8-1-2019

Collaborative UAV Surveillance

Winston Smith
Southern Illinois University Carbondale, wssmit95@gmail.com

COLLABORATIVE UAV SURVEILLANCE

by

Winston Smith

B.S., Southern Illinois University, 2018

A Thesis

Submitted in Partial Fulfillment of the Requirements for the

Master of Science Degree

Department of Computer Science

in the Graduate School

Southern Illinois University Carbondale

August 2019

THESIS APPROVAL

COLLABORATIVE UAV SURVEILLANCE

by

Winston Smith

A Thesis Submitted in Partial

Fulfillment of the Requirements

for the Degree of

Master of Science

in the field of Computer Science

Approved by:

Dr. Henry Hexmoor, Chair

Dr. Tessema Mengistu

Dr. Koushik Sinha

Graduate School

Southern Illinois University Carbondale

May 11, 2019

AN ABSTRACT OF THE THESIS OF

Winston Smith, for the Master of Science degree in Computer Science, presented on May 11, 2019, at Southern Illinois University Carbondale.

TITLE: COLLABORATIVE UAV SURVEILLANCE

MAJOR PROFESSOR: Dr. Henry Hexmoor

Autonomous collaborative robotics is a topic of significant interest to groups such as the Air Force Research Lab (AFRL) and the National Aeronautics and Space Administration (NASA). These two groups have been developing systems for the operation of autonomous vehicles over the past several years, but each system has several critical drawbacks. AFRL's Unmanned Systems Autonomy Services (UxAS) supports pathfinding for multiple tasks performed by groups of vehicles, but has no formal verification, very little physical flight time, and no concept of collision avoidance. NASA's Independent Configurable Architecture for Reliable Operations of Unmanned Systems (ICAROUS) has collision avoidance, partial formal verification, and thousands of hours of physical flight time, but has no concept of collaboration. AFRL and NASA each wanted to incorporate the features of the other's software into their own, and so the CRoss-Application Translator for Operational Unmanned Systems (CRATOUS) was created. CRATOUS creates a communication bridge between UxAS and ICAROUS, allowing for full feature integration of the two systems. This combined software is the first system that allows for the safe and reliable cooperation of groups of unmanned vehicles.

DEDICATION

To Paul Coen, who taught me to reach *ad astra*.

To McGwire and Amariah Hidden, who are the truest friends I could hope for.

ACKNOWLEDGEMENTS

I would like to thank those teachers and professors that went above and beyond in helping me to learn. The names of these people are: Dr. Bardh Hoxha, Dr. Tessema Mengistu, Dr. Henry Hexmoor, Dr. Norman Carver, Dr. Koushik Sinha, Mindy LaHood, Megan Yardley, and Kristi Grenda. Some of these people gave me knowledge, and some taught me how to be truly passionate about a project. Without any of them I would not have been able to complete this.

I would also like to thank my family and friends. I am incredibly lucky to have such a large group of people that is so willing to support me in my aspirations.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The field of collaborative autonomous robotics is an area of intense interest for many groups, not least among them NASA Langley and the Air Force Research Laboratory (AFRL). These two agencies have each created software that helps to enable airborne or spaceborne vehicles to safely complete missions, but the focus of each agency's software is, in many ways, the opposite of the other. NASA's ICAROUS software has thousands of hours of flight time on physical vehicles and is partially formally verified [2], but it has absolutely no concept of collaborative missions. AFRL's UxAS smoothly handles multiple vehicles collaborating to complete multiple tasks, but has less than a hundred hours of flight time and little to no formal verification on any of its modules. Both agencies wanted to integrate the other software's features into theirs, and so NASA Langley funded the research behind this paper - the research that lead to the creation of the CRoss-Application Translator for Unmanned Operational Systems, or CRATOUS. CRATOUS is a system that allows for full integration of functionality between UxAS and ICAROUS. This is the first general system for collaborative vehicle missions that has any guarantees regarding safety of the vehicles, non-violation of mission constraints, or completion of the mission.

In Chapter 2, we detail the systems currently included in CRATOUS, UxAS and ICAROUS. We cover the architectures of these systems as well as their features and limitations. Process flow diagrams are given and explained, and some of the core algorithms are discussed.

In Chapter 3, we move on to the functionality of CRATOUS, which combines ICAROUS and UxAS. We give an example problem for each side of the CRATOUS bridge, and walk

through what happens in each piece of software. Finally, we give a short discussion on the architecture of CRATOUS and its design philosophy.

In Chapter 4, we examine the results of our work that have not already been covered in previous sections. Included is a discussion on the limitations of CRATOUS and suggestions for the work that should be done on it in the future.

Some work related to this subject can be found in [5], a paper discussing the fine hardware and software details of one implementation of small autonomous UAVs. References [6], [7], and [13] discuss decentralized algorithms for consensus in UAVs, which is somewhat of an opposite to the approach used in UxAS. Reference [8] provides a mathematical approach to the creation of an algorithm for UAV tracking of unknown targets in low-information environments. Reference [14] discusses the implementation of real-time path planning, following the process from algorithm to hardware experiment, and [15] provides an algorithm for estimating position and attitude of a UAV with minimal sensors. The report in [18] details an experiment where UAVs are flown in a low-altitude, high-obstacle, high-interference environment and expected to inspect ground structures. Reference [22] approaches UAV routing from a communication-aware standpoint, routing UAVs so that they may efficiently communicate during a mission. Finally, [23] provides a system for forest fire monitoring using a team of autonomous UAVs.

CHAPTER 2

BACKGROUND: UxAS

UxAS is a modular system for computing the optimal paths for a set of UAVs, given a set of tasks for the UAVs to perform as a group, constraints on the tasks, keep-in and keep-out zones, and the parameters of the UAVs, such as initial position and maximum speed. UxAS is organized into a group of modules, which may send and receive messages across the Light-weight Message Control Protocol, or LMCP, Bus. Modules may send a generalized message with a topic that other modules may or may not subscribe to or they may send a targeted message, delivered only to a specific module or group of modules [21]. AMASE is connected to the LMCP bus through its own special connection, and may receive and send messages just like normal modules. Figure 2.1 shows a basic diagram of the architecture of UxAS.

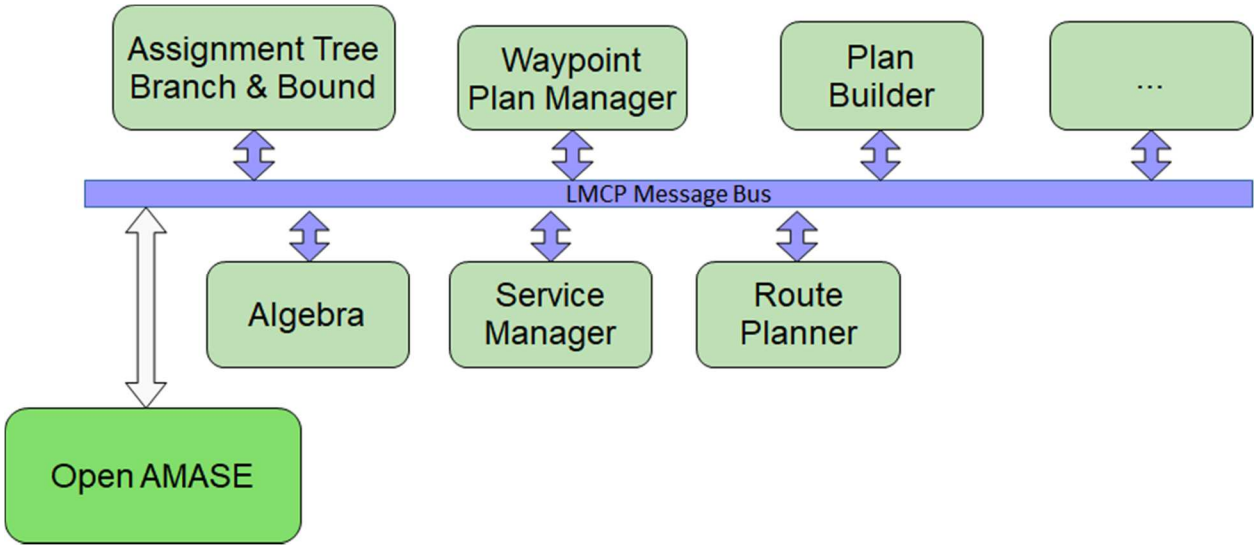

Figure 2.1: UxAS Architecture

A full examination of the workings of UxAS is beyond the scope of this paper, but a brief summary is necessary to the understanding of how CRATOUS, as a whole, functions. At startup, UxAS reads from a large set of files, mostly given in the XML format, and uses the

settings and input contained in the files to initialize its modules. Of special importance is the process algebra string, which is a string that tells UxAS which tasks should be performed, as well as any constraints on the tasks. This string is discussed further in the section titled "Process Algebra," below. UxAS expands the algebra string into TaskOptions, representing different ways to fulfill the tasks. For example, a LineSearch task may be searched starting from either end of the line. When the algebra string has been expanded and parsed into internal Task data, UxAS enters its main computational phase.

The core algorithm behind the optimality solving of UxAS is a branch-and-bound tree. This tree is roughly sorted from first branch to last branch by cost. Cost is determined by an optimization parameter given by the user before runtime. The tree is parsed from start to finish, and entire sections may be pruned - "bounded" - early if they are obviously worse than a solution already found. Given enough time, UxAS will find the best solution to the mission it was given, but given less time than that it will return the best solution it could find in the time it was given. Usually, the first branch of the tree is either the optimal solution or, more likely, close in cost to the best solution, since the branches are organized according to a basic estimation of their total cost. This allows UxAS to quickly find a solution that should be workable, but in the worst case it must consider every possible combination of assignments of TaskOptions to UAVs - an NP-hard problem that is roughly equivalent to a simpler version of the Travelling Salesman problem with multiple salesmen. Figure 2.2 shows a highly-simplified version of the flow of events in UxAS.

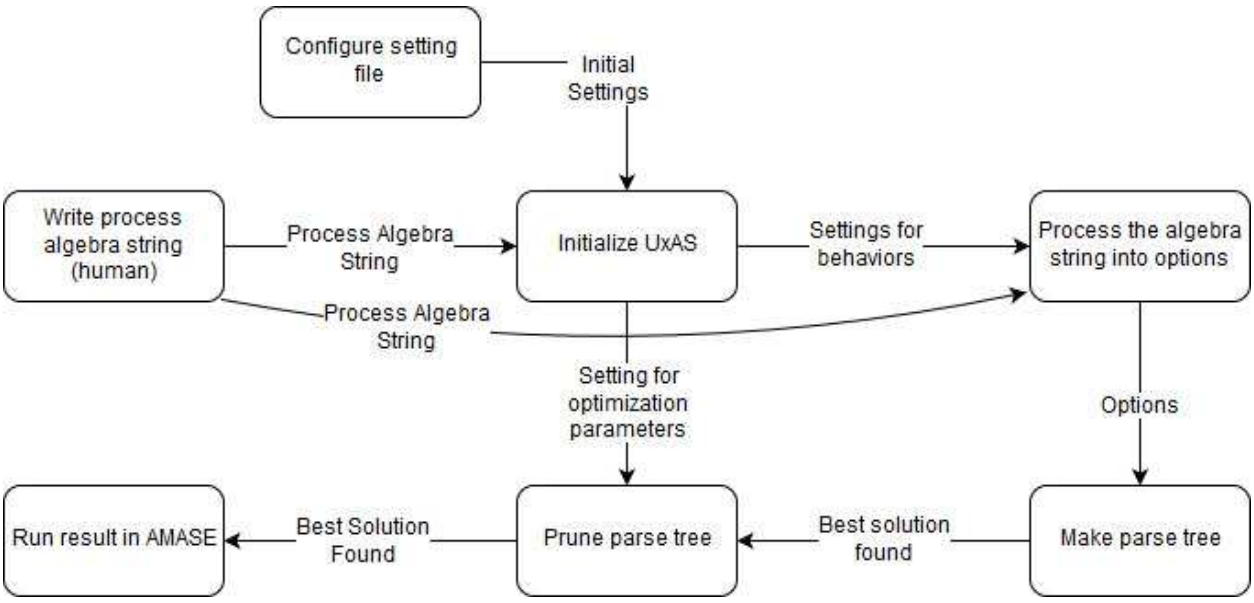

Figure 2.2: UxAS Sequence of Events (credit to Mr. Paul Coen for creating this figure)

According to [17], there are five levels of automation in military UAVs, as well as in general, ignoring the usual "zero" level of no automation. The first, and lowest, has been achieved. It is the automation of motion, and in our case this means that a UAV can fly if it is given a heading by its operator. The second level of automation is where UxAS falls; it is the automation of deciding where to go. In the case of UxAS, this means deciding where UAVs acting as a group should fly to, given the tasks the group should complete. The third level is the automation of deciding which tasks should be performed in an area, given the overall goal of the mission. To the best of our knowledge, no piece of software has been able to do either this or the higher levels of automation. The fourth level is deciding the overall mission in a small area, given an overarching goal in a region. The fifth level is deciding the overarching goal in a region, given only the end goal for the overarching project and information about the entire area of interest.

UxAS is a system for computing the optimal paths for multiple vehicles to complete multiple tasks. It is easily-extensible, open-source, and has a large and active development team

at the AFRL-RQ division. Its modules are entirely encapsulated from each other with the exception of message passing. AMASE, an easy-to-use simulation environment, integrates almost seamlessly into the LMCP Bus. Additionally, nearly all behavior of UxAS can be changed through the use of its settings files, which enables easy automated testing of the software.

Process Algebra

The mission specification language for UxAS is called process algebra. More information on process algebra as a mission and task assignment language can be found at [12]. This is a very small subset of Metric Temporal Logic, and also uses some different notation. In the default process algebra of UxAS, there is no concept of time, and there are only three operators. The first, "alternative," specifies that at least one task of a set should be performed. The second, "parallel," functions similarly to an "and" operator, specifying that all tasks in the set can and should be performed concurrently. The third, "sequential," specifies that tasks should be assigned in the same order they are given, but otherwise behaves the same as the parallel operator. A table of these operators and their symbols is given below.

Table 2.1: Default UxAS Process Algebra

Operator	Symbol
ALTERNATIVE	+
PARALLEL	|
SEQUENTIAL	.

The "timing" GitHub branch of OpenUxAS provides three additional process algebra operators, but is not part of the core software yet. The code for the "timing" branch can be found at https://github.com/afrl-rq/OpenUxAS/tree/timing and a discussion of these operators is in the next section.

Timing Constraints in Process Algebra for UxAS

The default process algebra for UxAS is quite basic. Aside from an "and" operator and an "or" operator, there is only one further operator, sequential, which in practice works much the same way as the "and" operator. To expand the flexibility of assignment specification, three new operators were added and the definition of the sequential operator was changed. The list of new operators can be found below, in Table 2.4.

Table 2.2: Timing Operators of UxAS

Operator	Symbol
TILDE	~
ABSOLUTE	_[t_1 - t_2]
RELATIVE	-[t_1 - t_2]

The definition of the sequential operator was changed to be an end-to-start constraint on the two tasks. The string ".(p1 p2)" specifies that both p1 and p2 should be assigned, but p2 should not start until p1 is complete. The tilde operator works the same way, but is a start-to-start constraint rather than end-to-start. These two operators allow for a task dependency hierarchy to be created. This is useful in many scenarios. One example is if a UAV should attack a specified area, but cannot enter the area until an anti-UAV defense site is cleared, either by that UAV or a different one.

The timing operators continue in the same vein. The absolute timing operator follows a list of tasks, and t_1 and t_2 give a range in milliseconds from the start of the mission. The task list cannot begin before t_1, and cannot end after t_2. The relative operator works the exact same way except for when the sequential and/or tilde operators are present. When these operators are present, the relative operator's meaning changes. If the sequential operator is present, then the second task, the one that cannot begin before the end of the first task, cannot begin before t_1 milliseconds after the completion of the first task, and cannot end more than t_2 milliseconds after the completion of the first task. The same logic follows for the tilde operator, but, as before, the constraint is start-to-start rather than end-to-start. For both relative and absolute timing operators, if more than one can apply to a given task - such as in the string "|(p1 |(p2 p3)_[1000-1500])_[100-3000]" - then only the timing constraint that applies to fewer tasks will apply to its tasks.

The code that powers these constraints is located in the process algebra parser, branch and bound tree analyzer, and loitering assignment services of UxAS. The algebra parser has been modified to accept the new operators, put the information about them into a message, send out said message, and strip the new operators from the algebra string before continuing with

normal operation. The loitering assignment service accepts messages from the branch and bound tree analyzer, and assigns loiter tasks to the relevant UAVs at the specified times. This allows UAVs to wait before performing a task if necessary.

The branch and bound analyzer service has been modified to accept information about the new operators from the algebra parser, change how the tree is parsed based on this information, and pass any necessary loiter commands on to the loiter assignment service. At initialization, the information from the algebra parser is ready, and it stored internally for later use. When execution passes to the point where the branch and bound tree needs to be analyzed, the modified service uses the algebra information to change how the tree is parsed. Every time a branch is considered, the estimated times for beginning and completion of each task with a dependency or timing constraint is checked, and if it does not satisfy the constraints then it is pruned, no matter how optimal it may otherwise be. If a task satisfies the constraints except that it would begin too early, then a loiter task is assigned to cause the UAV to wait long enough to follow constraints. This is a different process from how the "vanilla" operators are parsed; in default UxAS the operators are implicit in the construction of the tree. This departure from the norm is necessary because the tree is constructed before time estimation occurs, which means that timing operators would not function properly otherwise.

Limitations of UxAS

First and foremost, UxAS has no verification, and extremely limited testing outside of the AMASE simulation environment. It has approximately 150 hours of real flight time as of 2014 [21], and no formal verification at all as of the time of writing. Additionally, UxAS potentially computes every possible combination of assignments of all tasks to all UAVs, so its computational complexity is potentially factorial in nature

Background: ICAROUS

ICAROUS, or Independent Configurable Architecture for Reliable Operations of Unmanned Systems, is a relatively new project from NASA Langley aiming to create, as the name implies, a verifiable and verified system for operations of autonomous vehicles. The architecture of ICAROUS is nominally similar to that of UxAS: There are many apps, connected through a messaging system via subscriptions. The only major architectural difference is that ICAROUS can, and is in fact designed to, run on a single autonomous drone, using no operating system, whereas UxAS must run on one of the major operating systems - Mac OS X, Ubuntu Linux 16.04, or Windows 7 or 10. Figure 2.5 displays a simple representation of the architecture of ICAROUS.

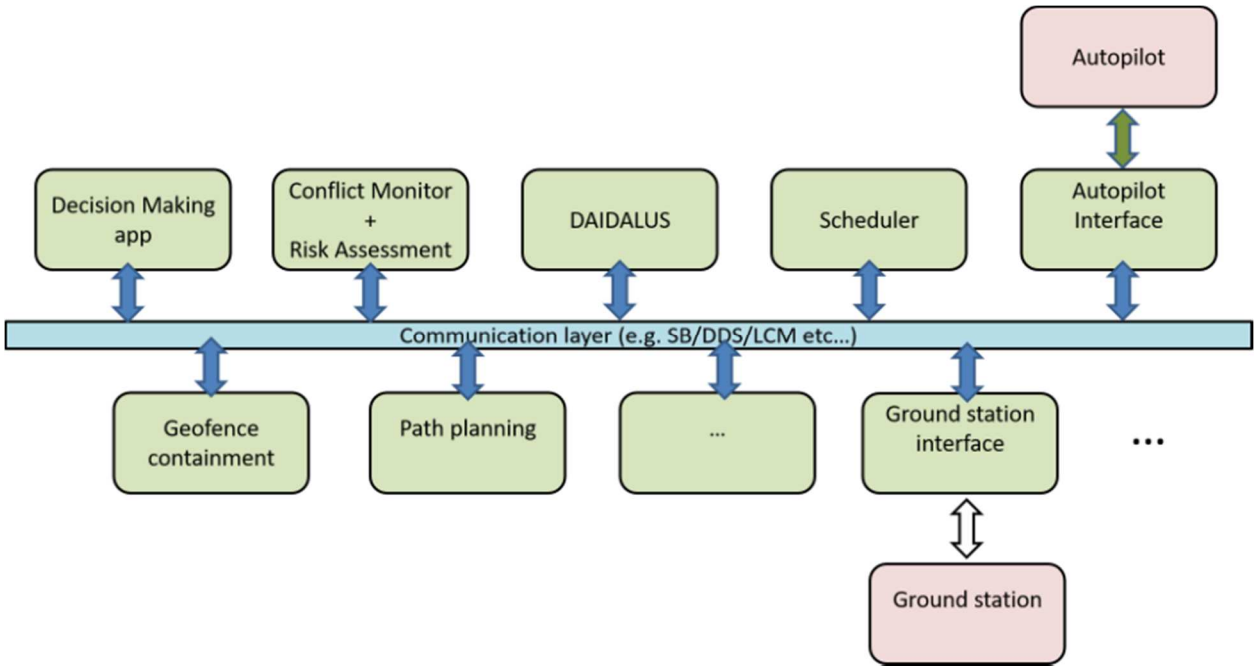

Figure 2.3: ICAROUS Architecture

Portions of ICAROUS are formally verified by the use of PVS Version 6.0 and the NASA PVS library. PVS is a verification system used for automated formal verification of software systems. ICAROUS is also being developed in a verification-driven manner, as

discussed in [11] and [20]. The two portions of ICAROUS that are fully formally verified at the time of writing are DAIDALUS [19], which handles the detection and avoidance of collisions, and PolyCARP, which handles computation behind the exact location of the points in polygons. This functionality is mainly used in allowing ICAROUS to know where geofences are located. Together, these two pieces of software can guarantee that collisions are avoided and mission constraints are respected given that it is possible to do so. The third guarantee mentioned in this paper, that of mission completion, is given as a result of extensive real-world testing of ICAROUS. The system has thousands of hours of flight time, and though new bugs are still found occasionally, mission completion can be practically guaranteed. The PVS proofs for DAIDALUS and PolyCARP can be found at https://github.com/nasa/WellClear/tree/master/PVS and https://github.com/nasa/PolyCARP/tree/master/PVS respectively. These proofs are far too long to directly quote in this paper.

ICAROUS is a project of interest because, to the best of our knowledge, it is the first and only system for the control of autonomous vehicles with any thought given to formal verification during development. Thus, it is the only such system that is at least partially proven to be safe and to follow mission constraints.

Limitations of ICAROUS

ICAROUS was designed to only control a single vehicle. It has absolutely no concept of cooperation, or of any vehicle -- other than the one it controls -- that is not an intruder. It is also only partially formally verified, and large parts of the code governing regular usage are not formally verified. The simulation environment for ICAROUS, MAVLink, is quite difficult to users, and is relatively feature-poor to developers. Additionally, it is relatively difficult to define

new missions in ICAROUS; there is no system equivalent to the process algebra of UxAS, nor the GUI editor for missions.

That said, these points are mostly either usability concerns or not as severe as they may appear. The partial formal verification allows the developers to continue developing ICAROUS, since they are not allowed to modify verified portions of code, and the thousands of hours of live flight time help to assure code correctness in the meantime. The final two points are based in usability. All of these complaints, with the sole exception of the one concerning formal verification, are solved with the addition of CRATOUS. CRATOUS provides ICAROUS with a connection to UxAS, which can control multiple vehicles collaboratively. ICAROUS is also provided with the OpenAMASE simulation environment, a much more sophisticated tool than MAVLink, and OpenAMASE includes a relatively easy-to-use GUI editor for missions. On top of that, CRATOUS missions are specified in process algebra through UxAS, making it much easier to quickly define new missions.

CHAPTER 3

APPROACH/METHODOLOGY: CRATOUS

The CRoss-Application Translator for Operational Unmanned Systems, or CRATOUS, is an ICAROUS module created to allow ICAROUS to connect to any server that has been configured to accept it. In the work done for NASA Langley in the summer of 2018, a module that can handle ICAROUS was added to UxAS, enabling these two pieces of software to collaborate on mission planning and runtime rerouting. As was mentioned in the introduction section, the combination of ICAROUS, CRATOUS, and UxAS has created the first system for solving general collaborative automated vehicle missions with any sort of guarantees towards safety, constraint nonviolation, or mission completion. In this section, we will expand further on these technical details, as well as detailing exactly how CRATOUS works.

CRATOUS is both an application that sets up a TCP/IP bridge with the server, communicates with the server, and provides an interface to ICAROUS for the server; and the overall messaging bridge between ICAROUS and the server. We have said that the modules of ICAROUS and UxAS communicate through a messaging bridge, and CRATOUS does not break this convention. Every change in behavior exhibited between ICAROUS and UxAS with and without CRATOUS running can be explained by the messages sent and received by the respective modules in each piece of software. This allows CRATOUS to function with, more or less, black-box applications. So long as the messaging interface between each side of CRATOUS and the overall pieces of software is the same, the internals of the applications do not affect how CRATOUS behaves on a basic level.

CRATOUS operates on an Internet connection between two pieces of software, and so in the real world, where there are no guarantees of message delivery via the Internet, it is non-

algorithmic and provides no hard guarantees. However, with a reliable Internet connection, there are many useful properties of the system as a whole. With ICAROUS running on each vehicle, all of the guarantees ICAROUS gives are provided here as well. Namely, vehicles will not violate mission constraints given that a nonviolation is possible, vehicles will not collide with each other or with intruders given that it is possible to avoid such collisions, and missions will be completed in finite time if it is possible to do so. With UxAS running as an overall mission "commander," these features are brought into a collaborative environment, with one caveat. The instances of ICAROUS that run for each vehicle do not communicate with each other, so in very rare, perfectly-symmetrical edge cases, vehicles may avoid each other and end their avoidance maneuver in the exact same positions they started the maneuver in, forever. This requires that the vehicles approach roughly the same point at nearly the exact same time, have the exact same maneuverability parameters as each other, and start and end their avoidance maneuvering at nearly the exact same time. Additionally, altitude and speed resolutions must be impossible or disabled for this scenario to occur. Examples where the guarantees do and do not break down are seen in Figures 3.1, 3.2A, and 3.2B.

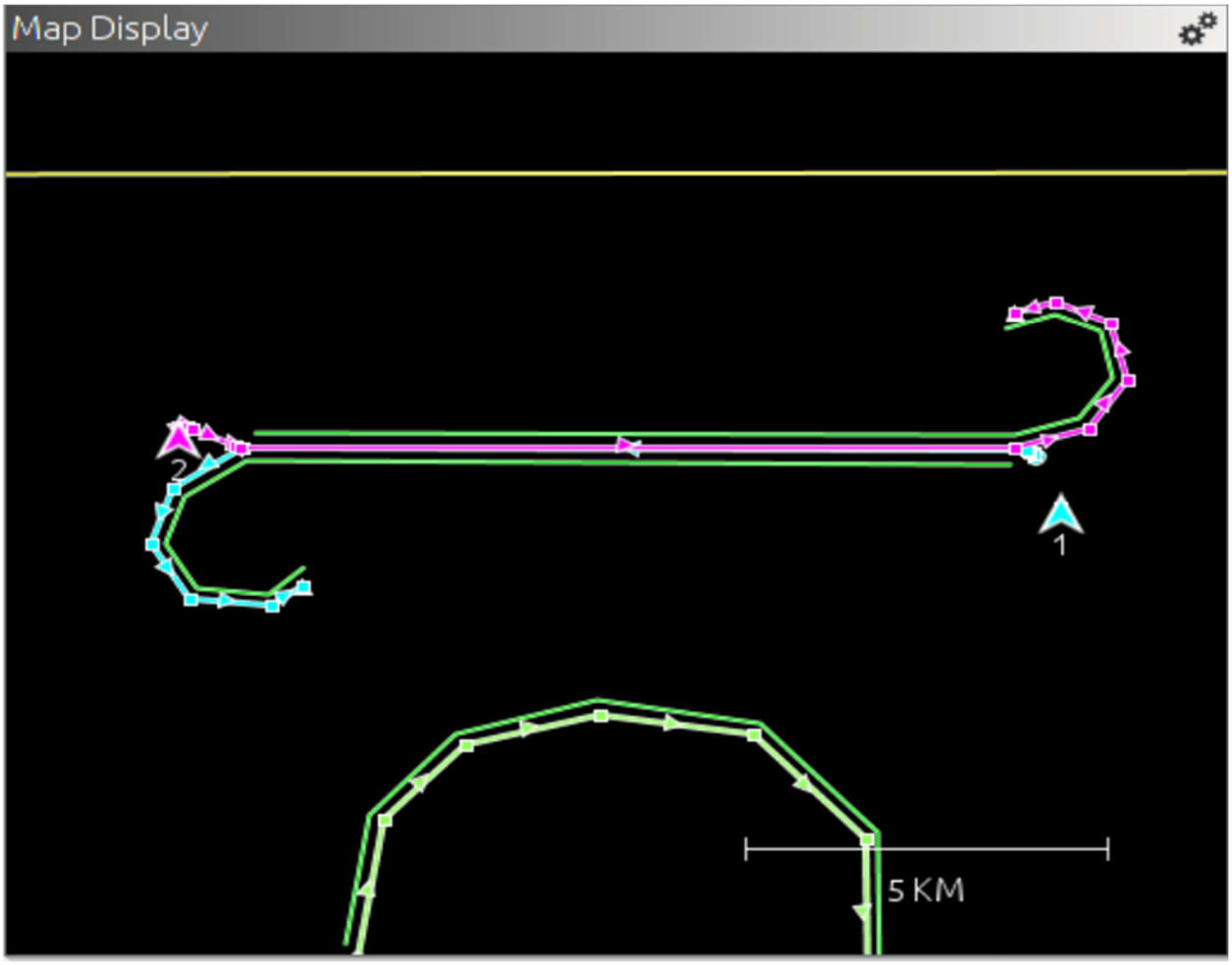

Figure 3.1: A case where the ICAROUS guarantees break down

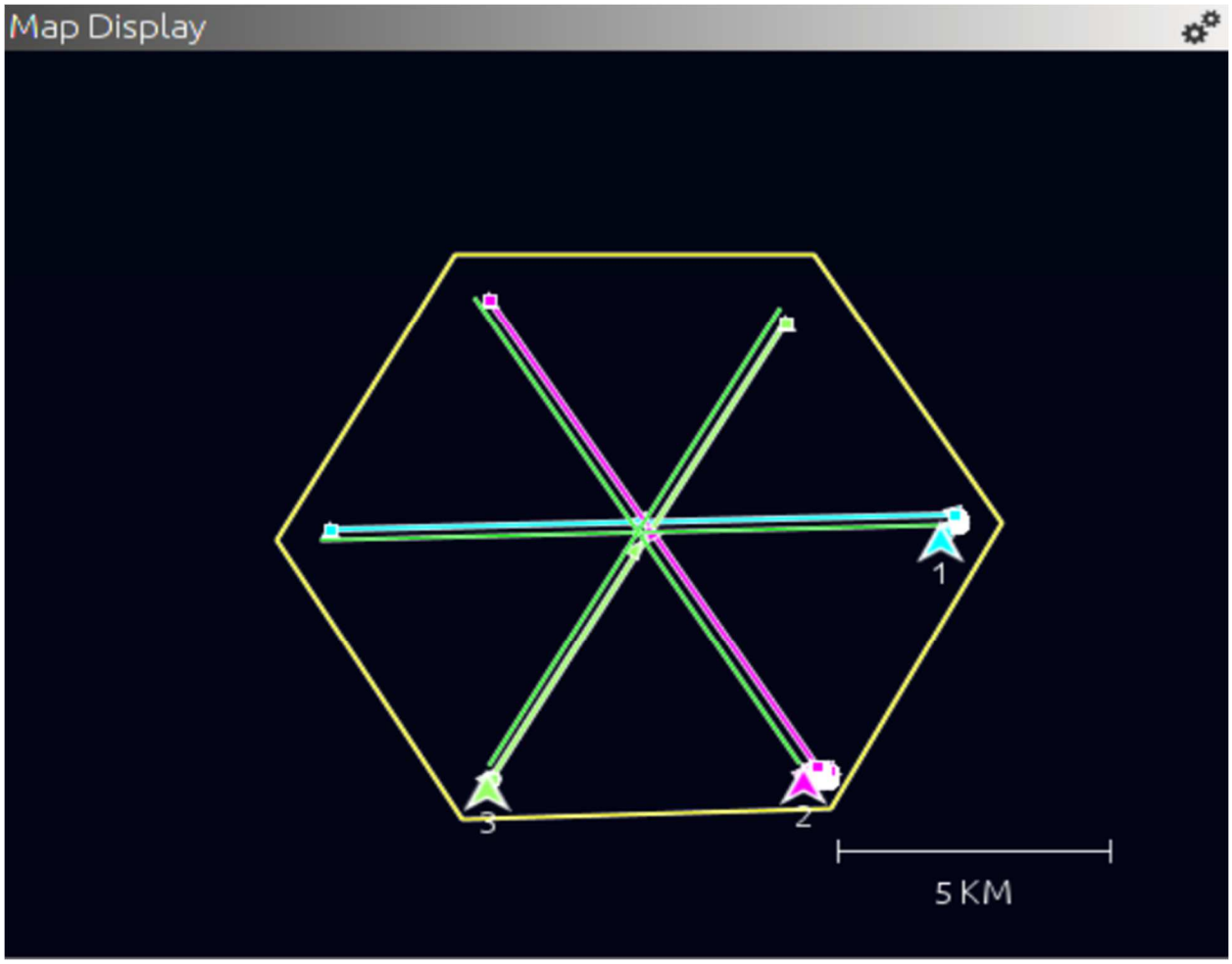

Figure 3.2A: A case where the ICAROUS guarantees look like they will break down, but do not.

Beginning of simulation

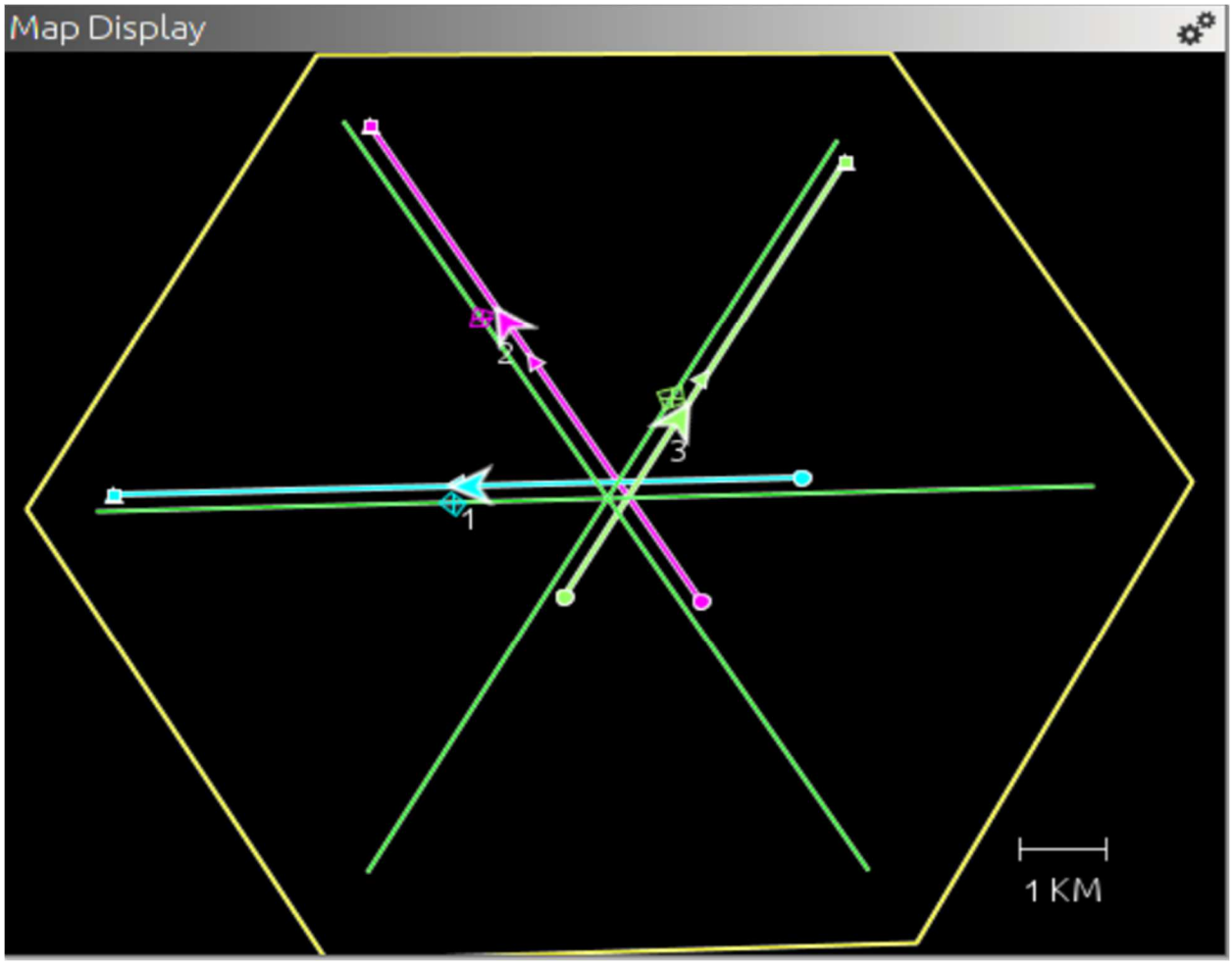

Figure 3.2B: Example of a case where the ICAROUS guarantees look like they will break down, but do not. After avoidance maneuvers

CRATOUS Architecture

In this section, we examine the overall architecture of CRATOUS, including an example walkthrough for a simple mission from the perspective of UxAS, then ICAROUS, and finally an overview of what goes on in both systems throughout the mission. However, before that, we must first examine the CRATOUS messaging system. When ICS sends a message to the CRATOUS app in ICAROUS or vice versa, a very specific format is used. A five-letter identifier of the message type is given, and then each field of the message is specified. The format is label-value, with the label in capital letters and the value given immediately after. A comma is inserted at the

end of the value. The fields can be in any order, but the message identifier must occupy the first exactly five characters of the message. An endline character marks the end of a message. An example message that CRATOUS might send is:

WAYPTtotal1,index1,lat51.433647,long-0.215516,alt100,speed100,\n

This message describes the first waypoint of a mission that has only one waypoint. A waypoint is one entry in an arbitrarily-long list of points that the vehicle should navigate to. The waypoint is located at the given latitude and longitude, 100 meters above the ground, and UAVs should be moving as close to 100 meters per second as possible when moving toward the waypoint from the starting position or previous waypoint. There are several other types of messages in CRATOUS. Their identifiers and purposes are given in the table below.

Table 3.1: CRATOUS message types

Tag	Purpose
GEOFN	Geofence specification
POSTN	Owned ship's position, velocity, and heading
ATTUD	Owned ship's attitude (pitch, roll, and yaw)
OBJCT	Other vehicles, and non-vehicle objects
COMND	Command messages
WPRCH	Indicates a given waypoint was reached
WPREQ	Request for a path between two waypoints

UxAS with CRATOUS

As we have discussed previously, UxAS is a modular system for optimal path planning between sets of autonomous vehicles on collaborative missions. However, it has nearly zero flight time and no formal verification. Each module of UxAS interacts with the others through a messaging interface, and messages can be sent to an individual module, to a group of modules, or, most commonly, as a general announcement that any subscribed module can receive. When UxAS runs with the IcarousCommunicationService, or ICS, module included, this architecture is unchanged except for the addition of the extra module. This can be seen in Figure X.X: UxAS Architecture with CRATOUS. When ICS starts up, it will look inside its configuration file to see how many instances of ICAROUS are to connect to UxAS and wait for these connections. Since

UxAS initializes modules sequentially, this causes the entire program to wait on the ICAROUS connections to be authenticated.

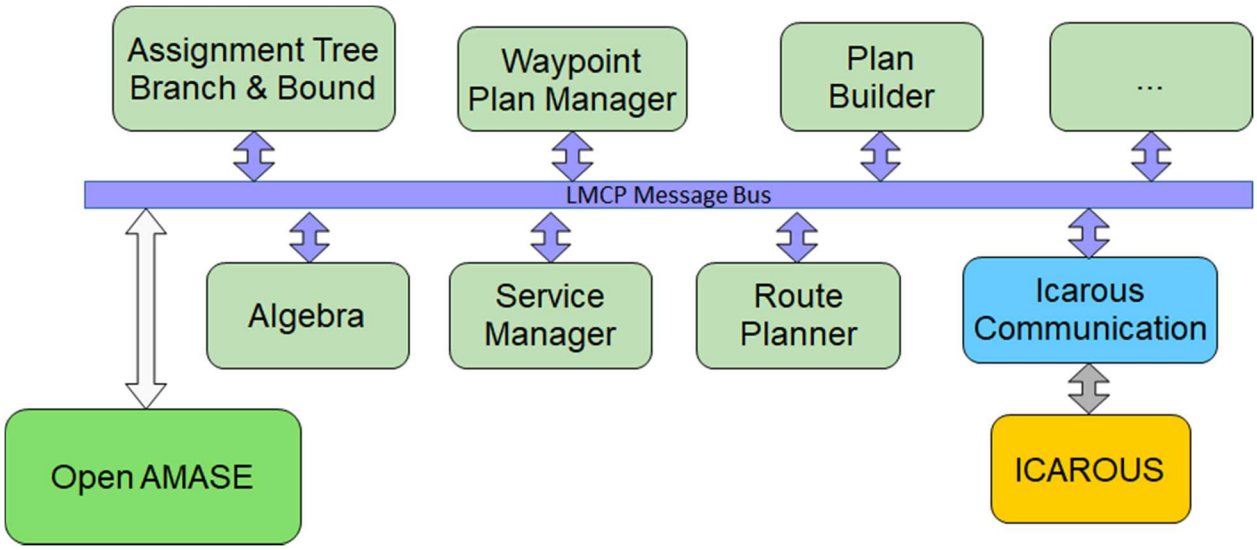

Figure 3.3: UxAS Architecture with CRATOUS

When all instances of ICAROUS have connected to UxAS successfully, then UxAS finishes its initialization of the other modules and proceeds with normal processing as outlined in Chapter 2 Section A until the time comes for simulation, unless the use of an ICAROUS path planner for pre-mission path planning is requested. In this case, a message is sent out in UxAS to disable the UxAS path planning modules and an instance of ICAROUS is chosen to handle pre-mission path planning. Further discussion of this scenario is in the next subsection of this chapter. UxAS will also send waypoints for all geofence vertices to ICAROUS so that ICAROUS will not send any UAV through a no-fly zone.

When the assignment tree has been parsed and AMASE begins to simulate the mission, the ICAROUS instances are each assigned to a UAV in order of ascending UAV ID number. The waypoints for each UAV that has an ICAROUS instance assigned to it are sent to the relevant ICAROUS instance, and the mission is ready to begin. There may be fewer ICAROUS instances than UAVs in the scenario, but there may not be more. The UAVs that are assigned an ICAROUS

instance will have collision avoidance enabled, and the others will not. The simplest case is where only a single UAV out of multiple has ICAROUS attached, so we will go through this example first, then move on to missions where there are more ICAROUS instances enabled. As discussed in Chapter 2 Section B, ICAROUS is an ownership-based system, meaning that it is only capable of guiding a single vehicle, its "owned ship." Thus, a single ICAROUS instance cannot guide multiple UAVs and a separate instance of ICAROUS must be created for each UAV where collision avoidance is desired.

In a mission where a single UAV is handled by ICAROUS and at least one other is not, only the UAV with ICAROUS has any guarantees about mission conformance, completion, or safety. The others should still complete the mission without violating the constraints, but these UAVs are handled exclusively by UxAS. ICAROUS is fed a constant "heartbeat" stream of information, carried in several messages that are sent from UxAS to ICAROUS via ICS and CRATOUS twice per second. One message is for the owned ship's position, speed, heading, and current waypoint; one message per non-owned vehicle is sent with the same information about that vehicle except for current waypoint; one message is for the owned ship's attitude (pitch, roll, and yaw); and the final message is an optional command message, which can cause ICAROUS to reset, forget geofences, forget waypoints, allow UxAS to take over live path planning, and so on. When ICAROUS predicts a collision or well-clear violation, it will send a message to UxAS through the same CRATOUS-ICS bridge and UxAS will allow it to take control of that UAV. Further information about what messages ICAROUS sends to control a UAV during a collision avoidance maneuver is in subsection B.2. UxAS will also update ICAROUS whenever a mission waypoint is reached, so that ICAROUS can plot a return-to-path maneuver for the proper waypoint. While ICAROUS is in control of a UAV, UxAS will keep track of any task the UAV was in the middle

of performing, if any, and judge whether or not the UAV was kept close enough to on-course to continue the task from where ICAROUS hands back control, or if the UAV needs to go back to some point during the avoidance maneuver and restart the task from there. The exact distance that determines how far "too far" is, is given in a configuration file at startup.

Command messages are usually only sent from UxAS to ICAROUS, and usually only in the case of a nonfatal error, such as ICAROUS sending a message indicating that it didn't receive part of a geofence. The only other cases where a command message might be sent are when UxAS encounters a fatal internal error, on the start of a mission, or immediately before and after an avoidance maneuver. In the last case, ICAROUS will send a message notifying UxAS that it would like to take or relieve control of its UAV.

In a scenario with multiple UAVs being controlled by ICAROUS, the multiple instances of ICAROUS do not communicate with each other. Collaborative avoidance maneuvers and intersection control are subjects beyond the scope of this work. Instead, each UAV will treat each other UAV as an intruder with an unknown path, and attempt to avoid these intruders. If both of the UAVs involved in a predicted collision are controlled by ICAROUS, they will both attempt to avoid the other, which is suboptimal. So long as avoidance is possible, it will still happen, but there may exist a temporary seeming deadlock of a given intersection. An example of such a seeming deadlock is given in Figures 3.1, 3.2A, and 3.2B, above.

ICAROUS with CRATOUS

As we have discussed previously in Chapter 2 Section B, ICAROUS is a modular system with partial formal verification for planning and executing missions for a single UAV while avoiding obstacles and other vehicles that may or may not be moving. It has thousands of hours of flight time on physical aircraft, and brings some guarantees of mission completion, restraint

compliance, and aircraft safety. However, unlike UxAS, it does not have an easy-to-use simulation environment, and it cannot command multiple vehicles simultaneously. When ICAROUS starts up, it will initialize its various modules, allocating memory for them and messages sent to them. The messaging system in ICAROUS uses the same basic "subscribe and send" system as UxAS, but there is no way to send a message to a specific module from another module, and no concept of groups of modules. Instead, all messages are pushed to the common message bus, and any modules that have subscribed to that type of message will receive it. Figure 3.5, ICAROUS Architecture with CRATOUS, is an illustration of this.

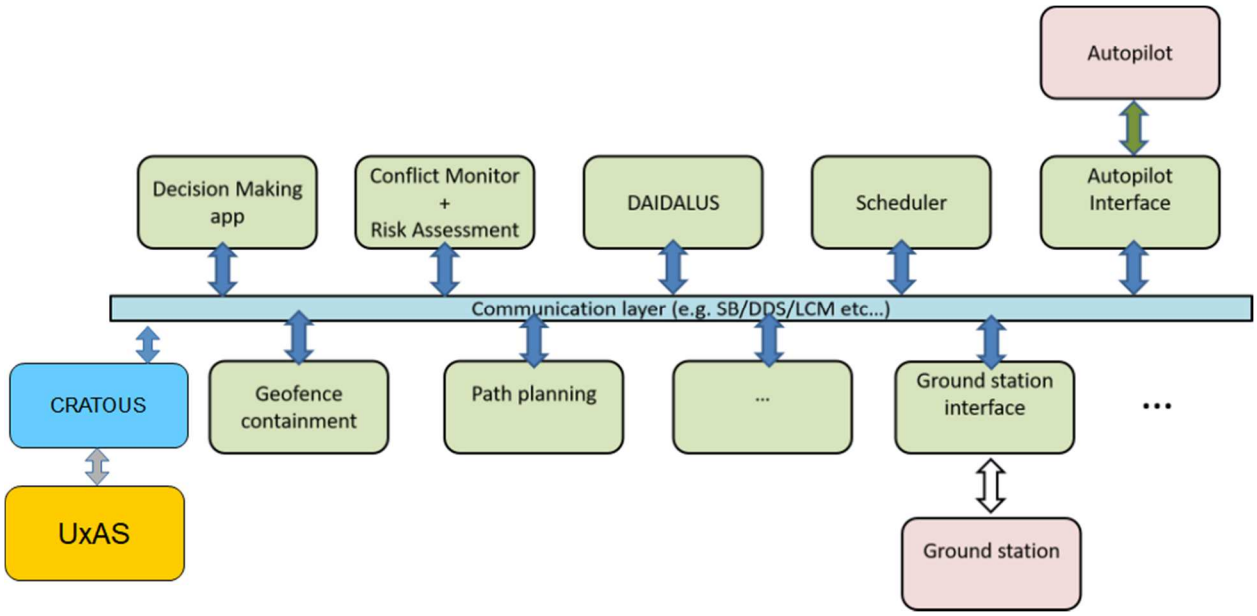

Figure 3.4: ICAROUS Architecture with CRATOUS

When ICAROUS starts up, one of the modules it initializes is named CRATOUS. CRATOUS will attempt to connect to UxAS using a given IP address, and if it cannot then it will exit and cause ICAROUS to crash. If the user has specified that an ICAROUS path planner should be used for pre-mission path planning, then UxAS will send a command message to CRATOUS, prompting it to request routes that are fed in by UxAS and send back the results. When UxAS has

sent the geofence information and the waypoints for the vehicle owned by ICAROUS, the mission is ready to begin.

When the simulation is started, the two-hertz heartbeat messages regarding vehicle positions to ICAROUS from UxAS will begin and ICAROUS will monitor for possible collisions or well-clear violations. Other than this heartbeat message, CRATOUS now acts as a listener to ICAROUS. If and when a violation is detected, ICAROUS will begin to attempt avoidance and CRATOUS will begin sending messages back to UxAS. CRATOUS translates the messages ICAROUS creates regarding desired headings into the string-based format discussed above and sends the translated messages to ICS. ICS will then translate the strings into LMCP messages for UxAS and AMASE, and broadcast these commands. CRATOUS will continue to send these control messages until such time as ICAROUS decides that the UAV may return to course, and at this point CRATOUS will return to just sending out the heartbeat message to ICAROUS.

Overview

CRATOUS as a full integrated system performs all of the tasks of UxAS and ICAROUS, with communication between them to control the simulation environment. The architecture of this complete system is shown below. The Ardupilot and Ground Station Interface modules are crossed out because UxAS fills this functionality for ICAROUS.

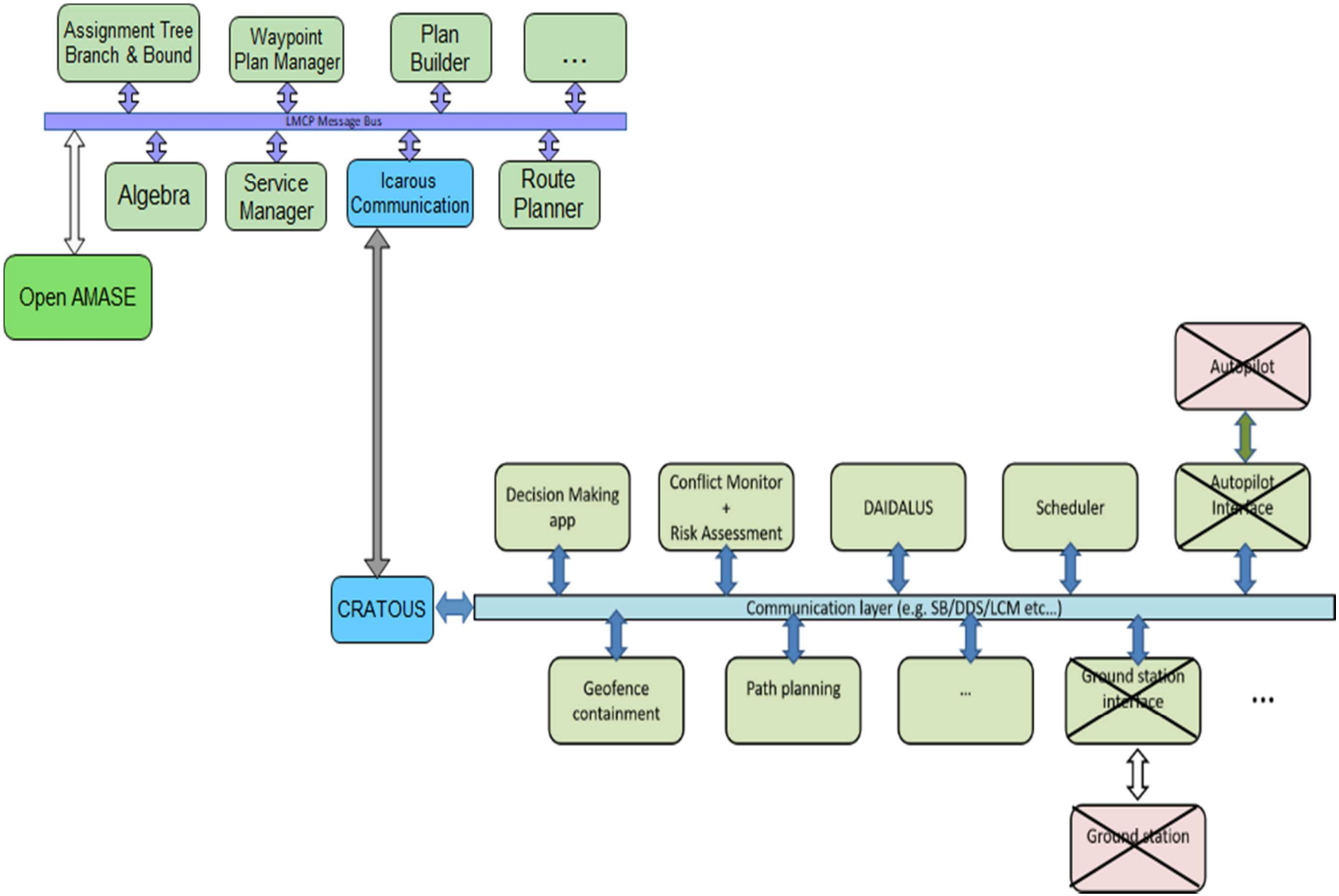

Figure 3.5: UxAS and ICAROUS Architecture with CRATOUS

CRATOUS was made with three major goals in mind: high extensibility, low programmer time to write code, and the ability to link UxAS and ICAROUS without affecting their internal processing. High extensibility was the primary concern, and a string-based messaging protocol is extremely extensible. The other approach that we examined that would have also been extensible was putting a software wrapper around ICAROUS and then putting ICAROUS directly inside of UxAS. This approach was abandoned when it became obvious that the secondary goals, writability and modularity, would suffer and that the Internet-based approach that was taken could fulfill all three goals simultaneously.

CHAPTER 4

RESULTS AND CONCLUSION

As we covered previously, the creation of CRATOUS fulfilled all major goals. Namely, ICAROUS and UxAS are each able to access the full functionality of the other program during their normal runtime, and while the current system has some limitations, it does allow for generally safe operation of collaborative unmanned vehicles. We have given ICAROUS the ability to perform collaborative missions, use the AMASE simulation environment, and do path planning through the UxAS path planners rather than its own. We have given UxAS access to the formally-verified collision avoidance system of ICAROUS, as well as the ability to use the ICAROUS path planners, some of which are formally verified.

Limitations

CRATOUS currently requires roughly one dozen software parameters to be manually tuned to each scenario. These parameters could be determined automatically, but time did not permit the implementation of this determination. If any of these manual parameters is set incorrectly, the vehicles may show undefined behavior. Most of the time, this undefined behavior results in a well-clear violation or nonconformance to mission constraints, but it may be as extreme as vehicles flying in a straight line, regardless of anything else, indefinitely.

One other, major, limitation that has been mostly ignored to this point is that CRATOUS requires an Internet connection to function. In practice, this may cause CRATOUS to be unusable - for instance, in a hostile environment, smoke or purposeful jamming may disconnect a given vehicle from the network and cause CRATOUS to stop functioning as intended. In summary, CRATOUS only works when its manually-set parameters are tune correctly, and when it has a dependable Internet connection.

Future Work

The immediate next step in CRATOUS is to determine the above parameters automatically, allowing for a much more reliable system and automated testing. The next step is to begin verification and flight testing, and to improve the generality of CRATOUS so that it could work with different protocols, or even as a server rather than as a client. This would allow for much broader research into how ICAROUS can function with other programs, and would allow the developers of ICAROUS to connect it to programs that provide desired functionality as opposed to implementing it themselves.

A more long-term goal for CRATOUS is to allow it to use more direct communication methods than an Internet connection - a direct connection between vehicles that cannot communicate with the base station. This would not remove the requirement for an Internet connection in order for vehicles to connect to UxAS and function collaboratively as a whole, but it would allow vehicles to collaborate in a limited fashion when communication with the controlling base station is cut off.

REFERENCES

[1] Balachandran, S., Muñoz, C., & Consiglio, M. C. (2017). Implicitly coordinated detect and avoid capability for safe autonomous operation of small UAS. In 17th AIAA Aviation Technology, Integration, and Operations Conference (p. 4484).

[2] Balachandran, S., Muñoz, C. A., Consiglio, M. C., Feliú, M. A., & Patel, A. V. (2018, September). Independent Configurable Architecture for Reliable Operation of Unmanned Systems with Distributed Onboard Services. In 2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC) (pp. 1-6). IEEE.

[3] Balachandran, S., Narkawicz, A., Munoz, C., & Consiglio, M. (2018). A Geofence Violation Prevention Mechanism for Small UAS.

[4] Balachandran, S., Narkawicz, A., Muñoz, C., & Consiglio, M. (2017). A path planning algorithm to enable well-clear low altitude UAS operation beyond visual line of sight. In Twelfth USA/Europe Air Traffic Management Research and Development Seminar (ATM2017).

[5] Beard, R. W., Kingston, D., Quigley, M., Snyder, D., Christiansen, R., Johnson, W., ... & Goodrich, M. (2005). Autonomous vehicle technologies for small fixed-wing UAVs. Journal of Aerospace Computing, Information, and Communication, 2(1), 92-108.

[6] Beard, R. W., McLain, T. W., Nelson, D. B., Kingston, D., & Johanson, D. (2006). Decentralized cooperative aerial surveillance using fixed-wing miniature UAVs. Proceedings of the IEEE, 94(7), 1306-1324.

[7] Casbeer, D. W., Kingston, D. B., Beard, R. W., & McLain, T. W. (2006). Cooperative forest fire surveillance using a team of small unmanned air vehicles. International Journal of Systems Science, 37(6), 351-360.

[8] Cao, Y., Muse, J., Casbeer, D., & Kingston, D. (2013, December). Circumnavigation of an unknown target using UAVs with range and range rate measurements. In 52nd IEEE Conference on Decision and Control (pp. 3617-3622). IEEE.

[9] Consiglio, M., Muñoz, C., Hagen, G., Narkawicz, A., & Balachandran, S. (2016, September). ICAROUS: Integrated configurable algorithms for reliable operations of unmanned systems. In 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC) (pp. 1-5). IEEE.

[10] Dowek, G., Munoz, C., & Geser, A. (2001). Tactical conflict detection and resolution in a 3-D airspace (No. ICASE-2001-7). INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING HAMPTON VA.

[11] Feliú, M. A., Rocha, C., & Balachandran, S. (2017, July). Verification-driven development of ICAROUS based on automatic reachability analysis: a preliminary case study. In Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software (pp. 94-97). ACM.

[12] Karaman, S., Rasmussen, S., Kingston, D., & Frazzoli, E. (2009, June). Specification and planning of uav missions: a process algebra approach. In 2009 American Control Conference (pp. 1442-1447). IEEE.

[13] Kingston, D. B. (2007). Decentralized control of multiple UAVs for perimeter and target surveillance.

[14] Kingston, D. B. (2004). Implementation issues of real-time trajectory generation on small uavs.

[15] Kingston, D., & Beard, R. (2004, September). Real-time attitude and position estimation for small UAVs using low-cost sensors. In AIAA 3rd" Unmanned Unlimited" Technical Conference, Workshop and Exhibit (p. 6488).

[16] Kingston, D. B., & Schumacher, C. J. (2005, June). Time-dependent cooperative assignment. In Proceedings of the 2005, American Control Conference, 2005. (pp. 4084-4089). IEEE.

[17] Kingston, D., Rasmussen, S., & Humphrey, L. (2016, September). Automated UAV tasks for search and surveillance. In 2016 IEEE Conference on Control Applications (CCA) (pp. 1-8). IEEE.

[18] Moore, A. J., Schubert, M., Rymer, N., Balachandran, S., Consiglio, M., Munoz, C., ... & Schneider, P. (2017). UAV Inspection of Electrical Transmission Infrastructure with Path Conformance Autonomy and Lidar-based Geofences. NASA Report on UTM Reference Mission Flights at Southern Company Flights November 2016.

[19] Muñoz, C., Narkawicz, A., Hagen, G., Upchurch, J., Dutle, A., Consiglio, M., & Chamberlain, J. (2015, September). DAIDALUS: detect and avoid alerting logic for unmanned systems. In 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC) (pp. 5A1-1). IEEE.

[20] Narkawicz, A., Munoz, C. A., & Dutle, A. M. (2017). The MINERVA software development process.

[21] Rasmussen, S., Kingston, D., & Humphrey, L. (2018, June). A brief introduction to unmanned systems autonomy services (uxas). In 2018 International Conference on Unmanned Aircraft Systems (ICUAS) (pp. 257-268). IEEE.

[22] Sabo, C., Kingston, D., & Cohen, K. (2014). A formulation and heuristic approach to task allocation and routing of uavs under limited communication. Unmanned Systems, 2(01), 1-17.

[23] Sujit, P. B., Kingston, D., & Beard, R. (2007, December). Cooperative forest fire monitoring using multiple UAVs. In 2007 46th IEEE Conference on Decision and Control (pp. 4875-4880). IEEE.

VITA

Graduate School
Southern Illinois University

Winston Smith

wssmit95@gmail.com

Southern Illinois University Carbondale
Bachelor of Science, Computer Science, May 2018

Thesis Paper Title:
Collaborative UAV Surveillance

Major Professor: Dr. Henry Hexmoor