Robust Multi-agent Mission Operations Management For Heterogeneous Aerial And Space-based Robotics

by

Sami T. Mian

B.S.E. in Computer Systems Engineering, Arizona State University, 2016

B.S. in Computational Mathematics, Arizona State University, 2016

M.S. in Computer Engineering, Arizona State University, 2018

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2020

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Sami T. Mian

It was defended on

September 9, 2020

and approved by

Zhi-Hong Mao, Ph.D., Professor
Department of Electrical and Computer Engineering

Ahmed Dallal, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Alan D. George, Ph.D., Professor and Chair
Department of Electrical and Computer Engineering

Brandon Grainger, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

James Martin II, Ph.D., Professor and U.S. Steel Dean
Department of Civil and Environmental Engineering

Dissertation Director: Zhi-Hong Mao, Ph.D., Professor
Department of Electrical and Computer Engineering

Robust Multi-agent Mission Operations Management For Heterogeneous Aerial And Space-based Robotics

Sami T. Mian, PhD

University of Pittsburgh, 2020

Unmanned Aerial Vehicles (UAVs) have become cheaper and more technologically advanced over the past five years, with uses in academia, industry, and government projects. A promising application of UAV technology is multi-agent swarms: using multiple drones to accomplish a group of tasks cooperatively. Currently, drone swarms have been used to aid search and rescue efforts, increase security systems, and produce awe-inspiring art installations. They are even used by groups like NASA to simulate advanced distributed systems, such as satellite constellations. However, as the hardware and sensing capabilities of UAVs increases, so too does the complexity of managing these swarms. Most swarm deployment systems are homogeneous: platforms are identical and assigned one overarching task. There is no allowance for specializations in platform capabilities. This undercuts the benefits of distributed computing: it is operationally restrictive to use custom, specialized UAVs in a large swarm, as the platform management is problematic and impractical. This leads to heavy implementation restrictions for novel sensors in swarms, due to high costs of integration and deployment.

The research of this dissertation creates a fleet mission management system that allows for multiple UAVs to cooperate and accomplish a multitude of mission types. The system employs new control law methods and flight software standards to coordinate the autonomous flight of drones in restrictive environments, while also optimizing for scarce resources like power, communication capabilities, and payload specialties. Furthermore, this research creates a system that allows for the inclusion and use of diverse, unique platforms and sensor payloads without considerable system modifications. The fleet management system is built on top of NASA's cFS architecture and includes features from open-source software. A novel optimal control technique, called heterogeneous decentralized receding horizon control is developed and tuning using a UAV simulator. Lastly, exploratory research has been conducted

on integrating dynamic vision sensors with UAV flight controllers, to test the integration of novel sensors with this fleet management system. The resulting system is readily deployable and can allow groups like NASA to mimic dynamic, diverse UAV swarms with relative ease.

Table of Contents

List of Tables

List of Figures

Acknowledgement

Over the past three years, I have had the pleasure and enjoyment of pursuing a doctorate of philosophy degree in Electrical and Computer Engineering at the University of Pittsburgh. It has been a difficult and trying journey, but it is finally coming to an end. I am greatly thankful to all of the people who have helped and encouraged me every step along the way.

First and foremost, I would like to express my gratitude and indebtedness to my main advisor, Dr. Zhi-Hong Mao. His support and encouragement have been a great source of motivation over the past few years. His constant guidance and mentorship have helped shape me into a great researcher. He has lead by example in the classroom, and helped me develop my skills as a teacher and instructor. And his willingness to allow me to explore my interests in research and topics has allowed me to find the niche of work that I find really engaging and personally satisfying. Lastly, his enthusiasm and invaluable advice have helped shape my outlook not only in research, but life in general. It has truly been an honor to work with Dr. Mao.

I would also like to thank Dr. Alan George for his support over the past couple of years. Dr. George is the reason I first came to Pitt, and he is the person who introduced me to my advisor, Dr. Mao. Without Dr. George, I would not have come to the University of Pittsburgh. He welcomed me into his research lab, helped fund my research over the years, and has exposed me to some unique opportunities I would not have found anywhere else. It has been an absolute pleasure working in the NSF Center for Space, High-performance, and Resilient Computing (SHREC), and has greatly impacted my graduate studies and research direction.

I am also grateful to Dr. Ahmed Dallal and Dr. Brandon Grainger, for their help and support throughout my time at Pitt. Dr. Dallal was one of the first professors I had at the university, and it was very rewarding studying under him. Dr. Grainger has been very open about research and realistic expectations, and has provided impactful support and advice for a few key decisions. These two professors are also past students of my advisor, so I was able to learn from them some of the expectations and requirements that I would face during my time at Pitt.

I would also like to extend my gratitude to Dr. James Martin, for his support and encouragement over the past couple years. I met Dr. Martin during my first week at the university, and we quickly formed a friendship of sorts. We both care deeply about education and the impact of universities in a community, and we have had many hour long conversations about the future of education and its role in society. It has been amazing listening to his thoughts on education and innovation, and I have been lucky enough to work with Dr. Martin on some of his projects. His background in systems engineering has also helped me to consider my work through a new viewpoint and approach problems differently, in a way that few people in my field do.

During my time at Pitt, I have had the opportunity to collaborate with and learn from some amazing colleagues and friends. I want to specifically thank my research collaborators: Tyler Garrett, John Hill IV, and Nik Salvatore; working with each of them was a wonderful opportunity and lead to some phenomenal research output. I would also like to extend this thanks to all of my fellow grad students in the Interactive Controls Lab and the NSF SHREC Center, for their support, ideas, and feedback. I want to thank my senior graduate student friends, Mohamed Bayoumy, Amr Mahmoud and Alvaro Cardoza, who have provided advice and guidance during the challenging parts of the program. Also, I would like to extend my warmest thanks to Sandy Weisberg and Nicole Klan of the Department of Electrical and Computer Engineering; their support and efforts have helped make my time at Pitt seamless and stress-free.

I also need to thank my many acquaintances from my previous adventures prior to Pitt, especially during my time at ASU. From my professors and classmates, to my work colleagues and industry mentors, I am very grateful to those who have helped motivate me and push me to finish my education. Specifically, I want to thank my dear friends Lauren, Alex, and Mark, whose shared love for education and constant support have made the process of leaving AZ and starting fresh a little more manageable.

Last, but certainly not least, I want to extend my deepest thanks and appreciate to my wife, my parents, and my sister and brother, for their patience, love, and support. They have always been a constant source of help and motivation, and I would not have come this far without their involvement every step of the way.

1.0 Introduction

Advances in embedded hardware, sensor technologies, and general robotics have allowed the development and usage of unmanned aerial vehicles (UAVs) to skyrocket the past five years. Unmanned aerial vehicles, also known as drones or small Unmanned Aerial Systems (sUAS), are used for a variety of applications. They have seen widespread adoption in sensing tasks, especially those in unknown or dangerous environments, mainly due to their customizable payloads and high maneuverability. These applications include search and rescue relief efforts, package delivery, and security perimeter searches. More recently, cities around the United States have been looking at FAA-sponsored pilot programs to integrate aerial systems into their smart city environments.

Recent excitement has been directing research to focus on the deployment and use of multiple UAV agents (swarms) to accomplish a joint effort. There are two primary limitations to deploying UAV swarms: First, many operational environments and missions require in-situ adaptability to mission events. While this is the primary motivation for deploying a swarm, the swarm's overall ability to handle situations is limited by the payload (sensors, power, tools, etc.) each UAV carries. It is not ideal to deploy hundreds or even thousands of identical UAVs as there is a threshold to scalability. Second, technology advances rapidly with business and cultural needs, thus swarms must evolve with the technology. However the costs incurred by comprehensive fleet upgrades are substantial. In both cases, the ability to utilize a heterogeneous swarm is paramount to increasing the overall flexibility and cost of the technology. These pose new challenges to control architectures and fleet management system implementations, as applying a uniform architecture/scheme to the swarm is no longer applicable. For this dissertation research, we look at two specific use cases to frame the motivation and driving force of this work. The first has to do with addressing the increasing difficulty of ambiguous search and sensing scenarios where UAV swarms are applied, and the second has to do with using UAVs as testbeds for space robotics.

The first motivation for this research is the lack of efficiency in using UAV swarms for large-scale search and sensing tasks. The most common use case is search and rescue of

individuals in disaster-stricken areas, but also applied to battlefield surveillance and generic data collection tasks. As sensor technology has improved drastically with the increase in computing capabilities and MEMS technology, so too are the capabilities of UAV sensor systems. Custom payloads can now be designed to work for specific tasks, such as using low-powered thermal cameras for tracking the spread of heat (fire), or hi-res RADAR cameras for imaging large areas in poor lighting conditions. Although these custom payloads can greatly increase the efficiency and success metrics of a mission, they are rarely used in multi-agent systems. This is because currently almost all deployable systems that manage drone swarms are built for homogenous systems: they only work with one type of platform. The few systems that do use manage multiple platforms are horribly inefficient, and still limited in their support scope. In order to use different UAV platforms efficiently, an end user will need to set up multiple systems, one for each platform specialty.

Another focus of improvement for this work is the efficient use of all agents within a swarm for polymorphic tasks. For any given scenario requiring multiple platform-types, there will be multiple different mission tasks. In an ideal scenario, each task will be structured so that it can achieved by one specific type of platform; however, this is not realistic. In a more conceivable deployment setting, each unique platform will be capable of accomplishing a subset of mission tasks during deployment. However, the set of mission capabilities are not mutually exclusive, there will be some overlap. In this situation, multiple UAVs will be able to accomplish the same mission, possibly with varying levels of speed/accuracy. In order to be as efficient as possible, a heterogenous swarm controller needs to take advantage of these overlaps. Platforms need to be assigned missions based on a variety of parameters, instead of just payload-task fit. This introduces more complexity into an optimal control scheme, which is not usually used to manage diverse possibilities.

In addition to this, there is also to consider situation of adjacent/secondary mission parameters. In our scenario, we look at wide-area sensing for disaster relief efforts. In addition to specific mission goals, where a certain UAV payload is needed for specific sensing task, there is also a secondary set of missions, focused on generic data collection and situational awareness of the surrounding environment. This secondary task can be accomplished by any UAV platforms, as long as they are in the right vicinity. However, this adds another

additional layer of complexity to the task. The goal of this dissertation research is to simplify the process for dealing with these scenarios.

The second motivation for this work is the application of swarm robotics to space-based robotics. One of the newest research uses of multiple UAVs by groups such as NASA and Boeing, is to use them as test platforms for satellites. Because the cost of launch for prototype satellites is extremely expensive and prototyping space objects in lab environments can get quite resource-intensive, these large companies have wanted to use other types of robots to simulate satellites and their sensor payloads. UAVs are the ideal platforms because they travel in 3-dimensional space, use similar flight hardware and software systems, and face similar resource challenges as satellites. It is also much easier to model satellite constellations using UAV swarms, as opposed to other robot platforms. Specifically, UAVs are used to model the "cube" satellite, a small U-class spacecraft. These cubesats are usually made up of minimal computing power (sometimes automotive grade components), and cost a fraction of the price to develop compared to regular satellites; the cost for deployment/delivery to LEO is also greatly diminished when compared to traditional satellites. This lower launch cost, as well as the easy replicability and production speeds, is why cubesats are more widely used today for small-scale space computing needs. For companies using cubesats, the return on investment is much greater compared to traditional satellites. Nowadays, instead of sending out one giant satellite to complete a task, a company can send up 5 to 10 smaller cubesats to accomplish that same task. The benefits to sending up a swarm or group of cube satellites is two-fold: the size of the payload for sending up the entire group is smaller than a satellite, and it can be split up among several different launches, meaning the overall launch cost can be diminished. And the risk of damage is also minimal, as a small piece of space debris would only take out one of these satellites and still leave 9 functioning units, whereas the same size of space debris can take out a full traditional satellite, rendering it unusable for its missions/operations. One of the nice things about using multiple platforms (both on the ground and in space) is that you can always add more functionality by sending up an additional platform. As many space companies are moving towards using cubesats for their space missions, there is now a need for platforms for research, development, and testing for cubesats in preparation. As mentioned above, a

new emerging method for simulating cubesats is by using unmanned aerial vehicles, or UAVs. However, there currently does not exist any fleet or mission management software that allows for control of UAV systems in a way similar to how it's done for cubesat constellations. One of the goals of this dissertation research is to overcome this limitation by creating a fleet mission management system that allows for robust, intelligent control of UAV platforms similar to what is needed for constellation arrays of satellites.

Just as with all robots, there are several factors that limit the capabilities of UAVs. The first common limitation that is shared by all robots is power resources: a drone is limited by its power source. For a regular robot, if it runs out of power, it is stuck unmoving in its original location; it can be retrieved by human operators and restored to full working condition. This is not an option with UAVs, which operate by flying through the air; if a UAV runs out of power, it falls out of the sky and sustains damage. Even worse, a UAV falling out of the sky can hit a human or vehicles, causing damage and potential injuries. Another crucial limiting factor for robot deployments is the platform's communication abilities. Unless a robot is fully autonomous, there needs to be some sort of communication protocol that allows for them to send and receive information from the controller; this includes commands, decisions, and safety-critical management. Once again, if a ground robot receives no commands or loses a communication connection, it will stay in place until it regains a connection. A UAV will do the same, until it runs out of battery and once again falls from the sky. Both of these problems scale in complexity and importance when you more from one robot to many robot platforms working in conjunction.

When it comes to dealing with fleets or swarms of robots, all of these problems magnify in importance. When developing mission parameters for multiple robots, the planning system needs to take into account the state, current operating status, and available resources for each platform, which becomes a huge optimization problem. This becomes even more difficult to solve when all of your robots are not in the same starting position. For example, managing package delivery with drones spread out across an entire city, you need to maximize their flying potential with regards to power storage. Drones can only fly so far on their battery reserves, and it is not optimal to have a drone fly from one end of town to another when there are other platforms closer that can accomplish the same task. As such, its important

to have a management system that understands all of the platforms, available resources, and possible anomalies when creating mission plans.

The main contribution of this research is the development of a fleet management system that allows for coordination of multiple, heterogeneous robot platforms in constrained environments. This work focuses on UAV systems, and will be adapted to allow for scientific implementation and research on space-based robots. This project accomplishes three main tasks. The first task is to develop a multi-drone management system using NASA's core flight software system; this system is a robust Mission management system that operates in real time and allows for flight readiness for Mission critical platforms. The second task is to develop a new control law for the robot coordination and path planning using optimal control methods. This control technique pulls data from robot resource availabilities, mission health records, and available sensor data to optimize the best mission plans for the group of robots. Part of this algorithm is to develop a distributed strategy for maximizing utilization of a swarm of drones, making sure each drone platform it used to the extent of its abilities (mainly for specialized platforms). The third task focuses on further developing sensor fusion techniques to use with these UAV platforms, and will specifically focus on integrating neuromorphic camera technologies onboard UAV systems. Novel sensor fusion techniques will be developed to utilize the event camera data for faster perception loops and flight controller responses. The results of this work have been published in peer-reviewed venues [1, 2, 3].

1.1 Research Questions

- How to develop a robust fleet management system for UAV missions that is flight tested, safety critical-compliant, and adaptive to diverse platforms and environments.
- How to use optimal control to allow for multi-agent, disjointed resource-based planning in highly restrictive operating conditions.
- How to integrate brand new sensor data into existing control/management frameworks for platform perception and planning.

1.2 Dissertation Organization

This dissertation is broken up into several major parts: an extended background on all the related technologies, the research conducted on each of the three major components of this work, and a detailed discussion and analysis that concludes all of the results from this work. These parts are broken up into six chapters, consisting of the following organization:

- Chapter 2 contains an overview of each of the systems or technology stacks that are involved in this research; common or widely researched topics will have shorter explanations with citations from numerous foundation papers, while newer/less known systems/topics will have in depth sections. This includes an extensive overview of the NASA Core Flight System architecture, as this is a minimally documented system.
- Chapter 3 covers the work accomplished with regards to fleet management for UAV swarms. Specifically, this chapter details the development of the UAV swarm management system designed using both Robot Operating System (ROS) and NASA's cFS architecture. This section covers the work started in conjunction with NASA Langley's Formal Methods team, and the extensions completed at the University of Pittsburgh.
- Chapter 4 details the research conducted on utilizing optimal control techniques for high-level mission planning for UAV swarms. This includes missing and path planning for high-dimension swarms, and covers both homogenous and heterogeneous swarm systems.
- Chapter 5 is focused on the integration of the neuromorphic camera system with traditional UAV flight controllers. This includes the reinforcement learning work completed with both raw event camera data, as well as the research on developing a novel non-linear representation for event camera bytestreams.
- Chapter 6 contains an overview of the work noted in the previous three chapters, and summarizes the contributions from each stage of research. This chapter also connects together all of the aforementioned research with the underlying goals and accomplishments of this dissertation, and discusses the next step in research for each of these topics.

2.0 Background

2.1 Overview of Flight Software

Flight software (abbreviated as FSW) is software that "flies" which means that it is onboard a UAV or spacecraft. IT can be a part of the spacecraft Bus (fundamental systems like mechanical structure or attitude control system), Flight Software is hosted within the flight electronics CPU, and starts immediately when the platform receives power. FSW acts as the brains of the mission. FSW is classified as embedded software, which is computer software written to control machines or devices that are specialized hardware, i.e. have time and memory constraints, and are designed for specific purposes. FSW must be handled in real time, as it allows it to be deterministic, reliable, and guarantees a response within required time constraints. FSW is also mission critical; it must keep your platform safe through unexpected anomaly and must be able to act autonomously.

The aerospace domain is unique with regards to computing. Missions require the use of specialized, radiation tolerant hardware; Consumer-grade Off The Shelf (COTS) solutions simply do not exist. This hardware is required to work in a fixed, constrained environment; the two major constraints are processor speed and available memory and storage capacity. For example, The Lunar Reconnaissance Orbiter uses a 166 MHz processor with 2MB of memory available for mission software. The software used in space-based devices is also highly complex. Software needs to be highly reliable and fault tolerant, must support autonomous operations and in-mission maintenance, and must be able to complete computations for high speed science operations. All of these challenges greatly increase the cost of satellite software.

In the past, flight software was rarely reused. There was no product line; heritage missions were used as starting points for new software. There was no version control or management of major software changes; all changes were made at the discretion of the developer. Since the hardware was very restrictive and specialized, any upgrades in hardware or operating systems would require vast changes throughout the flight software to make it

compatible. And lastly, there was poor documentation and no uniform testing procedures. Because of all of the above problems, NASA spent ludicrous amounts on software development, and never learned from past lessons. This had to change, so Goddard Space Flight Center developed a solution: An adaptive set of flight software systems that could be used as a majority of the software basis for most future missions; specifically, an all-purpose software solution to the spacecraft bus, flight software and avionics software. This project resulted in the development of NASA's Core Flight System [4].

2.2 Overview of CFS

NASA's Core Flight System (CFS) is a set of mission independent, modular software services and applications, running in a unique operating environment. It has three key aspects to its architecture: a dynamic runtime environment, layer software systems, and a component-based design [5]. CFS has a layered architecture that allows it to support a variety of hardware platforms, which is ideal for flight software due to the ever changing hardware requirements/availability. CFS provides a standardized application programmer interface (API) which makes it easier to interface with other external software; this also makes developing for CFS easier for those not as familiar with the system. CFS is built around supporting multiple software applications, including those used for core flight software. Individual applications can be added or removed at run-time, which allows for easier system integration, testing, and software maintenance. On top of this, CFS contains platform and missions specific configuration parametres, which are used to tailor the software apps to work optimally on specific hardware platforms or for specific mission focuses. In terms of extended development, CFS supports software development in three ways: through onboard flight software, through a desktop-based development environment, and through simulators. These allow for software applications that run on CFS to be developed using a variety of requirements and techniques.

The CFS software framework is built on top of years of successful GSFC flight software systems, all of this have been flight testing and gone through rigorous analysis and valida-

tion. It addresses the ever-present challenge of continually increased software development costs, mainly due to the perpetually changes and improvements to hardware used in flight missions. The size and complexity of flight software has and will continue to grow dramatically as the years progress, and CFS provides a way to manage the complexity that is intrinsic to this continuous growth and modularization. In order for the system to be independance and adaptable over a wide array of use cases, the baseline architecture contains a configurable set of code and base requirements. These parameters allow CFS to be specifically customized and tailored for each type of operating environment, from simulations to barebone CPUs. The CFS architecture greatly simplifies the development process of slight software by establishing an underlying infrastructure and hosting a runtime environment for development of project/mission specific applications; The cFS architecture also simplifies the flight software maintenance process by providing the ability to change software components during development or in flight without having to restart or reboot the system [5]. At a high level, CFS has a complete development environment that allows for development for multiple processor processor architectures, operating systems, and missions software needs. One of the greatest development capabilities of CFS is the ability to run the same software on varying platforms: to run and test applications on a development machine and then deploy that same software to an embedded system without any changes it extraordinary. Even before specialized hardware is made available, mission software can be prototyped, tested, and iterated on very early in the project while remaining hardware agnostic. The baseline missions and algorithms can be created, and then the flight software can be tuned to work on whatever hardware platform is chosen or provided. CFS development tools allows for managing different builds for individual processors, and allows for different applications to be loaded to each processor (depending on hardware support). The architecture also contains an automated testing suite which can be used to perform a large battery of basic tests, both unit and integration, on the software without hardware requirements. The architecture also contains other resources for developers, including design documentation, development standards, user guides, and testing procedures with expected results. According to NASA engineers, CFS and its underlying architecture has helped mission development processes in the following ways: It has reduced the overall time to create and deploy flight software

that is high quality, it has simplified the software engineering process for flight software, it has provided a platform for advanced prototyping and development, it has encouraged formalized software lifecycle adherence (including version control) and it has reduced costs and project scheduling immensely.

2.2.1 CFS Technical Details

Core Flight System is built using a layered architecture; this means that each layer is hidden from the other layers with regards to implementation and technology details. Each of these internal layers are modular and can be changed without affecting the components or operations of any other layers. This allows for individual aspects of CFS to be selectively improved or modified to meet growing specifications. This is what also allows CFS to be hardware agnostic, as well as removing limitations and reliance on specific operating systems and accompanying middleware. Figure 1 shows a diagram of the CFS System Architecture.

CFS runs using PROM Boot Software and utilizes a Real Time Operating System (RTOS). In addition to this, the software is mainly comprised of an OS Abstraction Layer (OSAL), Platform Support Package (PSP), core Flight Executive (cFE) Core, CFS Libraries, and several standard CFS Applications.

2.2.2 PROM Boot Software

This software is the local software that conducts the early initialization and bootstraps the operating system used to run CFS. It is stored in local PROM on the hardware platform. This software includes an EEPROM/Flash loader, which allows for installation on most if not all hardware platforms. It has been designed to be as simple as possible to minimize changes to the PROM. For CFS, the main two commonly used boot softwares are the RAD750 BAE SUROM and the LEON3 uBoot software; Goddard also created their own custom image based on open source code available from Coldfire.

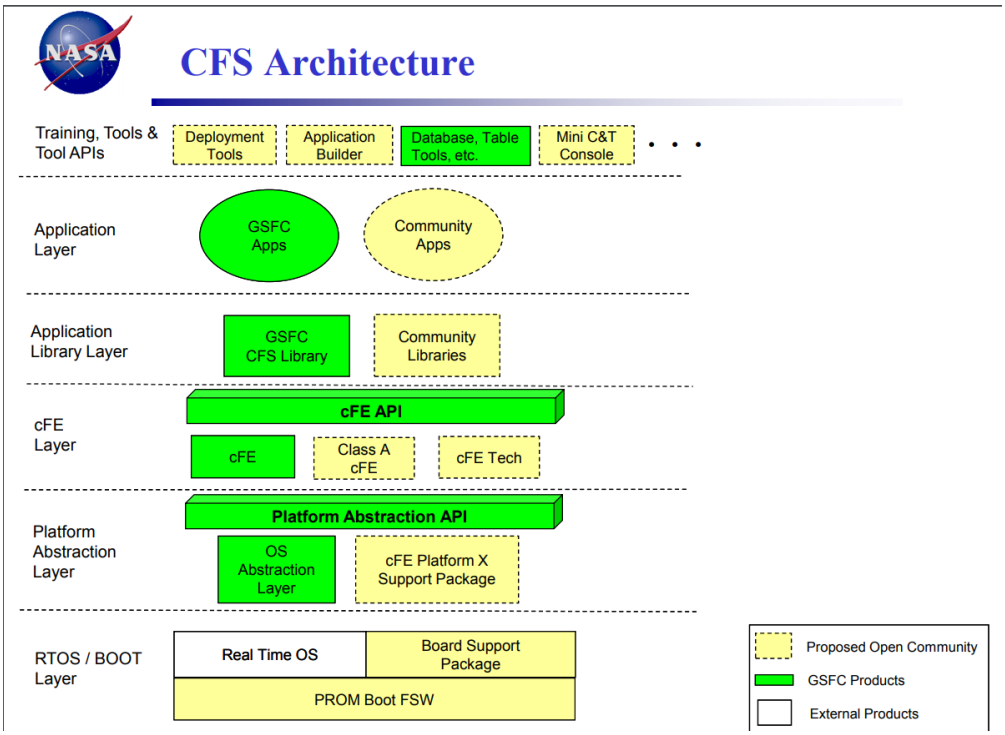


Figure 1: cFS architecture diagram [4]

2.2.3 Real Time Operating System

The real time operating system used to run the lowest levels of CFS is in charge of hardware-level operations. This includes handing message queues and semaphores/mutexes (to help alleviate problems with lockup or runtime conditions). The RTS is also in charge of handing preemptive priority based multitasking. It also has the standard support for interrupt and exception handling, as well as basic shell and file system support. The current RTOS systems that meet the requirements for CFS are VxWorks and RTEMS.

2.2.4 Platform Abstraction Layer OSAL

The reason that CFS is able to run on numerous different operating systems without any modifications is because of the platform abstraction layer. This software, called the operating system abstraction layer (OSAL) is a basic library that separates the flight software from the

RTOS, handing all of the calls in between using a library system built to be compatible with most operating systems. This way, the rest of CFS does not need to work about specifying system-level calls, like a basic print function. Figure 2 is a diagram of the implementation of OSAL.

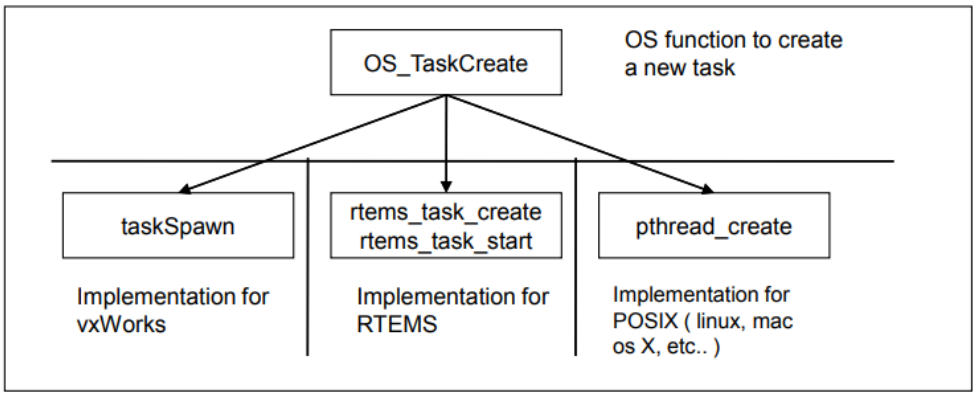


Figure 2: OSAL implementation system diagram [4]

2.2.5 Platform Abstraction Layer PSP

Another of the critical pieces of CFS is the platform abstraction layer. The purpose of this layer of software is to contain all of the software needed to adapt the core parts of CFS and CFE to any type of high level operating system and processor card. This piece of software is called a Platform Support Package (PSP), and also includes all of the tool chains that are required for make rules to build and deploy code of the given system. Functions of the PSP includes simple startup code, commands to read, write, and protect the onboard memory (EEPROM, RAND), processor reset functions, and timer functions. Currently the widely used PSPs are Linux x86, Power PC's VxWorks, and RTEMS.

2.2.6 Application Library Layer

CFS also uses a separate application library layer, which manages all of the shared libraries used by different applications. This includes standard C libraries as well as mission-specific libraries, such as those used for scientific computing work. The most important

use of this layer is for libraries that are crucial for working with onboard hardware, such as FPGA boards or specialized sensors (cameras, lasers, etc.). The main CFS library provides common utility functions including string manipulation, CRC computation, and file path/name verification.

2.2.7 Core Flight Executive: CFE

Core Flight Executive (CFE) is a framework of flight software services and an operating environment that are independent of any mission. It is the core of the CFS system, and is available for use with CFS out of the box; its applications are considered the "default" applications for CFS. CFE is where the platform and mission configuration parameters are stored; these are the parameters that are modified to tailor the program for a specific hardware platform and mission. CFE is composed of five core services: Executive Services, Event Services, Software Bus Service, Table Service, and Time Services.

The executive service managed the basic operations of the CFS system. This includes all of the aspects of system startup: powering on the system and processor, starting and resetting applications, spawning child tasks, keeping records of all tasks running, and managing which applications will be built/run for the current mission. In addition to this, executive services maintains a system log which records all execution, reset, and exception information; this is used for performance analysis and debugging. This service also provides the ability to use shared libraries, support different device drivers, and restored critical data after any processor resets.

The event services provides the basic interface between applications used to send and receive asynchronous messages. These messages include informational, debugging, and error messages, and each is timestamped for accurate message tracking. Event services also supports filtering, and has its own optional event log.

The software bus service serves as a portable massage service in between all of the running applications. It provides a subscription/publisher messaging interface similar to that used in Robot Operating System (ROS), which allows all running instances to send and receive information over open data pipelines. This system has built-in error detection for message

transfers, and records and provides statistics packets and relevant routing information for messages when necessary.

The table services manages all of the interaction between applications in CFS and the table data structure image. A Table is a specific group of parameters used by an application. The Table Registry is populated at runtime by loading in information from the table.h file; this eliminates cross-coupling of applications with the flight executive during compile time. This service also allows for synchronous table updates with relevant applications in order to verify data/parameter integrity. Tables can be shared by applications, although more core applications have their own reserved tables. The time service is used to provide time and timestamp data to all of CFS systems. Specifically, this service provides spacecraft time (based on mission start time), a time correlation factor, and time corrections to all services. Each application is able to use the time service to query the current time, which is used in turn for a number of calculations (trajectory and odometry, timestamps, efficiency analysis, etc.). The service can also distribute a wakeup time message, usually set to 1 Hz rate.

2.3 ICAROUS

NASA's Independent Configurable Architecture for Reliable Operations of Unmanned Systems, or ICAROUS for short, is a software architecture that allows for the fast and verified development of applications for unmanned aircrafts [6]. It allows for the robust integration of cFE and other core algorithms along with mission-specific application modules that is both safety-focused and flight assured. Currently, most UAV autopilot systems have limited computing capabilities, since all that is needed is basic navigation and control functions. However, with the increasing use of UAVs for all kinds of applications, there is more demand for high level decision making capabilities in autopilot software; this includes sense and avoid algorithms, conforming to airspace regulations (avoiding restricted airspace) and working with more advanced sensors. The ICAROUS software system allows for the higher level decision making capabilities to be available to work alongside traditional autopilot systems. One primary use of ICARUS is to easily enable beyond visual line of sight (BVLOS) missions

for UAV platforms without relying on human monitoring or intervention [6]. ICAROUS depends on an external autopilot system, and is designed to run on a companion computer that interfaces with the autopilot computer to modify flight plans/parameters to allow for safe flight.

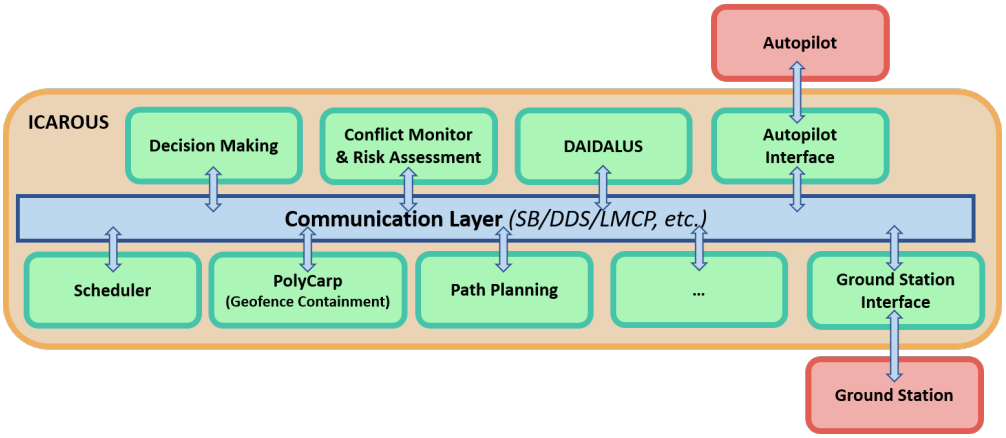


Figure 3: ICAROUS architecture [1] ©2020 IEEE

ICAROUS is comprised of several software modules that all interface using a software bus. Using modular apps allows for rapid development and incorporation of new features. The software bus is a data pipeline that is accessible to all applications running on the system, and uses the traditional publish/subscribe paradigm for message communication [6]. ICAROUS is implemented using NASA's CFS middleware, which allows it to operate as a distributed architecture. This modular, distributed approach allows ICAROUS to interface with a majority of available autopilot software, requiring only an interface program to be designed. The default build of ICAROUS is configured to work with MAVLink and ArduPilot, an open source autopilot flight stack that is used by the majority of hobbyists and academic researchers.

The major features in ICAROUS have been compressed into libraries for easy integration. They include DAIDALUS, PolyCarp, and PLEXIL. DAIDALUS is a detect and avoid library that allows for the monitoring of well clear violations against other vehicles [7]. It also provides contingency actions for horizontal and vertical speeds, as well as altitude, to avoid losing contact with a platform [7]. PolyCarp is a library that verifies in a point is inside or

outside a polygon. This functionality is used to compare flight plans with existing geofences, and detects any potential violations (both for white lists and black lists of points) based on vehicle velocity and flightplan. ICAROUS uses this information to avoid flying in restricted areas and modifies flight plans based on potential violations. ICAROUS is able to replant the flight plans using a combination of search algorithms and existing waypoints to avoid static or dynamic obstacles. PLEXIL, which stands for Plan Execution Interchange LAnguage, is a language that is used to represent plans for use by autonomous agents. ICAROUS uses PLEXIL to schedule different tasks for the platform based on priority and resource requirements.

2.4 Control Systems and Planning Background

2.4.1 2D Path Planning: Motion Planning

The problem of defining a path from a starting point to an ending point has been explored in great detail in numerous different studies and papers. There are two main types of solutions: graph search methods and optimal control methods. For most common graph search methods, the search space is discretized into a connected graph made up of nodes; each node represents a step in motion or an adjacent location. These nodes are connected together using a measurable metric (i.e. distance, reachability) to create a tree structure. The tree can then be searched for multiple paths from one point (node) to another point (node). Several popular search techniques used for graphs include A*, Dijkstra, and depth/breadth first algorithms. Although these techniques have been rigorously testing and are used in numerous settings (like A* used in video game AI), there are several drawbacks of the graph-based approach. Graph-based methods are not designed for scalable dimensionality; as the dimension of the search space increases, the complexity of the graph and all solutions increases exponentially.

In more recent literature, sampling-based methods have been introduced in order to help overcome the increased complexity of graph-based solutions. Two of the most common are the Probabilistic Road Maps (PRMS) [8] and Rapidly Exploring Random Trees (RRT) [9]. The main idea behind PRMs is to take random samples of the entire configuration space of a robot, test to see if they are in the robot's possible operating space, and use a basic pathplanner to try and attempt these configurations to other nearby configuration spaces. This first part is used to create a large graph configuration of all the possible motion plans for the robot. After this, a simple graph-search method such as A* is used to find the shortest path (or a path that meets other requirements based on the cost function). The RRT algorithm works by drawing a random tree of possible paths which stems from the starting configuration; random points are picked a certain distance away from the existing (or starting) points and then those are connected to the closest points using a line (if feasible). This technique is similar to a monte carlo approach, and is biased towards creating the largest voronoi regions in the available search space [9]. Numerous variations on this technique have been developed that allow for optimizing different attributes; the RRT* algorithm is optimized to give extremely straight paths. There have been numerous examples of successful motion plan generation using PRMs and RRT, although most of those have been confined to 2D movement (X and Y place). A couple of prominent self-driving car companies actually utilize the RRT* planning model for their main navigation algorithms [10, 11].

In optimal control methods used for motion planning, the problem objective tends to be represented as a cost function; the objective can include parameters such as finding the shortest path, using the least amount of power, or reaching the destination in the fastest time possible. A parametric representation for the path or the control inputs to the system is chosen. Then, an optimization algorithm solves for the parameters that minimize the cost function subject to various constraints on the environment and dynamics. One of the drawbacks of using the optimal control route is that these equations can be hard to solve analytically; the complexity is further increased by the presence of nonlinear constants in the functions. Therefore, usually numerical methods are used to solve these methods, although sometimes minimization/maximization functions can get stuck in local minimum/maximum solutions instead of global solutions.

2.4.2 Geofencing

Geofencing is another method in which hard boundaries are set for path planning for airborne vehicles. A geofence is a virtual perimeter for a geographic area that is usually composed of GPS waypoints. With respect to UAVs, a geofence is usually used to designate either a flight-approved zone, or to show obstacles or areas where UAVs are not allowed to operate. Geofencing strategies and integration into path plans for UAVS have been explored in several papers [12, 13]. The main focus of work thus far has been around creating algorithms that compare flight plans with geofences in order to detect any conflicts, and determining corrective action. Usually basic corrective action entails stopping flight or landing/returning home, although more advanced capabilities include maneuvering around an obstacle or completely re planning based on the available data. In order to allow for fully autonomous operations, any kind of replanning or conflict resolution needs to be able to consider mission parameters and dynamically create new flight plans that avoid geofences while still accomplishing all mission parameters.

2.4.3 Obstacle Avoidance

Sense and avoid (SAA) systems are implemented on most types of autonomous systems, from self-driving cars to space platforms currently orbiting the Earth [14]. As the operational environment grows in complexity, the sense and avoid capabilities of platforms must also improve in both detection quality and system response. This has largely been aided through the use of newer sensing technologies and advanced control architectures.

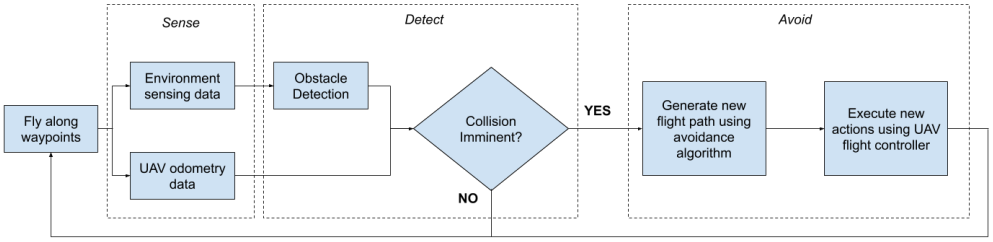


Figure 4: Traditional UAV sense & avoid system flow [1] ©2020 IEEE

Figure 4 shows the basic diagram of how a sense and avoid system operates. One or several sensors are responsible for detecting and identifying any obstacles or possible hazards along the vehicle's intended trajectory or surrounding environment. The data from these sensors is then processed using sensor fusion techniques, and a collision avoidance approach is devised. Some examples of techniques include graph search algorithms, nonlinear model predictive control, field potential modeling, and convolutional neural networks [15]. For UAV platforms, the main challenge for implementing collision avoidance is the usability of sensors; due to the size, weight, and power constraints of an aerial platform, sensors for the UAV payload are limited [16]. These sensors can be split into two types: cooperative and non-cooperative sensors. Cooperative sensors are those which are placed on not just on the target UAV, but also on potential obstacles, such as other UAVs. These sensors consistently broadcast their flight information, so that all nearby platforms are aware of intended flight plans. Two popular examples of this are the Traffic Alert Collision Avoidance System (TCAS) used on early aircraft, and the more popular Automatic Dependent Surveillance and broadcasting (ADS-B) which is now being implemented on many UAV platforms [17, 18, 19].

Non-cooperative sensors, which are sensors that only exist on the target platform, are more common on modern UAVs since they can meet the payload constraints, and do not rely on external sensors. Extensive work has been done with standard RGB-D cameras [20, 21], LIDAR systems [14], RADAR sensors [22], acoustic and sonar sensors [23], and infrared sensors [16, 24]. There has also been some cursory work on using dynamic vision sensors for obstacle tracking and avoidance on-board UAVs, but this area of research is still in its infancy.

2.4.4 Optimal Control for Robust Planning

Most literature related to optimal control techniques for path planning is focused on industrial robotics. Usually, the environments for these robots are manufacturing plants, where the robots are multi-DoF arms that have highly constrained motion and operating parameters. However, we can take a lot of inspiration from industrial robots when looking at UAV planning due to the similarity of these constraints: Limited motion options, power

constraints, maintenance considerations, and collision avoidance. Focusing on integrating power consumption and battery resource monitoring is especially important for UAVs, as battery is the leading constraint when it comes to mission planning.

Obstacle avoidance capabilities is also extremely important with regards to path planning, especially when looking at robot swarms. There is extensive literature in the industrial robotics space that details approaches to solve this [25]. A strategy to use spline interpolation to optimize trajectory and dynamic behavior is detailed in [26]. The paper by [27] focuses on obstacle avoidance using dynamic optimization through interpolation techniques [28]. With regards to controls, a time optimal control strategy is studied in [29], and the use of different performance indexes for the optimal control technique (in an obstacle free environment) is studied in [30]. Paper [31] has an interesting use of dynamic programming to use for point to point collision free motion planning, which can be good for planning around direct path obstacles in 3 dimension space. There is an approach to use penalty function to avoid collisions in a workspace in papers [32, 33, 34, 35]. And lastly, in paper [36] there is an approach to optimal robot motion planning with obstacle considerations using the non classical formulation of Pntryagin's maximum principle.

2.4.5 Receding Horizon Control (RHC) for Swarm Behavior

Another type of control system that uses optimization problems as its basis is the receding horizon control method (RHC). In receding horizon control, an optimization function is used to determine the next action for a platform/plant for a projected time frame that ends with a "receding horizon;" the next time step during which an action must be determined [37]. In order to develop an RHC, first the objective, constraint, prediction method,and horizon are specified; each of these tends to have a "natural choice" that is suggested by the application in question [37]. Unlike a PID controller, an RHC rarely needs to be tuned to achieve high levels of performance; these systems are also able to run in implementation at extremely high speeds, usually around the kilohertz sampling rate. Due to the high speeds at which these controllers can run, they are ideal for real time systems, as well as for use in rapid simulation settings (i.e. Monte Carlo) for iterative development. The real-time nature and

minimal tuning requirements of these controllers has led to the use of RHC for mission critical systems, where robust and speedy decision making is critical. By determining the proper cost functions and objectives, an RHC can perform near the top limits of a controller. RHC is used in numerous robotics fields, with particular interest in swarm robotics over the past few years [38, 39].

Receding horizon control has also been proven to act as a phenomenal control strategy for decentralized swarm systems [40]. This method, known as decentralized receding horizon control (D-RHC), focuses on using predictive information about nearby swarm members and collective goals to determine and optimize the appropriate cost-based objective functions. This prediction is done at each timestep, in order to try and predict the actions of each swarm robot at the next horizon time. The predictions are usually based on current state information of the surrounding robots (trajectory, velocity, etc.) but can also be derived from any model information that is available about the controllers on the neighboring robots (including their optimization functions). After the predictions are made, the system developed a constrained cost minimization problem to model the state of the swarm; the system solved this problem in order to determine the optimal solution for the next time horizon. The solution always involves modifying a variable that is controllable for the specific platform, such as velocity.

Although a receding horizon controller does not require much tuning in order to run optimally, it is still hard to design a decentralized version due to the need to combine a set of differing goals, costs, and constrains all to form an efficient optimization function [41]. Furthermore, in a complex swarm, it is even more difficult to determine the most efficient combinations of costs that can remain static due to the dynamic nature of the operating environment. Communication issues, network delays, noisy sensors readings, and other environmental factors can render a pre-set optimization function useless and ineffective. In the paper by Henderson et al, it is proposed that these different variables can be easily quantified and set using a meta-learning process: cost adaptation. This method uses a set of user-generated cost and constraint functions, as well as other external heuristics, to generate the optimization objective. A heuristic-based search algorithm is used to determine the best way to generate the objective based on a large list of pre-set cost and constraint functions.

This search is guided by the swarm's interactions with the environment, as well as data collected on neighboring platforms. Through the use of simulated annealing, we can allow the adaptive agents to perform tasks while optimizing for efficiency and safety [42]. The goal of this approach is to allow for the simple combining of goal-based objectives while still allowing for the system to adapt to all types of unforeseen obstacles presented by the operating environment. Since the learned objectives are based on user-entered goals and cost functions, the outcomes of the controller are also human-readable, which allows for insight into the decision making process and the ability to modify to match desired decision bases.

2.5 Distributed Systems and Computing Background

Distributed Satellite systems (DSS) are one of the types of robot "swarms" where autonomy is becoming more prevalent. A good synopsis of the uses of autonomy is found in a paper by Araguz et al [43]. Some of the issues facing satellites that are hoped to be solved with satellites include communication delays/timing, mission robustness and failure tolerance, and reduced control windows due to ground station availability. The goal of most research in this area is to enable these platforms with more forms of autonomy; the autonomy is not just another feature of these platforms, but will be the key method in which future platforms manage operation dynamics and complex systems [44].

Mission Planning Systems (MPS) are a major part of distributed satellite systems, as its important to coordinate task execution across multiple platforms. Currently, the main approaches used for MPS are heuristic-based approaches. A prime example is the EO-1 approach, which has a preset schedule that implements replanning whenever there are anomalies detected in operating conditions. Another interesting approach to MPS is described by Beauemet et al [45], using a reactive algorithm that created new, instantaneous plans based on preset system operating rules/parameters. For each task, a priority, start time, duration, and required resources is assigned, and then an optimization algorithm determines the best order of execution. This optimization can be performed by one of many math-based scheduling solutions, underneath the topic of Earth Observation Satellite (EOS) scheduling;

the most common techniques are described in detail in [46] for single satellites and [47] for multi-satellite systems. The goal of these approaches it to provide optimal schedules, assuming no deviations; this is only valid for missions where anomalies are not expected. Other pieces of relevant literature include the general field of distributed computing, where there has been a plethora of research focused on interdependent task scheduling, with a good list of work cited in [48]. There is also a large body of relevant work for terrestrial autonomous systems, including Autonomous Underwater Vehicles (AUVs) in [49], and UAVs, in [50] which is the focus of this paper. With regards to these last two, these systems do not have the same constraints as space-based systems, and are instead usually constrained by other factors (such as path planning).

One of the milestone papers in the area of autonomous spacecraft software contains an overview of the software used for Earth Observing One (EO-1) [51]. EO-1 is a satellite that flies in Low Earth Orbit, which uses cameras and other related sensors to detect notable events autonomously and then react to them. For its planning/replanning system, EO-1 uses the Continuous Activity Scheduling Planning Execution and Replanning (CASPER) software developed by NASA's Jet Propulsion Laboratory (JPL) [52]. The mission plans created by CASPER are used as input into the Spacecraft Command Language (SCL) executive, which uses CARPER's plan to perform low-level actions to achieve the predetermined actions. CASPER is able to create plans that span anywhere from a few minutes of planning to far into the future; this is due to the strict requirements and availability of processing power available on some spacecrafts [52]. The further out the actions to be completed, the higher level and more abstract the instructions would be. This scalability and adaptive schedule planning is important for UAVS, satellites, and any other platforms with limited storage and computing power, as detailed plans can take up a large amount of memory and can become obsolete in an instance due to anomalies or conflicting requirements.

In the paper [53], a new technique is discussed that allows for a more standardized approach to autonomy, using a modular architecture, that builds on top of the capabilities used for EO-1; this method is hardware and software independent, and can be translated to other platforms. This method, called the Autonomous Mission Manager (AMM), uses a Service-Oriented Architecture (SOA) which allows software to be divided up into multiple processes

that are able to communicate with each other using a predefined communication bus [44]. This method of process partitioning is similar to those used in ROS and NASA's CFS system. AMM also uses CASPER for its task planning, but also integrates a software executive platform called the Cooperative Intelligent Real-Time Control Architecture (CIRCA); AMM also has its own middleware messaging system that it uses to communicate in between its software modules, called the Adaptive, Scalable, Portable Infrastructure for Responsive Engineering (ASPIRE) framework [44]. Like with CFS, the goal of the AMM architecture is to standardize the interface and transfer of data between the modular components, which allows for easier software development and more robust systems.

2.6 Robot Operating System (ROS) Background

Robot Operating System (ROS) is an open-source, meta-operating system used for robots. It sits on top of a basic POSIX operating system, and allows for most of the hardware-software interfacing to give a robot functionality. ROS provides services such as hardware abstraction, low-level device control, package management, and other commonly-used OS functionality. ROS also has its own messaging service, which allows all processes and programs to send and receive data in one of many open channels of communication. ROS also provides all of the basic tools and libraries needed to write, build, and run code on a hardware platform. ROS is not a realtime framework, but it can be integrated with realtime code.

The ROS runtime system can be pictures as a "graph" of processes that are connected in a peer-to-peer network; usually these processes are connected using Rostopics, which is ROS's communication infrastructure. ROS processes can be distributed among multiple platforms, as long as they are all on the communication platform. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server [54, 55].

ROS is designed to be a thin system that can work with other robot software framework. It does not contain its own main() function, meaning it can be used by or with other software packages. ROS is also designed to be used with standard, ROS-agnostic libraries. ROS has supported implementations in C++ and Python, meaning that development is not tied to a specific language (language agnostic); in fact, ROS has support for cross-compilers that allow for use of multiple languages [54, 55]. ROS also has built-in testing and integration frameworks, under the ROStest library, which allows for quick unit and full integration testing of all written software. And lastly, ROS is built to be scalable, allowing it to be used on large runtime systems and with large development processes. ROS has been designed and developed to allow a few key features to be usable for robot platforms: Distributed computing, software modularity, and rapid testing.

2.6.1 Distributed Computation

Modern robot systems rely on software that utilizes many different processes. These processes can all be run on a single platform, or can be distributed across several different mediums (robots, servers, cloud systems, etc). With current best practices, it's best to divide software systems into many smaller processes as opposed to a few large behemoth processes. This is known as complexity via composition [54, 55]. Whether a single robot is running all of these processes, or there are multiple agents collaborating on a task, there is a common need in order to reach success: all of these processes need to be able to communicate to each other, for coordination and data sharing. ROS provides multiple mechanisms to facilitate this level of communication. In fact, the messaging system is so advanced that there is theoretically no limit to the number of processes that can be running simultaneously and sharing/using information amongst themselves.

2.6.2 Software Modularity

Robotics as a field has seen constant development and progress over the past 50 years. As a result, there is a large repository of robust, vetted algorithms that are used for common tasks such as mapping, navigation, computer vision, and sensor fusion. These can serve as very useful resources if they don't need to be implemented from scratch. With ROS, many of these existing algorithms can be used in a plug-and-play fashion. First of all, ROS has a group of standard packages that provide stable, fully integrated versions of many of these important algorithms. Two of the most well known are GMapping, which is used for building maps, and AMCL, which is a particle filter used for accurate 3D localization. These packages can be added to any ROS project because they use ROS's default messaging system. This messaging system has become the de facto standard for robotics software communication, and as such, any package using this system can interface with ROS. This also means that ROS can interface with most existing and new hardware, as it's designed around this communication protocol. Because of the uniform adoption of this system, there is also less need to write integration code to make existing code work with new packages; since they all use the same messaging system all that needs to be "modified" if the message processing scripts and algorithmic input/output [54, 55].

2.6.3 Rapid Testing

When it comes to developing robotics software, testing is one of the most challenging aspects. This is primarily because testing can be time-consuming and error-filled. Hardware is notoriously hard to debug, and sometimes the proper platforms are not always available, or have their own limitations. Integration tests tend to require a fully functioning system for even basic tests, which can be hard to have available during distributed development. Using ROS provides a couple of alternatives to help solve this problem. The first way is by using a simulator. Because of the way ROS is designed, a good ROS system can separate the low-level control system from the higher-level software system; the low-level system tends to handle interfacing with the hardware while the high level is more of the processing, navigation, and decision making systems the robot uses. If the low level systems can be placed in a separate

process, they can be replaced with a hardware simulation environment. This would allow for the higher level functions to be testing using "ideal" hardware data and a fully- controllable environment. ROS is also able to record all sensor/process data using its ROSBag software; this can always be used to simulate the robot platform by resending all of the data down the communication bus to the higher-level software. In both cases, simulator and rosbag, the change is seamless. Both options are able to create and provide identical interfaces to those of the physical robot, which means the software does not need to be modified in any way. As such, the system has no idea it is not being testing on live hardware. This in turn makes teasing much easier and faster with ROS.

2.7 Overview on Swarm Robotics

Using multiple robots for a joint task has been a large focus of robotics research over the past 30 years. As such, there have been numerous systems and studies carried out, many offering unique hardware and software architectures for multi-robot systems. There are two types of multi-agent systems: homogeneous and heterogeneous. As the names suggest, homogenous robot swarms are comprised of many units of the exact same platform, whereas heterogeneous swarms are comprised of different varieties/types of robot platforms; sometimes, heterogeneous swarms include robots of different operating environments (i.e. drones and underwater systems, or different types of space-capable platforms). Most research that has been performed in depth on these systems from a controls perspective has been performed on homogenous systems, as assumptions about group performance can be extrapolated based on individual platform characteristics and dynamics. A drawback of heterogeneous systems is that it is challenging for robots to model other platforms in the system, as not all platforms are uniform. This causes the systems robustness to diminish in cases of unique platform failure. Because of this, most researchers in the space believe heterogeneous swarms work should be avoided when possible, leading to a majority of research focusing on homogenous systems [56].

A "swarm" of robots is a large group of robot agents that are all working cooperatively

to achieve some kind of common goal or collective behavior [56]. Traditionally, this term is associated with joint movement through space, although it applies to all types of collective behavior, from movement to sensing and decision making. Individual agents in a swarm tend to have limited capabilities, in terms of computing, sensing, or actuation. Swarm intelligence refers to the collective intelligence that is created from the interactions between autonomous individuals within a large group [57, 58]. The goal of swarm robotics research is to use many of these limited platforms to achieve bigger accomplishments not possible for just single agent systems. In order for a robot swarm to accomplish different kinds of tasks, the swarm must be flexible, robust, and easily scalable [59]. Some examples of optimal uses for robot swarms include: using multiple agents to quickly explore and map an unknown area, using multiple platforms to accomplish a common task (such as transporting a large object), and using a group of agents to accomplish different pieces of a large-scale task (such as deep learning-enabled computer vision). Swarms of robots are useful for implementing superior situational awareness [60]. They also support greater levels of robustness in regards to mission failures, since the use of multiple platforms allows for redundancy and error checking [61, 62, 63]. One of the main features of swarm robotics is location-based distributed computing; they are able to distribute tasks and workload among agents uniformly throughout a large physical area, allowing for increased spacial awareness as well as manipulation of a larger portion of the surrounding environment [58, 64]. This is especially evident in hazardous environments, as the use of swarm robots allows for increased situational awareness and mission capabilities without the inclusion of a human.

There are also several drawbacks that are associated with swarm-based systems. First of all, it is very hard for a human operator to control or direct a swarm system, for both centralized and decentralized control schemes. For centralized systems, the communication structure and control schemes do not scale easily when the number of individual actors is increased, and are particularly sensitive to the loss or replacement of the central leader [65, 66, 67]. For decentralized systems, the main problem comes with the lack of access to all global data. For decentralized swarms, not all robots are connected to each other, meaning it is hard to get a snapshot of the entire state of the system, or to easily synthesize all of the information currently held by the collective group. This lack of full situational awareness and

the ability to predict full group behavior makes it difficult for one agent, in this case a human operator, to make appropriate calls to direct the swarm as a whole. However, even with these drawbacks, the benefits of using multiple robot platforms in a swarm far outweighs any of these concerns. Swarm robots have been implemented and testing in a variety of challenging scenarios, including space exploration [65, 68], search and rescue operations [69, 70, 71, 72], autonomous security [73], and battlefield surveillance [61].

2.7.1 Testbed Hardware Platforms

For heterogenous work in this paper, there are two main types of platforms we are focused on : UAVs and cubesats. UAVs will be used in this case to simulate cubeSats. What makes these platforms different is the different sensor packages and other hardware available on each platform. In order to allow for a collection of UAVs that are able to do a wide variety of tasks (both sensing and computing), we need to make sure the swarm is fully equipped to meet all of these needs. Since power and weight budgets and extremely restrictive on UAV platforms, these capabilities will be divided up among numerous physical platforms. Some UAVs will focus solely on sensing, and will be equipped with one of these specialized sensors: HD Camera, Thermal Camera, LIDAR sensor, IR Camera(s), Ultrasonic Sensors + microphones, wind sensors, etc. Other UAV platforms will have specialized computing hardware in order to meet special computational requirements, such as chips designed for Video Processing or machine learning. Some of this demand will be alleviated by embedding FPGA systems on these drones; as FPGAs are reprogrammable hardware, their software can be customized to meet the task at hand. And other UAVs will be equipped with hardware that allows them to conduct their duties, such as a package delivery mechanism, an arm to acquire experimental samples, a parachute for safety, or a speaker system to convey audio messages.

2.8 Overview of Dynamic Vision Sensors and Event Simulation

One of the major areas of interest in robotics is computer vision. With regards to UAV systems, computer vision and image processing is extremely important, both for situational awareness (sense and avoid) as well as for localization and mapping purposes. However, cameras, especially good ones, require large amounts of power, and do not operate well under high speed conditions. As such, UAV researchers are always looking at alternative methods to capture and analyze vision data. One of the latest sensors is called a dynamic vision sensor, also known as a neuromorphic camera or an event camera. The Dynamic Vision Sensor (DVS) is an event-based, neuromorphic sensor that mimics properties of the human eye in order to dramatically increase sampling rate, while simultaneously reducing effective data rate. Rather than sensing the magnitude of luminance at each pixel, the DVS detects changes in the log-luminance over time, resulting in image data with high temporal resolution and without redundant, static background information. This behavior allows the sensor to operate with a sampling rate in the MHz range and enables the high dynamic range of the sensors to detect exceptionally small changes in luminance. However, event-based sensors differ from conventional cameras due to the image data being represented as one-dimensional sequences of events rather than traditional 2D images. While the small-size and low-power consumption of dynamic vision sensors make them ideal for embedded platforms, the asynchronous nature of the image data necessitates new computer vision and image processing techniques for conventional applications [74, 75, 76].

Due to the relatively high cost of dynamic vision sensors, it is advantageous to be able to test and verify computer-vision applications using conventional images from either RGB cameras or rendered simulations. With the low availability of event-based datasets, event simulation (i.e. extrapolating a one-dimensional event stream of events from conventional images) is a particularly useful method for evaluating event-based applications. Given a conventional sequence of images, one-dimensional event stream data can be extrapolated via the intensity difference and current exposure value found at each pixel. Since event simulation can be a time-consuming process as a byproduct of the large number of spikes generated and the need for timestamps to be ordered temporally, a naive, linear implementation is

infeasible for real-time processing in this work. Instead, the process is converted into a sequence of matrix operations and parallelized via GPU acceleration in order to achieve real-time integration with AirSim [77, 78, 79].

Using a real-world DVS in the context of robotic navigation and obstacle avoidance has already been implemented in several previous works. The work in [80] explores how a reduction in visual latency can positively impact the maximum speed with which a quadrotor can navigate an environment by comparing monocular and stereo cameras to event-based cameras. The paper suggested that an event camera has a more beneficial impact on performance when a robot is more agile and able to react quickly to changing information, as compared to traditional monocular and stereo cameras. This assertion was tested by having quadrotors perform collision avoidance maneuvers against incoming balls, using DVS data as visual input. [81] implements an impressive on-board object collision method for quadrotors using a dynamic vision sensor and an Extended Kalman Filter. [82] takes the experiments performed in [81] and conducts a theoretical analysis on the joint role of perception latency and actuation latency in robotic navigation tasks.

2.8.1 Benefits of Using Dynamic Vision Sensors

The use of the neuromorphic vision system onboard UAVs and in space-based applications has numerous benefits. One major benefit of this camera is that it is event-driven; the event-based sampling works by collecting data from only the pixel sensors that sense a changing event, as opposed to all of the pixel sensors in the camera. Because of this reduced number of pixel data being recorded, this leads to very low data rates along with extremely high accuracy [83]. This is important for UAV and satellite systems as it allows for more efficient use of the limited onboard memory as well as helps resolve the usual problem of limited downlink bandwidth issues by transmitting back minimal data. Dynamic vision sensors have the capability for more efficient object tracking, and an increased ability to ignore overpowering light sources [84]. In the field of aerospace research, there is currently a major push for developing new technologies that can help with space situational awareness (SSA) [85]. The ability for a satellite to track objects and predict their motion patterns it vital in

space, as it allows for detection of threats and application of collision avoidance measures; this capability requires instant detection of objects as well as real-time processing. Dynamic vision sensors are ideal for this kind of instant object tracking, as they are optimized for real-time trajectory mapping and motion prediction. In one study by Valeiras et al, objects were able to be tracked at a Khz rate in real time using these cameras [86]. This kind of instant tracking is also crucial for UAV systems, where sense and avoid algorithms need to be perfected to work in real time; this is especially crucial when UAVs are operating at high speeds in dense areas, such as a forest or cityscape.

2.8.2 Space-Based Applications and Background

There are several major problems in the field of aerospace and space-operations research in which the neuromorphic camera can be a solution. Many of these problems have been thoroughly researched with a variety of solutions posed as results; we think the benefit of neuromorphic cameras can far outweigh many of the drawbacks of these other solutions.

As mentioned before, space situational awareness (SSA) has been an important area of research both in civilian and military situations with regard to space-based applications. The research done by Ender et al. demonstrates the use of radar systems for SSA, in use of applications for collision detection, orbit estimation, and propagation [87]. Radar is a commonly used technology chiefly because of its wide range; it can sense over a very wide area of coverage, with consistent results.

However, this method is only viable to detecting large objects; this is due to the large wavelengths of the radio waves used, as well as diffraction properties [87]. Another researcher, Demars, has presented a method that uses SSA for predicting non-linear orbits for objects in space. The proposed method uses Gaussian mixture modeling to exploit properties of a linear system to extrapolate information about a nonlinear system [88]; in order to reduce the error from extrapolation, Gaussian splitting is then introduced to the resulting model. This allows for the accurate mapping of complex, non-linear orbits, which allows for better interpretation of collision paths and probabilities. Although not as useful for human operating systems, this level of modeling and prediction becomes crucial for autonomous systems that are conducting

surveillance. A major downside of this method is the complex mathematical calculations that are part of the modeling process; these methods are too complex for operation on any standard satellite computing platform. There is also possibility of statistical errors to be introduced into the calculations, based on the method being used. Another paper by Abbot and Wallace also looks at ways to track orbiting space objects, albeit in larger numbers, using SSA. Their initial claim that the small number of sensors used for gather data is what leads to inconsistent observation of the objects, which leads to poor prediction results. They propose that multiple satellites orbiting in geosynchronous earth orbit (GEO) work cooperatively to monitor space objects, to allow for more robust collision avoidance systems. This is especially useful for satellites in GEO, as their orbit tends to oscillate (an example is shown in the diagram below) [89].

The drawback of this method of using SSA, like the last one, it the amount of processing power and local memory that is required. This method utilizes bayesian modeling, which tends to require large amounts of processing power to be done correctly, as well as relies on the available data regarding orbital information for all tracked objects, which is difficult to find and access. This can lead to many problems when considering classified space objects, those not known to the public, or those with incorrect public access data.

2.9 Machine Learning

2.9.1 Reinforcement Learning

Reinforcement Learning is a subset of machine learning that teaches an agent behavior based on its interactions with an environment and the resulting outcomes of specific decisions. Based on behavioral physiology, the basis of reinforcement learning is to attribute a "reward" for each action taken by an agent, and to shape that agent's behavior in order to maximize the cumulative reward in the course of a trial. Reinforcement learning allows us to give a framework and set of tools for robots in order to design and improve a hard-to-engineer set of behaviors in both known and unknown environments. For this dissertation, reinforcement

learning is used to develop control policies for UAVs in simulation, specifically teaching them how to interact with their environment to efficiently accomplish a list of tasks. Training these policies in simulation is the fastest way to develop a proper set of model behaviors, as this method relies on hundreds of thousands of training trials in order to reach a decent model. For this dissertation work, the two major reinforcement learning techniques used at Deep Q learning Networks (DQNs) and Proximal Policy Optimization (PPO).

Deep Q learning is a prominent area of reinforcement learning research that leverages deep learning neural networks to learn effective policies given high-dimensional input data [90]. The agent action policy is the core of the Q-learning algorithm, which dictates which action an agent should take given the current state of its environment in order to maximize expected reward. The Q-value indicates both the immediate reward for a given action and a discounted summation of the expected rewards gained from subsequent states, allowing the agent to effectively plan into the future what actions should be taken over time. Using techniques taken from classic deep learning tasks, such as image classification, the authors of [90] trained a custom CNN architecture as the means of estimating Q-values based on visual observations of an environment, with a target Q-value as the loss metric used to train the network weights. In order to smooth the training distribution, the authors also introduced experience replay, a manner in which the network could be trained over batches of previously encountered states and actions [90]. This resulting neural network function approximator, called the Q-network, was trained in order to achieve state-of-the-art results in playing a selection of five Atari games. [91] expands upon this work, achieving human-level performance across a wide variety of Atari games with a single, model-free deep Q-network agent. [91] uses human-in-the-loop demonstrations as a form of semi-supervised learning to reduce the size of the data set necessary for deep Q-learning. These demonstrations serve to pre-train the agent and accelerate the training process by initializing the network with non-random weights.

Several previous works have explored memory-based deep reinforcement learning methods in order to avoid obstacles in unknown, indoor environments. Reference [92] opts for a Deep RL approach using a simple monocular camera instead of computationally intensive SLAM or SfM approaches because of the cost of such methods and their inability to avoid

dynamic objects. The work successfully used a recurrent neural network (RNN) module to enable learning an obstacle avoidance control policy that can utilize relevant past information. Experimental results showed significant improvement over both DQN and other DQN variant algorithms. This work is similar to our approach, as it performs training in a virtual environment, modifies the basic DQN architecture, and performs obstacle avoidance with a simulated UAV. The primary difference between these approaches is that [92] uses a monocular camera to estimate depth (using a depth estimator obtained from training a network with a conditional GAN) and computationally-expensive LSTM modules to enable temporal attention; Our agent relies on the interaction between the DVS data and spiking architecture to capture temporal information.

2.9.2 Spiking Neural Networks

Work done in formulating SNN-specific training algorithms stems from developments in both deep learning and neuroscience. However, fundamental research in SNN-specific training relies heavily on biologically-inspired processes from neuroscience research that are not entirely proven in the realm of deep learning. A number of papers concerning biologically-realistic models for learning have resulted in real-world implementations that demonstrate promising results [93, 94, 95]. The work in [96] outlines a framework for propagating an additional network parameter, a reward signal, through neurons in a biologically-inspired way of mimicking conventional reinforcement learning. A supervised multi-spike learning algorithm is used to train spiking neurons to converge towards desired spiking rates in the work of [97], achieving competitive accuracy in time-based pattern classification tasks. [98] demonstrates a novel method of SNN learning using spike-time dependent plasticity (STDP) for image classification tasks. For deep learning-based approaches to training SNNs, the reader is referred to [99].

For practical applications, the low-level implementation of spiking architectures can be incredibly time-consuming, especially without native support in machine learning frameworks such as Tensorflow or PyTorch. To facilitate conversion between traditional, nonspiking networks to equivalent spiking models, [93] outlines a framework for combining deep learning

with the strengths of neuromorphic modeling. The Nengo framework is specifically designed to allow users to experiment with biologically-plausible neural networks and then apply them to meaningful tasks within the same unified environment. Additionally, the NengoDL framework used for this research enables users to apply conventional deep learning methods to spiking neural networks and supports the conversion of models built with the popular Keras libraries. "Neuromorphic modelling" refers to building models that incorporate high levels of biological realism, such as the three-factor learning outlined in [96]. In object tracking and collision avoidance tasks, SNNs have been shown to perform well in closed-loop control systems [100]. A SNN is used in the ROLLS neuromorphic processor to realize an obstacle detection and avoidance method on a physical robot with both static and moving objects. An interesting observation from this paper was that avoiding moving objects was more robust than avoiding static objects, as moving objects generate more events than static objects. This observation provides experimental evidence for the claim that performance of a control method is positively correlated with the speed at which the robot moves relative to its surroundings [100].

2.10 Simulation

Since it is difficult and time-consuming to tune controllers and UAV configurations physically, many multi-agent system designs relies on high-fidelity simulation. There are several simulation environments that are designed for testing and training UAV flight controllers. Some are equipped with the popular open-source autopilot system called ArduPilot [101]. The Gazebo simulation environment for Robot Operating System™ (ROS) has support for UAVs and is a free open-source simulator. Additionally, NASA provides several open-source UAV simulation environments, including the Independent Configurable Architecture for Reliable Operations of Unmanned Systems (ICAROUS) system which supports several libraries for geofencing, sense and avoid, and formal method verification for safety-critical applications [6].

The majority of the work in this dissertation utilizes Microsoft's AirSim simulator [102], which has been used for numerous first-stage UAV research applications. AirSim is a high-fidelity physical simulator built using the Unreal Engine which allows modeling & evaluation in a variety of environments and flight conditions. Previous work has shown that controllers designed and tuned within AirSim are transferable to physical UAV platforms with minimal changes [95]. Though recent advances have been added in support of the Drone Racing Lab [103], this research included the development and addition of several mechanisms to support swarm controls engineering. These include modeling of centralized & mesh communication networks, a framework to provide HD-RHC control to UAV platforms, and tools to tune & analyze HD-RHC performance. For the neuromorphic vision research, a stand alone event camera simulator was integrated into the AirSim egosystem, which has been used in place of the actual camera hardware system.

3.0 Task 1: Fleet Management System

3.1 Task Overview

The first task in this dissertation research is to develop a robust fleet management system for multiple UAV platforms. For this portion, the focus is on the high-level management system for interfacing and controlling the drones, as opposed to any controls work. Implementation focuses on homogeneous UAV platforms (all the same) as this is the easiest for testing and validation. The scenario used for this portion of the research was to use UAVs to simulate freelying spacecrafts (cubesats), and have them accomplish a joint mission of detecting damage on an object of interest. The payload for the UAV platforms is a standard RGB camera, and the fleet management system incorporates a computer vision algorithm that completes the sensing portion of the mission tasks. For this portion of research, the fleet management system was built using flight-proven open source software developed by NASA. This choice was crucial because it allows for the testing and simulation of space-based robotic software to be completed on UAV platforms. The software architecture was modified to add multi-agent functionality and control, as well as several interfaces to allow for using multiple different types of hardware platforms. The controller/planning portion of this system was also abstracted into a separate module, so it could be changed in order to reflect the work completed in Task 2. This work was completed in conjunction with the NASA Langley Research Center, and was presented in a conference publication [1]. This section includes the computer vision work completed for the custom sensing task as it shows the extend to which the fleet management system is able to operate.

[1]Sami Mian, Tyler Garrett, Alex Glandon, Chris Manderino, Swee Balachandran, Chester Dolph, and Cesar A Munoz. Autonomous spacecraft inspection with free-flying drones. In 2020 IEEE/AIAA 39th Digital Avionics Systems Conference (DASC). IEEE, 2020 [In Press].

3.2 System Design

This work deploys ICAROUS with cFS in UAV mission computers to simulate free-flyers operating in-orbit. A major contribution of this work is the development of several new modules for the ICAROUS system to provide high-level mission management and multi-agent coordination. This work also develops a cFS VICON interface for indoor localization during research and testing. We also implement a novel computer vision program in cFS for accurately detecting several types of damage in-situ.

3.2.1 System Software Architecture

Thanks to the ICAROUS system developed by NASA Langley, the same software system used on satellite platforms can now be used for autonomous UAVs. In order to use ICAROUS for the application of multi-agent damage detection, three new modules were added to the system to allow for extended capability. Figure 5 shows the various software modules that comprise the flight software system used for controlling the freeflyers.

Cognition determines various levels of mission tasks for each of the free-flyer including takeoff/land, assigning specific waypoints for each platform, and positioning for capturing data with available sensors.

Guidance issues low-level commands to each free-flyer based on their allocated tasks, such as changes in directional velocity, position estimation, and local trajectory planning functions.

Coordination manages the multi-agent aspects of the mission. This application accepts mission input from the ground station, determines how to split up the mission parameters/tasks based on requirements and number of agents available to deploy. The Coordination application also handles the dynamic addition or loss platforms of free-flyer workers at any time during the mission.

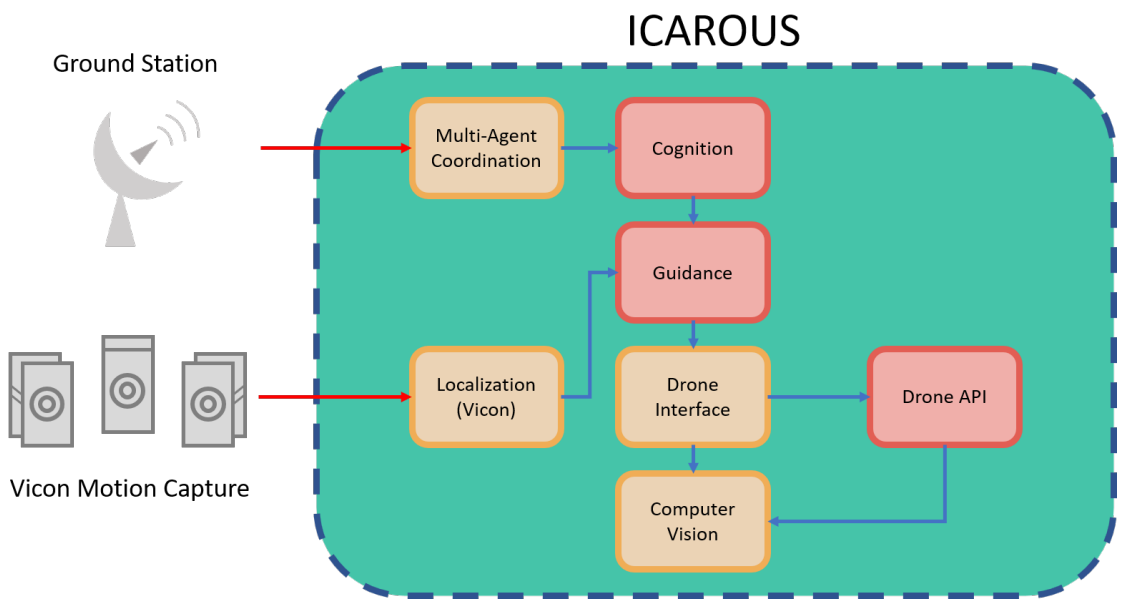


Figure 5: Flight software architecture [1] ©2020 IEEE

3.2.2 Platform Support Applications

Hardware Interface – In addition to ICAROUS suite applications, a hardware-interface module allows the ICAROUS system to interface with the free-flyer platform firmware. This application will be expanded on the implementation section.

Positioning System – As autonomous systems, the free-flyers require the ability to accurately determine position in orbit, with respect to themselves and an object of interest. To use traditional GPS localization, something common to UAV and spacecraft flight systems, indoors, a cFS localization application was developed to interface with a motion capture (mocap) system. This application translates local area positioning to GPS coordinates for real-time autonomous navigation, using a Vicon mocap setup. The Vicon system is a commercially available indoor motion capture system, our flight space utilized 16 HD motion capture cameras. Each free-flyer platform is marked with several tracking tags and individually registered in the system. In order to integrate into the system, a cFS application was created to interface with the Vicon system.

Inter-craft Communication – Each system uses a specialized cFS application, Software Bus Network (SBN), in order to share communications. SBN enables each instance of cFS to receive messages published to the software bus by any member of the swarm. For example, if the message is received from another flight unit indicating its position is too close to the aircraft itself, modification can be made to its flight path in order to avoid potential collisions while still progressing to its next way point.

3.2.3 Inspection Protocol Using Computer Vision

The inspection protocol (the custom sensing module added to the fleet management system) uses computer vision techniques on video provided by each UAV platform to identify potential damage or anomalies. There are two subtasks for the computer vision processing: first, the object of interest (a mock satellite) is segmented from the background; second, damage is detected in a window area corresponding to the segmented module. Damage is highlighted and visualized for the operation team in real-time. As a complementary feature to 2D damage detection, 3D reconstruction for visualization is also implemented for post-mission analysis. The 2D algorithm isolates regions of interest for autonomous operations, and a human in the loop can inspect the 3D rendering of the selected region after the mission. The computer vision has been integrated into the entire system as a standalone cFS app. Figures 6 and 7 below show the pipeline for the two computer vision processes.

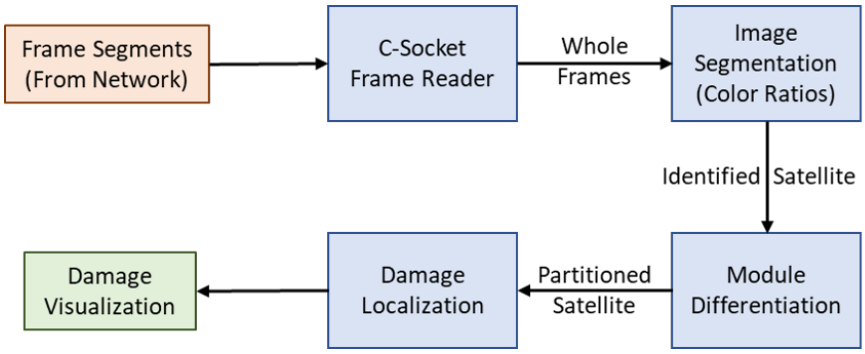


Figure 6: Real-time 2D Damage Visualization [1] ©2020 IEEE

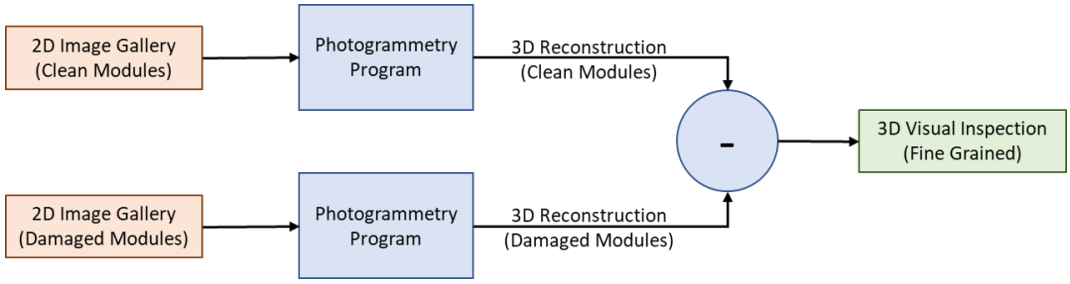


Figure 7: Post-Processing 3D Reconstruction [1] ©2020 IEEE

3.3 Implementation

Several technologies are developed for this project. The development of the control and coordination for the UAVs is realized through the implementation of new ICAROUS modules: a two-way communication system between the UAVs hardware and ICAROUS, a Vicon Tracker interface for providing vehicle telemetry, a custom flight controller based on a Proportional Integral and Derivative (PID) architecture, a mission coordinator for decentralized task distribution, a networking module that enables UAVs to share flight plans and mission objectives, and a visual damage inspector. The communication module added new message structures to the existing publish/subscribe protocol. These new messages

allow for the distribution of tasks among available free-flyers. The custom flight planner is designed to adapt the ICAROUS planner for multiple UAVs. The coordination planner analyzes mission objectives and dynamically distributes mission subtasks to each available UAV. Several libraries are created to autogenerate nominal flight plans for the UAVs with optimized video stability and field of view. Finally, a new set of computer vision applications were developed and integrated into the ICAROUS system to enable the damage detection and analysis portion of this project.

3.3.1 Drone Control Software

Since the cFS software used to control the drone's motion planning was not placed on-board the drone, a separate module was created in order to fill in the necessary gaps, and mimic how the system would work if the NUC computers were onboard the AR drone platforms.

The Drone Control module provides two high-level functions: convert velocity commands from cFS to low level control commands an AR drone accepts (AT commands, takeoff/land, Roll/Pitch/Yaw/Gaz), and serve as a flight controller to maintain the drones trajectory from point to point, with minimal error. The Parrot AR drones come built with a standard API, written in multiple languages, which supports sending commands to the drone to takeoff and land, hover in place, activate emergency mode, and modify the drones current roll, pitch, yay, and gaz (altitude). Several different control systems were also implemented in order to control the target drone's precise movement. A Proportional-integral-differential (PID) controller was used for managing the drone's 2D grid based navigation, a bang-bang controller for controlling the altitude, a double setpoint controller for managing the drone's yaw and field of view, and a normalized proportional controller for controlling the ground speed. The equations used for each of these controllers are listed at the end of this section. The resulting velocity output matrices were multiplied with three sets of transformation matrices, to convert the values from the local frame to the global frame of reference, for proper application onboard the drones. Several experiments were run to tune these controllers and determine their effectiveness in comparison to off-the-shelf solutions, the results of which are detailed in

the results section. A figure of the high level control scheme for the drones is shown in figure 8.

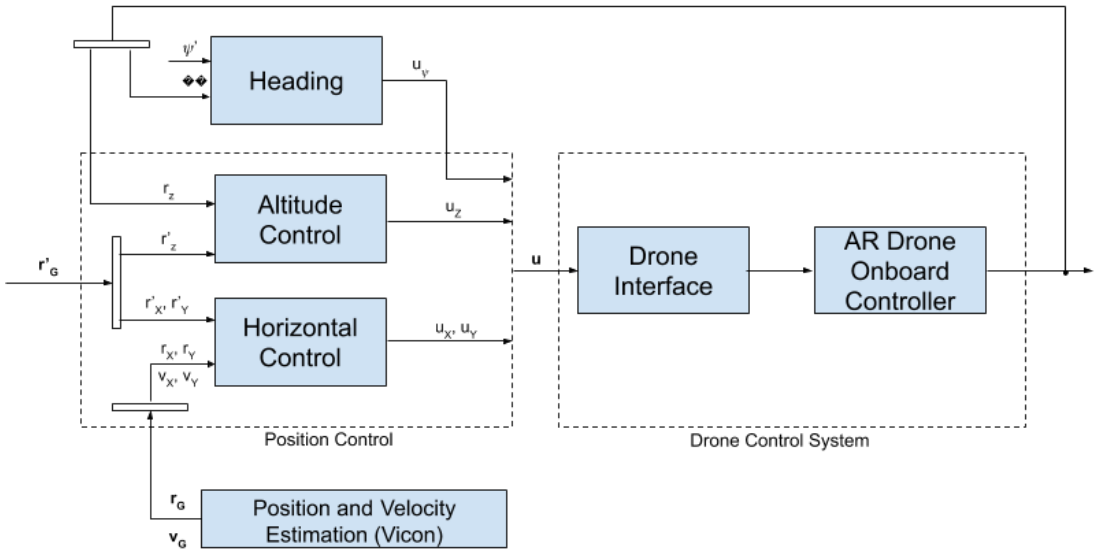


Figure 8: High level control overview [1] ©2020 IEEE

These are the equations used to design the various controllers used for the custom flight controller. Equation (3.1) is used to determine the desired viewing angle for the UAV to focus on the object of interest.

$$\theta = arctan2\frac{Y_{drone} - Y_{object}}{X_{drone} - X_{object}} \tag{3.1}$$

Where X and Y are the Cartesian coordinates for the object, in the local frame Equation (3.2) is used to calculate the Yaw velocity of the drone to change its camera orientation:

$$V_{YAW} = \begin{cases} 0.25\omega_{max}, & \theta - \psi > 5^\circ \\ 0.75\omega_{max}, & \theta - \psi > 15^\circ \\ 0.75\omega_{max}, & \theta - \psi < 15^\circ \\ 0.25\omega_{max}, & \theta - \psi < 5^\circ \end{cases} \tag{3.2}$$

Where ψ is the current UAV heading, θ is the desired heading, and ω_{max} is the maximum UAV angular velocity.

Equation (3.3) is used to calculate the thrust needed to change the UAV's current altitude. Velocity input for drone thrust:

$$V_{GAZ} = \begin{cases} V_x * \tau + V_{ALT} & \Delta_{ALT} > 0.5m \\ V_x * \tau + V_{ALT} & -\Delta_{ALT} > 0.2m \\ 0.1m/s & otherwise \end{cases} \tag{3.3}$$

where Δ_{Alt} is the required change in altitude, V_x is the current velocity in the X direction, τ is the yaw scaling factor, and V_{ALT} is the vertical velocity required to stabilize the UAV.

3.3.2 Multi-Drone Coordination

In the most basic implementation of an inspection, the ground station uploads a single flight plan to a lone aircraft which then travels to each waypoint. The mission concludes once all points have been reached. The Cognition and Guidance applications are able to guide the

drone effectively and safely, however, rely on receiving the initial flight plan to carry out the mission. This becomes increasingly complicated as more than one drone is introduced into the system. The complexity is further exacerbated by needing to handle a fluctuating fleet size. To address these issues, a new application is introduced called Coordination. The role of the Coordination application is to monitor the fleet and dynamically allocate and distribute updated flight plans to each drone to achieve optimal traversal of the search space around the object being inspected. The application considers fleet size and remaining waypoints from a master flight plan. Coordination is implemented within the software architecture between the ground station and Cognition. A master flight plan is uploaded initially to a single drone. The flight software on this drone then evaluates, computes, and distributes the new flight plans for itself and all other fleet members based on their aircraft ID. In the event that a drone is removed or added to the system, a reassessment of the remaining list of waypoints and drone positions is evaluated and updated flight plans are redistributed. Coordination can handle several different scenarios including:

1. Mission starts with one or more available agents
2. A new agent is added to the available group of platforms
3. An existing agent is no longer able to perform a mission (loss of platform, communication, etc.)
4. A discrepancy in data is detected and new mission tasks need to be added for robustness
5. The object undergoing inspection has moved and new mission waypoints need to be determined

3.3.3 Automated Waypoint Generation

As the goal of this mission is to use various freeflyers in order to inspect an object and look for damage using computer vision approaches, it is necessary to obtain images of the object being inspected from a variety of angles and distances. In order to make sure the UAVs are able to obtain substantial visual data to ensure the detection of any and all simulated damage, several tools were created that auto generate various UAV flight plans. The tool requires a number of parameters, including object's size and GPS coordinates, as well as the

number of images desired, the resolution of the images, the desired yaw and pitch angles of the photos, and any unique flight patterns (a helix, a circle, raster photos, etc.). The planner first calculates all of the requested waypoints in a 3D cartesian coordinate system, placing the object at the origin. Then, these coordinates are converted into Geodesic coordinates using an open source UTM library, which simulates the projection of 3D space onto a sphere (the earth). Lastly, these new coordinates are formatted and combined into a mission input file that can be provided to ICAROUS.

3.3.4 Localization System

The Vicon motion capture was integrated into ICAROUS as a new application which used the Vicon SDK to open a socket connection that received telemetry data as the drones motions were tracked in real-time. Capturing 3D frames of the flight space at up to 200Hz, the drone's position and rotation are recorded relative to the global center. From here the Vicon application performs several calculations to derive velocity (by taking the difference in position between frames), heading, and translating the coordinates from the local frame (NED) to the spoofed global frame (geodesic). The data is then piped to the Guidance application where based on the current location of the drone coupled with the assigned destination from the Cognition application, adjustments are made to the velocities in the local coordinate frame to keep the drone on course. These adjustments are then passed to the drone interface module to be translated to the raw commands accepted by the UAV platform's control API.

3.3.5 Computer Vision Implementation

First, for background subtraction, several techniques are tested experimentally for performance on segmenting our given modules. For the satellite segmentation several techniques are compared. Adaptive Otsu thresholding is considered as shown in figure 9a. It is fast, but ineffective at precise segmentation. The intensity between the cylinders and the background is insufficient for Otsu segmentation. Template matching is considered as shown in figure 9b. Template matching works well when the mock satellite is at a fixed distance or image size.

When the mock satellite is close or far, the predefined template will not match. This can be fixed, by performing multiple searches through the image with different size templates. In this case the speed is drastically reduced. GrabCut is applied for segmentation also as shown in figure 9c. It performs well, but is not considered for final implementation as it requires user interaction and is slow to process.

(a) Otsu thresholding

(b) Template matching

(c) GrabCut

Figure 9: Initial segmentation experiments [1] ©2020 IEEE

Finally a solution is developed using color based thresholding that is able to robustly and precisely extract the location of the space module. To improve segmentation performance, the next step was to paint the mock satellite a gold color. The gold color was chosen because of its similarity to that of the polyimide-based insulation usually found on the outside of satellites.

The color based segmentation is based on ratio matching. Standard color matching is based on color channel vector distance as in Equation 3.4. R refers to the red color channel intensity, and so on for G and B. R_{ref} refers to the target channel intensity and so on.

$$\sqrt{(R - R_{ref})^2 + (G - G_{ref})^2 + (B - B_{ref})^2} \tag{3.4}$$

This gives stable results invariant to lighting and background noise using ratio based matching, where distance is now described in Equation 3.5. Reference RG_{ref} refers to a target red to green ratio, and likewise for green to blue ratio and blue to red ratio.

$$\sqrt{(\frac{R - G}{RG_{ref}})^2 + (\frac{G - B}{GB_{ref}})^2 + (\frac{B - R}{BR_{ref}})^2} \tag{3.5}$$

Once the cylinder image is determined, convolutional neural network (CNN) and sobel edge detection are compared experimentally. The first CNN model is trained to consider the 3-class decision problem of “background”, “module - damage”, and “module - no damage”. This did not achieve good performance. Including ”background” made the classification problem more complex. Next, the CNN is trained to return a binary decision, representing damage “present” or “absent”. This required building a training set of many example views of the cylinder including normal and damaged sections, with a variety of lighting and distance conditions. The CNN is applied as a sliding window operation to detect damage in regions of interest. The CNN performance was fair, however the computation was slow, as the CNN output is computed over several windows.

Sobel edge detection is considered as it is fast to operate this method over an entire image. Sobel returns a filtered image, where edges are highlighted. Two damage detection algorithms are applied to the sobel output. The sobel output is integrated over windows of interest to determine regions of damage. The sobel output is also visualized at the granularity of pixel level.

For post-mission processing 3D damage visualization using photogrammetry is performed. A necessary condition for fidelity in 3D object rendering is the presence of discriminatory image features at each location of the object. Unlike the expectations for a spacecraft, the gold painted cylinders did not have identifiable features over the extent of their clean surfaces. To correct this deficiency, a speckle paint pattern is applied to the surface of the damage cylinder. The photogrammetry pipeline uses an algorithm called structure from motion to generate the 3D representation giving a gallery of images from different perspectives [104].

3.4 Experimentation & Results

For this demonstration, a damaged aluminum cylinder is used as a mockup of a damaged satellite. The free-flyers cooperate to scan the cylinder for damage at a high resolution. Multiple UAVs use ICAROUS to autonomously navigate around the mockup satellite while keeping it within their cameras' field of view. The free-flyers cooperate and maintain a safe distance between each other vehicle and the mockup. The coordination module creates a unique flight plan for each of the UAVs based on the shared mission plan. The coordinator also allows dynamic task reallocation when the number of UAVs systems available for the mission is changed (though new additions or platform loss). The UAVs systems complete a full successful scan of the mockup, highlighting the damaged surfaces in real-time and providing a video visualization during mission execution. A publicly released video of the demo and project overview is available at [105].

3.4.1 Drone Hardware Platform

The UAV platform used in this demonstration is the Parrot AR 2.0 Drone equipped with an ARM Cortex A8 processor, 1Gb of RAM, and a barebones version of Linux 2.6 [106]. These platforms come equipped with a built-in WIFI b/g/n chip for both establishing and connecting to wireless networks. The sensors onboard each platform included a 3 axis gyro, a 3 axis accelerometer,a magnetometer, an ultrasonic sensor (for altitude measurements) and 2 cameras. A 720p 30 FPS camera faces forward on the drone, and was used to collect video for the damage analysis in this demonstration; the other camera is a downward facing wide angle lens sensor which is used for optical flow tracking, which allows for smoother movement and hover.

Due to a limit of 100g payload and insufficient computing power, a secondary offboard computer was chosen as a proxy. The Intel NUC miniature PC was chosen, as it is small enough to be placed onboard custom UAV platforms and hence serves as an acceptable substitute. Each NUC was connected to one AR drone platform via WiFi, running the cFS and ICAROUS software platform and issuing low level actuation commands. The UAVs stream back live video from its forward-facing camera to the NUCs, where the computer vision cFS application would analyze the video for damage patterns.

3.4.2 Full System Demo

The mission success of this simulator demonstrates multiple-agents can cooperatively inspect an object for damage in real time.Two UAVs were used to simulate two free flyers in space; the space modules were stacked together, with the damaged cylinder on top of the pristine cylinder, to mimic the large cylindrical body of a fuselage. A single flight plan was autogenerated for inspecting the cylinders, with three distinct parts: An orbit of the top cylinder, a downward spiral in the pattern of a helix with two full revolutions, and a full orbit of the bottom cylinder. With this flight plan, each portion of the object's surface would be viewed at least twice by one of the cameras. In this demonstration mission, one free-flyer initiates the mission and a second free-flyer joins halfway through the mission. The second UAV would then be abruptly removed from the space, simulating a loss of spacecraft. The

Coordination application coordinates the dynamically changing environment to account for mission resources and mission progression.

The demo progressed with the following events:

1. UAV 1 enters flight space
2. Coordination module detects one freeflyer and passes full flight plan from ground station
3. UAV 1 begins orbit of the top cylinder and processes the video stream for real-time damage analysis
4. UAV 2 is dispatched
5. Coordination detects new freeflyer and dynamically updates mission into two subtasks: one for orbiting the top cylinder and one for orbiting the bottom
6. Each subtask is distributed to UAVs 1 and 2 respectively
7. UAVs 1 and 2 commence new operation
8. UAV 2 abruptly departs
9. Coordination detects the loss and recalculates a new mission for the UAV 1 consisting of the remaining waypoints
10. UAV 1 completes visual analysis of the top cylinder and enters a helix to spiral down to finish analyzing the bottom cylinder using the waypoints inherited from the loss of UAV 2
11. UAV 1 completes the remainder of the mission

Figure 10 shows the original mission flight plan as well as the flight patterns for both drones generated from the flight logs from this demo.

3.4.3 Flight Controller Performance

Several experiments were performed during the development of the UAV control software, to assess the impact of the different control approaches and tuning methods.

For the 2D trajectory planner, the PID controller provided the capability to allow even the cheap AR drone the ability of high precision in regards to movement. Although the actuation system on board the platforms was inaccurate, the controls provided a critically

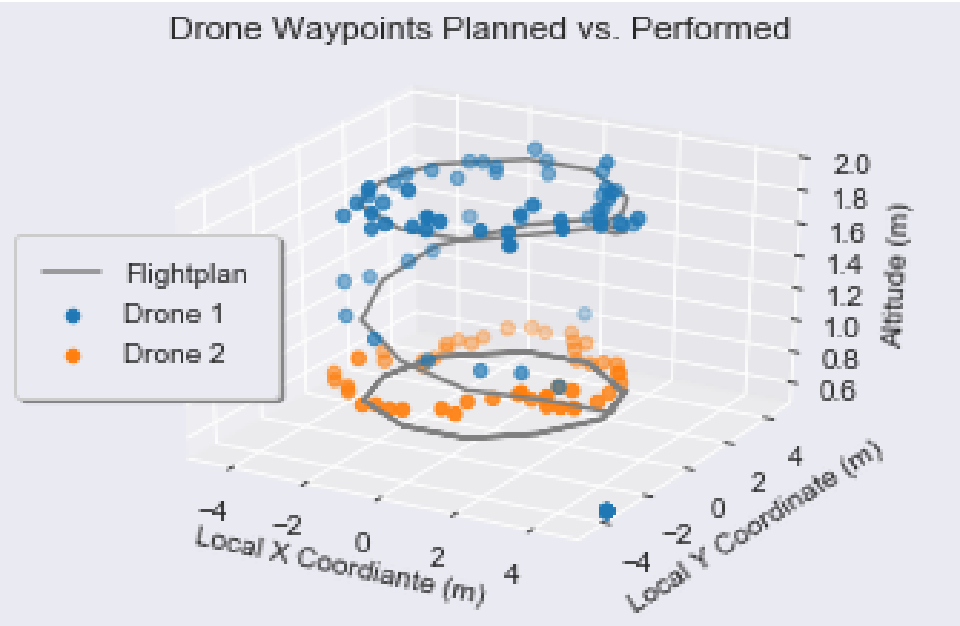


Figure 10: Drone waypoints visualization [1] ©2020 IEEE

damped system, allowing the UAVs to smoothly follow their waypoint lists. This was extremely important for some of the more complex flight patterns, including the double helix and several overlapping flight plans. During flight testing, each drone's actual position was recorded and compared to the intended position determined by ICAROUS, in order to ascertain the effectiveness of the PID controller. Figures 11 and 12 shows the maximum error of the drone's position across five trials; Figure 11 shows the error in terms of absolute distance (meters), and Figure 12 shows the error in terms of relative distance (percentage).

For the Yaw control, both a proportional controller and a double setpoint controller were considered. The proportional controller proved to be a poor choice, as it would often overdampen, causing the drone to jerk violently back and forth, failing to keep its focus on the object of interest. The double setpoint controller provided a smoother choice, as it allowed for the drone platform to turn at a constant rate, which was recorded and used for post-processing of the video information. As the drone's yaw orientation came closer to the target orientation, the drone would slow to a more appropriate angular velocity.

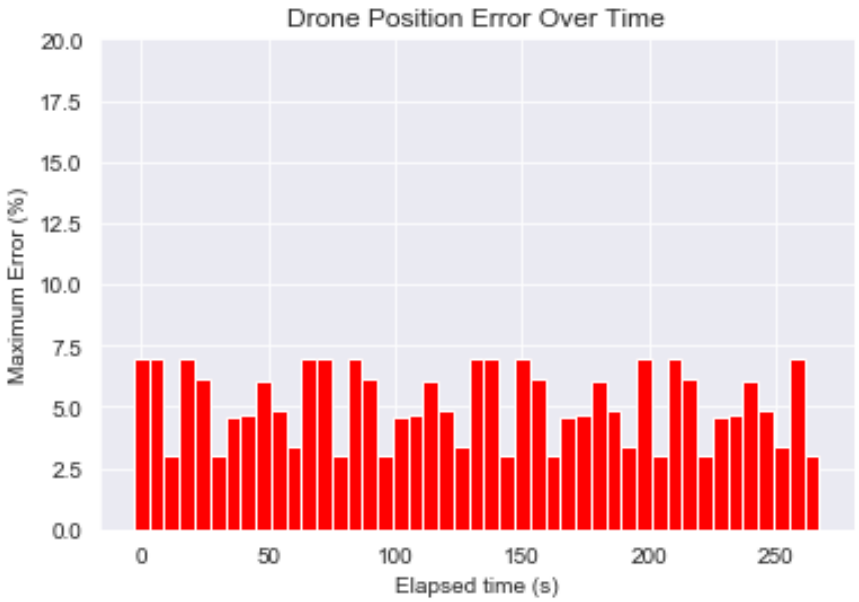


Figure 11: Drone position absolute error [1] ©2020 IEEE

Figure 12: Drone position relative error [1] ©2020 IEEE

For the altitude controller, a simple bang-bang optimal controller proved adequate for thrust control. The system would be able to reach the targeted altitude within 1.5 seconds, based on the ground speed of the platform, while maintaining smooth motion. Due to the nature of UAVs' need for constant thrust, this provided additionally beneficial during hover/holding patterns, where supplying a constant set rate of thrust allowed for near-perfect altitude hold.

3.4.4 Computer Vision Results

The color ratio segmentation algorithm is shown to perform robust satellite segmentation as shown below in figure with the blue and green windows. Our CNN is trained for damage detection. The CNN is trained on all the surfaces, but the more variety of surfaces, the more complex the CNN becomes. This is less robust for a simple space module surface than sobel. Figure 13 shows the output of the CNN on a sliding window damage detection where green represents normal and red represents damage.

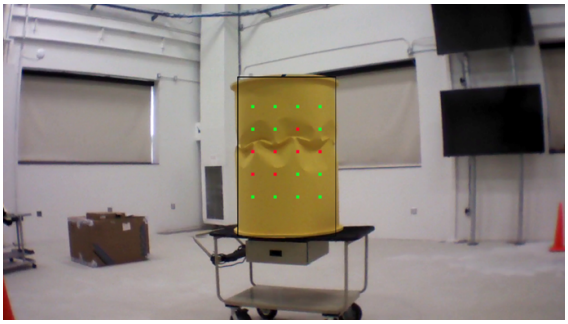


Figure 13: CNN sliding window output [1] ©2020 IEEE

Based on performance, the final system incorporated sobel filtering for damage detection. Given that our surfaces are rather smooth and have low inter class variance, the CNN did not match the performance of the sobel filter. False negatives are crucial to avoid as not detecting damage is system failure. Our final system incorporates a tunable sobel filter

threshold which enables the false negatives to be minimized by increasing false positives. This parameter is fine-tuned and found that the damage can be robustly isolated with very low false negatives, and at the same time the false positives are very low. The overlaid final damaged detection is also shown in window based damage detection in figure 14a, and edge based highlighting in figure 14b.

Figure 14: Computer vision output of segmented satellites and damage highlighting [1] ©2020 IEEE

The final segmentation and damage detection algorithms are written in OpenCV, which requires C++. However, the C++ code is wrapped into a C application, so that the computer vision algorithms are self-contained in a standalone ICAROUS module. In addition the concept of 3D modeling is demonstrated to supplement and improve perception of damage to the space module. 3D visualization shows a human-navigable depth perspective for further investigation. Given a space module with arbitrary lines (features that the sobel would pick up but that don't represent damage), the original surface can be modeled for comparison with an after damage 3D reconstruction to detect discrepancies due to damage. Figure 15 shows the before where the speckle paint is applied to the cylinder, and the after 3D reconstruction.

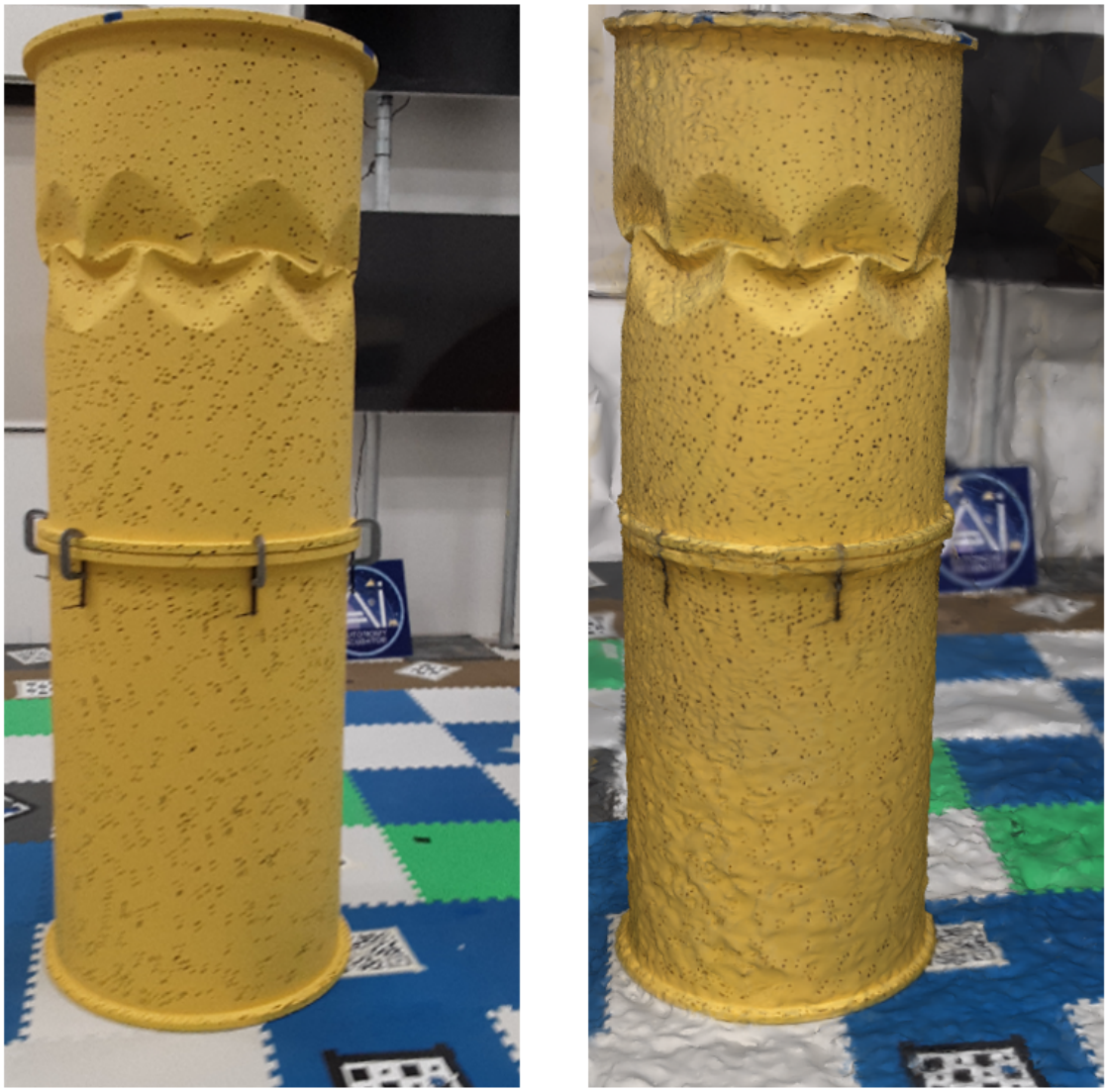


Figure 15: Speckled painting and 3D reconstruction [1] ©2020 IEEE

3.5 Discussion

The high-level system designed in this project was made to show the beneficial capabilities of freeflying agents in orbit around the Earth. Using simple UAV platforms as freeflyer substitutes, a software framework was designed to show the applications of computer vision onboard these platforms, specifically for the purpose of in-situ inspections of objects of interest. The demonstration that was held at the end of this project showed that such a system is indeed possible, and a large portion of the enabling software technologies for this have been created as a result. The implementation and experiments carried out have shown that the existing ICAROUS framework allows for testing conducted using UAV platforms to properly emulate those of actual satellites/systems in orbit, and the additional software modules have added new capabilities to the system. The following is a discussion on the outcomes, implementation results, and challenges for each major part of this project.

3.5.1 Individual UAV Control

The AR Drone was adequate for the requirements of this mission simulator. Although the built-in control system was subpar, the development of the ICAROUS drone integration module and custom control scheme allowed the UAV flight to be controlled with a high level of precision. The PID system allowed the UAV to follow the directed velocity commands, and maximized power efficiency by avoiding unnecessary acceleration and jerk. The original design of the yaw controller tended to overdampen, causing the UAV to jerk back and forth while focusing on the cylinder. This caused noise in the video data, which inhibited the damage analysis. The double setpoint control scheme for the yaw allowed for smooth control of the UAV's angular velocity, and the small margin of error allowed the video to stabilize sufficiently for video data analysis. The upgraded control system improved stability while hovering. The simple bang-bang controller for movement in the Z-axis proved sufficient, as the system would always have to provide a non-zero thrust to ensure stability. The adaptive control technique allowed the UAV to reach the desired altitude quickly, while utilizing the UAV's current directional momentum and improving thrust. It was shown that this scaled

approach to altitude hold allowed for the smoothest transition between altitudes, both during UAV movement and during position hold.

3.5.2 Localization System

The indoor localization system was able to be successfully integrated with the ICAROUS framework, allowing for pseudo-GPS data to be provided for any of the tracked objects within the space's vicinity. The GPS location of the system was dynamic, allowing for simulated testing in numerous real world locations; this was important for ensuring that geofence and airspace restrictions would be followed. The use of the indoor localization system also allowed for a high degree of accuracy with respect to tracking the UAVs' movement, and enabled the coordination of multiple platforms to work in a highly confined space. The system designed to provide localization data for over one hundred unique agents. Each agent's position and orientation data was provided at a rate of 200Hz, which allowed for the tracking of vehicle acceleration and jerk at an interval measured in milliseconds. This is comparable to the tracking necessary of any freeflyers in orbit, where the distance traveled in this short time frame is significant.

3.5.3 Multi-Drone Implementation

The main challenges of using multiple drones related to determining how to split the mission properly between available free-flyers and how to coordinate the motion planning for all of the active platforms. This proved to be challenging as the available free-flyers were treated as a dynamic resource: platforms could be added or removed from service at any time. In order to ensure all mission waypoints and tasks were achieved, the Coordination application tracked all assigned subtasks, and verified mission success with each agent at every milestone. At the beginning of each time step, the group of platforms would be checked to see if any platforms had been removed or added to the available group; if so, only then would the Coordination module re-distribute the mission parameters.

This method of UAV coordination and mission allocation proved successful in all testing scenarios. In addition to tests with variable number of UAVs, some tests were run where platforms were suddenly removed from the system (to simulate sudden damage). Each agent's data was marked with a timestamp including the platform's spacecraft ID. This way, during 3D reconstruction, the CV module would be able to use each platform's position/velocity data to match up the relevant video data in relative space. One of the challenges of the joint motion planning was preventing any well-clear violations (making sure the UAVs didn't fly too close to each other or the object). This was accomplished by having each platform broadcast its current and next waypoint goal throughout the mission; a UAV's flight controller would run a check to make sure there would be no intersection or adjacency of paths during movement.

Another major challenge was verifying that each waypoint was visited the correct number of times, by any of the platforms. To solve this, each platform kept track of the list of waypoints all the other platforms visited, based on the previously mentioned broadcasts; if there was a discrepancy, the affected waypoints would be visited again to verify data redundancy.

3.5.4 Computer Vision System

The CNN-based algorithm was trained as demonstrated for damage detection. This is less robust for a simple space module surface than sobel. However, if the damage is more diverse in character, and the modules have a complicated surface with equipment and junctions, a CNN can be trained if an existing dataset of labeled damage is given. The final system incorporated sobel filtering for damage detection. The advantage of sobel is the tunable filter is able to minimize false negatives.

The advantage of 3D modeling in addition to 2D, is two fold. First, 3D modeling can be used as a comparison of before and after assembly to detect damage. Secondly, when our 2D pipeline is in place, and is producing detections of damage in real time, the 3D modeling can be used to visualize damage offline interactively. For example, if there is damage on the rims, the 3D modeling can find this, where the edge detection based sobel would miss this difference. Also, a CNN can miss damage when it is not given a specific set of training images corresponding to a class of damage. 3D modeling places a human operator in the loop, minimizing the likelihood of detection failure.

3.6 Chapter Conclusion

The work accomplished in this chapter shows the successful development and deployment of a robust fleet management system for multiple UAVs conducting joint missions. For the scenario specific research, this work implements and demonstrates the feasibility of the computer vision and navigation techniques necessary to perform inspection of a large spacecraft using the ICAROUS framework. UAVs were successfully used to simulate freeflying spacecraft in this demonstration mission, and various damage patterns were detected on a uniform metallic surface using a variety of computer vision techniques. Advanced networking and multi-agent control techniques proved capable of supporting a dynamic number of agents, allowing for efficient mission planning and completion based on a variable number of resources.The results of this work suggest that UAV platforms can be used to successfully simulate and develop control software for space-based robotics, and can support the needs of multi-agent mission collaboration.

4.0 Task 2: Optimal Control Techniques for UAV Swarms

4.1 Task Overview

The goal of Task 2 in this dissertation is to develop a new control algorithm that allows for heterogeneous platform support in multi-agent swarms. Specifically, this work focuses on allowing fleet management systems, like the custom ICAROUS-based control system developed in Task 1, to manage and control multiple different types of UAV platforms for a variety of mission objectives. This poses new challenges to control architectures as applying a uniform architecture to the swarm is no longer applicable. Each platform needs to be quantified based on its unique characteristics, including payload capabilities, operating requirements/restrictions, and other identifying factors.

Most research on these controls problems focus on homogenous systems, assuming group performance may be extrapolated based on uniform platform characteristics and dynamics (a single type of platform is used across the system). This is not the case for several missions, such as Search & Rescue [107], Structural & Materials Analysis [108], and Space Exploration [65]. A variety of platforms and sensor payloads are used in order to obtain mission success, thus indicating a heterogeneous swarm as the optimal force. There is limited capability to manage a large fleet of varied platforms using a robust control framework with current market products.

[1]Sami Mian, John Hill IV, and Zhi-Hong Mao. Optimal control techniques for heterogeneous UAV swarms. In 2020 IEEE/AIAA 39th Digital Avionics Systems Conference(DASC). IEEE, 2020 [In Press].

This work applies an optimal control technique called Decentralized Receding Horizon Control (D-RHC) in context of a search and rescue mission. Though the technique itself has been previously implemented for swarm organization, this application poses novel challenges: While victim discovery can succeed with any agent detection, hazard discovery is specialized, therefore a heterogeneous swarm is ideal. We therefore introduce a novel technique: Heterogeneous Decentralized Receding Horizon Control (HD-RHC). This new control technique takes into account an individual platform's capabilities and limitations, usually derived from sensor payload and power resources, and modifies the optimization functions used in D-RHC to factor these aspects in the swarm motion and mission planning. In this work, we define the requirements and restrictions for the control system, derive the mathematical models used to create the control scheme, and implement the controller using the python controls library. The HD-RHC controller is then integrated and tuned using the AirSim physics-based UAV simulation environment.

4.2 Deriving the HD-RHC Controller

We are extending RHC control for high-level motion planning, dictating which missions and locations each UAV platform is responsible for surveying. The swarm implements a mesh-network [109], where UAV platforms rely on their neighbors to communicate with the whole swarm.

A key contribution to this project is formulating objectives uniquely suited to specific members of the swarm without introducing the need to implement a separate controller for each different UAV. Specifically, every member of the swarm should be able to accomplish some common tasks in the mission, while members with targeted configurations will only be applicable to specific objectives. As an example implementation, we investigate a targeted surveillance mission in which the swarm must observe a given search area, and the area has specific regions of interest that require specialized sensory payloads to analyze. Figure 16 describes an example search area A with two classes of mission points.

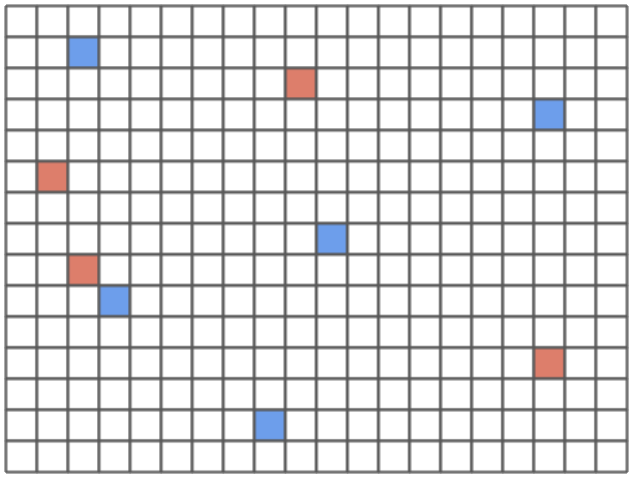


Figure 16: Search area A with mission cells of two classes depicted in *red* and *blue* [2] ©2020 IEEE

4.2.1 Problem Concept

Suppose an operational area in which a heterogeneous UAV swarm executes a mission. This swarm is composed of UAV platforms, each with a payload (sensors, equipment, etc.) that makes it suited to accomplish a specified sub-mission. Cells that contain sub-mission objectives are called mission points. The swarm is provided with an *a priori* specification of the operational area and tasked with providing surveillance of the entire area, including specialized inspection of the mission points.

4.2.2 Parameter Definition

Payload Classes

The payload that each UAV carries provides that platform with unique capabilities and properties which may not be shared amongst the rest of the swarm. The variance of these payloads within the swarm is the basis of its heterogeneity. These payload classes are described by the set:

$$P : \{p^1, ..., p^k\} \tag{4.1}$$

Note that each UAV will have exactly one payload; we consider combinations of payloads to be a unique class in our problem, as it affects overall sub-mission acceptability and energy-costs.

Mission Points

The operational area contains some finite number of mission points, however these areas of interest are best assessed with specialized payloads. For our problem, we consider that each mission point has a mission uniquely satisfied by exactly one class of payload.

As such we define the set of mission points:

$$G : \{g_1, ..., g_m\} \tag{4.2}$$

Since the mission point is only applicable to a specific payload, we will use a superscript to denote its applicability with respect to a payload class. Hence, the major mission can be described as

$$x \mid g_j^x \in G \rightarrow p^x \in P \tag{4.3}$$

UAVs Individually

Each agent in the swarm has properties shared by the entire swarm. Particularly, we assume that any region without a mission point is capable of being surveyed by any member in the swarm. While this seems intuitive, it is important to recognize that payloads do not make UAV platforms orthogonally unique to one another – otherwise the problem is reducible to operating concurrent swarms.

Nonetheless, other non-mission properties are evident. These include pose (r) and velocity ($\dot{r}$) with respect to the origin of A and the energy-cost function (e) of UAV operations. The action of a UAV, $u_i(t)$, is idealized to the transit activity moving from one place to another. This includes the acceleration to achieve target flight velocity, deceleration to reach the location of interest, and surveillance maneuvers needed to be effective with the payload. The energy-cost function, therefore, is a function of both the activity and underlying properties of the payload (mass & its own energy needs during usage), hence its definition as $e(u(t); p)$.

An additional property is considered with UAV operation: reliability (ζ). This property represents the overall health of the individual UAV platform. For our problem, it does not matter if this property is formulated as a probability or a score. What we do consider, however, is that this property diminishes over the lifetime of operation. That is to say, reliability has the following dynamics:

$$\arg\max_{t} \zeta(t) = 0 \tag{4.4}$$

$$\lim_{t\to\infty} \zeta(t) = 0 \tag{4.5}$$

$$\frac{d\zeta}{dt} < 0, 0 \leq t < \infty \tag{4.6}$$

With payload, we succinctly describe a UAV in swarm D as follows:

$$d_i \in D : \{p^x \in P, r_i, \dot{r}_i, e(u;p), \zeta\} \tag{4.7}$$

Each platform has a membership function as a consequence of sub-mission applicability. This function is idealized as:

$$m(g_k^x) = \begin{cases} 1 & p^x \in d_i \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

This concept is crucial when our problem considers both high-level (mission) and low-level (motion) planning objectives considering the payload-classes of members of the swarm.

4.2.3 Mission Planning Problem Formulation

The controller splits between high-level objective planning and low-level motion planning. High-level planning primarily focuses on the swarm satisfying the objectives stated by the mission points. A bidding scheme executed in each planning epoch is defined by the following function:

$$b_{i,j}(d_i, g_j) = \begin{cases} \|r_i - g_j^x\| - \zeta & m(g_j^x) > 0, g_j^x \in G^* \\ \infty & \text{otherwise} \end{cases} \tag{4.9}$$

Here, we define $G^* \subseteq G$ as the set of mission points that have not been visited by a UAV with an appropriate payload requirement matching d_i's payload. A mission point is won by having the lowest bid. UAVs can only have a single mission point during a planning epoch. So when a UAV is able to win multiple mission points, it will select the mission point that generated the lowest bid (greedy approach). In the event of a tie, a random-draw is negotiated by the UAVs. Bidding in this scheme is exercised at each planning epoch. This helps ensure robustness in the event that when a platform fails, another appropriate one can take its place.

In our experiments, we satisfy conditions (4.4), (4.5), and (4.6) for reliability by reducing ζ for each action taken and when a UAV visits a payload-applicable mission point. The additional reduction characterizes the reduced reliability of the UAV itself because its specialized payload was energized.

There may arise cases where a UAV simply does not win bidding on any mission point, such as when there are more platforms available than mission points. In that event, the motion planning function will drive the UAV to explore the operational area, maximizing search coverage. Likewise, we expect cases where $G^* = \emptyset$, meaning all mission points have been visited. We define the time of this event T_g.

4.2.4 Motion Planning Problem Formulation

4.2.4.1 Search Coverage Cost Search coverage itself is a function of the swarm providing a "sweep" of the entire operational area. That is, we consider that UAVs may or may not have observability into neighboring cells with their default sensor suite. We define $v_i(r)$ as the region in the operational area which is being observed by UAV d_i. Therefore, the total area searched over time by the swarm can be defined as

$$a_i(t) = \int_0^t v_i(r(t))dt \tag{4.10}$$

Therefore we can consider a search cost c_a:

$$c_a = -v_i(r_i(t); u(t)) \tag{4.11}$$

Here, $u(t) \in U$ represents an action in the action space of the UAV. In our example case, we consider the function $v_i(r; u)$ as an update to a one-hot heat map describing the operational area with newly observed cells incrementing by 1, and repeated observed cells having no change. Therefore actions that lead to greater newly discovered cells have lower costs than those which do not. One consequence of describing the covered search area is that we can detect a time T_a such that the entire operational area A is covered. That is,

$$\lim_{t \to T_a} \sum_{i=1}^{N} a_i(t) = A \tag{4.12}$$

4.2.4.2 Mission Point Visualization Cost Considering mission point selection, we consider the definition of error-distance to the mission point as the cost:

$$c_g = \begin{cases} \|r - \tilde{g}\| & \tilde{g} \in G^* \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

Two results can be observed with this formulation. First, as a UAV with appropriate mission payload reaches the current mission point, its cost will increase so that it can move onto the next unvisited mission point whose bid it won. The second is that the situation when the cost reaches zero is only when there are no more appropriate mission points that the UAV should visit. Note that we define *time of mission complete* T_g as the time when $G^* = \emptyset$.

4.2.4.3 Energy Cost Overall energy costs are important for any UAV's operation, They are critical when considering reusability and the likelihood of mission success – no one wants a UAV to fail before it completes its specified mission. Intuitively, a number of factors play into this cost, specifically its payload and the amount of force required for it to transit from one location to another. Here, we idealize the energy function of a given activity as $e(u; p)$:

$$c_e = e(u(t); p) \tag{4.14}$$

4.2.5 Full Optimization Problem

We characterize the total cost to be optimized as the weighted-sum of the aforementioned costs with weights w_a, w_g, and w_e respectively. Thus the swarm action process is defined as the solution to this optimization problem:

$$\begin{aligned} &\min_{u \in U} w_a c_a + w_g c_g + w_e c_e \\ \text{s.t..} &\|r_i - r_j\| > \delta_{min}, \quad \forall d_{i \neq j} \in D \\ &\lambda_{i,j} < \infty, \quad \forall d_{i \neq j} \in D \end{aligned} \tag{4.15}$$

We consider two constraining factors for the swarm: **safety** to keep UAVs from inadvertently colliding with one another; and **cohesion** to keep the swarm from separating too far so that mesh-network communication is impossible. These are competing constraints with respect to distance.

4.2.5.1 Safety Constraint For our problem, we consider a safety-radius of δ_{min} from which each UAV must stand-off from one another. The constraint itself does not need to be uniform. In later work, we consider such cases where directional active-sensors need to be oriented away from each other to reduce interference during mission operations.

4.2.5.2 Cohesion Constraint Unlike safety, cohesion cannot be considered as keeping within a maximum radius. Rather, consider that the swarm implements a *mesh network* [109]. We, therefore, must redefine this property. A swarm is cohesive when the swarm has an unbroken mesh network allowing any UAV to be in communications with any other UAV. Figure 17 shows how reachability is defined, with this example: : $\|A - B\| > \delta_{max}$; B is reachable because $\|A - C\| < \delta_{max}$ and $\|C - B\| < \delta_{max}$ and therefore $\lambda_{a,b}$ is finite.

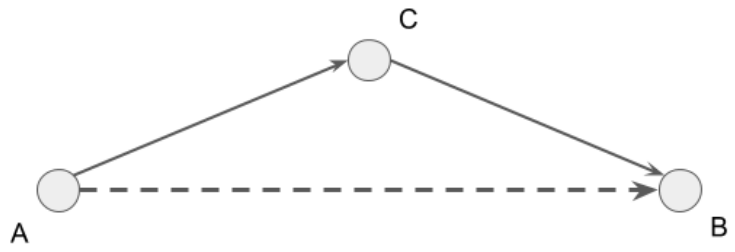


Figure 17: Node reachability example [2] ©2020 IEEE

As such, we consider the swarm itself as a weighted K-graph with weights, $\mathcal{L}_{i,j}$, between the nodes as:

$$\mathcal{L}_{i,j} = \begin{cases} \|r_i - r_j\| & \|r_i - r_j\| < \delta_{max} \\ \infty & \text{otherwise} \end{cases} \tag{4.16}$$

Here, δ_{max} represents the maximum range in which point-to-point communications is considered successful, beyond which, we consider the cost to be infinite. Following this, a number of path search techniques can be employed to provide a minimal-cost path for a message from any one UAV to reach with path-length $\lambda_{i,j}$. When there are no reachable paths between UAVs d_i and d_j, we define $\lambda_{i,j} = \infty$, hence the constraint for the swarm:

$$\lambda_{i,j} < \infty, \quad \forall d_{i \neq j} \in D \tag{4.17}$$

4.2.6 Cost Adaptation

As found in [41], this formulation allows for cost-adaptation, which is the tunability of the controller given characteristics of the swarm itself. To achieve this, we search for tuning that minimizes time to mission completion. Here, we define time to mission complete as:

$$T = \max(T_a, T_g) \tag{4.18}$$

Therefore, we tune the tuple (w_a, w_g, w_e) to minimize T.

4.3 System Design

We provide a simulation setup to tune and evaluate the controller, demonstrating its scalability. This system consists of a client-end model of each UAV of the swarm and the mesh network model, which can connect to an optional server-end AirSim instance. A single discrete controller was provided to all platforms of the swarm, demonstrating its extensibility to a heterogeneous composition. This is made available to the public for inspection and testing.

4.3.1 Discrete Controller Design

In general, the controller has three high-level functions, described in Algorithm 1. At each planning epoch, each UAV accumulates the latest telemetry and bid information of the swarm, assesses its bid for mission points, and determines its next action to take.

Algorithm 1 Discrete Controller Loop [2]

Input: $t; H : \{(r_i(t), \zeta_i(t))\} \forall i = 0..N-1$

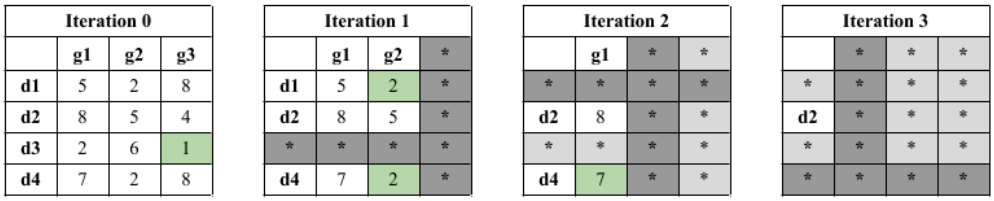
Output: $u \in U$

1: updateSwarmHistory(H)
2: g = selectMissionPoint()
3: u = computeMotionVector($g; H$)
4: **return** u

Each UAV uses the telemetry and bid information to update its understanding of the internal mission state. That is, it updates its understanding of search coverage and mission point visitation in addition to the location of each UAV in the swarm to understand constraint satisfaction.

4.3.2 Mission Point Selection

Mission point selection is assigned based upon the bidding scheme discussed in the previous section. However, each UAV can assess and determine the entire swarm's selection, given the information passed at each step through the mesh network. This is done by constructing a table of bids and allowing assignment to go in order of minimum bid.



Iteration 0

	g1	g2	g3
d1	5	2	8
d2	8	5	4
d3	2	6	1
d4	7	2	8

Iteration 1

	g1	g2	*
d1	5	2	*
d2	8	5	*
*	*	*	*
d4	7	2	*

Iteration 2

	g1	*	*
*	*	*	*
d2	8	*	*
*	*	*	*
d4	7	*	*

Iteration 3

	*	*	*
*	*	*	*
d2	*	*	*
*	*	*	*
*	*	*	*

Figure 18: Example of mission point assignments [2] ©2020 IEEE

Bid tables comprised of elements $b_{i,j}$ computed by Equation (4.9) is constructed for each mission point class. Searching in order of minimum bid, rows and columns are eliminated when an assignment is selected. By going in order of minimum bid, each UAV is able to select mission points taking the greedy approach, and ties are resolved by simply looking up the pre-generated random rolls each bid makes for itself.

Using the example described by Figure 18 above, Iteration 0 describes an initial composition of bids. UAV d_3 has the minimum bid overall in the table, and is awarded g_3. Thus for Iteration 1, the row denoting d_3 and column denoting g_3 are eliminated, and the loop continues. Here, both UAVs d_1 and d_4 provide the lowest winning bids for g_2; in this scenario, d_1's random-draw was a lesser value than that of d_4, so it is awarded g_2. Iteration 2 demonstrates that though d_4 did not win the bid of g_2, it instead wins g_1. In Iteration 3, with all mission point columns eliminated, d_2 is left without assignment and thus favors search when evaluating action cost. Note that assignment also ends in the case where there are no UAVs left in the table, yet mission points remain.

4.3.3 Action Selection

With a mission point assignment, we can now consider potential actions. For a proof-of-concept, we explore a discretized grid map and provide an action-space such that each UAV selects which neighboring cell to visit (four cardinal directions and four diagonal directions). To keep the computational load small for this study, the controller does not consider potential next-states of the swarm. This is done so that the action space is not **size 8N** where **N** is the size of the swarm. Future work will consider reducing this space.

Algorithm 2 Motion Vector Selection [2] ©2020 IEEE

Input: $g \in G^*; H : \{(r_i(t), \zeta_i(t))\} \forall i = 0..N-1$

Output: $u \in U$

Initialisation : $U^* = \emptyset$

for $u \in U$ **do**
  if $\|(r_i(t) + u) - r_j(t)\| \geq \delta_{min}$ and $\lambda_{i,j} < \infty \; \forall j \neq i$ **then**
    Add u to U^*
  end if
end for
if $U^* \neq \emptyset$ **then**
  $u^* = \arg\min_{u \in U^*} w_a c_a + w_g c_g + w_e c_e$
else
  u^* set to null-action
end if
return u^*

This example implementation first filters for actions that do not violate the optimization constraints, then searches for the action with minimal cost. In the event that every action violates the constraints, the UAV remains stationary (the "null-action") for its safety; this is necessary for the case where the entire swarm begins from a common staging location.

4.3.4 Controller Tuning

The weights w_a, w_g and w_e are calibrated to strike a balance between seeking to cover the search area and favoring mission point visitation. To do so, we consider two different techniques: Monte-Carlo (MC) and Simulated Annealing (SA) inspired by [40]. Both are uninformed search techniques that are commonly applied to such problems. As such, several trial iterations with different weight settings are analyzed to find the optimal combination, which minimizes time to mission complete (see Equation (4.18)).

4.4 Experiments & Analysis

Here, we execute two experiments to evaluate the controller: Benchmarking Tuning Effort, and Assessing Scalability of Tuning. In all cases, we are predominantly interested in three aspects: 1) Mission point visitation, 2) Search Area Coverage, and 3) Efficiency of Search Coverage. While the first two are parts of the controller's cost function, the third is interested in understanding how frequently cells in the space are revisited.

To tune, a search area of 64x64 is proposed with two different payload types. There are four UAVs whose positions were chosen arbitrarily near the origin. There are 20 mission points split evenly among the payload types. Our preliminary modeling was integrated with AirSim for accuracy and visual analysis.

Through the remainder of the document, we will review both mission coverage results via odometry and the efficiency of the swarm. Figure 3 provides an example: The coverage plots (a) provide an assessment of mission point visitation and total search area coverage; mission points (in black) which are not visited have no odometry paths joining them. Overall coverage can be understood by observing the odometry data spanning the graph field. The heat map (b) represents cells that have been repeatedly visited with hot/red, and cool/blue represents rarely or unvisited cells. This offers an effective understanding of loitering behavior.

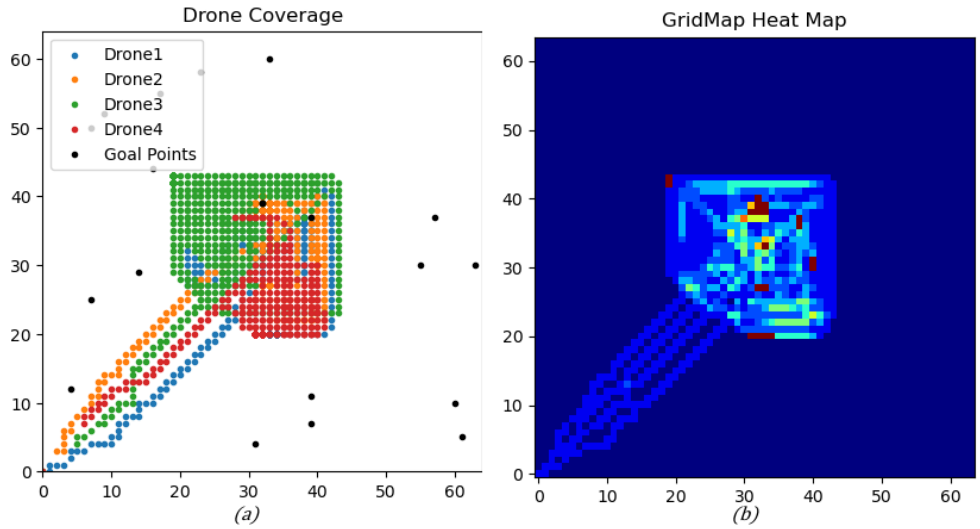


Figure 19: Example mission coverage and efficiency plots regarding a failed configuration [2] ©2020 IEEE

4.4.1 Controller Tuning

Cost-adaptation is a key feature of D-RHC, and so it too is required of HD-RHC. There are several ways this controller can be tuned. However, we chose to explore two uninformed search techniques, Simulated Annealing (SA) and Monte-Carlo Method (MC).

In these setups, mission points are randomized between each simulation trial; UAV starting locations were not due to simulator configuration requirements. In all cases, it is expected that poor cost-weights may be analyzed, and so the simulations are capped with a maximum step-count.

4.4.1.1 Simulated Annealing Simulated Annealing is a randomized search that attempts to explore local neighborhoods searching for optimal solutions [110]. This tuning method was chosen as it is a popular default heuristic search to locate local-optima and explore sub-optimal regions about it. It is, however, a non-deterministic search, so even when optimal configurations are determined after the set, it is recommended that multiple setups are executed and results compared with each other.

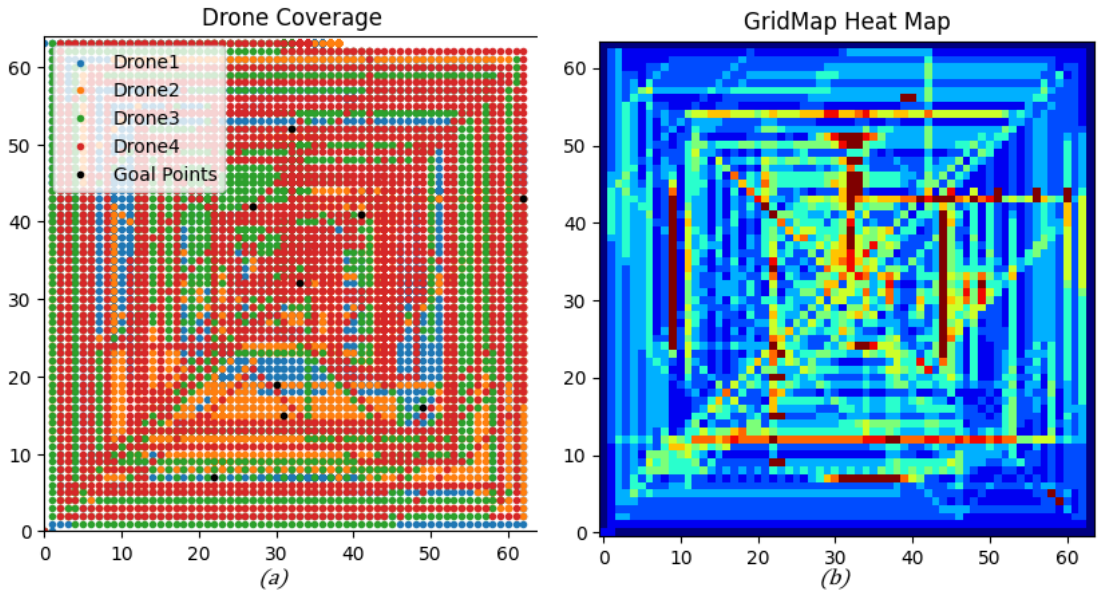


Figure 20: Example results from SA Tuning [2] ©2020 IEEE

In our implementation, we restricted the search space to only the positive-domain as negative weights would create a repulsion-effect in the controller (for example, if $w_g < 0$, then the UAV would favor going away from its assigned mission point). The search itself began at (1.0, 1.0, 1.0), randomly searching the neighborhood with a maximum difference magnitude of 1.0 from its currently selected "best" candidate configuration.

4.4.1.2 Monte-Carlo Method Monte-Carlo Method is another uninformed search technique in that the search randomly polls about a neighborhood given a probability distribution. Readers wishing to explore implementations can refer to [111] and [112]. This method was also chosen because of its popularity and simplicity to implement. We restricted the search space to weights $0 < w \leq 1$ with uniform distribution.

Figures 21, 22, and 23 show a selection of results from the trial runs, where we look at the dynamics of each of the components of the cost function (by setting the other two weights to 0). Figure 24 shows the results of one of the highest performing trials.

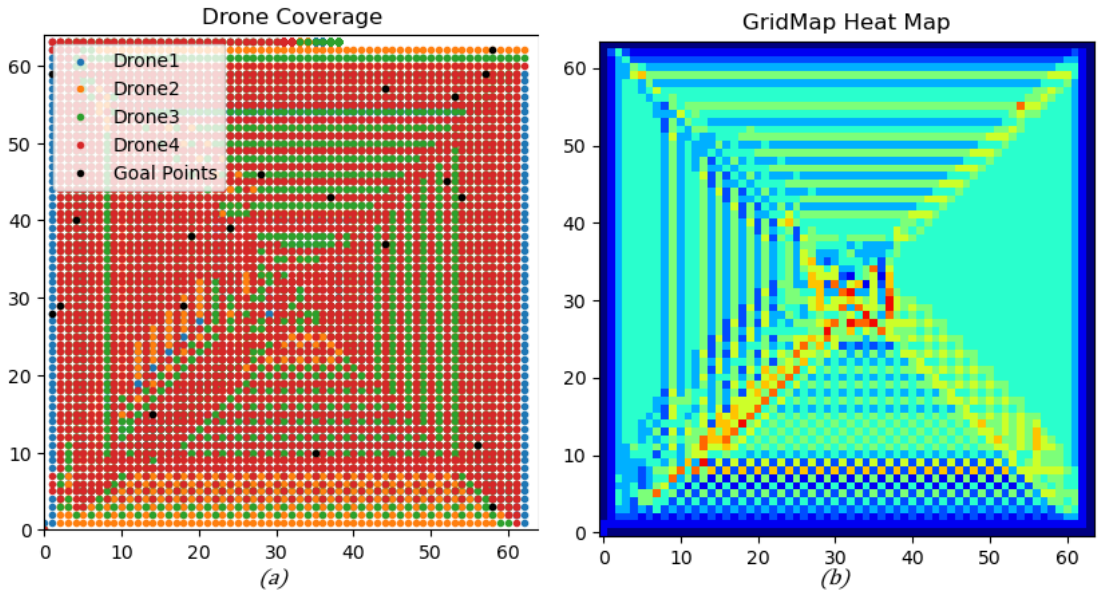


Figure 21: MC trial with weights $w_a = 1$, $w_g = 0$, $w_e = 0$ [2] ©2020 IEEE

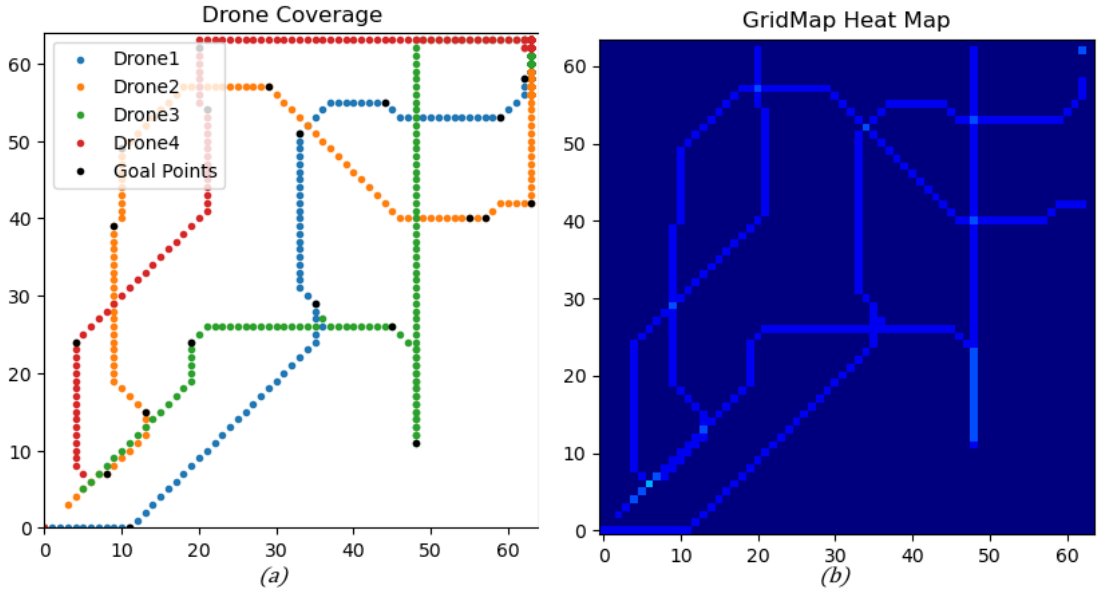


Figure 22: MC trial with weights $w_a = 0$, $w_g = 1$, $w_e = 0$ [2] ©2020 IEEE

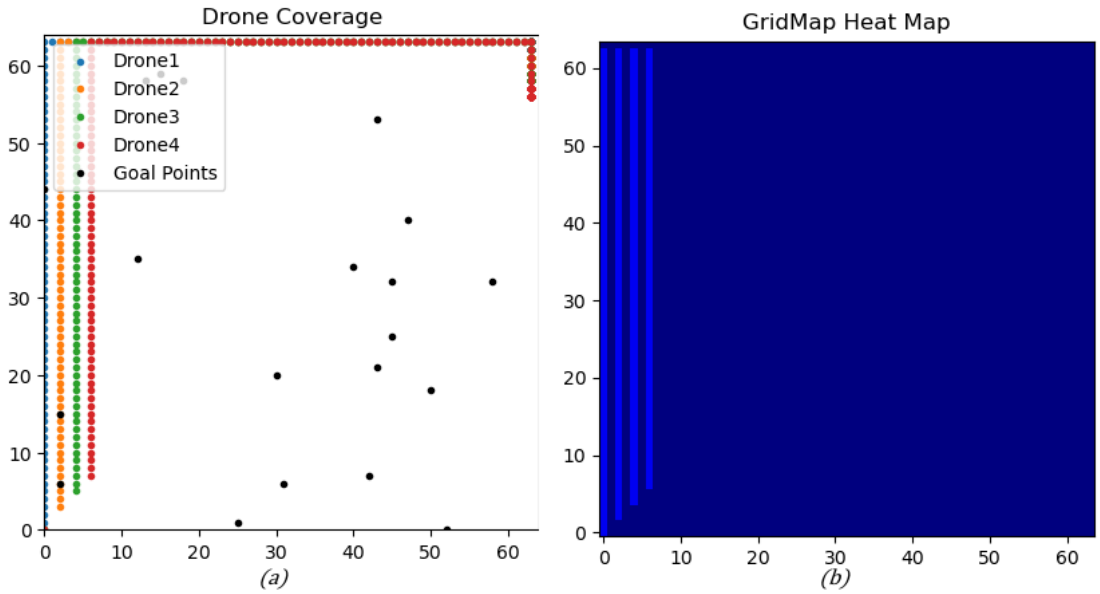


Figure 23: MC trial with weights $w_a = 0$, $w_g = 0$, $w_e = 1$ [2] ©2020 IEEE

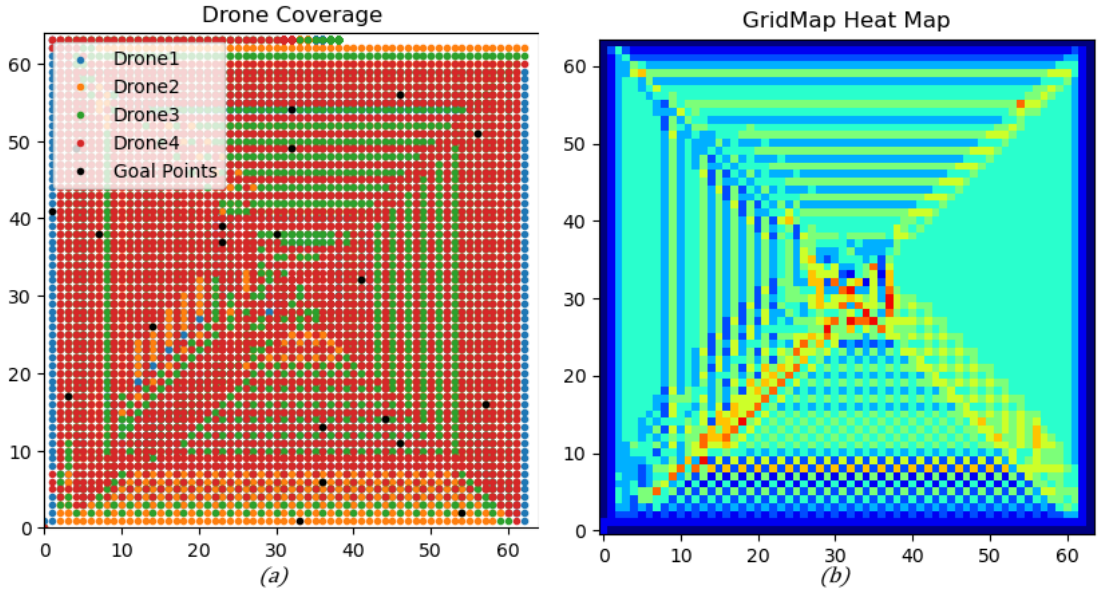


Figure 24: MC trial with the best overall performance [2] ©2020 IEEE

4.4.2 Scalability Experiments

We also analyze the applicability of tuning parameters to larger mission areas and swarms. In tuning, the swarm itself was a four UAV configuration with only two payload classes. Here, we extend to a larger area (100x100) with five payload classes and a swarm size of 48 UAVs.

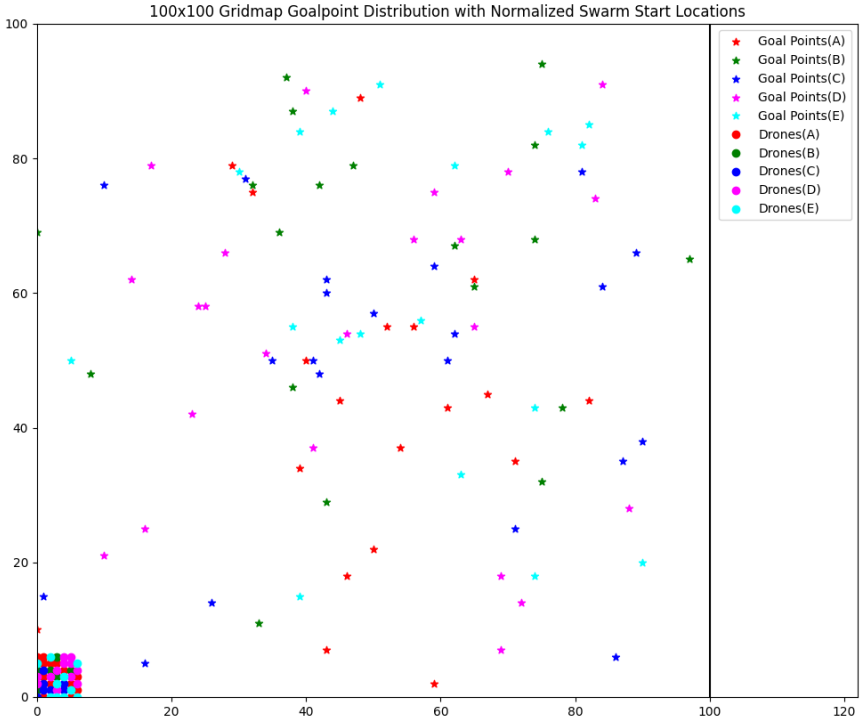


Figure 25: Normalized swarm initial locations with mission point distribution [2] ©2020 IEEE

There are two starting configurations for the swarm. The first is a pre-deployment starting location: all of the swarm agents are starting off in a cluster at the edge of the map, as if being deployed out into the environment for the first time (i.e. "first flight" configuration). The other configuration has each of the swarm agents starting in randomized locations spread out throughout the environment; this is to simulate a pre-deployed swarm obtaining a new mission after operating for some time, or a pre-distributed swarm receiving its first mission orders.

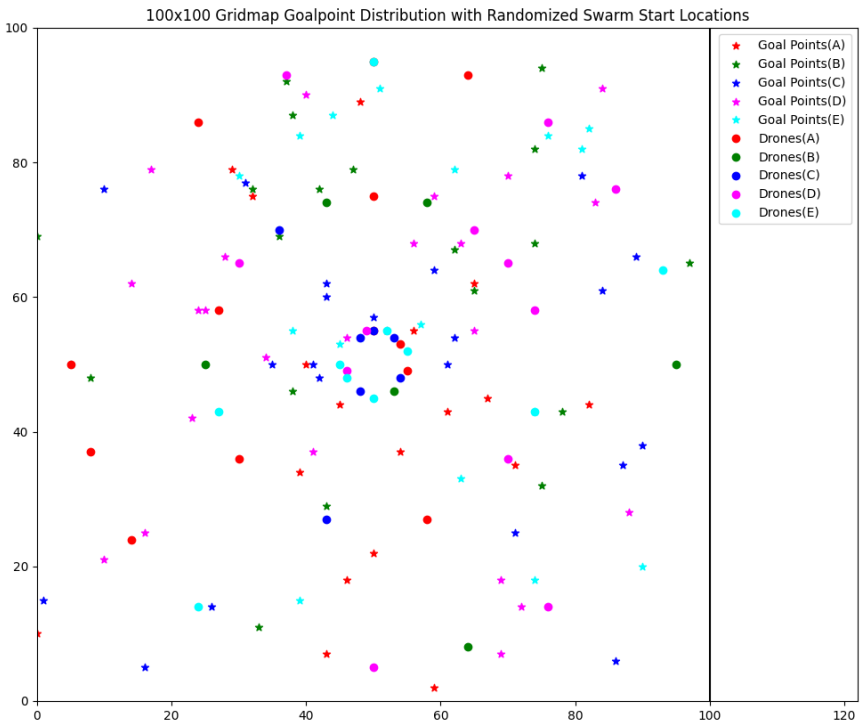


Figure 26: Randomized swarm initial locations with mission point distribution [2] ©2020 IEEE

4.5 Discussion

After several iterations of tuning and experimentation with the HD-RHC controller, we were able to discover the impact of each of the weights on the controller, and better categorize how to configure the HD-RHC controller for different mission approaches. The system allows for a variable number of heterogeneous agent types to be considered and can optimize for different mission costs (efficiency, coverage, mission-completion speed). The integration of this controller with AirSim allowed training and evaluation of the system using high-fidelity UAV odometry. This provided a realistic understanding of the dynamics present within these autonomous systems.

4.5.1 Mission Completion Analysis

Equation (4.15) describes a cost function balancing three different trade-offs. Understanding the dynamics between each is crucial to understanding phenomena whilst tuning the controller. For example, in Figure 20, odometry (a) shows mission point visitation, but demonstrates a missed region of coverage; the heat map (b) demonstrates contention in favoring which direction to expand search coverage as highlighted by the hot/red rows & columns. Intuitively, costs focusing exclusively one one of the three demonstrate mission focus in that area exclusively: non-zero weight for w_a maximizes area coverage; non-zero weight for w_g maximizes mission point visitation; non-zero weight for w_e favors momentum preservation.

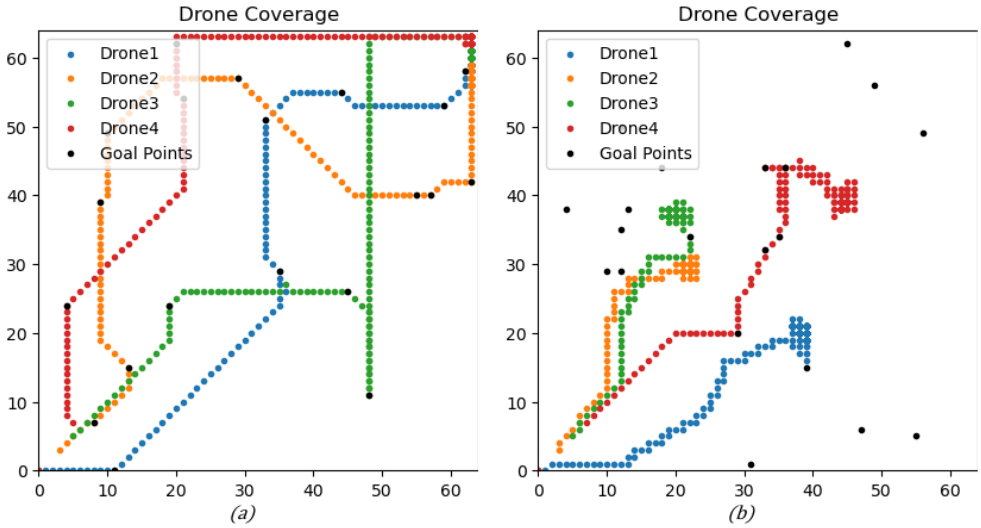


Figure 27: Odometries caused by slight perturbation of w_a [2] ©2020 IEEE

Here, we noticed certain effects: particularly when $w_a \geq 0.1$, we begin to observe mission points never being explored. Further analysis of the odometry noted that the controller would oscillate between favoring a mission point visitation or favoring search coverage exploration near $w_a \sim 0.1$. An interesting revelation is how sensitive the controller was to changes for each of the weights; changing one weight would cause unexpected interference with the other respective weight parameters. As seen in Figure 27, by adding a minor weight to search area coverage cost, the swarm begins to miss mission points entirely and struggles

by alternating its focus between mission point visitation and coverage, essentially leading to a standstill. From this, we conclude that naive uninformed searches cannot produce optimal tuning without exhaustive highly-resolute searches. Better calibrations are possible when the weights are first set to cost-normalized values. As a result, when mission costs are changed, these parameters are not easily transferable. Nonetheless, when a new cost function is considered holistically, a set of initial weights can be derived.

4.5.2 Tuning Scalability

Determining optimal weights with a smaller swarm & mission-scope, then applying them to a larger swarm hasn't been thoroughly analyzed. So after deriving an optimal tuning, we applied it to the large swarm configurations mentioned in the Scalability Experiments section. Below are the results of these trials.

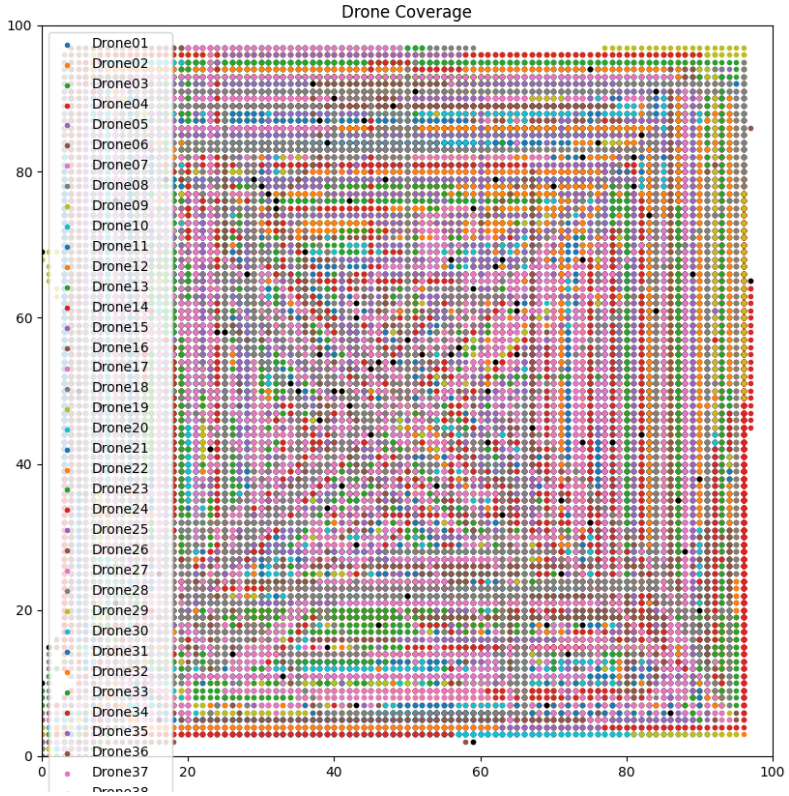


Figure 28: Coverage with normalized start location [2] ©2020 IEEE

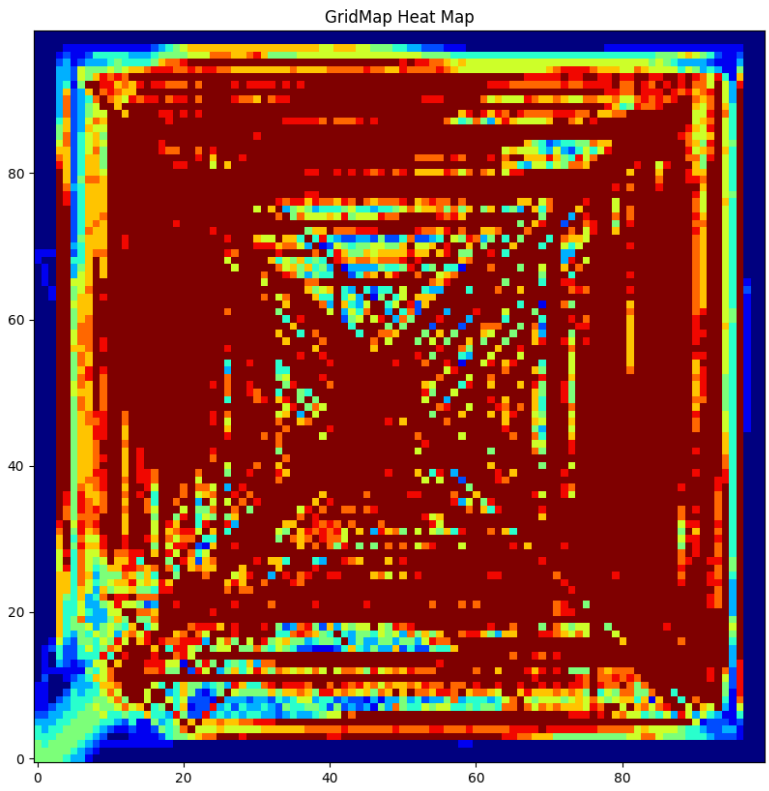


Figure 29: Swarm efficiency with normalized start location [2] ©2020 IEEE

In both cases, the swarm was able to achieve full search coverage and mission point visitation in roughly equal runtimes (2934 and 3288 time steps, respectively) with the same parameters used in the four UAV / two payload classes normalized configuration. This effectively demonstrates that tuning in simulation need not match the deployed UAV size nor capability. Rather, tuning only needs to focus on striking a balance between objective costs. Therefore, it is possible to increase training efficiency by using only a handful of models instead of a comprehensive modeling of the target swarm.

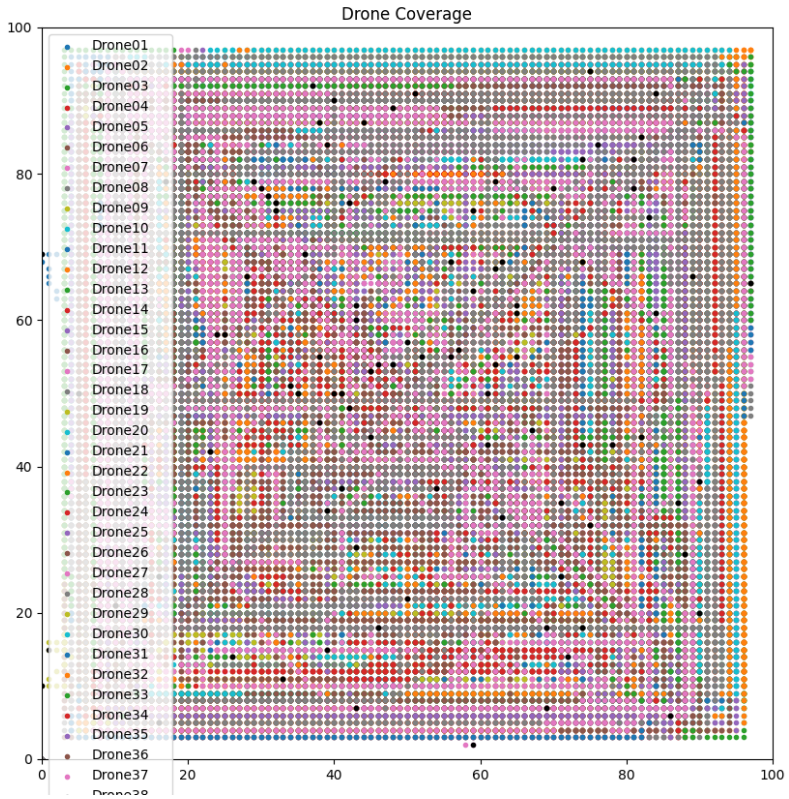


Figure 30: Coverage with randomized start location [2] ©2020 IEEE

4.6 Chapter Conclusion

For this portion of the dissertation work, we were able to build on existing work using receding horizon control to manage multiple UAV agents, by introducing a new type of controller: Heterogeneous Decentralized Receding Horizon Control (HD-RHC). With HD-RHC, we are now able to provide scheduling and planning for a heterogeneous UAV swarm in both a known and unknown environment. Each platform can be assigned a unique payload, which correlates to specific mission capabilities (sensing, actuation, etc.). Scheduling and planning work the same as a standard RHC system, but now include a scalable number of unique platforms and mission capabilities. This controller also considers communication constraints, currently designed for use with a mesh network of variable strength. This system

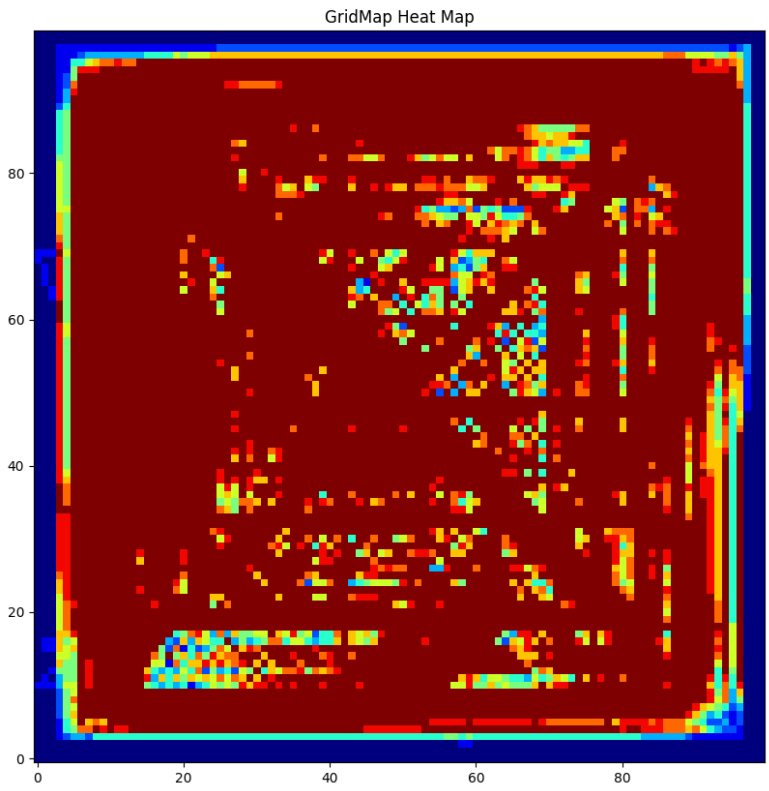


Figure 31: Swarm efficiency with randomized start location [2] ©2020 IEEE

is able to be trained both headless, using basic graph representation for the search space and UAV positions, as well as with the AirSim high-fidelity simulation environment. Through initial tuning and testing, a small range of weights has been selected that allow for fast mission complete time and full search coverage of unknown areas. The sensitivity of the controller to the weights allows for a high level of mission adaptability; with proper tuning, the HD-RHC controller can be used for a number of different mission outcomes, such as large-scale area exploration (search and rescue), as well as mission-focused deployments (i.e. package delivery).

5.0 Task 3: Novel Sensor Fusion Techniques for UAV Perception

5.1 Task Overview

The final task in this dissertation is focused on platform-specific solutions for motion planning and navigation, using new and existing sensor techniques. Although the fleet management system and HD-RHC controller are able to direct UAV platforms to specific goals, each of the UAVs still needs to be able to reach that mission objective, by navigating through the surrounding environment. Currently, there are a multitude of techniques used for motion planning and obstacle avoidance that are used onboard modern UAVs. However, one of the overarching goals of this research is to make sure the fleet management system is able to work with novel, cutting edge sensors and payload systems. So for this work, we decided to turn to a dynamic vision sensor, also known as an event camera or neuromorphic camera. A majority of the novel research here focused on integrating the event camera into a UAV's flight control system, specifically for obstacle avoidance. Reinforcement learning Is used to train the flight controller for perception inputs, using the event cameras; this was implemented and tuned using a UAV simulation environment. This method was then integrated into the fleet management system and compared with other approaches using event cameras.

[1] Nikolaus Salvatore, Sami Mian, Collin Abidi, and Alan George. A neuro-inspired approach to intelligent collision avoidance and navigation. In 2020 IEEE/AIAA 39th Digital Avionics Systems Conference (DASC). IEEE, 2020 [In Press].

5.2 Task-Specific Background

Unmanned Aerial Vehicles (UAVs) are ideal remote-sensing platforms due to their ability to maneuver quickly at high speeds in terrains which may be difficult for humans to access. With a wide variety of sensor payloads available, UAVs are able to operate in diverse environments while performing complex sensor tasks. For autonomous vehicles to safely operate in shared airspace, their collision avoidance system should have the ability to react as quickly as possible to avoid objects on an intersecting trajectory. Designing a control system that achieves this goal involves coupling sensors with a control architecture that minimizes processing overhead and response time. A common choice of sensor is the image-based camera, which has seen widespread use in robot perception tasks. One of the major limitations, however, for object avoidance using image-based data is the sampling rate of conventional cameras and laser-based systems. In high-speed autonomous navigation situations, low latency is key to successfully performing avoidance maneuvers. In general, the faster that a UAV moves, the more detrimental the sensing latency [82].

As sensor latency is a restricting factor for UAV agility, choosing a sensor that balances low latency with sufficient visual information should lead to improved actuation ability. Among novel sensors proposed, the Dynamic Vision Sensor (DVS) [113] is an emerging type of event-based camera that registers changes in luminosity to construct a power-efficient visual representation of the environment. The DVS has high dynamic range, low latency, and low computational overhead, making it an interesting on-board sensor candidate for autonomous systems that require low latency.

Finding a good control policy is important for ensuring that a UAV not only performs actions quickly, but also appropriately. Recently, a large body of research using deep learning (DL) for robotic perception tasks has emerged for these specific applications. More specifically, reinforcement learning (RL) methods have consistently demonstrated promise in their ability to train an agent to make good decisions when facing problems with difficult formulations for standard deep learning techniques. Reinforcement learning algorithms typically find a good control policy by training an agent iteratively within a simulated environment for many thousands, or even millions, of trials. The Deep Q-Learning method combines advances in deep and reinforcement learning techniques to allow an agent, guided by the Deep Q-Network (DQN), to select actions in a way that maximizes cumulative future reward [91].

5.3 Methodology

For our training procedure, we opted to create a series of lanes in Unreal Engine 4 that feature elementary object shapes as obstacles with which to train our DQN architecture for collision avoidance. Training trials were performed by repeatedly commanding the simulated drone to fly down each obstacle course lane and allowing the DQN architecture to issue collision avoidance commands as needed. Successive frames were pulled from the simulated drone's on-board camera and passed through an event-based vision simulator in order to mimic the behavior of event-based sensors. After training the conventional DQN architecture, observations were pulled from successful trials and used to train an equivalent spiking architecture. Both trained networks were then evaluated on additional, distinct obstacle course lanes created for the purposes of testing.

5.3.1 Event Simulation

To assess the benefits of using dynamic vision sensing for collision avoidance within the AirSim environment, it is necessary to emulate the behavior of the sensor using the RGB luminance values of frames provided by the simulated drone. In this work, we use an emulation method adapted from [77] in which the brightness change, in terms of logarithmic luminance calculated at each pixel, is accumulated overtime and compared to a crossing threshold chosen at runtime as shown in Equations 1 and 2.

$$L(u,t) = log(0.299I_R(u,t) + 0.587I_G(u,t) + 0.114I_B(u,t)) \tag{5.1}$$

$$p(u,t) = \begin{cases} +1, & if L(u,t) - L(u,t-1) > \Delta L \\ -1, & if L(u,t) - L(u,t-1) < -\Delta L \end{cases} \tag{5.2}$$

In Equation 1, $L(u,t)$ indicates the log luminance of the given pixel at pixel position $u = [x,y]$ at time t, while $I(u,t)$ represents the luminance magnitude with subscript indicating the color channel of the corresponding values. Equation 2 shows the generation of simulated DVS events with polarity $p(u,t)$ based on the comparison of the log luminance difference between frames with the chosen crossing ΔL threshold. In order to mimic the spiking behavior of a DVS, the number of events generated at a given pixel is determined via linear interpolation of the log luminance values observed between subsequent frames as shown in Equation 3

$$N_s(u,t) = min(N_b, \frac{\Delta B}{\Delta L}(u,t)) \tag{5.3}$$

where N_s indicates the number of events generated at pixel u, N_b is a maximum constraint placed on the number of events that may be generated, and ΔB is the total log luminance

change occurring at pixel u and at time t. In order to accomplish real-time event simulation for DQN training, these operations were performed in parallel using OpenCV's built-in GPU acceleration to produce both matrices containing numbers of events generated at each pixel $N_s(u,t)$ and a gray-scale, integrated frame with values shown in Equation 4.

$$I(u,t) = \begin{cases} 255, & p(u,t) = +1 \\ 125, & p(u,t) = -1 \\ 0, & p(u,t) = 0 \end{cases} \tag{5.4}$$

In order to train the spiking neural network architecture with realistic DVS data, the number of events $N_s(u,t)$ is used to generate a one-dimensional stream of events with the native DVS form (x,y,t,p), where (x,y) indicates the x and y coordinates of the pixel, t is the timestamp of the generated event, and p is the polarity of the event as calculated above. The timestamp of each event generated at the same pixel is incremented by a value relative to the time interval between received conventional frames as shown in Equation 5

$$t = t_o + \frac{\Delta T}{N_s(u)} \tag{5.5}$$

where t_o is the timestamp of the first event generated at the current time-step and ΔT is the time interval between frames in microseconds. In order to properly imitate a DVS, it is also necessary to sort these simulated events into temporal order according to their associated timestamps. Figure 2 shows a sample image taken from within the AirSim environment, while Figure 3 displays the corresponding integrated frame output yielded from event-based vision simulation.

Figure 32: Drone view of UE4 environment [3] ©2020 IEEE

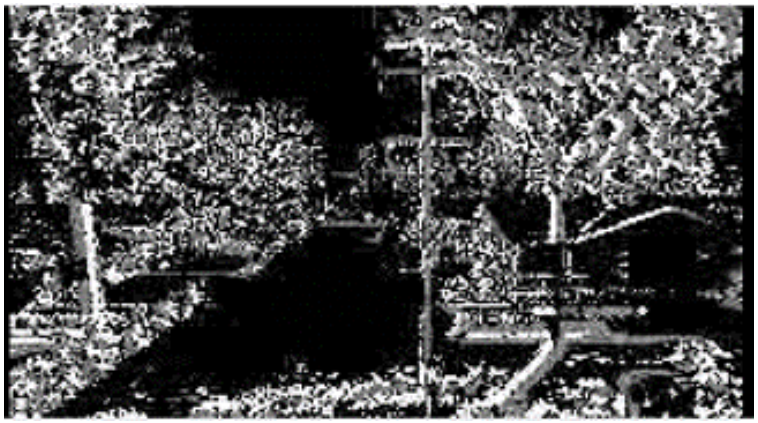


Figure 33: Integrated frame of event simulation output [3] ©2020 IEEE

5.3.2 Deep Q Network Training

In order to reduce the expected training time, we opted to use a shallow architecture for the Deep Q network, consisting of only two convolutional layers for feature extraction and two fully connected layers for action value determination. Similar to other previous works, the input to this network is a stack of three consecutive frames taken at regular intervals during flight time. The output of the network indicates the expected value of each of the possible avoidance maneuvers available to the agent as detailed in Table 1. Although the network is given five total maneuvers to choose from, the exact nature of these avoidance maneuvers is arbitrary and dictated by the options available within the AirSim API. The overall DQN architecture is shown in Figure 34.

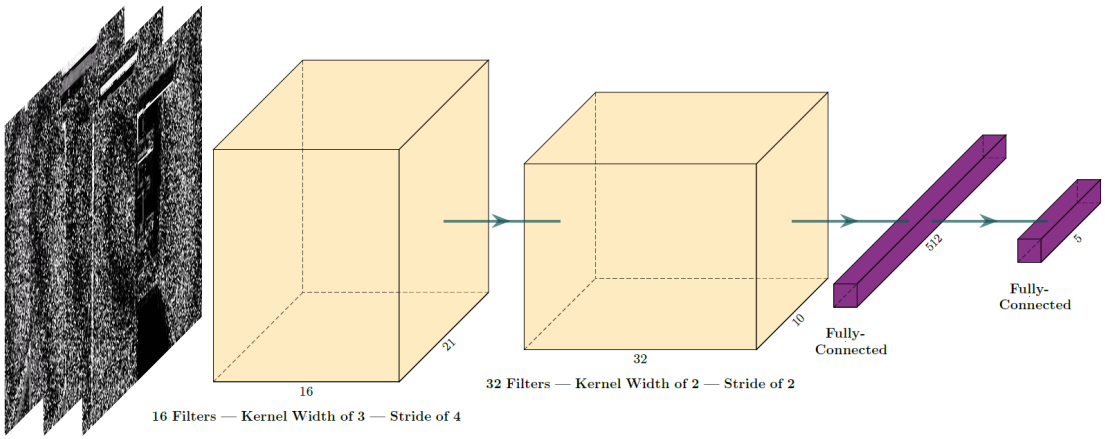


Figure 34: DQN architecture [3] ©2020 IEEE

Table 1: Network Output Actions [3] ©2020 IEEE

$ActionNumber$	$AvoidanceManeuver$
0	"Avoid Left"
1	"Avoid Right"
2	"Avoid Up"
3	"Avoid Down"
4	"Maintain Course"

The reward function used to assign values to the drone actions and states was chosen in order to reward clear progress down the current lane as well as penalize both collisions and repetitive, unnecessary collision avoidance maneuvers. In our scheme, the current state of the drone considers both the current, visual observation of the sensors and the y position and collision state as relayed by the AirSim environment. Given the arrangement of the training lanes used, movement in the positive y direction corresponds to successful progress in a lane. The reward function is therefore defined as follows

$$\begin{gathered} R(s,a) = \gamma_P \Delta y - \gamma_C C - \gamma_a a \\ \gamma_a a = 0, if a = 4 \end{gathered} \tag{5.6}$$

where $\gamma_P, \gamma_C, \gamma_a$ are the reward values assigned to progress, collisions, and actions respectively, Δy is the distance traveled down the current lane, C is the number of collisions currently registered, and a is the index of the last action taken. As indicated, the reward is not penalized when the agent chooses not to make a collision avoidance maneuver.

In order to train the network, state, action, reward, and state transitions are stored in an experience replay buffer, which is randomly sampled and batched to calculate network weight updates at regular intervals. For network training, we used a Double Q learning approach [114] in which two, identical networks are used to estimate the current Q value and target Q value separately. Every n steps of the simulation, weights of the DQN network are updated according to

$$\Delta w = \alpha[R + \gamma max_a \hat{Q}(s', a, w_t) - \hat{Q}(s', a, w)]\nabla\hat{Q}(s', a, w) \tag{5.7}$$

where $\hat{Q}(s', a, w)$ represents the value prediction of the DQN network, $max_a\hat{Q}(s', a, w_t)$ represents the discounted maximum value predicted by the target network, and α is the learning rate. After training on each batch is completed, the weights of the target network are updated with a chosen τ.

$$w_t = \tau w + (1 - \tau)w_t \tag{5.8}$$

As with most reinforcement learning schemes, a decaying exploration value, ϵ, was used to indicate the probability of the agent taking a random action during the training procedure, regardless of the DQN's output.

5.3.3 SNN Conversion and Evaluation

Although several methods exist for performing reinforcement learning natively with spiking neural networks, we focused on using more mature training methods in tandem with the conventional reinforcement learning network in order to encourage faster convergence of the SNN. In order to create and train the spiking architecture, we made use of the NengoDL framework's converter functionality, allowing us to create a spiking model equivalent to

the conventional DQN previously trained. This spiking DQN was then trained in a semi-supervised fashion, using a labelled dataset collected using the experiences of the trained, conventional DQN. This dataset was formed by taking the stacked input frames representing the state of the agent at regular intervals in each lane and labelling it with the action index chosen by the conventional DQN. To avoid training the SNN with erroneous inputs, observation and action pairs were only recorded from trials in which the conventional DQN successfully traveled the entire length of the lane. After training the SNN in the NengoDL framework with this labelled dataset, the SNN performance was then evaluated using integrated frames with predetermined presentation times on the additional testing course lanes. Additional parameters of synaptic scaling and smoothing were also included as necessary with the spiking function of the network, although no extensive parameter optimization of these values was explored.

5.4 Experiments

As mentioned previously, network training was conducted on a series of five obstacle course lanes built in Unreal Engine 4 with a series of elementary shaped obstacles as pictured in Figures 35 and 36. For each trial, the simulated UAV was commanded to progress down a given lane with the trial terminating after reaching the endpoint, i.e. 150m in the y direction, or registering at least two collisions with an object in the environment. At the beginning of a predefined command window, the last three event-based frames from the simulation were stacked and passed to the DQN as input, after which the network's action and subsequent reward were then stored in the experience replay buffer.

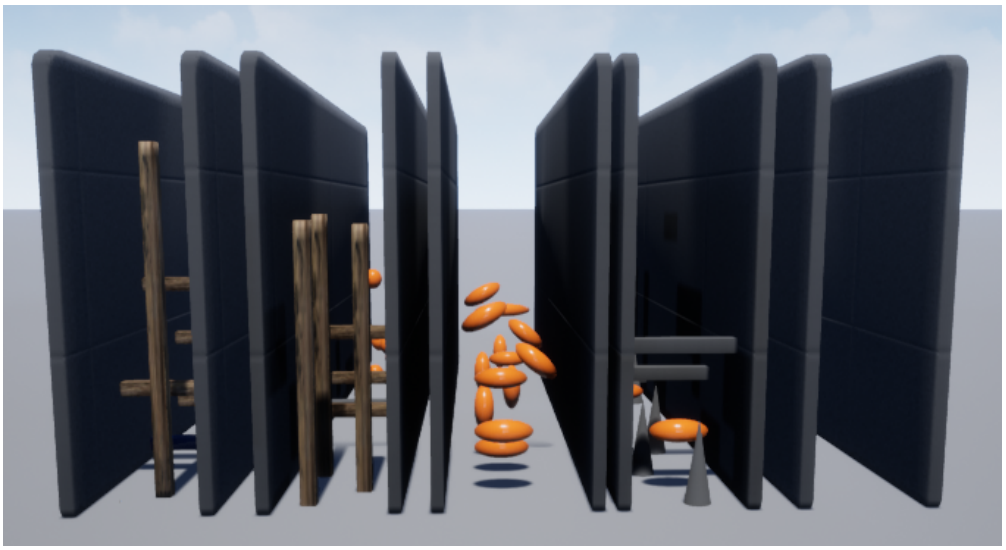


Figure 35: Training environment: front view [3] ©2020 IEEE

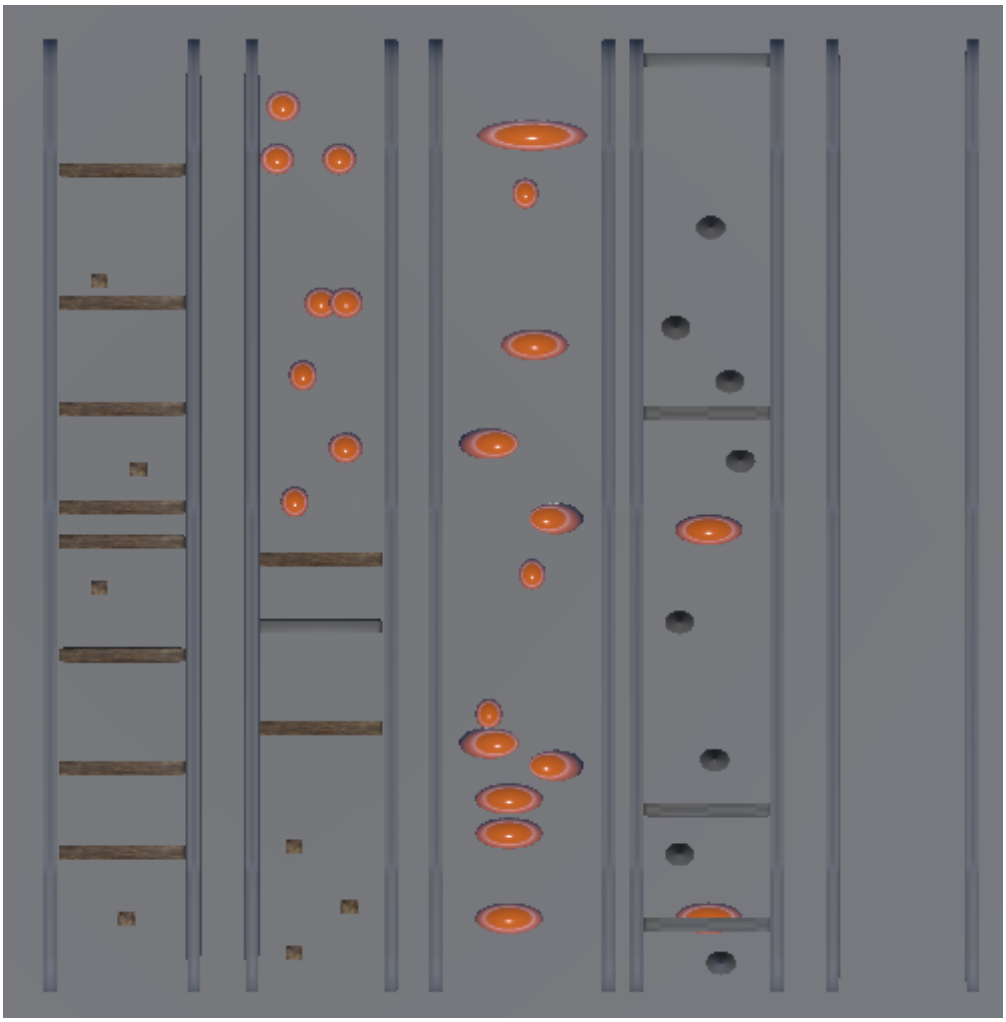


Figure 36: Training environment: top view [3] ©2020 IEEE

After a certain number of experiences, i.e. state, reward, action, and transition pairs, are collected, the AirSim simulation is paused and the weights of the DQN networks are then updated as stated previously. Once sufficient training was performed with the conventional DQN, a series of trials were performed on the training lanes where all observations occurring within successful lane trials were recorded and used to create the labelled dataset for SNN training. Finally, the performance of both the conventional and spiking network architectures were then evaluated using the distinct testing lanes pictured in Figures 37 and 38.

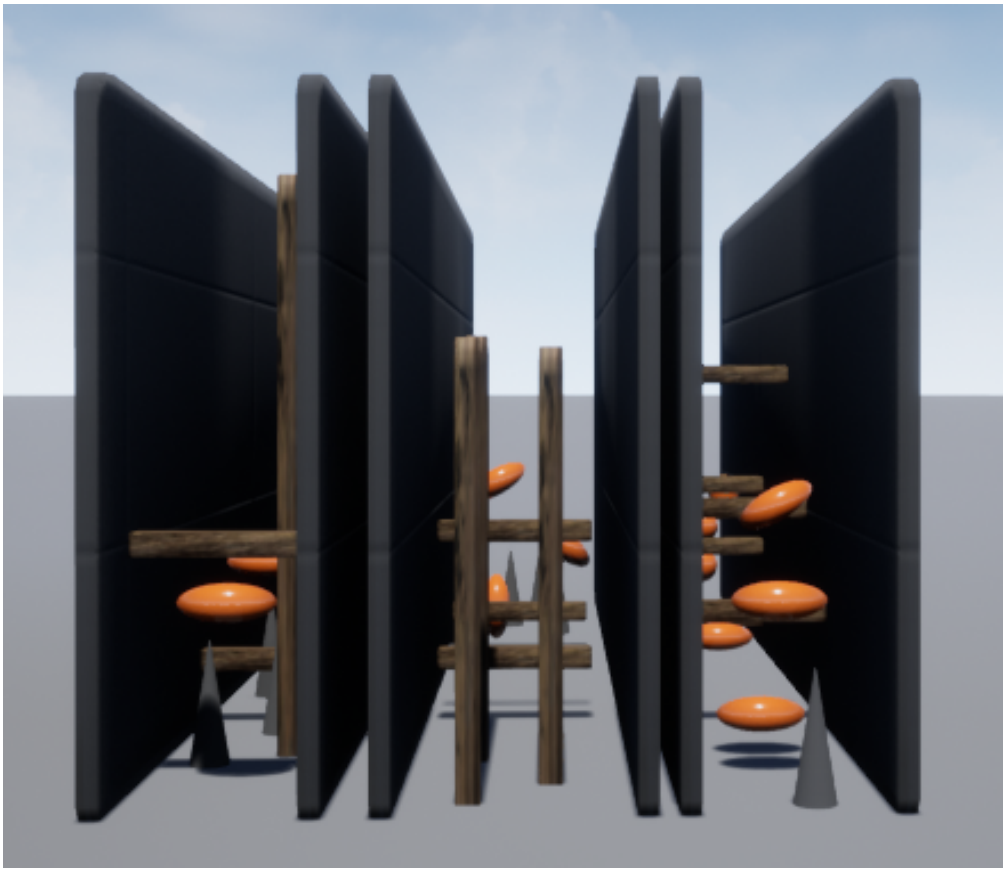


Figure 37: Testing environment: front view [3] ©2020 IEEE

Figure 38: Testing environment: top view [3] ©2020 IEEE

5.5 Results

50,000 training trials were performed for conventional DQN agent, evenly distributed amongst the five lanes. Each trial consisted of the drone travelling down a given course lane with the global Cartesian coordinates, lane number, and trial number being recorded at the trial's conclusion. Training trials terminate when either the drone reaches the end of the lane ($y \geq 145\ meters$) or collides with an object more than once, which is recorded as either a success or failure respectively. A log-luminance threshold of $\Delta L = 0.25$ was used in the event-based simulation of the drone's observations. The results in Figure 39 show the rate of success over time, whereas Figure 40 shows a spatial visualization of successes and failures

over the training period. While Figure 39 highlights the general trends in collision avoidance performance over time, Figure 40 serves to pinpoint which obstacles are most difficult for the drone to avoid as well as show the successful distance travelled for trials marked as failures. Table 2 lists the statistical metrics calculated to analyze the correlation between success rate and training iterations during the training process.

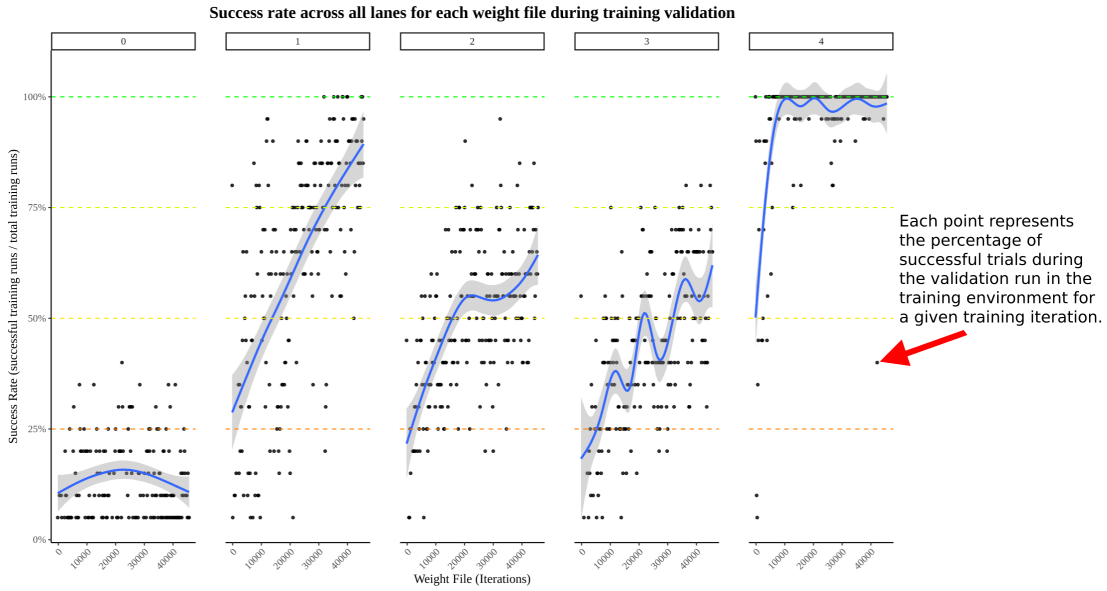


Figure 39: Success rate vs. training iterations for training lanes [3] ©2020 IEEE

After validating the training data and weights, a set of testing trials was conducted using the testing environment pictured in Figures 37 and 38. For these trials, the operating parameters were the same as those used for training (namely, using a DVS simulation threshold of 0.25) as well as the data collection process. The success rate results are shown in Figure 41 as well as the corresponding correlation analysis in Table 3.

In order to compare the results of this training with a random baseline, a separate set of trial runs were performed where the UAV was directed to take a random action during each timestep. Additionally, the spiking architecture, trained using labelled data derived from the most trained, conventional DQN, was evaluated on the same testing lanes as the conventional

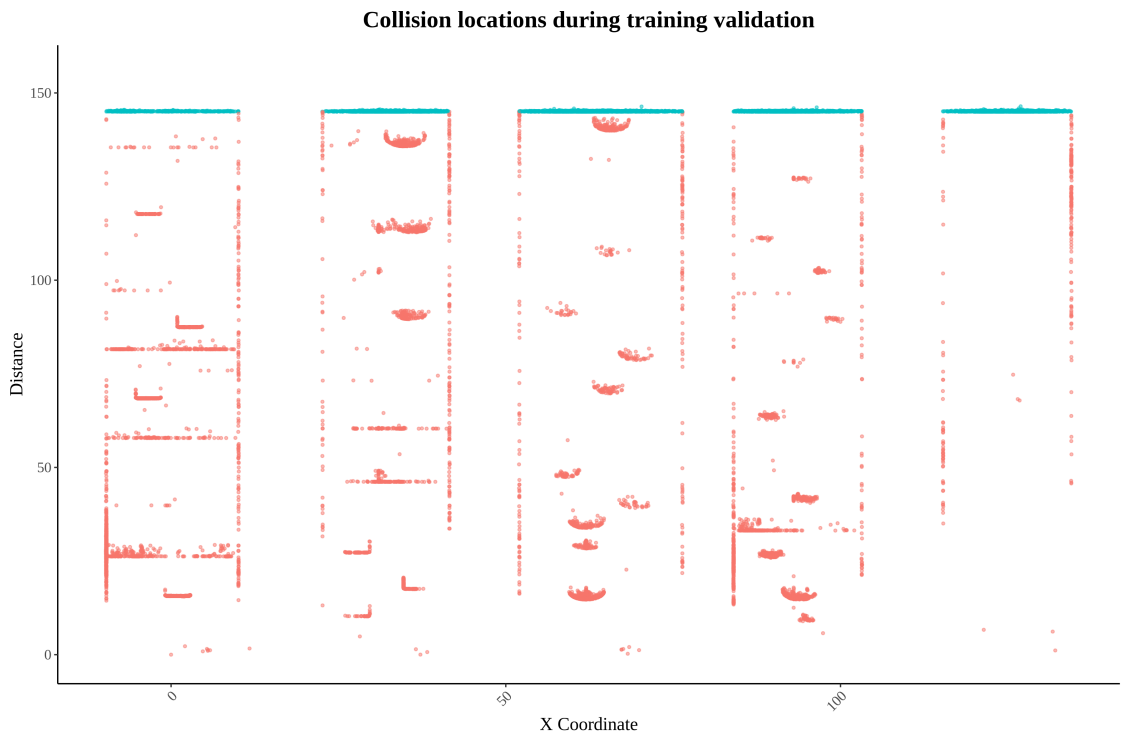


Figure 40: X/Y coordinates for trial termination [3] ©2020 IEEE

Table 2: Statistical Analysis of Distance and Iterations in Training Environment [3] ©2020 IEEE

$Lane$	$d.o.f.$	$p-value$	$corr.coeff.$	$t-value$
0	456	0.5175	0.278	-0.648
1	2749	2.2e-16	0.590	38.344
2	2233	2.2e-16	0.451	23.879
3	1901	2.2e-16	0.517	26.296
4	4318	2.2e-16	0.286	19.602

and random agents. Figure 42 shows the success rates of the random agent, spiking agent, and most trained conventional agent trials using the same testing environment. The spiking

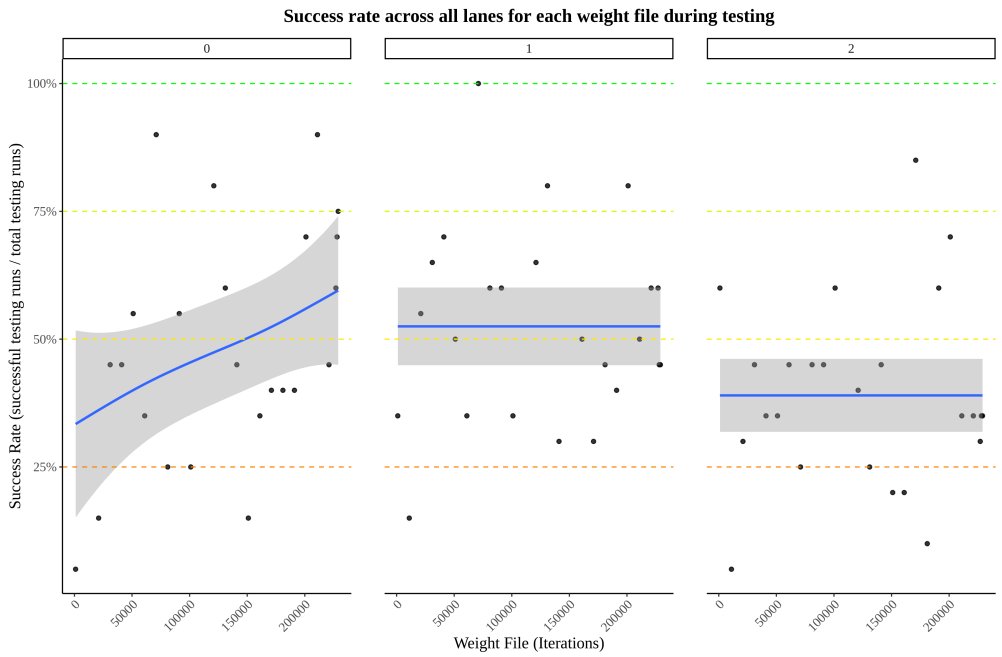


Figure 41: Success rate vs. training iterations for testing lanes [3] ©2020 IEEE

Table 3: Statistical Analysis of Distance and Amount of Training in Testing Environment [3] ©2020 IEEE

$Lane$	$d.o.f.$	$p-value$	$corr.coeff.$	$t-value$
0	230	1.734e-05	0.278	4.389
1	250	0.075	-0.1122	-1.786
2	193	0.164	0.100	1.397

agent was trained for 500 epochs on batches of observations collected from 50 successful trials in each training lane, resulting in a action selection accuracy of 91.35% in relation to the conventional agent's actions.

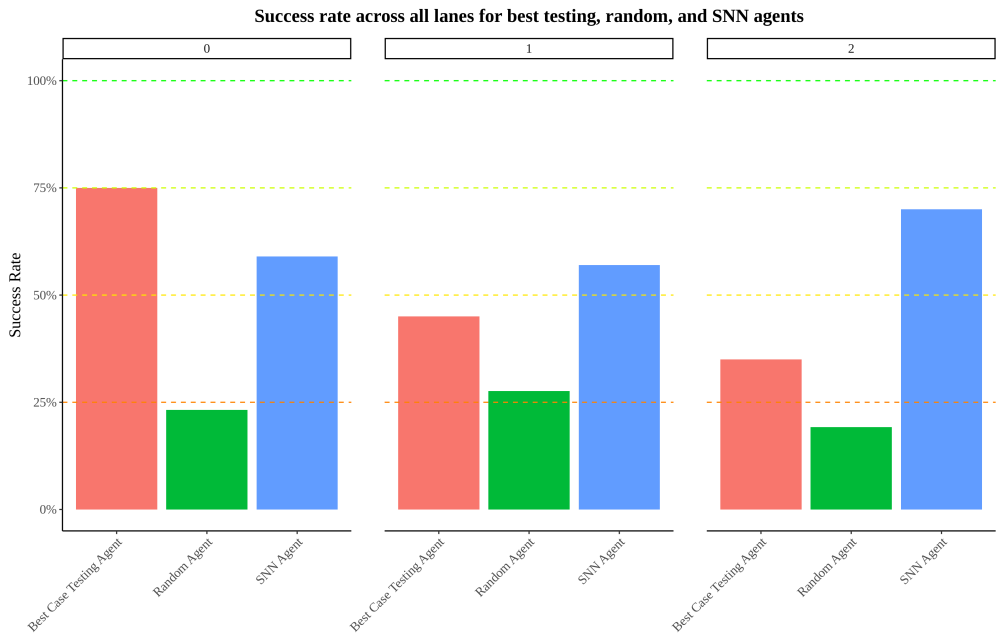


Figure 42: SNN vs. random agent vs. training [3] ©2020 IEEE

5.6 Discussion

Due to the overhead of the Unreal Engine 4 and AirSim simulation, the time required to perform a single training iteration for the agent in this work is far greater than that of previous works in far simpler environments. As a result, the number of training iterations performed in this work is far fewer than in similar reinforcement learning tasks, and the outcome can be clearly seen in the mediocre results of the testing course success rates. However, the success rate of the agent in each training lane over time shows a statistically significant, positive correlation with increasing training iterations, indicating that the agent is learning to avoid obstacles despite the limited training data available. The one exception to this improvement was training course lane 0, which featured obstacles with large, flat surfaces that the agent found difficult to avoid. One issue that the lack of training diversity introduces is the difficulty of generalizing the agent to new obstacle environments, as can be seen with the testing lane results. While the trained agent does show superior performance

as compared to the agent choosing random actions, there is a weak correlation between training iterations and test lane success rate, indicating that the agent requires more varied training data to be able to generalize it's decisions to new environments.

Another potential issue that the agent faced over the course of training is the inherent nature of the DVS simulation output. While the edges of obstacles were clearly visible throughout training, the smooth surfaces of some obstacles and the walls of each course lane had limited or overly high activity depending on the contrast threshold chosen for event-based simulation. These issues also made it more difficult for the agent to learn avoidance maneuvers when confronted with large, flat surfaces, such as in segments of training lane 0. The agent's difficulty in avoiding these obstacles can be seen in the spatial visualization of 40, where the majority of collisions are registered on these flat surfaces. Using elementary obstacles within simulation also posed an issue as the surfaces of these objects lack the complexity of real-world objects and therefore appear differently to the DVS simulation than might be ordinarily expected. Later versions of our training and testing course lanes used various in-engine textures in order to make the obstacles more visible to the simulated sensor with larger contrast thresholds. However, it is as of yet unclear how the DVS simulation parameters might extrapolate to a real-world setting and a hardware DVS may have a much easier time of differentiating object edges and surfaces.

In regards to the spiking architecture, testing results showed that the SNN had variable performance as compared to the regular, non-spiking architecture. Across testing trials, the SNN exhibited superior success rates in testing lanes 1 and 2, but a slightly lower success rate in lane 0. Interestingly, lane 0 is also the only testing lane that showed positive correlation between training iterations and success rate, indicating that the additional training time would most likely have resulted in greater success rates for the conventional approach. It is also important to recall that the SNN agent is trained using labelled observations collected during successful conventional agent runs, resulting in the spiking agent relying on far fewer observations of the lanes than the conventional equivalent. Furthermore, the entire training process of the SNN agent resulted in an accuracy of approximately 91% in relation to the labelled observations of the conventional DQN, causing the SNN to choose different actions despite some observations being nearly identical to those encountered by the conventional

architecture. Nonetheless, the superior success rates in the other testing lanes suggest that the SNN agent may actually benefit from only training on a subset of the conventional agent's observations. Overall, the SNN testing results suggest that the training scheme used in this work can be used to train an equivalent spiking agent from a conventional one, although the true benefits of using a spiking architecture may only be seen when integrating the system with a hardware DVS.

5.7 Additional Training Approaches with Reinforcement Learning

After this first round of training and testing, it has been shown that reinforcement learning is a viable solution for developing a sense and avoid system for UAV flight control. The next steps in this research is to determine the optimal way to train these policies, both in terms of speed and efficiency. In order to do this, we device a new training environment focusing on one specific type of obstacle, a cone, shown in Figure 43. Several training lanes are created using this obstacle, including an "easy" course and a more difficult course; an example is shown in Figure 44.

Figure 43: Cone obstacle

In order to evaluate how well the training setup is working, we decide to plot the cumulative rewards over the course of training iterations. For this experiment, two versions of the reinforcement learning training pipeline are used. The first is the same pipeline used in

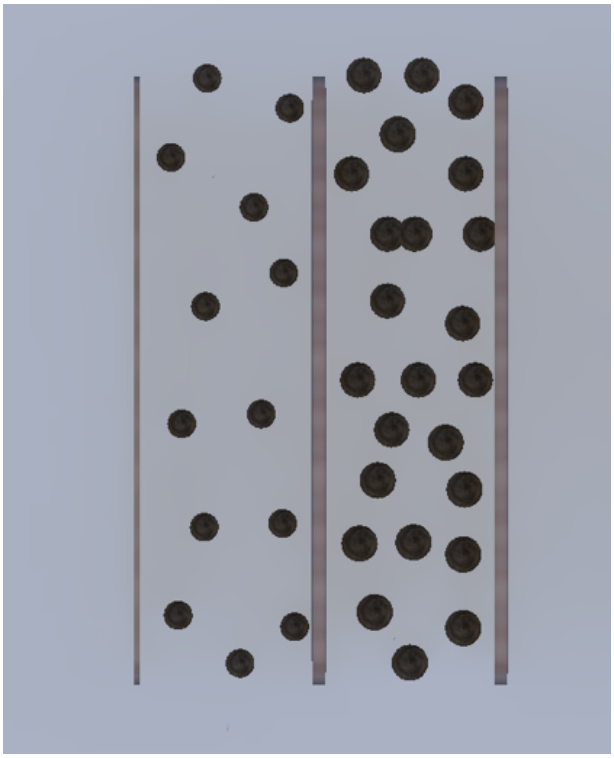


Figure 44: New training courses

previous experiments, where the drone is fed a steady stream of velocity commands. The second is a modified pipeline where the drone is sent position commands instead, and the AirSim built-in planner is used to fly the drones to the relevant positions. The initial results of these experiments are shown in Figures 45 and 52. The bounds of the cumulative rewards are from -2000 to 1000.

As we see from these results, the RL controller is slowly learning how to navigate the obstacle courses using the event camera data. In both of these experiments, the training loops manage to reach a cumulative reward greater than 0 in approximately 5000 trials; this shows that the drone is successfully navigating to the last portion of the testing lanes, but not successfully completing the trials. Analysis of individual results show this is due to two reasons: unreasonably high penalties for non-ideal actions, and sparsity of data for the RL system to use. Both of these problems are fixed in the next iteration of training.

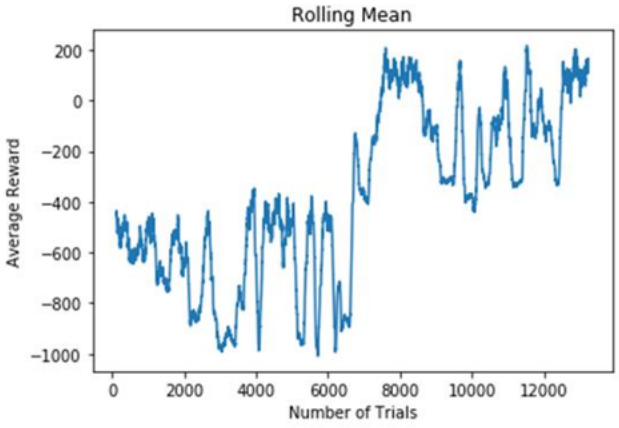


Figure 45: Velocity-based pipeline results [3] ©2020 IEEE

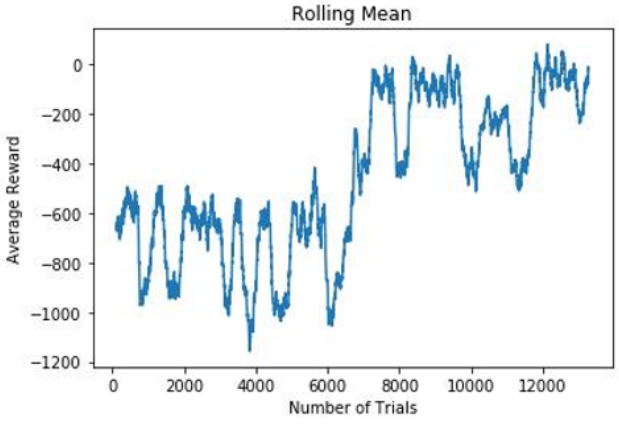


Figure 46: Position-based pipeline results [3] ©2020 IEEE

5.8 Chapter Conclusion

In summary, this work has introduced a reinforcement learning scheme that has the potential to train both conventional and spiking DQN agents to perform collision avoidance maneuvers using event-based vision sensors. Despite the rather small number of training set iterations performed, the conventional agent showed significant improvement in avoidance as compared to the baseline, with these benefits being successfully transferred to an equivalent spiking architecture. The AirSim simulation environment proves essential for allowing the creation of custom training environments and the large number of iterations needed to train a reinforcement learning agent. In the future, far more varied lanes and additional training time will be required to generalize the collision avoidance scheme to more complex environments. Furthermore, more exhaustive analysis of real-world event-based sensors will be necessary to establish the performance gains versus conventional sensors and the benefits of integration with spiking neural net architectures. At higher level, this portion of the dissertation research has shown that it is feasible to integrate new, cutting edge sensor packages on-board UAVs and can integrate their use with the fleet management program designed. The use of new sensors like this can take control of point-to=point motion navigation for UAVs during mission procedures, meaning this work no longer has to be directly overseen by a fleet management system.

6.0 Contributions Overview and Future Work

Over the course of this dissertation, work has been conducted on along many different aspects of multi-agent drone systems, from high-level management approaches to low-level sensor techniques. The overarching goal of this research is to develop and validate a scalable, modular system that will allow for the command and control of multiple UAV systems for any mission requirements. In order to ac hive this, comprehensive work has been done at each level of control. At the onset, the research focused on developing a fleet management system that would be able to control and direct any number of robots for a specific purpose. This was built on top of existing, proved software, in order for it to be readily deploy-able and meet all operating requirements by various sources (ISO, ANSI, SAEI, etc.). The second stage of work was to work on platform-specific motion planning, to determine what is the best way to coordinate these groups of robots at the platform level. This work focused on adapting and creating new control techniques that would take into consideration the mission and platform aspects while planning. These novel control techniques serve as the mission planning "brains" of the fleet management system. The third section of work in this dissertation was focused on novel sensor technologies. The goal of this work was to focus on how to integrate new types of sensors and payloads into a multi-agent swarm. The main focus of this section is for platform-specific processing and management; how does each platform accomplish the goals given to it by the fleet management system. Specifically, I chose to work on obstacle avoidance for motion planning, as this is the main issue the robots face with following orders given by the motion planner. This was also a change to work with brand new, minimally-studied technologies, and be one of the first research projects to integrate them onto UAV systems.

6.1 High-level Fleet Management

The goal of the fleet management system implemented in this research was twofold: create a system that can interface with and control any number of robotic systems for joint mission completion, and built it on top of a pre-existing system that has already been validated and approved for deployment. This was accomplished by utilizing two open source software architectures currently available: Robot Operating System (ROS) and NASA's Core Flight System. Using these two systems, a set of modules was developed that would allow for the core flight system to interface with a large number of remote platforms, all connected over a distributed network. In order to develop and prove that this system works, we devised a test scenario: have a number of UAVs cooperatively scan an object for visible damage. This scenario was perfect because it required several important operational constraints: manage multiple UAVs flying in close proximity, manage and modify the flight plans of these UAVs based on mission factors, manage multiple high-bandwidth payloads and run analysis on the incoming data, and operate using a robust and safety-critical system. The system was first prototyped and tested using ROS, and was later deployed on top of NASA's ICAROUS suite, due to its specification for UAV systems. This system was then successfully tested for the scenario mentioned above. After these tests were completed, the system was then modified to work with the simulator of choice for this research project, AirSim.

6.2 Platform-specific Motion Planning

The next step in this research was to create a new process that would be able to direct and manage the actions of the numerous robots connected to the fleet management system. The best way to do that was to create a new control policy specifically designed for this application. As stated before, the parameters for the policy were that it needs to support multiple agents, it must be able to handle heterogeneous systems (many diverse platforms and payloads), and it needs to work for both centralized and decentralized systems, for a variety of mission types. In order to accomplish this, the new control policy was based on Receding

Horizon Control (RHC), a mixture between optimal and adaptive control techniques that allows for near-optimal performance with minimal tuning. The RHC control policy was first setup conceptually, using the base case scenario of a search and rescue mission with both known and unknown points of interest (mission points), The controller was designed and integrated with both the AirSim simulator and a graph-based training environment. The system was then tuned over the course of 10,000 trials, in order to find the optimal weights. Once it was verified that this new policy worked as intended, a cFS module was created for the RHC controller, so it can interface with the fleet management system with ease.

6.3 Exploration into Novel Sensor Technologies

The last portion of this research was focused on platform-specific solutions for motion planning and navigation, using new and existing sensor techniques. Now that the fleet management system is able to interface with multiple robots and send them intelligent actions to accomplish a mission, the individual platforms need to be able to carry out those orders. For this research, the focus is to make sure the UAVs can successfully navigate to their desired location while avoiding any unforeseen obstacles. This research ended up revolving around the dynamic vision sensor (DVS), also known as a neuromorphic or event camera. This is because very little work has been done regarding DVS systems and UAVs, except for the last six months. Event cameras are an ideal sensor for UAVs, as they work best on moving platforms and are able to capture data at faster rates than traditional sensors. For this research, we focused on training a UAV flight controller to use event camera input for perception and action loops in a diverse environment. The 2D image output from the event cameras were fed directly into a reinforcement learning algorithm, with the output being a set of actions for a UAV. Through 400,000 iterations of training in several environments, a model policy was created that would allow a UAV platform to successfully navigate a variety of obstacles in its way, with minimal deviations to its existing trajectory.

This research was then extended by developing a way to train and control the flight controllers using the raw bytestream output of the vent cameras. The results for this training showed that using the raw bytestream and a non-linear representation actually allows for the development of a more robust policy in a faster amount of time.

6.4 Novel Contributions

This dissertation research has produced a number of novel contributions, that have been published and shared with the research community through several conference and journal papers. Here is a list of the contributions broken up by each task.

Task 1:

- Develop a localization system plugin for the cFS architecture
- Add multi-agent support to the NASA ICAROUS system
- Add machine learning (CNNs) support to the NASA cFS architecture
- Integrate ICAROUS into the AirSim simulation framework, for rapid training/tuning

Task 2:

- Develop a new version of Receding Horizon Control that support heterogeneous systems
- Integrate this new controller with the AirSim simulator to allow for rapid tuning
- Develop a new formalization for denoting non-uniform payload capabilities for platforms
- Integrate health metrics, energy constraints, and communication limitations into the RHC controller

Task 3:

- Develop a reinforcement learning pipeline using AirSim and various cloud architectures
- Integrate an event camera into the Unreal Engine environment
- Develop the first use of event camera data in reinforcement learning
- Develop a novel non-linear representation for event camera bytestream data
- Train reinforcement learning systems using raw bytestream data

6.5 Future Work

6.5.1 Fleet Management Work

For future work the new ICAROUS modules can be improved to allow for more complex methods of mission allocation and coordination. The multi-agent approach can be improved upon, adding in support for heterogeneous collection of UAVs. For communications, some time was dedicated to exploring the use of distributed communication systems, specifically the Data Distribution Service (DDS). The incorporation of DDS into a multi-agent system would allow for more advanced network and data sharing techniques, which would prove impactful when coordinating among large numbers of systems. The next step for the computer vision work would be to implement 3D reconstruction capability in real time, as the free-flyer platforms scan its surface. Another avenue is the use of unsupervised machine learning techniques for feature detection.

6.5.2 RHC Controls Work

Though the controller has demonstrated effectiveness for heterogeneous swarms, we understand that this proof of concept is limited. To explore the true dynamics of this controller, we intend to introduce higher fidelity simulation models. This includes modeling varying payload energy and operational costs: various sensors require different lengths of time to properly measure certain phenomena (e.g., gas leak detection). This in turn would require changes to the controller's cost function to account for non-uniform visitation time costs.

Another avenue of future work is to implement and test the HD-RHC controller with several other simulation platforms. Of interest are the NASA ICAROUS system, due to its support for low-powered systems and formally verified architecture, and the new AWS RoboMaker simulation environment, which would provide extended support for tuning huge-number systems using cloud services [115]. Additional areas we wish to explore are extending beyond the one-hot membership function. There are two additional combinations of payload classes to consider: payloads which have multiple mission applicabilities, and payloads which have partial mission applicability. In the former, we expect that the energy and time costs

of motion for such a UAV to be greater than their dedicated-payload counterparts, thus needing to expand mission point selection to consider more than distance, and reliability scores. In the case of partial-membership, we expect the mission selection bids to be factored by applicability, but know that tuning is required to not extensively favor full-membership payloads.

The modeling of the controller itself assumes discrete actions and synchronous control epochs. We wish to further explore applications in continuous action space and non-synchronized planning. Doing so will more closely model real-world implementations, but also may change the cost function model to consider the probability mass of selected actions of neighboring UAVs in the swarm.

6.5.3 Neuromorphic Camera Work

While the results of the testing trials do show steady improvement in the agent's collision avoidance, there is of course considerable room for improvement and a number of methods that could be applied. Foremost among these would be to drastically increase both the number of training trials and the number and diversity of obstacle course lanes used for training. Given the elementary obstacles used in these trials, the trained DQN would most likely have difficulty generalizing to environments with less well-defined obstacles such as trees or power lines. For future training, it would be immensely beneficial to include some means of procedural generation that could produce large numbers of very distinct obstacle course lanes for training. Another modification to the training procedure that could have a significant impact would be to change the manner in which the experience replay buffer is sampled at training time. In this work, the buffer was simply randomly sampled in batches, however, previous works have shown that prioritizing certain memories based on difference in calculated loss can significantly improve results and decrease required training time [116].

Prior research has also shown that the training time of reinforcement learning models can be greatly reduced by including demonstration samples from human-in-the-loop trials [117]. In much the same way that the SNN was trained in a semi-supervised fashion with labelled observations, the conventional DQN could also be trained partially using observations labelled with actions performed by a human as ground-truth.

There are also significant changes that could be made to the DQN architecture itself. For this work, we opted for a shallow convolutional neural network with fewer layers that extracted features at a fairly large scale. The reasoning for this choice was an attempt to reduce training time as well as the assumption that fine-grained features were unnecessary for UAV-based collision avoidance. However, given the differences between the images supplied by a DVS and a conventional camera, it may be beneficial to include more convolutional layers in the network. The event-based data produced by a DVS also has a strong temporal component that could benefit from using a recurrent neural network architecture that could operate on the one-dimensional stream directly. Lastly, after making significant improvements to the architecture and training, the long-term goal of this work would be to transfer the network to a physical UAV in order to assess performance in a real-world environment. This testing will include integrating a dynamic vision sensor on-board a custom UAV platform, and flying the platform through a custom-built obstacle course. The amount of training required for real-world testing will hopefully be minimal, thanks to the extensive training conducted in the high-fidelity simulator [95].

Appendix A Software Architecture Diagrams

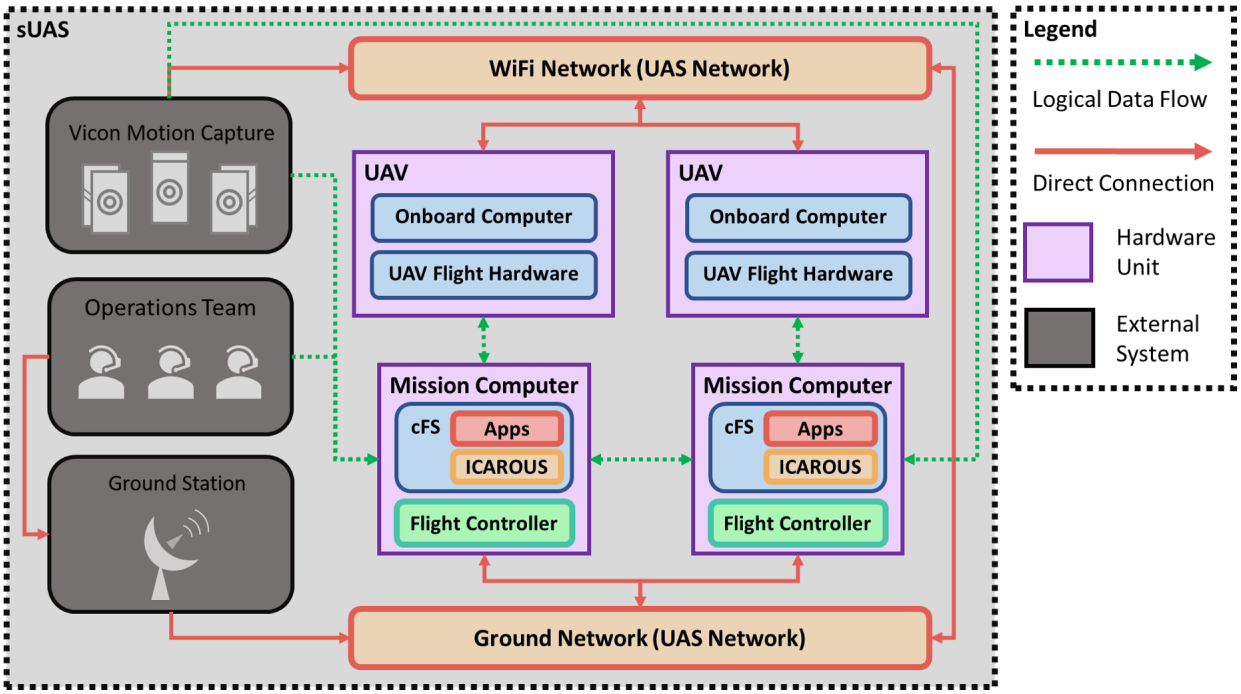


Figure 47: ICAROUS-based FMS diagram [1] ©2020 IEEE

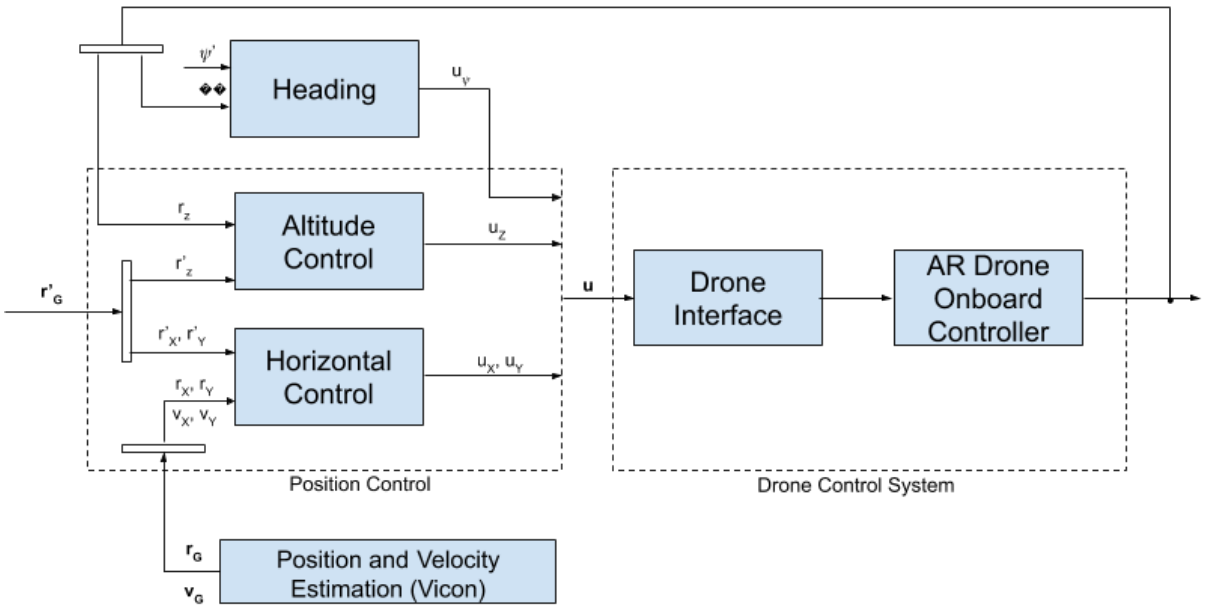


Figure 48: FMS control system architecture [1] ©2020 IEEE

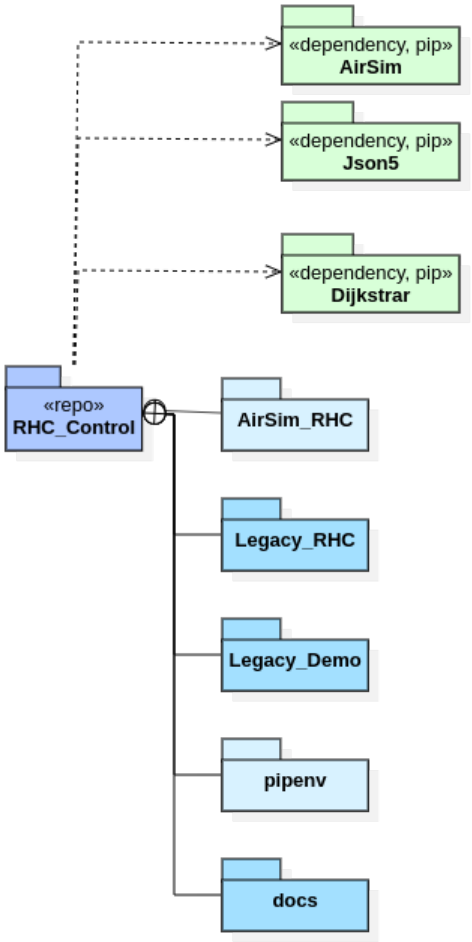


Figure 49: HD-RHC project organization

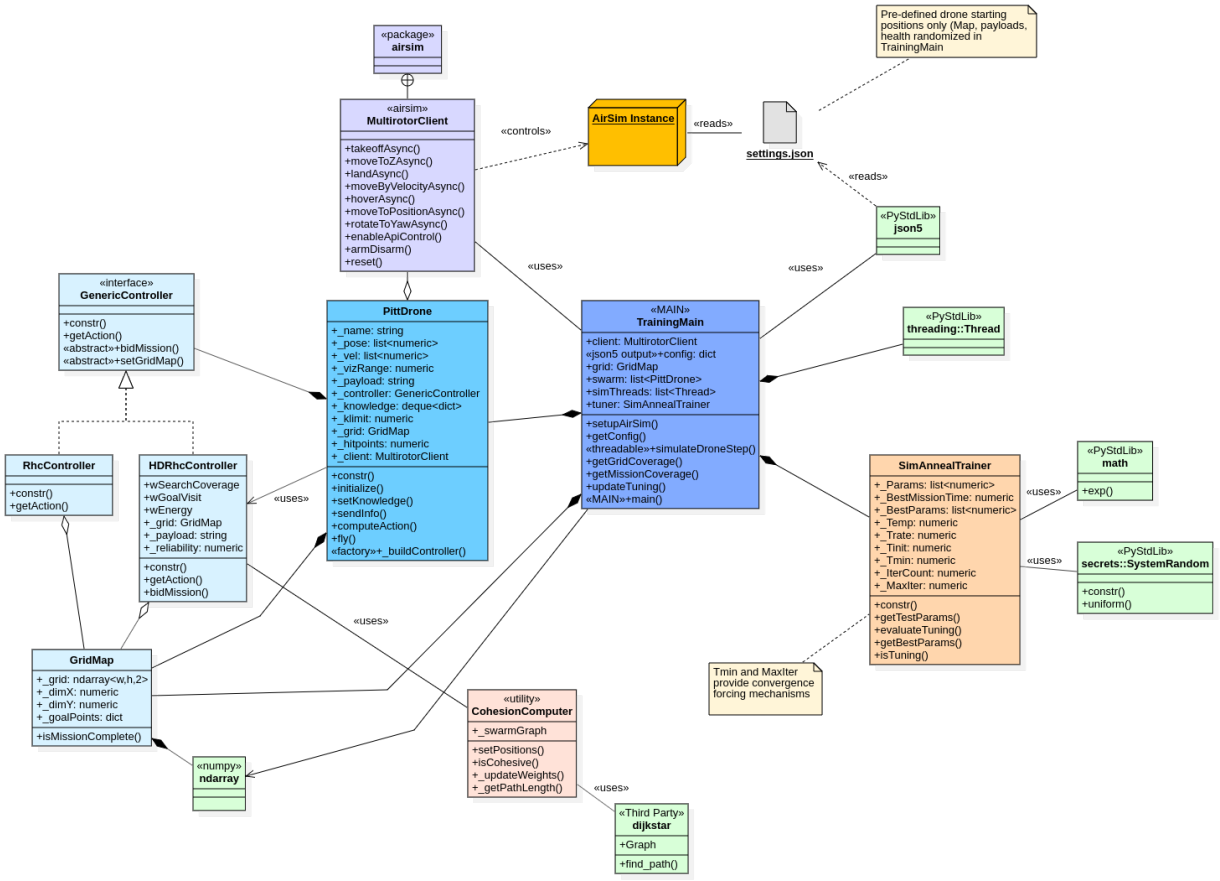


Figure 50: HD-RHC training class diagram

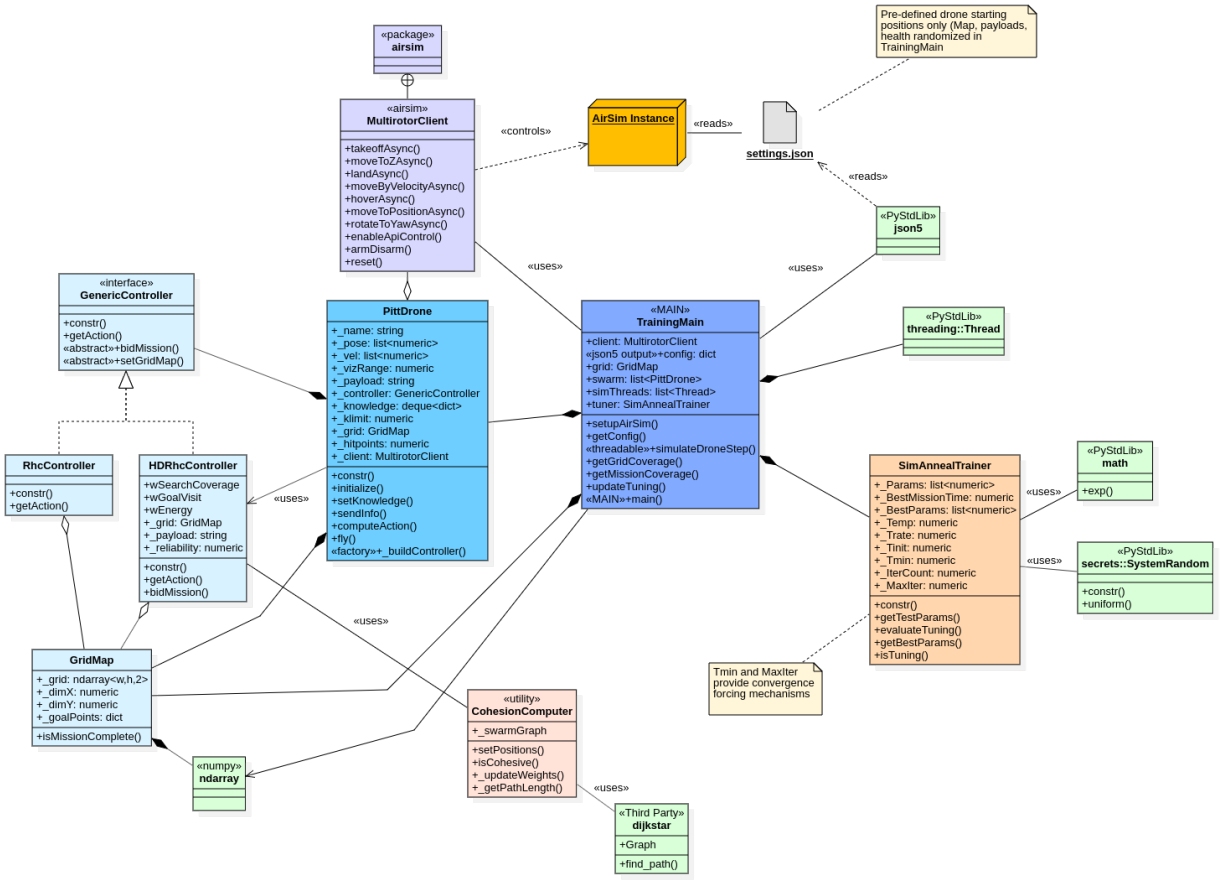


Figure 51: HD-RHC evaluation class diagram

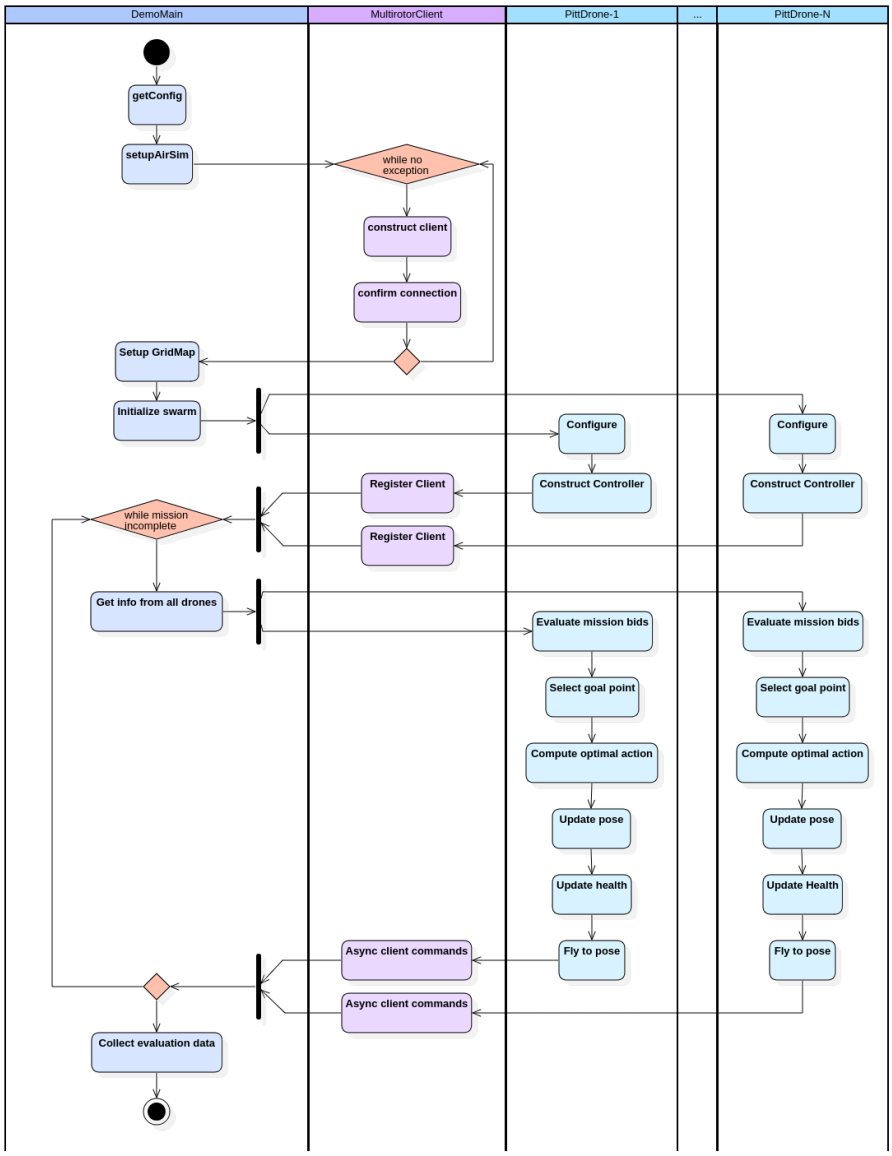


Figure 52: HD-RHC activity diagram

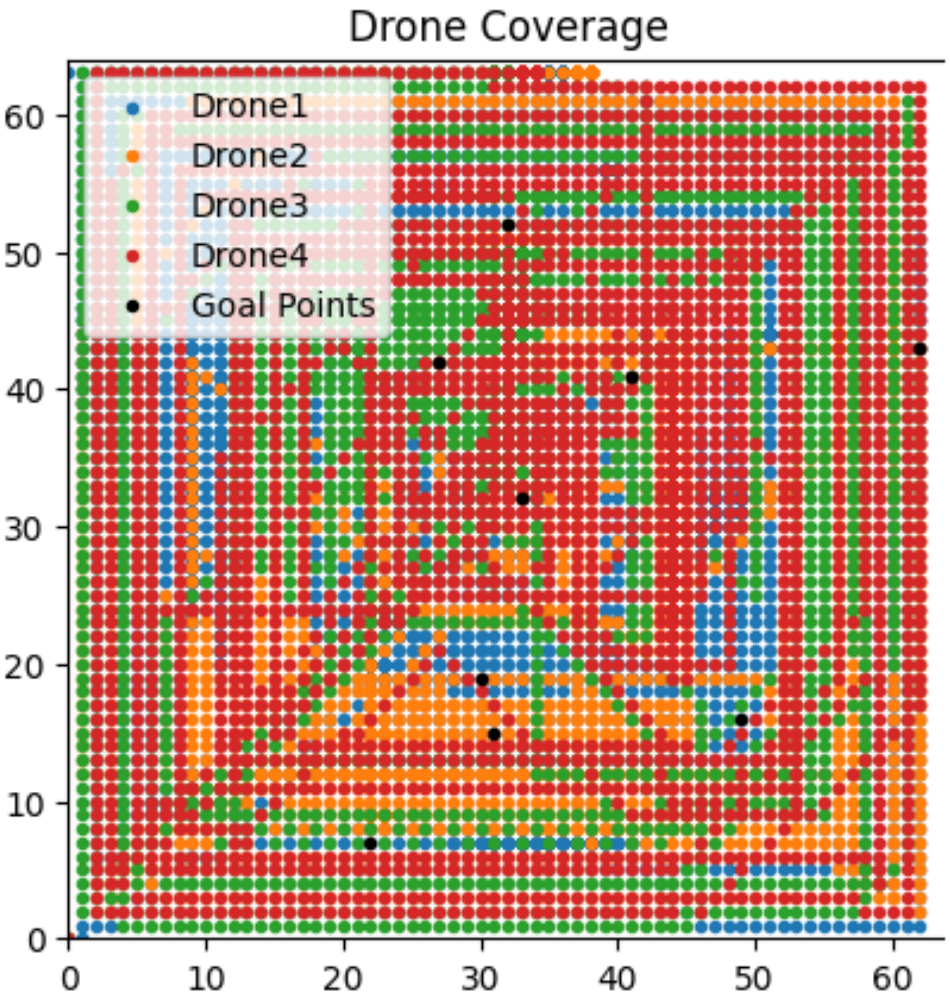


Figure 53: Odometry plot for weights (1.0, 1.0, 1.0) [2] ©2020 IEEE

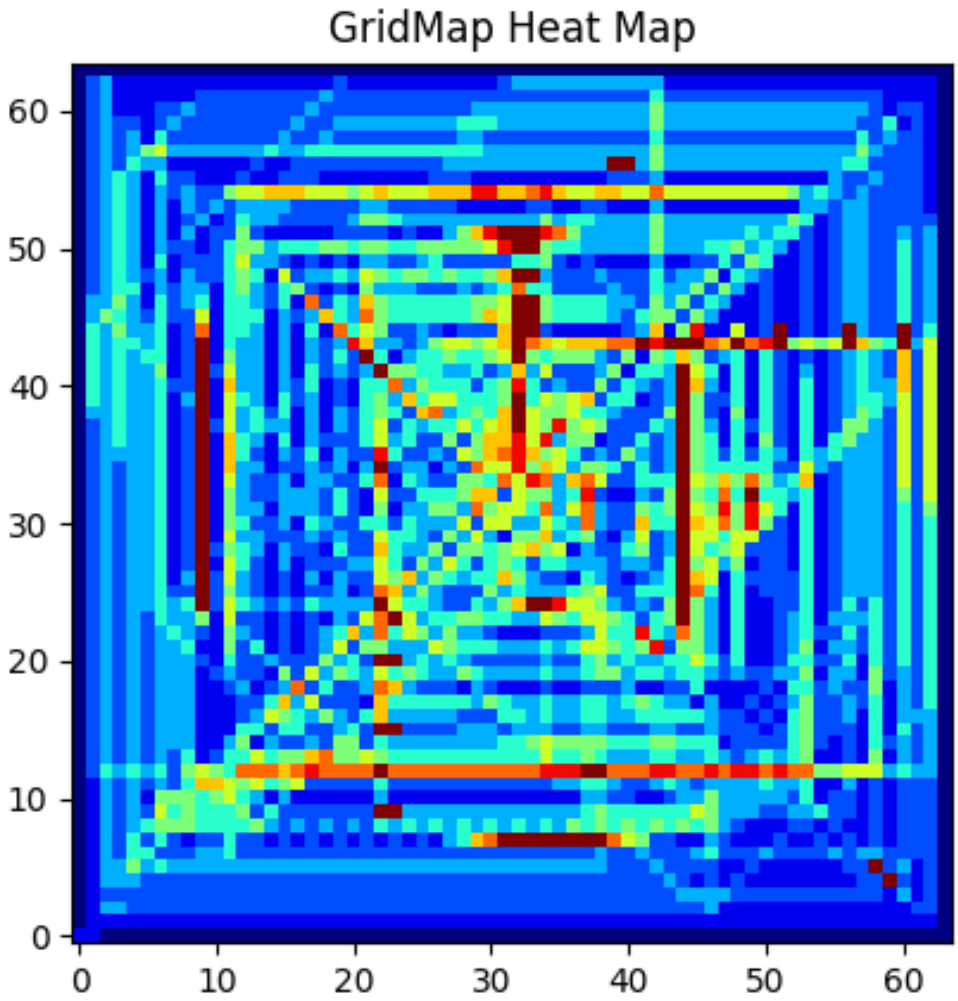


Figure 54: Heat-visitation map for weights (1.0, 1.0, 1.0) [2] ©2020 IEEE

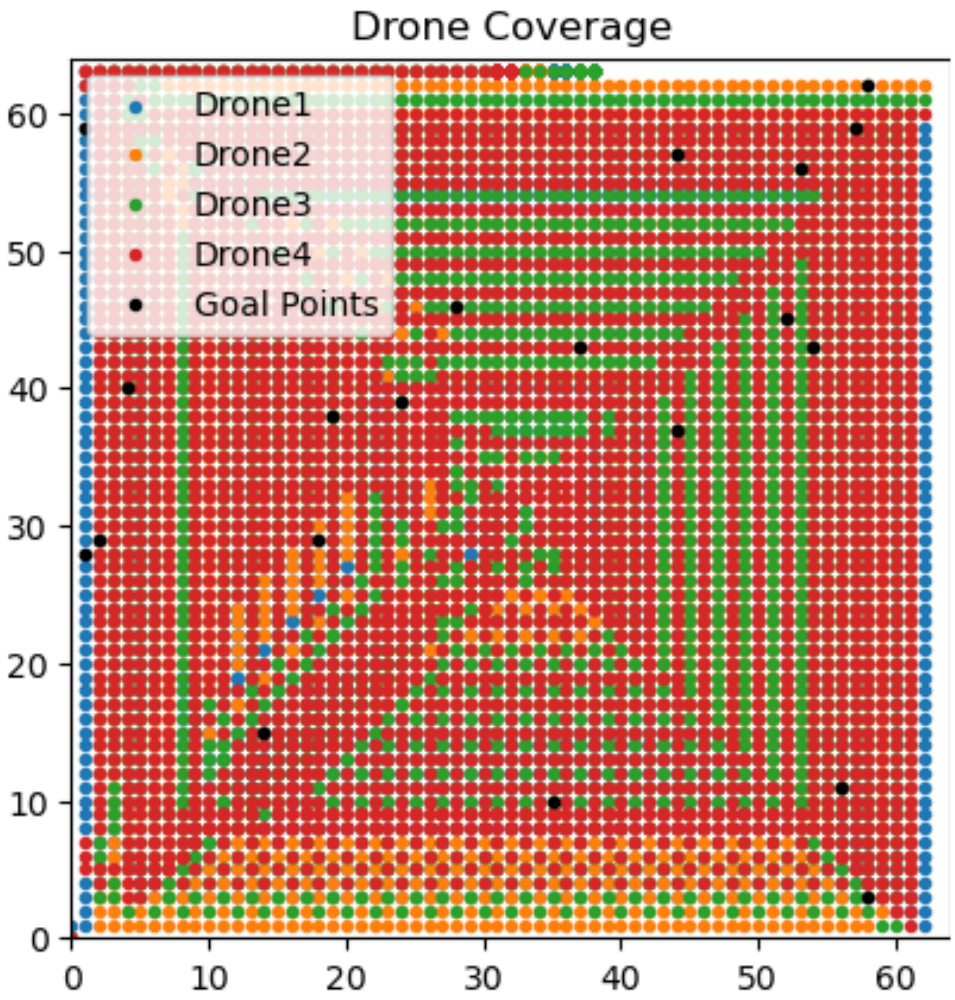


Figure 55: Odometry plot for weights (1.0, 0.0, 0.0) [2] ©2020 IEEE

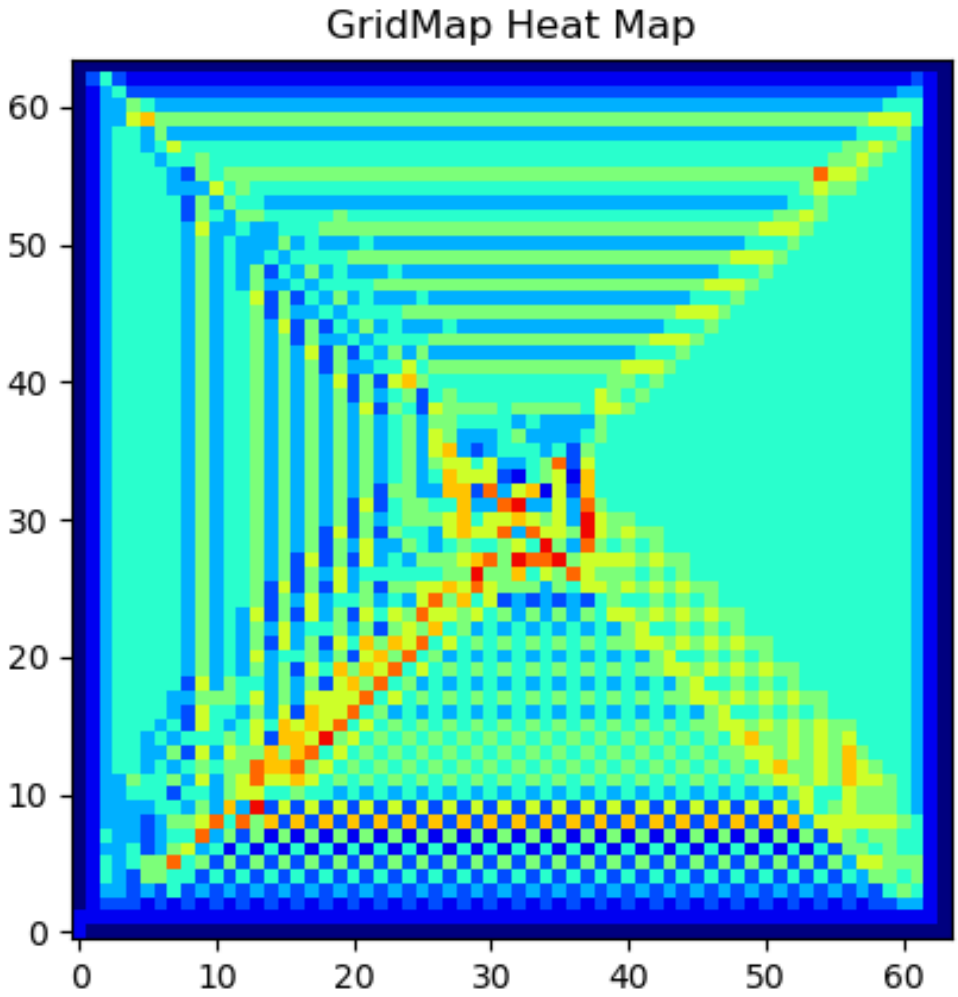


Figure 56: Heat-visitation map for weights (1.0, 0.0, 0.0) [2] ©2020 IEEE

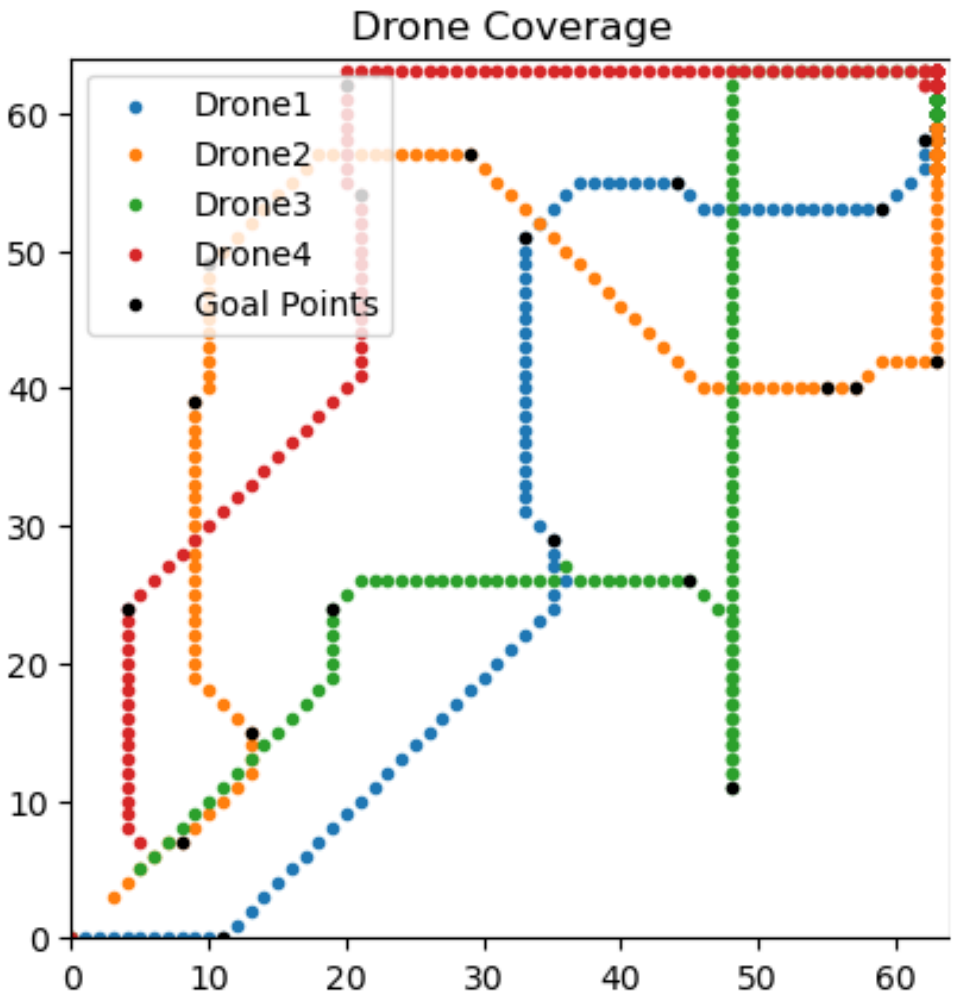


Figure 57: Odometry plot for weights (0.0, 1.0, 0.0) [2] ©2020 IEEE

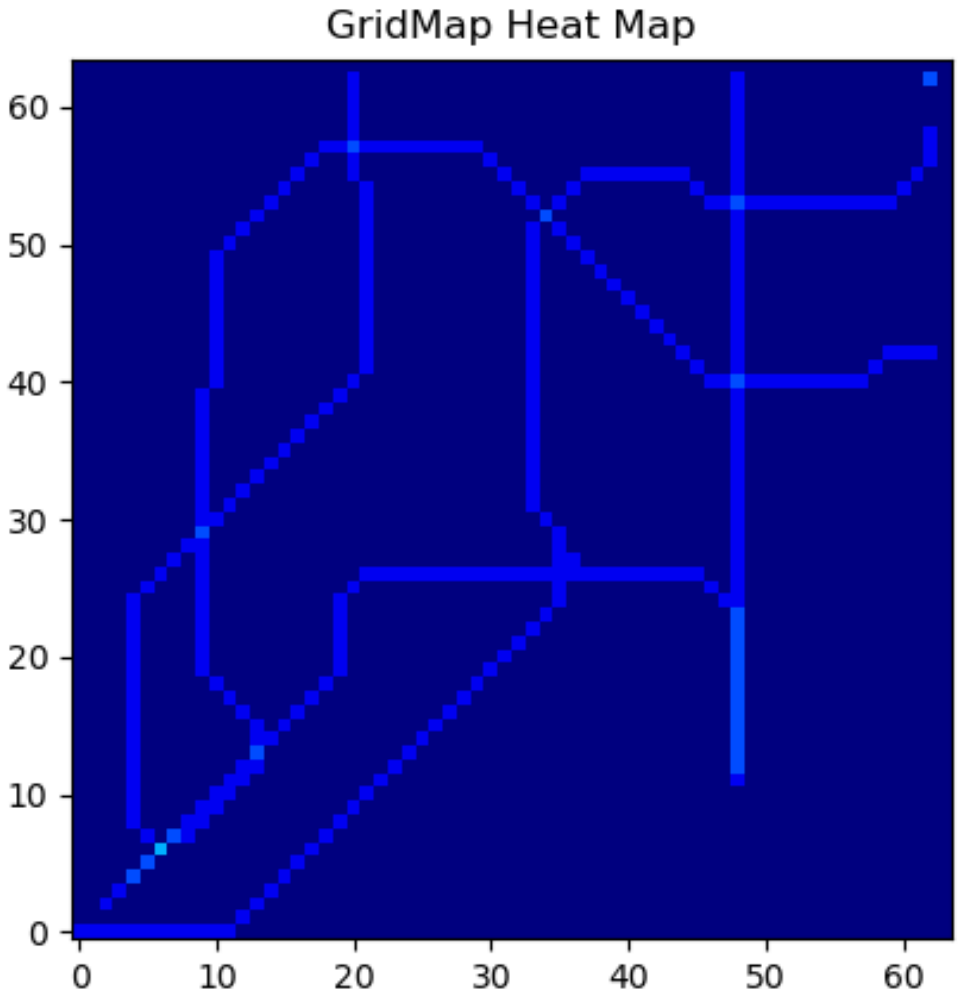


Figure 58: Heat-visitation map for weights (0.0, 1.0, 0.0) [2] ©2020 IEEE

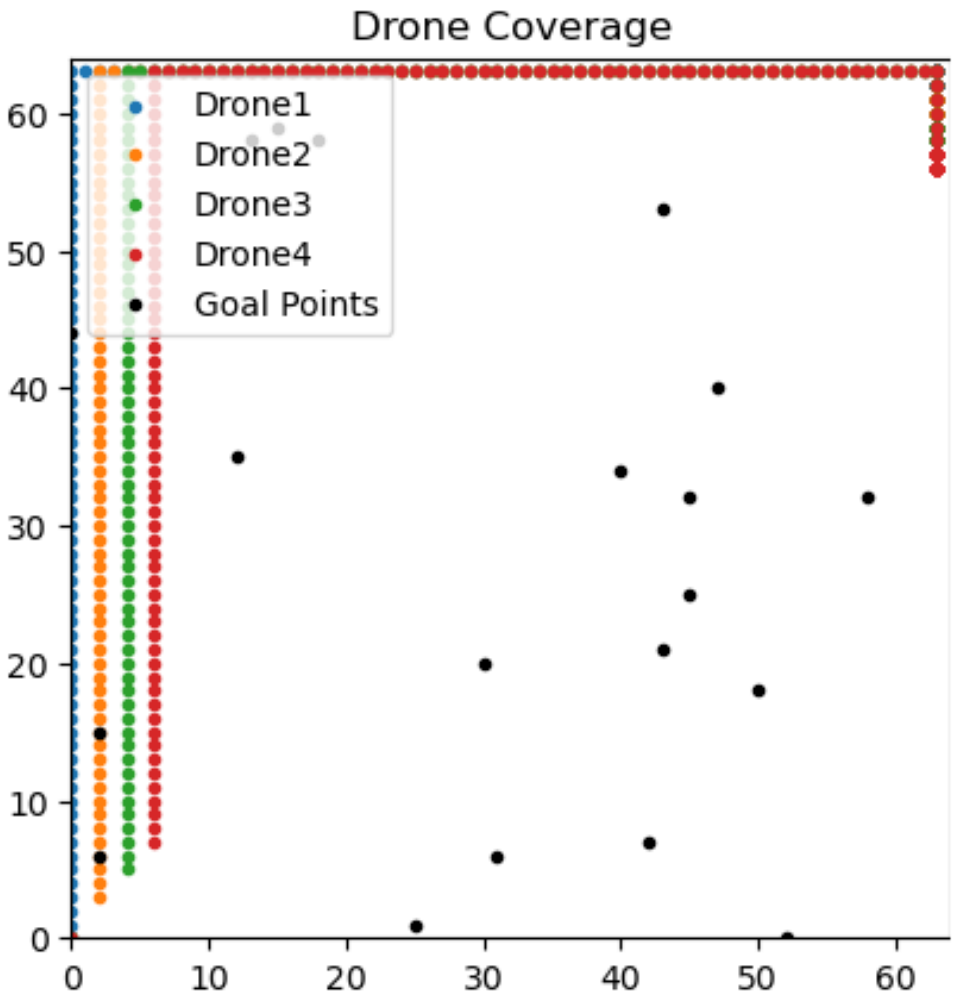


Figure 59: Odometry plot for weights (0.0, 0.0, 1.0) [2] ©2020 IEEE

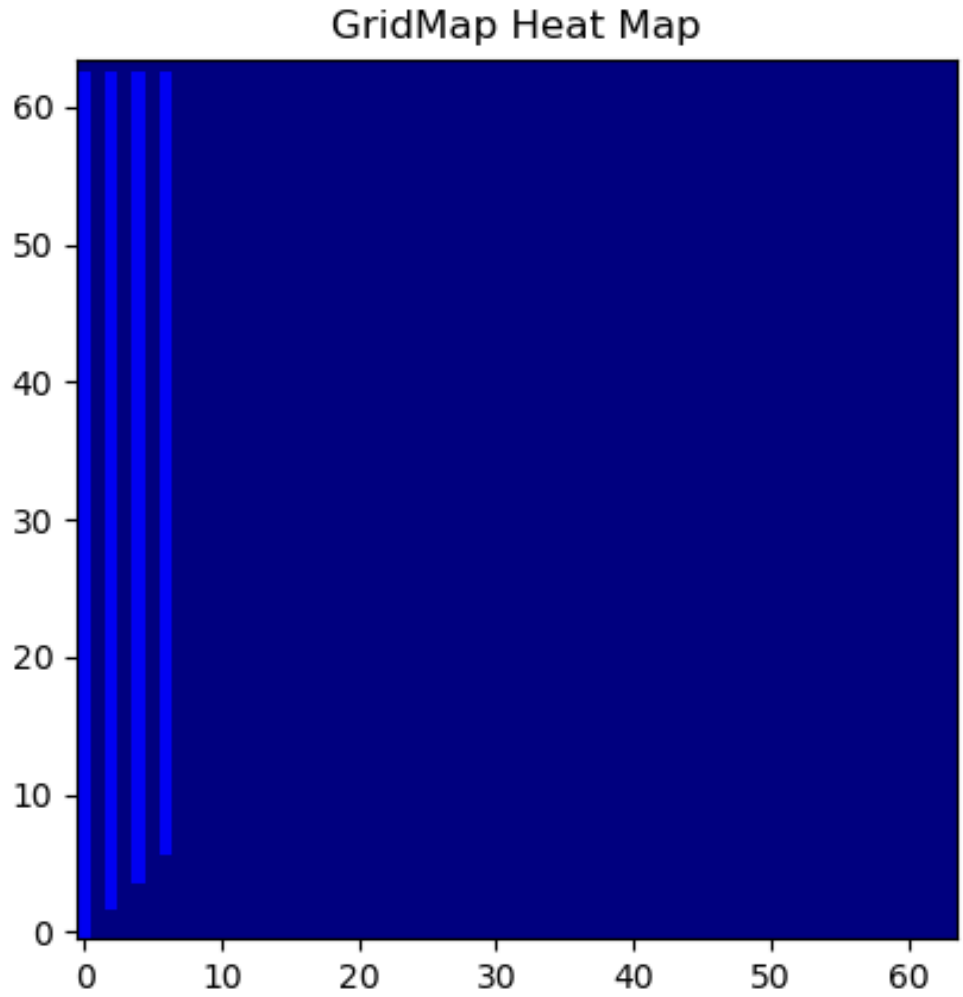


Figure 60: Heat-visitation map for weights (0.0, 1.0, 0.0) [2] ©2020 IEEE

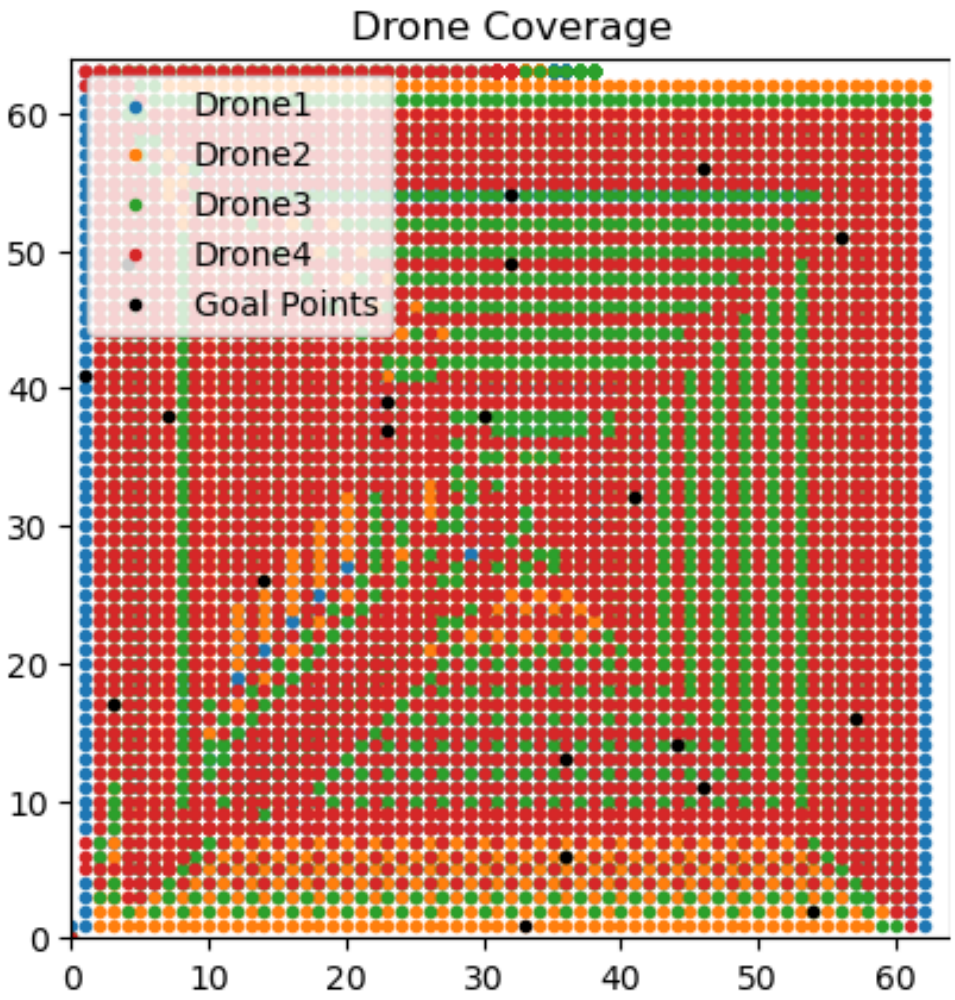


Figure 61: Odometry plot for best case results (1.0, 0.3, 0.0) [2] ©2020 IEEE

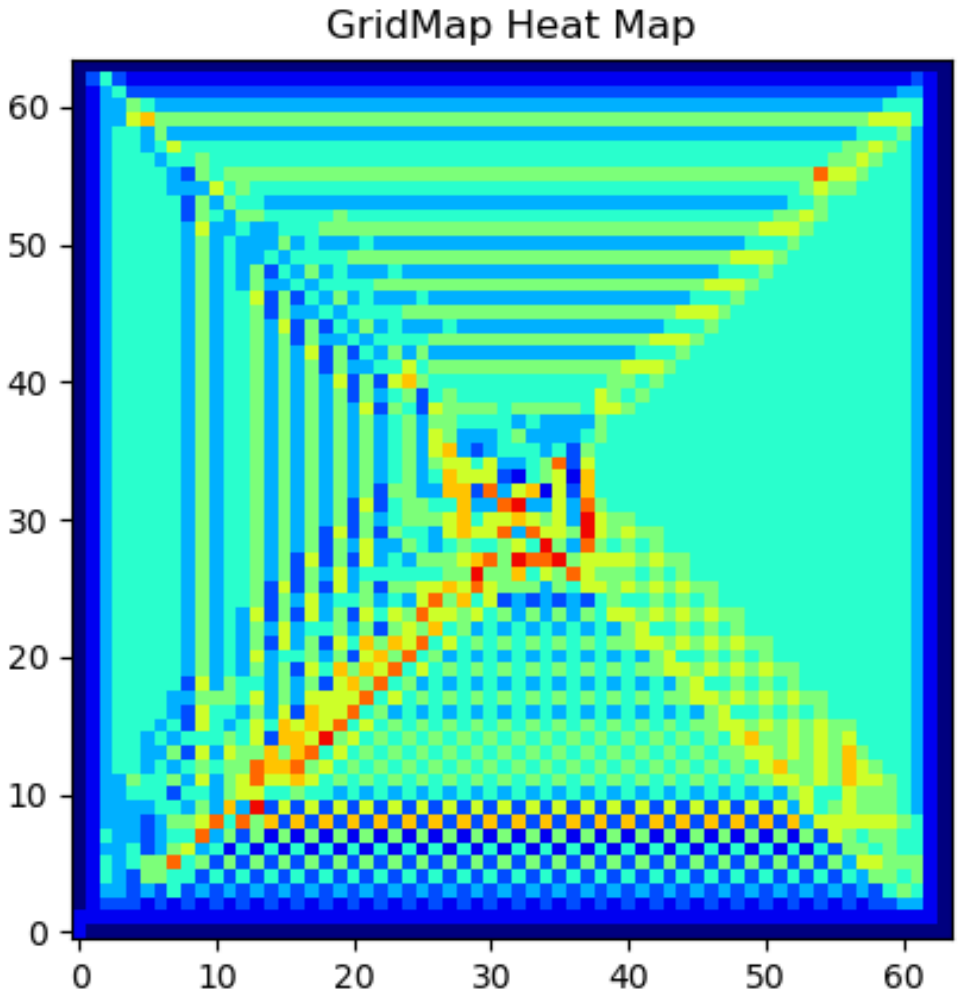


Figure 62: Heat-visitation map for best case results (1.0, 0.3, 0.0) [2] ©2020 IEEE

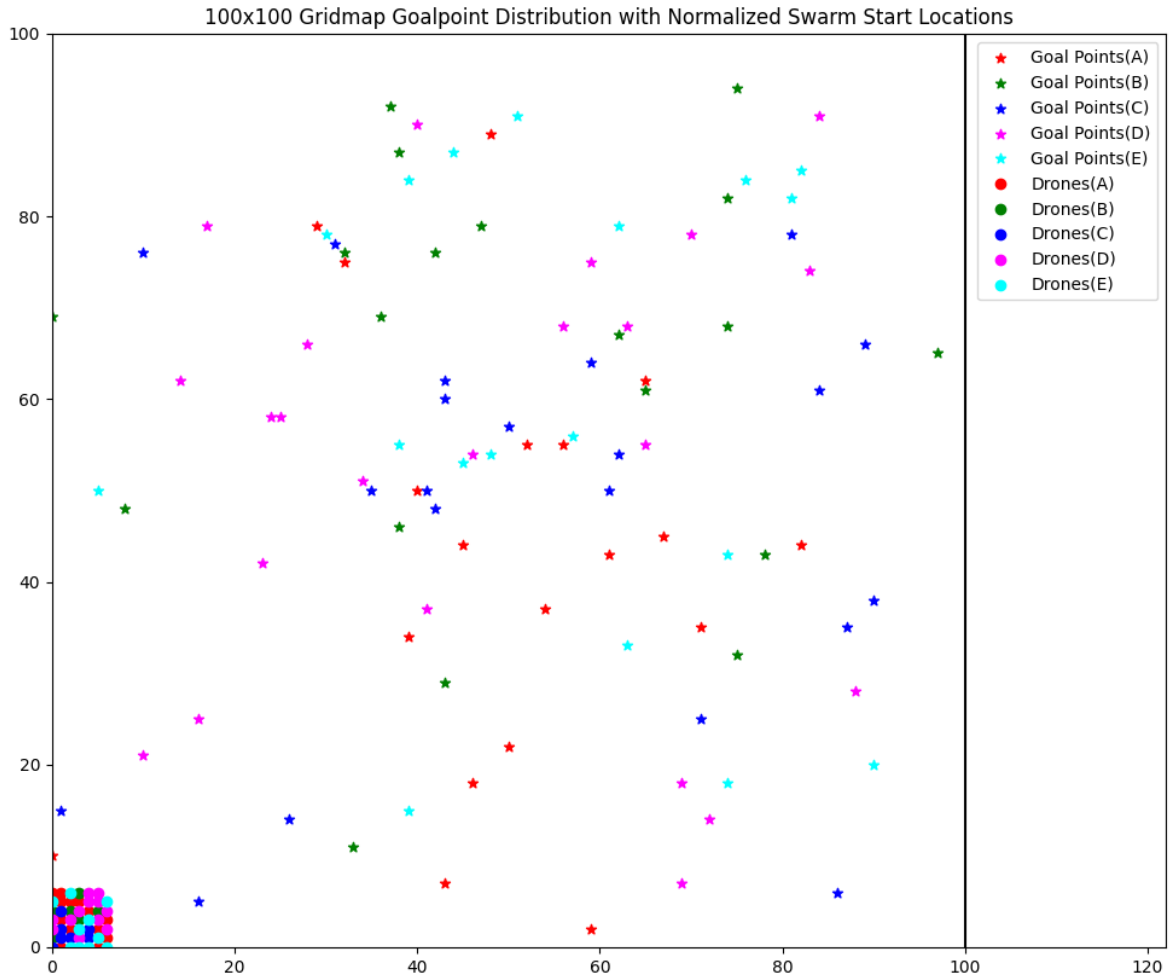


Figure 63: Initial distribution, normalized locations [2] ©2020 IEEE

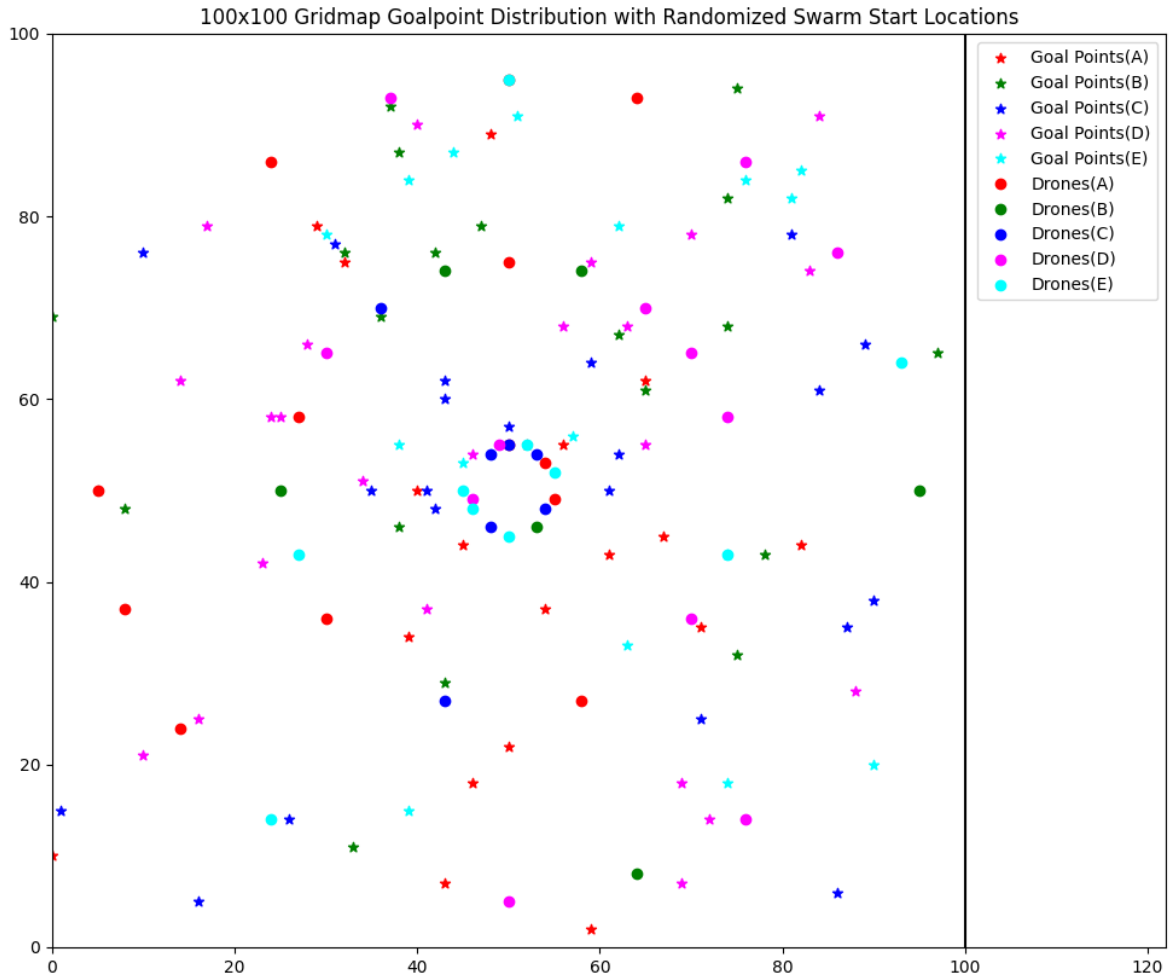


Figure 64: Initial distribution, randomized locations [2] ©2020 IEEE

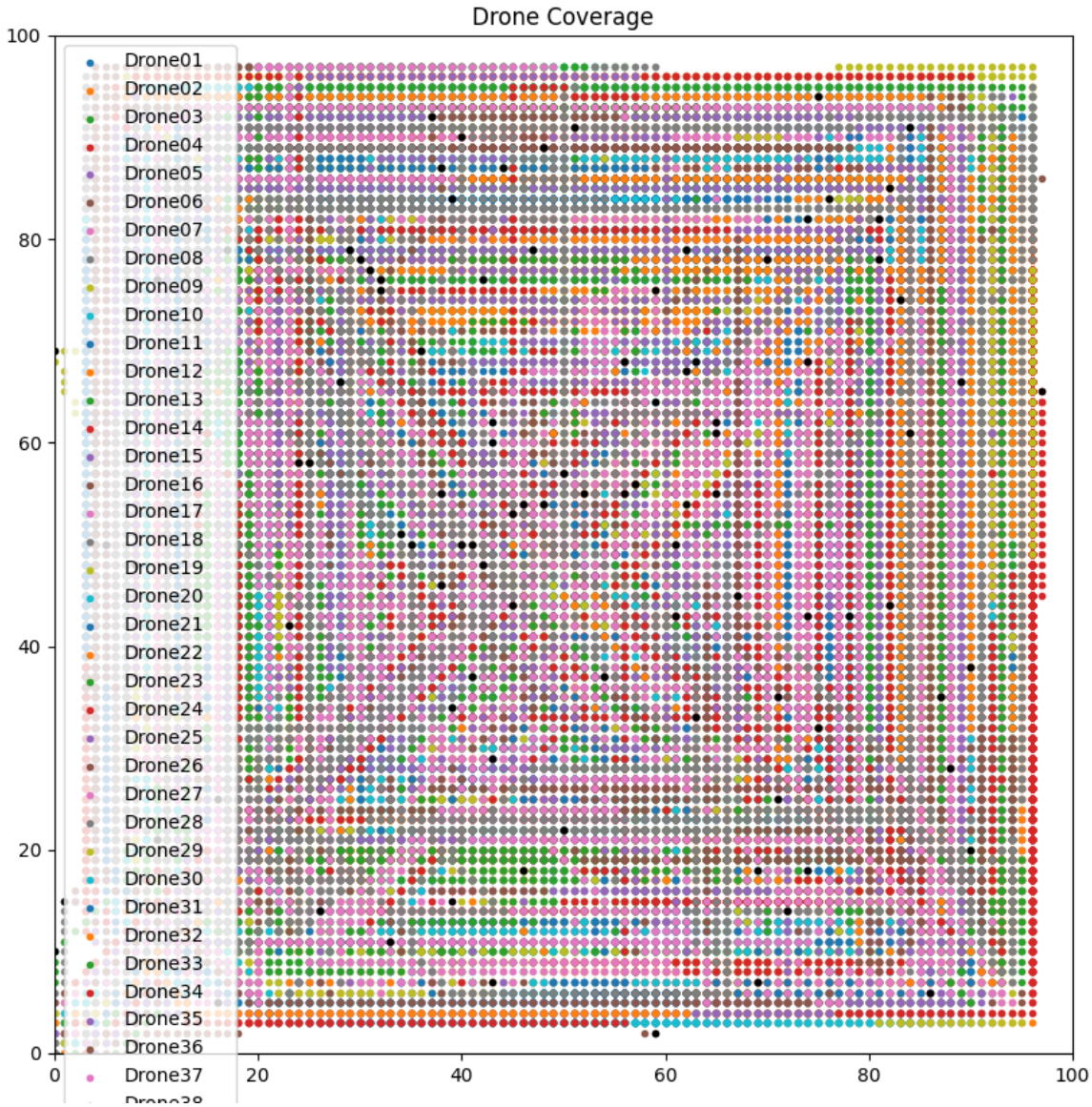


Figure 65: Odometry plot: normalized start location [2] ©2020 IEEE

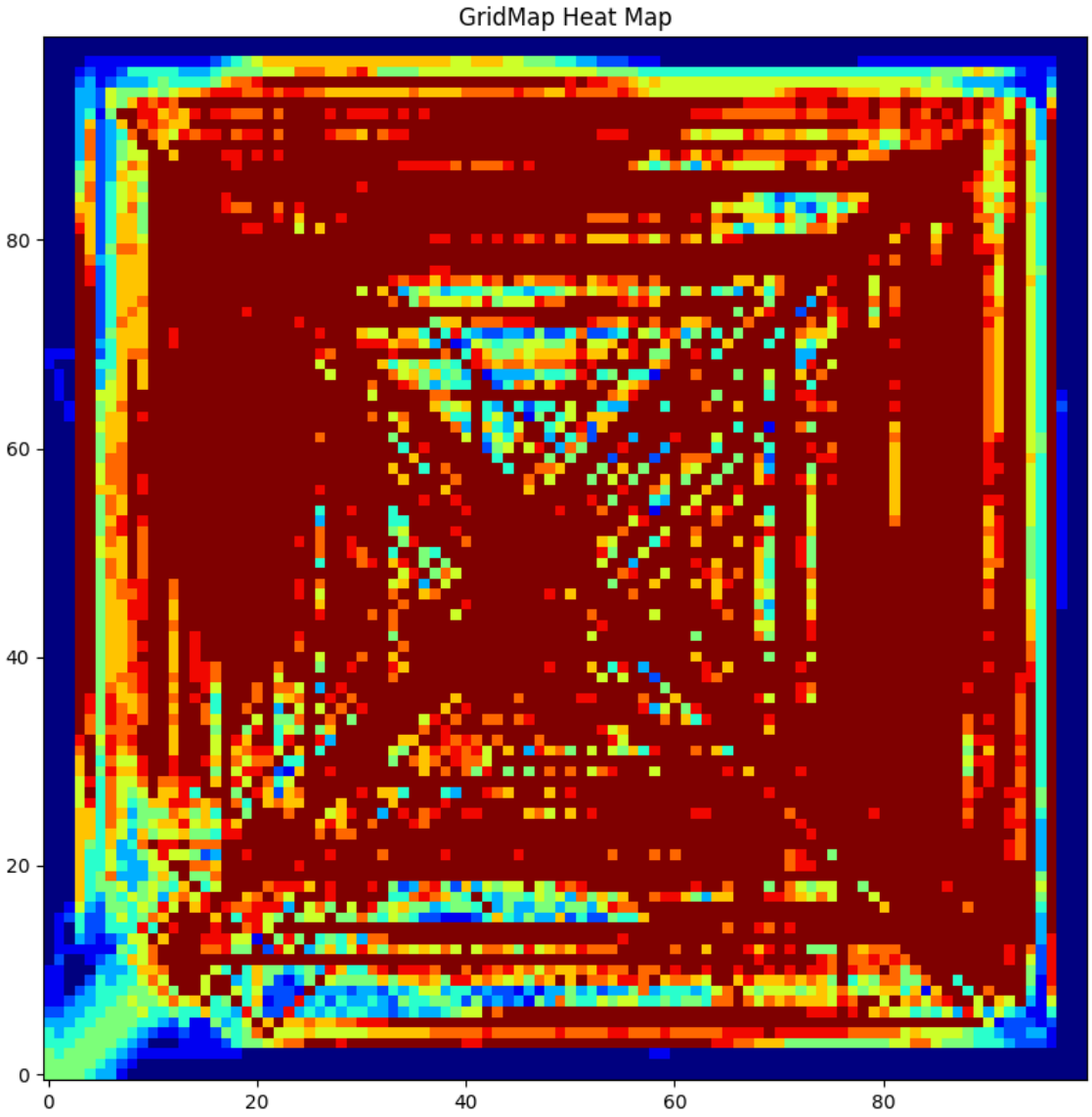


Figure 66: Heatmap: normalized start location [2] ©2020 IEEE

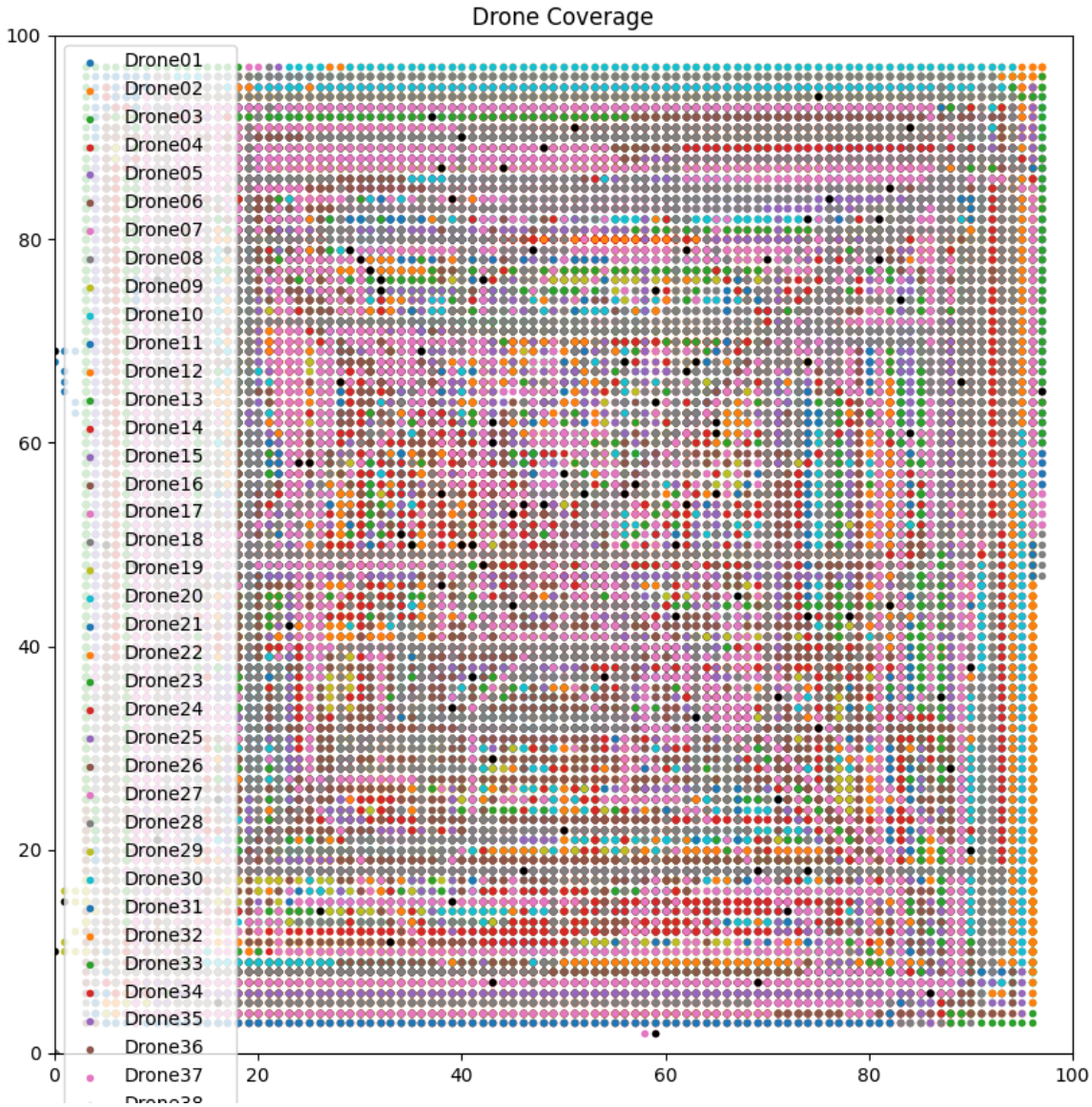


Figure 67: Odometry plot: randomized start locations [2] ©2020 IEEE

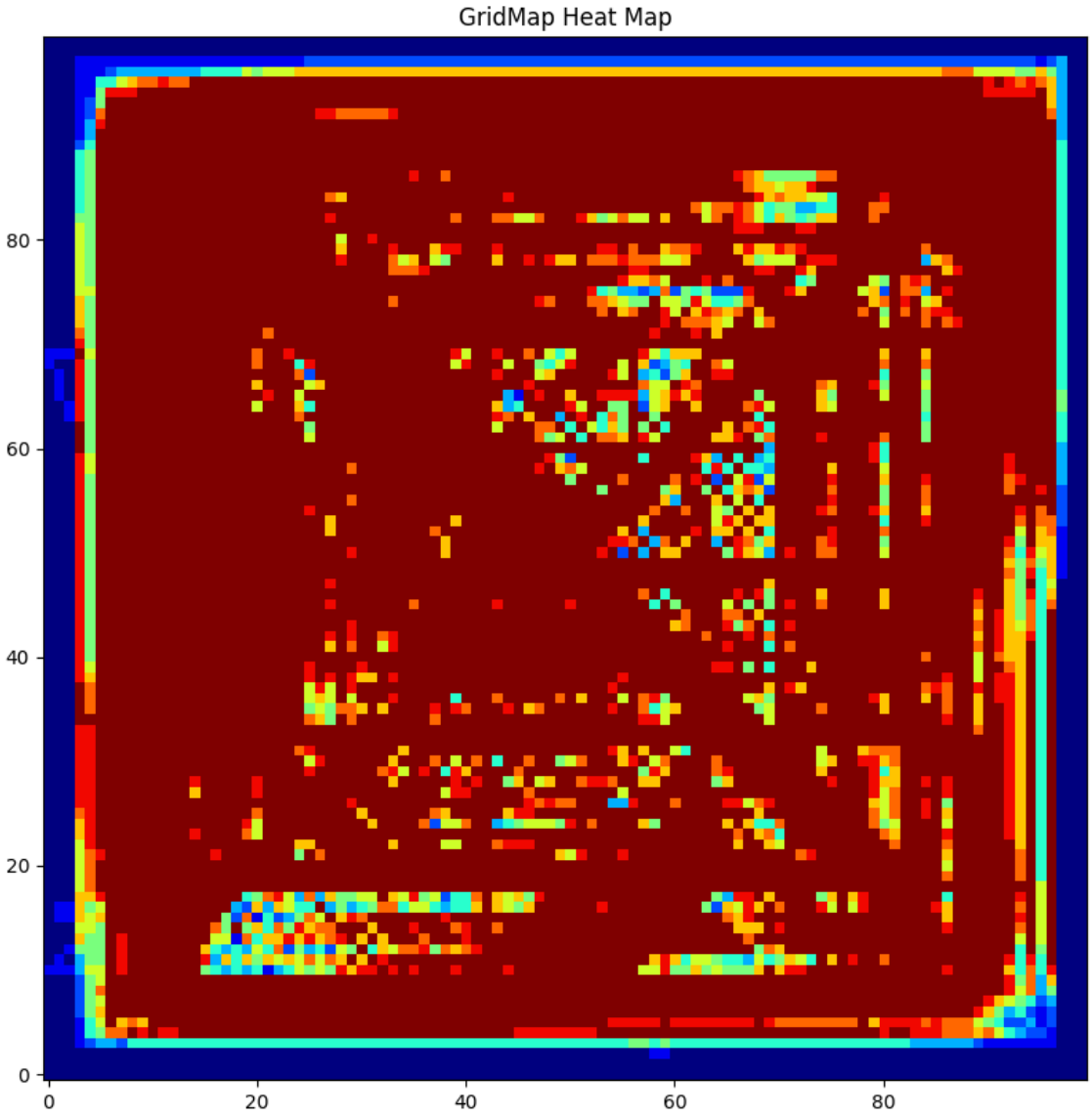


Figure 68: Heatmap: randomized start locations [2] ©2020 IEEE

Bibliography

[1] Sami Mian, Tyler Garrett, Alex Glandon, Chris Manderino, Swee Balachandran, Chester Dolph, and César A Munoz. Autonomous spacecraft inspection with free-flying drones. In *2020 IEEE/AIAA 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020.

[2] Sami Mian, John Hill IV, and Zhi-Hong Mao. Optimal control techniques for heterogeneous uav swarms. In *2020 IEEE/AIAA 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020.

[3] Nikolaus Salvatore, Sami Mian, Collin Abidi, and Alan George. A neuro-inspired approach to intelligent collision avoidance and navigation. In *2020 IEEE/AIAA 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020.

[4] GSFC NASA. core flight system (cfs) background and overview, 2014.

[5] David McComas. Nasa/gsfc's flight software core flight system. 2012.

[6] María Consiglio, César Munoz, George Hagen, Anthony Narkawicz, and Swee Balachandran. Icarous: Integrated configurable algorithms for reliable operations of unmanned systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–5. IEEE, 2016.

[7] Maria Consiglio, Brendan J Duffy, Swee Balachandran, Louis Glaab, and Cesar Munoz. Sense and avoid characterization of the independent configurable architecture for reliable operations of unmanned systems. In *13th USA/Europe Air Traffic Management Research and Development Seminar*, 2019.

[8] Lydia E Kavraki Jean-Claude Latombe. Probabilistic roadmaps for robot path planning. *Pratical motion planning in robotics: current aproaches and future challenges*, pages 33–53, 1998.

[9] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[10] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on control systems technology*, 17(5):1105–1118, 2009.

[11] M Otte, E Frazzoli, and X RRT. Real-time motion planning/replanning for environments with unpredictable obstacles. *Algorithmic Foundations of Robotics XI*, pages 461–478.

[12] Mia N Stevens and Ella M Atkins. Multi-mode guidance for an independent multicopter geofencing system. In *16th AIAA Aviation Technology, Integration, and Operations Conference*, page 3150, 2016.

[13] Evan T Dill, Steven D Young, and Kelly J Hayhurst. Safeguard: An assured safety net technology for uas. In *2016 IEEE/AIAA 35th digital avionics systems conference (DASC)*, pages 1–10. IEEE, 2016.

[14] Roberto Opromolla, Giancarmine Fasano, Giancarlo Rufino, Michele Grassi, and Al Savvaris. Lidar-inertial integration for uav localization and mapping in complex environments. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 649–656. IEEE, 2016.

[15] A Zhahir, A Razali, and M Mohd Ajir. Current development of uav sense and avoid system. In *IOP Conference Series: Materials Science and Engineering*, volume 152, page 12035, 2016.

[16] B Nikhil Chand, P Mahalakshmi, and VPS Naidu. Sense and avoid technology in unmanned aerial vehicles: A review. In *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, pages 512–517. IEEE, 2017.

[17] Armin Strobel and Marc Schwarzbach. Cooperative sense and avoid: Implementation in simulation and real world for small unmanned aerial vehicles. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1253–1258. IEEE, 2014.

[18] Xiaohua Wang, Vivek Yadav, and SN Balakrishnan. Cooperative uav formation flying with obstacle/collision avoidance. *IEEE Transactions on control systems technology*, 15(4):672–679, 2007.

[19] Yucong Lin and Srikanth Saripalli. Sense and avoid for unmanned aerial vehicles using ads-b. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6402–6407. IEEE, 2015.

[20] Massimiliano Iacono and Antonio Sgorbissa. Path following and obstacle avoidance for an autonomous uav using a depth camera. *Robotics and Autonomous Systems*, 106:38–46, 2018.

[21] Milton CP Santos, Lucas V Santana, Alexandre S Brandao, and Mário Sarcinelli-Filho. Uav obstacle avoidance using rgb-d system. In *2015 International Conference On Unmanned Aircraft Systems (ICUAS)*, pages 312–319. IEEE, 2015.

[22] Andrew Viquerat, Lachlan Blackhall, Alistair Reid, Salah Sukkarieh, and Graham Brooker. Reactive collision avoidance for unmanned aerial vehicles using doppler radar. In *Field and Service Robotics*, pages 245–254. Springer, 2008.

[23] Hordur K Heidarsson and Gaurav S Sukhatme. Obstacle detection and avoidance for an autonomous surface vehicle using a profiling sonar. In *2011 IEEE International Conference on Robotics and Automation*, pages 731–736. IEEE, 2011.

[24] Sandra Vieira, Walter HL Pinaya, and Andrea Mechelli. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience & Biobehavioral Reviews*, 74:58–75, 2017.

[25] Yong K Hwang and Narendra Ahuja. Gross motion planning—a survey. *ACM Computing Surveys (CSUR)*, 24(3):219–291, 1992.

[26] SFP Saramago and V Steffen Jr. Optimization of the trajectory planning of robot manipulators taking into account the dynamics of the system. *Mechanism and machine theory*, 33(7):883–894, 1998.

[27] RR Santos, V Steffen Jr, and SFP Saramago. Robot path planning: avoiding obstacles. In *18th International Congress of Mechanical Engineering (COBEM)*, 2005.

[28] Rogério R dos Santos, Valder Steffen, and Sezimária de FP Saramago. Robot path planning in a constrained workspace by using optimal control techniques. *Multibody System Dynamics*, 19(1-2):159–177, 2008.

[29] Daniela Constantinescu and Elizabeth A Croft. Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. *Journal of robotic systems*, 17(5):233–249, 2000.

[30] Oskar von Stryk and Maximilian Schlemmer. Optimal control of the industrial robot manutec r3. In *Computational optimal control*, pages 367–382. Springer, 1994.

[31] W Gerke. Collision-free and shortest paths for industrial robots found by dynamic-programming. *Robotersysteme*, 1(1):43–52, 1985.

[32] Steven Dubowsky, M Norris, and Zvi Shiller. Time optimal trajectory planning for robotic manipulators with obstacle avoidance: a cad approach. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1906–1912. IEEE, 1986.

[33] Zvi Shiller and Steven Dubowsky. Robot path planning with obstacles, actuator, gripper, and payload constraints. *The International Journal of Robotics Research*, 8(6):3–18, 1989.

[34] E Gilbert and Daniel Johnson. Distance functions and their application to robot path planning in the presence of obstacles. *IEEE Journal on Robotics and Automation*, 1(1):21–30, 1985.

[35] Miroslaw Galicki. Optimal planning of a collision-free trajectory of redundant manipulators. *The International journal of robotics research*, 11(6):549–559, 1992.

[36] Miroslaw Galicki. The planning of robotic optimal motions in the presence of obstacles. *The International Journal of Robotics Research*, 17(3):248–259, 1998.

[37] Jacob Mattingley, Yang Wang, and Stephen Boyd. Receding horizon control. *IEEE Control Systems Magazine*, 31(3):52–65, 2011.

[38] Tamás Keviczky, Kingsley Fregene, Francesco Borrelli, Gary J Balas, and Datta Godbole. Coordinated autonomous vehicle formations: decentralization, control synthesis and optimization. In *2006 American Control Conference*, pages 6–pp. IEEE, 2006.

[39] TamáS Keviczky, Francesco Borrelli, and Gary J Balas. Decentralized receding horizon control for large scale dynamically decoupled systems. *Automatica*, 42(12):2105–2115, 2006.

[40] Tamás Keviczky, Francesco Borrelli, Kingsley Fregene, Datta Godbole, and Gary J Balas. Decentralized receding horizon control and coordination of autonomous vehicle formations. *IEEE Transactions on Control Systems Technology*, 16(1):19–33, 2007.

[41] Peter Henderson, Matthew Vertescher, David Meger, and Mark Coates. Cost adaptation for robust decentralized swarm behaviour. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4099–4106. IEEE, 2018.

[42] Lester Ingber. Adaptive simulated annealing (asa): Lessons learned on simulated annealing applied to combinatorial optimization. *Control Cybern. https://doi.org/10.1115/2777*, 1996.

[43] Carles Araguz, Elisenda Bou-Balust, and Eduard Alarcón. Applying autonomy to distributed satellite systems: Trends, challenges, and future prospects. *Systems Engineering*, 21(5):401–416, 2018.

[44] Antony Gillette, Brendan O'Connor, Christopher Wilson, and Alan George. Spacecraft mission agent for autonomous robust task execution. In *2018 IEEE Aerospace Conference*, pages 1–8. IEEE, 2018.

[45] Grégory Beaumet, Gérard Verfaillie, and Marie-Claire Charmeau. Feasibility of autonomous decision making on board an agile earth-observing satellite. *Computational Intelligence*, 27(1):123–139, 2011.

[46] Sara Spangelo, James Cutler, Kyle Gilson, and Amy Cohn. Optimization-based scheduling for the single-satellite, multi-ground station communication problem. *Computers & Operations Research*, 57:1–16, 2015.

[47] Doo-Hyun Cho, Jun-Hong Kim, Han-Lim Choi, and Jaemyung Ahn. Optimization-based scheduling method for agile earth-observing satellite constellation. *Journal of Aerospace Information Systems*, 15(11):611–626, 2018.

[48] Wayne F Boyer and Gurdeep S Hura. Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments. *Journal of Parallel and Distributed Computing*, 65(9):1035–1046, 2005.

[49] Somaiyeh Mahmoud Zadeh. Autonomous reactive mission scheduling and task-path planning architecture for autonomous underwater vehicle. *arXiv preprint arXiv:1706.04189*, 2017.

[50] Yohanes Khosiawan, Youngsoo Park, Ilkyeong Moon, Janardhanan Mukund Nilakantan, and Izabela Nielsen. Task scheduling system for uav operations in indoor environment. *Neural Computing and Applications*, 31(9):5431–5459, 2019.

[51] Steve Chien, Rob Sherwood, Daniel Tran, Benjamin Cichy, Gregg Rabideau, Rebecca Castano, Ashley Davis, Dan Mandl, Stuart Frye, Bruce Trout, et al. Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing, Information, and Communication*, 2(4):196–216, 2005.

[52] S Knight, Gregg Rabideau, Steve Chien, Barbara Engelhardt, and Rob Sherwood. Casper: Space exploration through continuous planning. *IEEE Intelligent Systems*, 16(5):70–75, 2001.

[53] K Center, P Countney, R Adams, DJ Musliner, MJ Pelican, J Hamell, D Kortenkamp, MB Hudson, JL Fausz, and P Zetocha. Improving decision support systems through development of a modular autonomy architecture. In *Proceedings of the 2012 I-SAIRAS Conference*, 2012.

[54] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[55] ROS Wiki Page. Url: http://wiki. ros. org/ros. *Introduction (visited on 04/20/2017)*.

[56] Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *International workshop on swarm robotics*, pages 10–20. Springer, 2004.

[57] G Dudek, M Jenkin, E Milios, and D Wilkes. A taxonomy for swarm robots, intelligent robots and systems' 93, iros'93. In *Proceedings of the 1993 IEEE/RSJ International Conference on*, volume 1, 1993.

[58] Y Uny Cao, Alex S Fukunaga, and Andrew Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, 4(1):7–27, 1997.

[59] Jan Carlo Barca and Y Ahmet Sekercioglu. Swarm robotics reviewed. *Robotica*, 31(3):345–359, 2013.

[60] Sean J Edwards. Swarming on the battlefield: past, present, and future. Technical report, RAND NATIONAL DEFENSE RESEARCH INST SANTA MONICA CA, 2000.

[61] Patrick Vincent and Izhak Rubin. A framework and analysis for cooperative search using uav swarms. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 79–86, 2004.

[62] Jorge Cortes, Sonia Martinez, Timur Karatas, and Francesco Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on robotics and Automation*, 20(2):243–255, 2004.

[63] Kristina Lerman, Alcherio Martinoli, and Aram Galstyan. A review of probabilistic macroscopic models for swarm robotic systems. In *International workshop on swarm robotics*, pages 143–152. Springer, 2004.

[64] Jan Carlo Barca, Grace Rumantir, and Raymond Li. A concept for optimizing behavioural effectiveness & efficiency. In *Intelligent Engineering Systems and Computational Cybernetics*, pages 449–458. Springer, 2009.

[65] Ming Ma and Yuanyuan Yang. Adaptive triangular deployment algorithm for unattended mobile sensor networks. *IEEE Transactions on Computers*, 56(7):946–847, 2007.

[66] Lynne E Parker. Designing control laws for cooperative agent teams. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 582–587. IEEE, 1993.

[67] V. J. Lumelsky and K. R. Harinarayan. Decentralized motion planning for multiple mobile robots: The cocktail party model. *Autonomous Robots*, 4(1SN - 1573-7527):121–135, Mar 1997.

[68] Michael G Hinchey, Roy Sterritt, and Chris Rouff. Swarms and swarm intelligence. *Computer*, 40(4):111–113, 2007.

[69] Sonia Martinez, Jorge Cortes, and Francesco Bullo. Motion coordination with distributed information. *IEEE control systems magazine*, 27(4):75–88, 2007.

[70] Jacques Penders, Lyuba Alboul, Ulf Witkowski, Amir Naghsh, Joan Saez-Pons, Stefan Herbrechtsmeier, and Mohamed El-Habbal. A robot swarm assisting a human firefighter. *Advanced Robotics*, 25(1-2):93–117, 2011.

[71] Julián Colorado, Antonio Barrientos, Claudio Rossi, and Jaime del Cerro. Follow-the-leader formation marching through a scalable o (log 2 n) parallel architecture. In

2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 5583–5588. IEEE, 2010.

[72] Nojeong Heo and Pramod K Varshney. Energy-efficient deployment of intelligent mobile sensor networks. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 35(1):78–92, 2004.

[73] Sami Mian. A novel battery management & charging solution for autonomous uav systems. Master's thesis, ARIZONA STATE UNIVERSITY, 2018.

[74] Christoph Posch, Teresa Serrano-Gotarredona, Bernabe Linares-Barranco, and Tobi Delbruck. Retinomorphic event-based vision sensors: bioinspired cameras with spiking output. *Proceedings of the IEEE*, 102(10):1470–1484, 2014.

[75] Christoph Posch, Daniel Matolin, and Rainer Wohlgenannt. An asynchronous time-based image sensor. In *2008 IEEE International Symposium on Circuits and Systems*, pages 2130–2133. IEEE, 2008.

[76] Tobi Delbrück, Bernabe Linares-Barranco, Eugenio Culurciello, and Christoph Posch. Activity-driven, event-based vision sensors. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 2426–2429. IEEE, 2010.

[77] Garibaldi Pineda García, Patrick Camilleri, Qian Liu, and Steve Furber. pydvs: An extensible, real-time dynamic vision sensor emulator using off-the-shelf hardware. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE, 2016.

[78] Matthew L Katz, Konstantin Nikolic, and T Delbruck. Live demonstration: Behavioural emulation of event-based vision sensors. In *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 736–740. IEEE, 2012.

[79] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *The International Journal of Robotics Research*, 36(2):142–149, 2017.

[80] Rika Sugimoto Dimitrova, Mathias Gehrig, Dario Brescianini, and Davide Scaramuzza. Towards low-latency high-bandwidth control of quadrotors using event cameras. *arXiv preprint arXiv:1911.04553*, 2019.

[81] Elias Mueggler, Nathan Baumli, Flavio Fontana, and Davide Scaramuzza. Towards evasive maneuvers with quadrotors using dynamic vision sensors. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–8. IEEE, 2015.

[82] Davide Falanga, Suseong Kim, and Davide Scaramuzza. How fast is too fast? the role of perception latency in high-speed sense and avoid. *IEEE Robotics and Automation Letters*, 4(2):1884–1891, 2019.

[83] Ryad Benosman, Sio-Hoi Ieng, Charles Clercq, Chiara Bartolozzi, and Mandyam Srinivasan. Asynchronous frameless event-based optical flow. *Neural Networks*, 27:32–37, 2012.

[84] Seth Roffe and Alan D George. Evaluation of algorithm-based fault tolerance for machine learning and computer vision under neutron radiation. In *2020 IEEE Aerospace Conference*, pages 1–9. IEEE, 2020.

[85] Brian Weeden, Paul Cefola, and Jaganath Sankaran. Global space situational awareness sensors. In *AMOS Conference*, 2010.

[86] David Reverter Valeiras, Xavier Lagorce, Xavier Clady, Chiara Bartolozzi, Sio-Hoi Ieng, and Ryad Benosman. An asynchronous neuromorphic event-driven visual part-based shape tracking. *IEEE transactions on neural networks and learning systems*, 26(12):3045–3059, 2015.

[87] Joachim Ender, Ludger Leushacke, Andreas Brenner, and Helmut Wilden. Radar techniques for space situational awareness. In *2011 12th International Radar Symposium (IRS)*, pages 21–26. IEEE, 2011.

[88] Kyle Jordan DeMars. *Nonlinear orbit uncertainty prediction and rectification for space situational awareness*. PhD thesis, 2010.

[89] Richard I Abbot and Timothy P Wallace. Decision support in space situational awareness. *Lincoln Laboratory Journal*, 16(2):297, 2007.

[90] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. nips deep learning workshop, 2013. *arXiv preprint arXiv:1312.5602*, 2013.

[91] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[92] Abhik Singla, Sindhu Padakandla, and Shalabh Bhatnagar. Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[93] Daniel Rasmussen. Nengodl: Combining deep learning and neuromorphic modelling methods. *Neuroinformatics*, 17(4):611–628, 2019.

[94] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[95] Chia Yu Ho, Shau Yin Tseng, Chin Feng Lai, Ming Shi Wang, and Ching Ju Chen. A parameter sharing method for reinforcement learning model between airsim and uavs. In *2018 1st International Cognitive Cities Conference (IC3)*, pages 20–23. IEEE, 2018.

[96] Łukasz Kuśmierz, Takuya Isomura, and Taro Toyoizumi. Learning with three factors: modulating hebbian plasticity with errors. *Current opinion in neurobiology*, 46:170–177, 2017.

[97] Yu Miao, Huajin Tang, and Gang Pan. A supervised multi-spike learning algorithm for spiking neural networks. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2018.

[98] Taras Iakymchuk, Alfredo Rosado-Muñoz, Juan F Guerrero-Martínez, Manuel Bataller-Mompeán, and Jose V Francés-Víllora. Simplified spiking neural network architecture and stdp learning algorithm applied to image classification. *EURASIP Journal on Image and Video Processing*, 2015(1):4, 2015.

[99] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: opportunities and challenges. *Frontiers in neuroscience*, 12:774, 2018.

[100] Moritz B Milde, Hermann Blum, Alexander Dietmüller, Dora Sumislawska, Jörg Conradt, Giacomo Indiveri, and Yulia Sandamirskaya. Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system. *Frontiers in neurorobotics*, 11:28, 2017.

[101] Emad Ebeid, Martin Skriver, Kristian Husum Terkildsen, Kjeld Jensen, and Ulrik Pagh Schultz. A survey of open-source uav flight controllers and flight simulators. *Microprocessors and Microsystems*, 61:11–20, 2018.

[102] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[103] Ratnesh Madaan, Nicholas Gyde, Sai Vemprala, Matthew Brown, Keiko Nagami, Tim Taubner, Eric Cristofalo, Davide Scaramuzza, Mac Schwager, and Ashish Kapoor. Airsim drone racing lab. *arXiv preprint arXiv:2003.05654*, 2020.

[104] Ivan Reljić, Ivan Dunder, and Sanja Seljan. Photogrammetric 3d scanning of physical objects: Tools and workflow. *TEM Journal*, 8(2):383, 2019.

[105] Autonomy Incubator. Free flyers: Autonomous coordinated operations.

[106] Parrot. Parrot ar.drone 2.0 power edition.

[107] Carlos Sampedro, Hriday Bavle, Jose Luis Sanchez-Lopez, Ramon A Suárez Fernández, Alejandro Rodríguez-Ramos, Martin Molina, and Pascual Campoy. A flexible and dynamic mission planning architecture for uav swarm coordination. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 355–363. IEEE, 2016.

[108] John Lyle Vian, Ali Reza Mansouri, and Emad William Saad. System and method for inspection of structures and objects by swarm of remote unmanned vehicles, November 15 2011. US Patent 8,060,270.

[109] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer networks*, 52(12):2292–2330, 2008.

[110] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

[111] JE Hurtado and AH Barbat. Monte carlo techniques in computational stochastic mechanics. *Archives of Computational Methods in Engineering*, 5(1):3, 1998.

[112] Paolo Brandimarte. *Handbook in Monte Carlo simulation: applications in financial engineering, risk management, and economics*. John Wiley & Sons, 2014.

[113] David Tedaldi, Guillermo Gallego, Elias Mueggler, and Davide Scaramuzza. Feature detection and tracking with the dynamic and active-pixel vision sensor (davis). In *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pages 1–7. IEEE, 2016.

[114] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[115] Huaxi Zhang and Lei Zhang. Cloud robotics architecture: trends and challenges. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 362–3625. IEEE, 2019.

[116] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[117] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.