18]. The first part produces a graph of a vehicle-task route segment. The Visibility graph method used here produces a graph of visible paths between vertices in the environment, that is, paths that are not block by an obstacle, and between the vertices and the task and vehicle being investigated. Note that search areas can be regarded as obstacles to prevent routes from passing through them.

Fortunately, well developed algorithms exist that can graph the environment, search the graph for shortest routes, and solve the assignment problem in small fractions of a second for many more than 15 vehicles. Methods of graphing the environment include Voronoi Diagrams and Visibility graphs, among others [18]. The Visibility graph method, initially described by Lozano-Perez and Wesley [15], is used here from a code library created by VisiLibity [19]. This method first identifies vertices of polygons that define the edge of the environment and any internal obstacles, and then connects each vertex with every other vertex that is visible. In other words, the straight line between them does not cross an obstacle or the edge of the environment. For each vehicle-task combination,
2.1. ARCHITECTURE

2.1. METHOD

Vehicle Task

Figure 2.2 Visibility graph with shortest path shown.

the vehicle and location are added as two additional points and connected to visible vertices. The shortest path is contained within the graph, as well as all other possible paths connecting vertices with straight lines. The resulting graph might look like figure 2.2. Once the environment is graphed, the shortest paths through the environment from each vehicle to each task must be found. Several graph search algorithms exist with different strong points. Many descend from Dijkstra's method [20], including A* [16], the method used in this paper. Very generally, A* searches towards the destination, and uses the straight-line distance to the destination from where it is to inform its next choice. Once the environment is graphed and routes found, the min-sum assignment can be solved.

2.1.3 Vehicle Assignment

A frequent goal in AUAS path planning, because of limited fuel and flight duration requirements, is to determine vehicle routes by minimizing distance of travel of individual vehicles. An important variation on this goal, which is introduced in this paper, is the minimization of total distance of travel of all of the vehicles (a min-sum assignment). The minimization of total distance of travel leads to an importing routing principle: The routes determined by minimizing total distance of travel do not intersect in the open domain of the problem. The mathematical proof of this routing principle is presented in the appendix. Although non-intersection is guaranteed in the open domain, it is not guaranteed on the boundary of the domain. Indeed, vehicle-task segments can share vertices of obstacles, as shown below.

The crux of the problem is task assignment. In this problem there are \( n \) vehicles and \( m \) tasks. The goal is to assign only one vehicle to only one task. The cost of assigning vehicle \( i \) to task \( j \) is arranged in a matrix \( C_{ij} \) and \( x_{ij} \) is a matrix of zeroes and ones recording which vehicle \( i \) is assigned
to which task $j$. Then, the total cost is:

$$
\sum_{i=1}^{n} C_{ij} x_{ij}
$$

subject to:

$$
\sum_{i=1}^{n} x_{ij} = 1 \quad (2.1)
$$

$$
\sum_{j=1}^{m} x_{ij} = 1
$$

$$
x_{ij} = 0 \text{ or } 1
$$

where the constraints ensure that one and only one task is assigned to one and only one vehicle.

Finding the minimal sum of all the costs is non-trivial. In fact, simple combinatorial methods run out of memory rapidly. 15 vehicles and tasks have $15! \approx 1.3 \times 10^{12}$ possible combinations, which would take several terabytes just to store. The Hungarian algorithm uses matrix manipulation to solve the assignment problem [17]. Using a MATLAB program developed by Markus Buehren and available on MATLAB File Exchange [21], 20x20 matrices and larger can be solved in fractions of a second. The version of the Hungarian algorithm used can readily handle non-square assignments where the number of vehicles and tasks are not equal. A more detailed explanation of the Hungarian method is available in [3, 17].

2.1.4 Deconfliction

After the environment has been graphed, the graph searched, and the min-sum assignment found, potential collisions between vehicles must be identified and eliminated. In other systems, this is an iterative process; if a conflict is found, modifications are made to the conditions to prevent it, and then the routes are re-calculated and the conflict resolved again. This paper takes advantage of a unique property of the min-sum assignment that eliminates conflicts except in easily-checkable scenarios, and allows for those scenarios to be remedied without re-calculating routes. In short, the line segments in a route generated by a min-sum assignment do not intersect. A detailed explanation and proof is available in Appendix A. Non-intersection is a more powerful statement than deconfliction. A deconflicted route means two vehicles must not occupy the same space at the
same time, but the non-intersecting property shows that the entire length of the paths do not cross at any time. This decouples vehicle speed from the path planning and permits speed adjustments without altering the deconfliction property.

Obstacles in the environment modify the results slightly. While the non-intersecting line segments property continues to hold, it is possible for line segments of the route to share endpoints at obstacle vertices. In Fig. 2.3, two vehicles share a vertex in their assigned route. The non-intersection property of the assignment holds for all route segments, meaning only the shared vertex need be checked for conflict, and if a conflict is found, the vehicles can change speed to avoid conflict without altering the deconfliction of the remaining route segments. By loosening any initial assumptions about vehicle speed, shared vertices can be deconflicted by small speed adjustments without changing the horizontal path of the vehicle. Because the assigned path is known not to intersect with other paths, this speed adjustment cannot cause other routing conflicts, eliminating the need to re-check the final solution for conflicts. Interestingly, the assignment here is indeterminate. The total cost of either combination of routes are the same, $AC + BD = AD + BC$, therefore either vehicle can be assigned to either task without changing the total cost. This holds true for any set of vehicles and tasks that have a shared vertex in their route. This indeterminacy can be taken advantage of when one task conflicts with the route to another task. In Fig. 2.4 one task lies under a shared route segment. Because both possible assignments have the same total cost, we can select the assignment to prevent possible conflicts. This is done by assigning the closest vehicle to the vertex to the furthest task form the vertex. In this instance A is assigned to D, preventing the possibility of A stopping at point C (or loitering/orbiting) in the middle of B’s planned flight path. In extreme cases timing can
be corrected by placing vehicles in holding patterns, and is scalable to high-order AUAS when route segments are sufficiently long. Other remedies, like route adjustment (See Levedahl and Silverberg, 2005 [11]), become a last resort.

2.1.5 Autonomous Flight Controls

The autonomous planning block leads to vehicle assignments. The vehicle assignments represent low frequency and bandwidth information that can be performed off-board. The autonomous flight controls, in contrast, represent higher frequency and bandwidth processes that are ideally suited for on-board implementation. Best practices in autonomous flight controls are reviewed below. The first step is to convert assigned routes to guidance paths (G). At this stage routes are refined to account for turning radii and other properties to enable vehicles to follow the routes precisely. Indeed, slopes and curvatures of connected line segments are discontinuous at route vertices and can’t be followed precisely. Circular arcs between line segments, described by Dubins in 1957 /citeDubins1957, produce paths whose directions of travel (first derivatives) are continuous. Further refinement produces paths whose turning radii (second derivatives) are continuous, which allows time for the AUAV to change its flight direction through roll. The next step is to regulate the guidance paths. The k-th vehicle regulates its guidance path \( r_k \) independently and without awareness of other vehicles. An external disturbance \( u_{Dk} \), like a wind load, causes deviations from the actual flight path \( x_k \). The control surfaces are adjusted by a guidance input \( u_{Nk} \) and a regulation input \( u_{Rk} \). The guidance input serves to fly the vehicle along its guidance path in the absence of external disturbances and the regulation part compensates for errors. The primary errors are caused by external disturbances but others errors originate from prescribing non-smooth guidance paths, transducer error, and model errors in the plant \( L_k \). The guidance state \( x_{Nk} = [r_{Gk}^T \ \theta_{Gk}^T \ \psi_{Gk}^T \ \omega_{Gk}^T]^T \) is received while following the actual state \( x_k = [r_k^T \ \theta_k^T \ \psi_k^T \ \omega_k^T]^T \). Let \( u_k \) represent the control input vector. The equations governing the motion of the vehicle, the autonomous navigator, the autonomous regulator and the form of the control input are

\[
L_k x_k = u_k \\
K_{Gk} r_k = u_{Nk} \\
K_{Rk}(r_k - x_k) = u_{Rk} \\
u_k = u_{Nk} + u_{Rk} + u_{Dk}
\]

where \( K_{Gk} \) is the guidance operator, \( K_{Rk} \) is the regulator operator, and where the control input is the sum of a guidance input \( u_{Gk} \), a regulator input \( u_{Rk} \), and a disturbance input \( u_{Dk} \). This is called
the modern control form. (See Levedahl and Silverberg, 2009 [22]). Note that the operator notation above does not specify the realization, that is, whether the system is represented by differential equations, difference equations, algebraic equations, or a hybrid. Under ideal conditions ($G_{Gk} = L_k$ and the guidance input is smooth), the guidance input causes the vehicle to follow the guidance state vector exactly. It follows that the input-output relationship for the vehicle, the error, and the characteristic equation are:

\[
(L_k + G_{Rk})e_k = u_{Dk}
\]

\[
e_k = x_k - r_k
\]

\[
0 = |z I - (L_k + G_{Rk})|
\]

where $I$ is the identity operator and $e_k$ is the state error vector. The characteristic equation is independent of the guidance state vector. Therefore, the regulator gain matrix $G_{Rk}$ determines the vehicle's stability characteristics (settling time, peak-overshoot, and steady-state error) independent of the guidance state vector $r_k$. These best practices pertaining to autonomous flight controls are summarized as an AUAS Flight Controls Principle: Assume that a realizable guidance path has been produced and that a vehicle has the control authority capable of maintaining the guidance path within certain limits placed on settling time, peak overshoot, and steady-state error. Then autonomous guidance and autonomous regulation can be designed by separate and distinct methods. Following best practices in autonomous flight controls lead to vehicle paths that more closely follow their guidance paths, which is of particular interest in autonomous systems that are not instrumented with on-board collision avoidance systems.

### 2.2 Algorithm Complexity

Using the described architecture, the user is decoupled from control of individual vehicles. This removes human span-of-control limits on maximum number of vehicles, leaving only computational requirements of the algorithms and bandwidth as major limitations. Because there is no iteration between the steps in the architecture, the limiting overall complexity in scaling is the fastest growing of the individual complexities. The Visibility graph method of tabulating the environment has a Big O complexity of $O(E\log(V))$, where $E$ is the number of edges in the environment border and obstacles and $V$ is the number of vertices. However, a unique Visibility graph is made for each vehicle and task combination, resulting in a total complexity for the Visibility graphs of:

\[
O(E\log(V)nm)
\]
where $n$ is the number of vehicles and $m$ is the number of tasks. Similarly, the A* graph search algorithm has a complexity of $O(E)$, but is accomplished once for every vehicle-task pairing, and has an overall complexity of:

$$O(Enm)$$

(2.5)

Once the cost matrix is built, the Hungarian algorithm for min-sum assignments has a complexity of:

$$O(x^3)$$

(2.6)

where $x = \max(n, m)$ is the larger of tasks or vehicles when the assignment is non-square. This algorithm is only accomplished once for all vehicles and tasks. It is clear that for complex environments, $\log(V) > 1$ for $V > 10$, therefore $O(E\log(V)nm) > O(Enm)$ for $V > 10$, that is, computing the Visibility graph will be more complex than searching the graph with A*. However, it is interesting to compare the Hungarian complexity, Eq. (2.6), with the Visibility complexity, Eq. (2.4). The Visibility graph complexity will grow faster than the Hungarian algorithm if the number of edges, vehicles, and tasks all grow at a similar rate, that is $O(\log(x)x^3) \geq O(x^3)$, where $E, V, n, m = x$, for $x > 10$. However, if either the tasks or vehicles grow significantly faster than any other part of the problem, then the Hungarian algorithm will dominate the complexity. It is useful to note that this implementation of the Visibility graph recalculates the environment for each vehicle-task pair. If the environment is significantly complex, it becomes helpful to build the environmental graph once, and then connect the vehicles and tasks into it. This would require using new algorithms to solve this variant of the problem.

### 2.3 Hardware

Once the ground station has a set of vehicles assigned to tasks and associated routes, they must be transmitted to the vehicles. The next waypoint on each route is sent to the respective vehicles using the MAVLink 'guided' command. MAVLink is a communications protocol designed by Lorenz Meier for communicating with small unmanned aircraft [23]. Originally designed for use on the PXHAWK autopilot, it is also used on the PX4, APM, Parrot AR.Drone and other platforms and it can be used both for communication with the ground station and for onboard communication between components. A message in MAVLink consists of header information including sequential message number, system and component ID, and message type, body information consisting of parameters specific to the message, and a checksum count terminating the message. The 'guided' command is a particular type of 'mission item' command and has 37 bytes of message-specific information and 8 bytes of header and checksum data. Each vehicle transmits back its location and
other information with typical message payloads of 1 to 40 bytes, or a few hundred bytes per vehicle for each set of updates. Less important than the actual number of bytes transmitted, the bandwidth required scales linearly with the number of vehicles. In the demonstrated system, updates were sent to the vehicles at a rate of 1hz, and data was received from the vehicles at a rate of 1hz. The telemetry communications was set to 57,600 baud, providing a bandwidth limitation of around 10 vehicles. Increasing this limitation is fairly easy, the xbee transceivers can run multiple groups on different channels and radio communications with higher bandwidths are readily available. Moving from the guided message to transmitting more capable multi-point flight plans would increase the required bytes per aircraft significantly, but the communications would still scale linearly with the number of vehicles.

The four test vehicles seen in figure 2.5 are built around Hobbyking's Bixler V2 aircraft, a stable, beginner-level radio-control aircraft with reliable handling characteristics and a durable, easy-to-repair expanded polyolefin (EPO) structure. The aircraft is used as a trainer, powered glider, and, more recently, as a First-Person-View (FPV) platform, where its easy-to-modify EPO foam body lends itself to mounting of additional cameras and telemetry equipment that let the operator fly by viewing real-time camera footage from the actual aircraft. Radio-control linkage to each aircraft used Spektrum controllers and receivers. Two vehicles used the Spektrum DX6i, with channel mixing capability that made initial testing of the autopilot much easier. Two vehicles used the Spektrum DX5e, which lacks the ability to command multiple autopilot modes but was sufficient for...
emergency manual control. While imperative for initial development and testing, future versions of a multi-vehicle system could have no manual control ability at all. Each aircraft uses an ArduPilot 2.5 built by 3D Robotics for on board control. The Ardupilot receives GPS location information from an external GPS receiver and combines this with onboard gyros, accelerometers, and altimeter to determine aircraft location and state. While it has the ability to also use a pitot-static probe for airspeed measurements, this feature was not used for this experiment. The Ardupilot was used in manual pass-through and auto modes. In manual mode the servo inputs from the R/C receiver are exactly the outputs sent to the servos and motor. In auto mode the Ardupilot uses its own navigation and state information to autonomously follow a flight plan with waypoints and simple logic. Airborne communication with the autopilot is accomplished using Xbee Pro transceivers operating on 900 Mhz. These transceivers have an outdoor LOS range of up to 1.8 miles and can operate in a multi-point network enabling the GCS to use one transceiver to connect to all of the vehicles [24]. Two vehicles used wire antennae and two used RPSMA ‘rubber duck’ antennae. The Xbees were operated at 57,600 bps but are capable of faster transmission speeds. The aircraft were powered by Turnigy 2200mAh batteries driving Turnigy Aerodrive 2822 1275 kV motors spinning 7x5 props. Two vehicles used Castle Thunderbird 18 ESCs and two used Great Planes Electrify25A Silver Series ESCs. Estimated cruise flight time with this power train is 30 to 45 minutes. Servos were standard 9g Micro size made by HobbyKing. To reduce required ballast, elevator and aileron servos were moved to the tail. No additional payload was added. Servos were powered from the ESC battery elimination circuit (BEC). The GCS consisted of a Lenovo T430 with an Intel Core i5-3230M CPU running at 2.6GHz connected to an Xbee Pro 900Mhz for communications link to the vehicles. A custom program operating in MATLAB provided the user interface and recorded log data. Screencast-o-matic was used to record screen video of the test flights.
Figure 3.1 Aerial view of flight test area at Lake Wheeler Road Field Laboratory. [25]
Several demonstration flights were flown covering sets of user-selected points, areas converted into a set of lines, dynamic tasking of additional tasks, and obstacle avoidance. The results shown here represent three different flights of approximately 15 minutes each, limited by the recording ability of the screen-capture video software used. During each flight position, heading, tasking information and other data were recorded for each vehicle at a sample rate of 1hz. Selected portions of this data are graphed below. Test flights were flown at the Lake Wheeler Road Field Laboratory seen in Fig. 3.1, an agricultural lab near Raleigh, NC operated by North Carolina State University.

3.1 Multiple Point Tasks

![Figure 3.2 Multi-task initial assignment.](image-url)

![Figure 3.3 Assignment following task completion.](image-url)

In Fig. 3.2, six tasks have been input to the system and all four available vehicles have been assigned. Each vehicle determines autonomously how to get to the assigned task, as well as all flight and navigation control loops required operate onboard. In Fig. 3.3, we see vehicle 2 arrive near its first assigned task, be reassigned, and then be reassigned again as vehicle 4 completes its assigned task. The key behavior seen here is that at each task completion, all vehicles participate in the next assignment, there is no memory of what assignments were made previously. Deconfliction at each new step is guaranteed by having all vehicles participate in the reassignment. While this clearly does not find the complete optimal solution, it does easily adapt to a dynamic environment where the set of assigned tasks is likely to change before all tasks are completed. This example also shows an unexpected behavior from the aircraft, as vehicle 2 approaches its point, it does not proceed directly
towards it, but begins to turn to intercept an orbit around it. In guided mode, the vehicle will enter an orbit around the sent point. Here the vehicle has begun to enter that orbit before the ground station has identified the vehicle as having arrived at the task and assigned a new point to fly to.

### 3.2 Area Task

![Figure 3.4 Assignment to swaths.](image)

![Figure 3.5 Completing swaths.](image)

In the next example, the user defines an area which is converted into a series of swaths which are assigned as tasks having two endpoints. Fig. 3.4 shows the initial assignment, where three of the four available vehicles are assigned to the three tasks. Some time later, Fig. 3.5 shows the vehicles moving down the assigned swaths. Upon completion of the assigned swath, each vehicle will return to its home loiter position.

### 3.3 Dynamic Tasking

Both of these scenarios involved the vehicles accomplishing an initial set of tasks and then returning to their loiter positions. It is clear that more optimal solutions might be calculated ahead of time if all tasks were known a priori. The next example shows multiple retaskings where additional tasks or sets of tasks are assigned before the completion of the initial set. This simulates a dynamic work load in a changing environment and shows the flexibility of this set of algorithms. In Fig. 3.6 the vehicles begin at their loiter positions when a task is input and vehicles 1 and 2 are assigned. In Fig. 3.7, an area is input by the user, resulting in three swaths added to the set of tasks to be accomplished. All available...
3.3. DYNAMIC TASKING

Figure 3.6 Dynamic assignments Begin.

Figure 3.7 Adding swaths.

Figure 3.8 Beginning to complete swaths.

Figure 3.9 Adding more swaths
vehicles are subject to reassignment, and while vehicle 2 continues to its previously assigned task, vehicle 1 is sent to one of the new tasks. In Fig. 3.8, vehicles 1, 3 and 4 are accomplishing their swaths, while vehicle 2 completes one task and is assigned to the next. In Fig. 3.9, a new area has been input. This does trigger a new assignment, though vehicles 1 and 4 do not participate because they are actively executing a task, rather than traveling to a task. Vehicle 2 completes its task in Fig. 3.10 and is assigned to one of the new swaths. In Fig. 3.11 vehicles 1 and 4 have completed their assigned swaths and are assigned to the new area. The vehicles continue through the rest of the tasks, completing nine different tasks assigned in three sets before returning to their loiter positions.

3.4 Obstacle

This final example demonstrates behavior around an obstacle. In Fig. 3.12, the magenta aircraft has been tasked to a point on the other side of the obstacle area, in white. The vehicle is not aware of the obstacle, but the route found using the visibility graph and A* methods includes the corner of the obstacle as an intermediate point. The vehicle is commanded to fly towards that corner, and then to fly on to the task. This method of path planning, where the vertices of the obstacle are used for routing, requires that the shape designated as the obstacle be sufficiently large enough to allow for positional errors, including navigational error and environmental disturbances, and allow for turn radius and other vehicle path requirements. A video highlight of the user interface output recorded during these demonstrations is included with the electronic copy of this dissertation.
3.4. OBSTACLE

Figure 3.12 Routing around an obstacle.
These flights successfully demonstrated a system architecture that decouples the human operator from individual vehicles, thereby overcoming human span-of-control limitations. The GCS interpreted user-input goals and sent task assignments to available vehicles. Vehicle deconfliction was accomplished by the non-intersecting property of the assignment method used. The individual algorithms are arranged in non-iterative steps, making the scaling complexity of the entire system easy to analyze. The min-sum algorithm used incorporates a one-task horizon that might miss complete optimal solutions in a static environment, but adapts easily and efficiently to a dynamic environment. Immediate scaling limitations of the system as tested are likely to be bandwidth based, with bandwidth becoming saturated near 10 vehicles, however, bandwidth requirements scale linearly and are easily overcome with more channels or faster communications methods. As the system scales, the dominant algorithm depends on whether the number of vehicles and tasks or the complexity of the environment grows faster. In an environment of increasing complexity, the Visibility graph algorithm becomes the most computationally complex, while the Hungarian assignment method dominates if the growth of vehicles or tasks outpaces the complexity of the environment. Some unexpected vehicle behavior appeared, however, the system remained tolerant of this behavior and stayed stable in an environment with some dropped communications and other system lag.
Future improvements or area of study depend on what purpose the system is used for. As demonstrated, the system is able to perform persistent surveillance. The only adaption required would be adding specific tasks, such as loitering over a point or following a moving target, and the associated onboard sensors. Scaling the number of vehicles significantly in the multi-task dynamic environment demonstrated here would require a better understanding of how the deconfliction guaranteed with the straight-line routes used in the algorithm is affected by actual vehicle flight paths and tracking errors. Incorporating the ability to handle non-homogenous systems would enable multi-user systems that could respond to significantly different requests, such as dropping a payload vs. imagery, or particular spectral imagery rather than visible light. Clearly, the architecture presented here is widely applicable to how we interact with multi-vehicle systems.

Ultimately, the future of human interactions with multi-vehicle autonomous systems will require system architectures that overcome human span-of-control limitations and scale predictably. Whether we choose to use multi-vehicle systems for their damage tolerance, persistence, or to keep individual vehicles under a certain cost or weight limitation while maintaining system capability, the future will certainly involve systems of vehicles that are orders of magnitude larger than we currently fly.
BIBLIOGRAPHY


Consider \( \{Q_i\}_{i=1}^{i=n} \), called point set \( A \), and \( \{q_i\}_{i=1}^{i=m} \), called point set \( B \), in which the points \( Q_i \) and \( q_i \) are contained in domain \( D \) in plane \( \mathbb{R}^2 \) and \( n \leq m \). Next, construct a set of line segments \( \{C_i\}_{i=1}^{i=n} \) in \( D \), called connecting line segments. The connecting line segments connect each point in \( A \) to a unique point \( B \). Two connecting line segments are said to intersect (cross) if they share a single point that does not lie on the boundary of \( D \). The lengths of the connecting line segments are \( L_i \) and the total length is \( L = \sum_{i=1}^{n} L_i \). Under these assumptions, the connecting line segments that have the shortest total length are non-intersecting. 

**Proof**

This theorem essentially claims the following: When connecting two sets of points in an environment that may contain obstacles, the combination of line segments that has the shortest total length are non-intersecting (See Fig. A1).

To begin, let us show that any two pairs of points in an assignment can be examined separately from the larger assignment without changing the optimality of the assignment. Without loss of generality, the points in the set \( B \) are ordered so that the connecting line segments forming the smallest total length are \( \{Q_i,q_i\}_{i=1}^{n} \). Then, the sum of the lengths is \( L_{min}(n,m) = \sum_{i=1}^{n} Q_i,q_i \). The first connecting line segment and associated points, \( i = 1 \) in set \( A \) and set \( B \) are removed. The removed line
Segment length is $L_{\text{min}}(1, 1) = Q_1 q_1$ and the sum of the remaining points is $L_{\text{min}}(n, m) - L_{\text{min}}(1, 1) = \sum_{i=1}^{n} Q_i q_i - Q_1 q_1$, or:

$$Q_1 q_1 + \sum_{i=2}^{n} Q_i q_i = \sum_{i=1}^{n} Q_i q_i$$

(A.1)

This will be used to show that the remaining assignment forms the minimal sum of the remaining points in both sets. To do this, first assume that the remaining sum, $\sum_{i=2}^{n} Q_i q_i$, is not the minimal solution. This implies there exists a $L^*$ total length that is shorter than $\sum_{i=2}^{n} Q_i q_i$. This leads to $Q_1 q_1 + L^* < \sum_{i=1}^{n} Q_i q_i$, necessitating the existence of a shorter solution than $\sum_{i=1}^{n} Q_i q_i$, which was defined as the minimal solution. Therefore, $\sum_{i=2}^{n} Q_i q_i$ is the minimal solution of the remaining points in both sets. Furthermore, $n$ line segments are non-intersecting if all combinations of two of them is non-intersecting. Therefore if the generalized pair of two line segments in the solution are non-intersecting, then it holds for any $n \geq 2$, and the theorem is proved. Thus, it remains to prove the theorem for $n = 2$ connecting line segments.

To prove this theorem, one builds on some basic properties of paths between points. Archimedes provides Property 1, that the shortest distance between two points is a straight line [Archimedes]. Property 2 is Euclid's triangle inequality [Euclid], stating that any side of a triangle is shorter than the sum of the other two sides. Note that this can be used to prove Archimedes' straight-line theorem for any path that can be described as or approximated by a series of straight lines.

Fig. A.2 shows the set of points $[A, B]$ and $[C, D]$. In the assignment of two agents to two tasks, there are $2! = 2$ possible assignments, $AC$ and $BD$ or $AD$ and $BC$. As above, the proof of non-intersection will be shown by examining the opposite possibility. In the assignment of $AD$ and $BC$,
the total length is $AD + BC$. Defining the crossing point as $E$, we see that $AE + ED = AD$, and the total length $AE + ED + BE + EC = AD + BC$. This expanded sum can be compared to the other possibility $AC + BD$. Conveniently, the sums can be compared as edges of two triangles $AEC$ and $BEO$, and Euclid’s triangle inequality assures us that $AC < AE + EC$ and $BD < BE + ED$, leading to the conclusion that

$$AC + BD < AE + EC + BE + ED \quad (A.2)$$

showing that an assignment resulting in intersecting paths will have a shorter, non-intersecting, possibility, and thus cannot be the minimal assignment. By contradiction, it is proved that if the minimal solution is found, it will not have path segments that intersect. The case for $n = 2$ is proved, thus the case for $n > 2$ is also proved.

**End of proof**