University of Kentucky

UKnowledge

Theses and Dissertations--Electrical and Computer Engineering

Electrical and Computer Engineering

2022

Developing Reactive Distributed Aerial Robotics Platforms for Real-time Contaminant Mapping

Joshua Ashley
University of Kentucky, jashley2017@gmail.com
Author ORCID Identifier:
https://orcid.org/0000-0002-4971-8756

Digital Object Identifier: https://doi.org/10.13023/etd.2022.378

Right click to open a feedback form in a new tab to let us know how this document benefits you.

Recommended Citation

Ashley, Joshua, "Developing Reactive Distributed Aerial Robotics Platforms for Real-time Contaminant Mapping" (2022). *Theses and Dissertations--Electrical and Computer Engineering*. 186.
https://uknowledge.uky.edu/ece_etds/186

This Master's Thesis is brought to you for free and open access by the Electrical and Computer Engineering at UKnowledge. It has been accepted for inclusion in Theses and Dissertations--Electrical and Computer Engineering by an authorized administrator of UKnowledge. For more information, please contact UKnowledge@lsv.uky.edu.

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student’s advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student’s thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Joshua Ashley, Student

Dr. Biyun Xie, Major Professor

Dr. Daniel Lau, Director of Graduate Studies

Developing Reactive Distributed Aerial Robotics Platforms for Real-time Contaminant Mapping

---

Thesis

A thesis submitted in partial
fulfillment of the requirements for
the Master's degree in the College of
Engineering at the University of
Kentucky
By
Joshua A. Ashley
Lexington, Kentucky

Directors: Dr. Biyun Xie, Professor of Electrical Engineering and Dr. Michael Sama, Professor of Biosystems and Agricultural Engineering
Lexington, Kentucky
2022

Copyright© Joshua A. Ashley 2022

ABSTRACT OF THESIS

Developing Reactive Distributed Aerial Robotics Platforms for Real-time Contaminant Mapping

The focus of this research is to design a sensor data aggregation system and centralized sensor-driven trajectory planning algorithm for fixed-wing aircraft to optimally assist atmospheric simulators in mapping the local environment in real-time. The proposed application of this work is to be used in the event of a hazardous contaminant leak into the atmosphere as a fleet of sensing unmanned aerial vehicles (UAVs) could provide valuable information for evacuation measures. The data aggregation system was designed using a state-of-the-art networking protocol and radio with DigiMesh and a process/data management system in the ROS2 DDS. This system was tested to consistently operate within the latencies and distances tolerated for the project while being highly extensible to sensor configurations. The problem of creating optimal trajectory planning for exploration has been modelled accurately using partially-observable Markov decision processes (POMDP). Deep Reinforcement learning (DRL) is commonly applied to approximate optimal solutions within a POMDP as it can be analytically intractable for complex state spaces. This research produces a POMDP that describes this exploration problem and applies the state-of-the-art soft actor-critic (SAC) reinforcement learning algorithm to create a policy that produces near-optimal trajectories within this new POMDP. A subset of the spatially relevant input is used instead of complete state during training and a turn-taking sequential planner is designed for using multiple UAVs to help mitigate scalability problems that come with multi-UAV coordination. The learned policy from SAC can outperform a greedy and fixed trajectory on 1, 2, and 3 UAVs by a 30% margin on average. The turn-taking strategy provides small, but repeatable scaling benefits while the windowed input results in a 50%-60% increase in reward versus trained networks without windowed input. The proposed planning algorithm is effective in dynamic map exploration and has the potential to increase UAV effectiveness in atmospheric contaminant leak monitoring as it is expanded to be integrated on real-world UAVs.

KEYWORDS: aerial, robotics, planning, reinforcement learning, UAVs

Joshua A. Ashley

October 22, 2022

Developing Reactive Distributed Aerial Robotics Platforms for Real-time Contaminant Mapping

By

Joshua A. Ashley

Dr. Biyun Xie & Dr. Michael Sama

Directors of Thesis

Dr. Daniel Lau

Director of Graduate Studies

October 22, 2022

Date

ACKNOWLEDGMENTS

.

This work could not have been accomplished without the continuous mentorship and support of my research from my faculty advisors, Dr. Biyun Xie and Dr. Michael Sama. I would also like to thank my other committee member Dr. Jesse Hoagg for his leading role on the DART project for which my work was a part of and his enthusiasm for my contributions to the project.

Additionally I would like to thank the National Science Foundation (NSF Grant #1932105) and the University of Kentucky Aerial Research Lab for the continued support throughout the completion of this degree.

TABLE OF CONTENTS

Acknowledgments iii

List of Tables vi

List of Figures vii

Chapter 1 Introduction 1

1.1 Application and Significance 1

1.2 Objective 3

1.3 Literature Review 3

1.3.1 Long Range Network Protocols 3

1.3.2 Networking Middleware 4

1.3.3 Aerial Gas Detection Methods 5

1.3.4 POMDPs for Aerial Mapping and Exploration 6

Chapter 2 Designing a Data Aggregation System 7

2.1 System Design 7

2.1.1 Overview 7

2.1.2 Sensor Abstraction 7

2.1.3 High-Level Protocol 9

2.1.4 Message Serialization 10

2.1.5 Network Setup 12

2.1.6 GPS Timing 13

2.2 Design Validation 13

2.2.1 Experiment Setup 14

2.2.2 Results 15

2.2.3 Conclusion 15

Chapter 3 Creating a Sensor-driven Multi-UAV Trajectory Planning Algorithm 18

3.1 Problem Model 18

3.1.1 Overview 18

3.1.2 WRF Model 19

3.1.3 Belief Model 20

3.1.4 Aircraft Model 23

3.1.5 POMDP Model 23

3.2 Methodology 24

3.2.1 Using Soft Actor Critic to Approximate the Optimal Solution of a POMDP 24

3.2.2 Neural Network Architecture and Training 26

3.2.3 Reducing State Dimensionality by Accounting for Environment Dynamics 27

3.2.4 Extension to Multiple UAVs 29

3.3 Results 31

3.3.1 Model Verification Process 31

3.3.2 Simulation Comparison for One UAV 34

3.3.3 Simulation Comparison for Multiple UAVs 35

3.4 Conclusion 37

Chapter 4 Conclusion 39

4.1 Summary 39

4.2 Future Work 40

Bibliography 41

Vita 46

LIST OF TABLES

1.1 Mesh network protocol comparison 4

2.1 Messages from UAV to ground station 10
2.2 Messages from ground station to UAV 10
2.3 Packet delay metrics for one and two UAVs 16

3.1 Training Hyperparameters 27

LIST OF FIGURES

1.1 A visualization of how a mobile sensing platform can be used in a contaminant leak scenario is shown. As the platform passes through the contaminant it is able to measure the local atmospheric properties to help aid in estimating the contaminant trajectory. 2

1.2 A visualization of a flat mesh network. In a flat mesh network, all devices have the same role allowing them to send their own data along the network and act as a router for other data. 5

2.1 A visualization the data flow from a sensor to the ground station and ground applications through the described system. 8

2.2 The packet layout for an example ROS2 message type following the serialization method described in this section. The packet header consists of a sensor code and GPS time which consume 10 bytes followed by the byte-indexed sensor data. 12

2.3 The computer hardware that is put on the UAVs during flight is shown. All sensor data is aggregated to the Raspberry Pi and then transmitted over the DigiMesh network. The Neo-M8P RTK GPS is used to associate all sensor readings to a GPS time and location with high accuracy. . . . 15

2.4 Maps of the packet delay from the first two experiments. The position of the points correspond to the recorded position of the UAV when that packet information was recorded and the color of the points indicate the delay. 16

3.1 An overview of the data relationship of the each component model is shown. This overview shows the positive feedback relationship between new actions taken by the policy and information gain that increases model accuracy. 19

3.2 A visualization of how measurements within the flowfield affect the belief space in different ways is shown. The values within the belief map are color coded with higher confidences being assigned green and lower confidences being assigned red. The arrows represent the direction and magnitude of the wind vector for each cell. An image of the belief map before, Fig. 3.2(a) and after measurements from within two different flow fields is made at point $x = 20$, $y = 20$ is shown, Fig. 3.2(b) and Fig. 3.2(c). 21

3.3 The data-flow diagram for a training iteration in SAC. Replay memory is the input data which consists of a representative data set of the POMDP. Yellow boxes indicate the parameter update functions with the blue boxes showing the network that receives those updates. 27

3.4 The structure of the neural networks that are trained by SAC is shown. Both the DQNs and policy networks use this structure with only a different output layer. The output layer shown describes the stochastic policy output which is a distribution $\mathcal{N}(\mu_a, \sigma_a)$. 28

3.5 An example of the total state being sliced into a window based on the current location of the UAV. This slice is what is given to the trained agent which produces an action from that windowed input. 30

3.6 Training progress is made over 1,200,000 iterations. Reward is taken over an average of 50 verification runs. The yellow and grey dotted line shows the baseline average reward of the lawnmower and greedy policies respectively. 32

3.7 Training progress is made over 800,000 iterations. A 'crash' is qualified by the UAV going out of the boundary of the map. The yellow and grey dotted line shows the baseline average reward of the lawnmower and greedy policies respectively. 33

3.8 The generated policy by the proposed algorithm produces a trajectory that is compared to the other policies in this figure. Fig. 3.8(a) shows the starting value map. Fig. 3.8(b) shows the trajectory made by the agent produced by SAC. Fig. 3.8(c) shows the greedy policy trajectory. Fig. 3.8(d) shows the fixed trajectory. 35

3.9 The different policies' reward are compared over 100 runs for 1, 2, and 3 UAVs flying simultaneously. This shows a clustered column chart of the mean and standard deviation for the reward across these 100 runs for each. 37

3.10 Example trajectories made by multiple UAVs using the turn-taking strategy on the trained policy is shown. Fig. 3.10(a) is the starting value map that is given to the policy as input, and fig. 3.10(b) is the resulting total trajectories planned by the policy and the resulting impact on the map. 38

Chapter 1 Introduction

1.1 Application and Significance

Predicting airborne pollutant dispersion is a highly complex problem to model with real world relevance in emergency situations. A large amount of variables in the atmosphere determine contaminant transport such as local geography, turbulence, and heat which in most situations must be accounted for over a large area with these variables changing at small scales. This contributes to the problem of simulating contaminant transport even with sparse point measurements which may not adequately quantify the contaminant distribution in a dynamic environment [25]. Mobile sensing systems deployed on a fixed-wing unmanned aerial vehicle (UAV) provide an opportunity to obtain airborne measurements of information such as contaminant concentrations and wind speeds at optimal positions and times for detecting an unplanned release of contaminant. Therefore, UAVs used in conjunction with simulation have the potential to improve our understanding of contaminant distribution by moving the sensor through the contaminant or by quantifying the environmental conditions that control contaminant transport [27]. Fig. 1.1 shows a real world example scenario of such a contaminant leak with a mobile sensing platform. The colored lines in the figure correspond to UAV trajectories taken during the flight. In this example three fixed-wing UAVs are flying simultaneously.

In a 2010-2014 registry tracked by the U.S. Department of Health and Human Services (HHS), hazardous emergencies involving toxic substances reported from nine states were the cause of 5,134 injuries and 190 fatalities [20] with natural gas being the most frequent substance. At a national scale, the number of injuries and fatalities are likely much higher. In the US there have been a multitude of high profile natural gas leaks. These leaks pose the risk of explosion exemplified by an incident

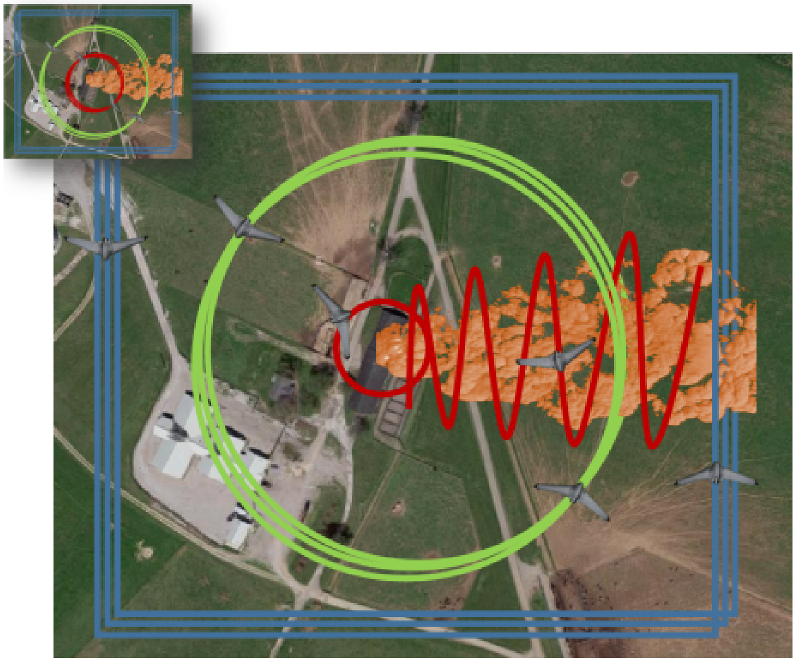


Figure 1.1: A visualization of how a mobile sensing platform can be used in a contaminant leak scenario is shown. As the platform passes through the contaminant it is able to measure the local atmospheric properties to help aid in estimating the contaminant trajectory.

in Edison, New Jersey in 1994 which destroyed or damaged 14 apartment buildings. They also pose the risk of exposure as was observed in the Aliso Canyon natural gas leak where 2,200 families were relocated due to an unusual increase in illnesses such as nausea, severe nosebleeds, and common infections [8, 15]. The Aliso Canyon leak in particular has a pre-existing example of aircraft being used for estimation of point source emissions with some effect [10]. This study also highlights a broader application of the work from this project in modelling urban greenhouse gas emissions. Better analysis of these urban environments could provide tools for planning and mitigation of greenhouse gas toxicity in urban populations. A system that better predicts real-time contaminant distribution would be able to better inform emergency response and evacuation which could prevent injuries and even save lives [5].

However, the problem of real-time contaminant tracking with mobile platforms poses a variety of technical challenges that must first be addressed before an effective system can be deployed. The overall goal of the project is to develop a real-time, sensor-driven flight system that can efficiently and accurately map and predict the surrounding atmospheric environment in the event of a major contaminant being introduced into the atmosphere.

1.2 Objective

The focus of this work is to describe two contributions in designing these sensor-driven flight systems. The first contribution, detailed in Chapter 2, addresses the challenge of real-time sensor data aggregation of multi-UAV systems. The contaminant simulation will be hosted on a centralized server in order to perform model adaptation in real time. This server also needs to be fed environmental information gathered by the UAVs in real time so that the system can produce the most recent and accurate model possible of the current weather environment. Therefore, some real-time data aggregation system is required. The second contribution, detailed in Chapter 3, shows an approach to sensor-driven trajectory planning specifically tailored to near-optimal contaminant mapping using deep reinforcement learning (DRL). This algorithm quantifies prediction confidences in the model from simulation to inform the UAVs' trajectories towards areas that would best increase the simulation confidence in the future. The remainder of this chapter is dedicated to the literature review investigating the current technologies applicable to these problems.

1.3 Literature Review

1.3.1 Long Range Network Protocols

The application of distributed aerial robotics presents challenges not only at the application level of the network, but additionally at the lower levels of protocol and hardware. The problem, as described in [11], is that UAVs require an ad-hoc network

Table 1.1: Mesh network protocol comparison

Protocol	**Range (km/node)**	**RTT Delay (s @ 60 Bytes)**	**Reconfigurablity**
DigiMesh	0.1-4	0.14	High (Flat Network)
LoraMesh	1-13	1	High (Flat Network)
ZigBee	0.01-4	0.14	Medium (Hierarchical)

that can reconfigure itself at high speeds. This survey categorizes the objective of sensing with multiple UAVs as an infrastructure-based mesh network with centralized control.

[21] uses LoRa mesh networking in emergency environmental monitoring for UAVs but notes the low network bandwidth in addition to high end-to-end delay that makes for difficulties in centralized planning. [31] and [17] compare implementations of the two mesh networking protocols Zigbee and Digimesh on Xbee radio hardware. The results show that DigiMesh has advantages in throughput requirements and versatility given that DigiMesh is a flat network while Zigbee is not [17]. Additionally, while Zigbee does have lower packet round trip time (RTT), the difference is small enough to warrant the trade-off in selecting DigiMesh. Table 1.3.1 shows a very short synopsis of relevant factors compiled from the literature leading to choosing DigiMesh.

1.3.2 Networking Middleware

With the low-level protocol selected, there needs to be an application-level system that manages the transport of sensor information over the network coming from a wide variety of sensor types asynchronously. Robot operating system 2 (ROS2) is a robotics and networking middleware that utilizes a data distribution service (DDS) for efficient and reliable inter-process communication both locally and globally within a network [26]. ROS2 therefore provides a scalable solution for multi-UAV networked communication system. [23] use ROS2 to scale micro-aerial vehicles (MAVs) swarms in a truly distributed system. [33] uses ROS and ROS2 to create a decentralized

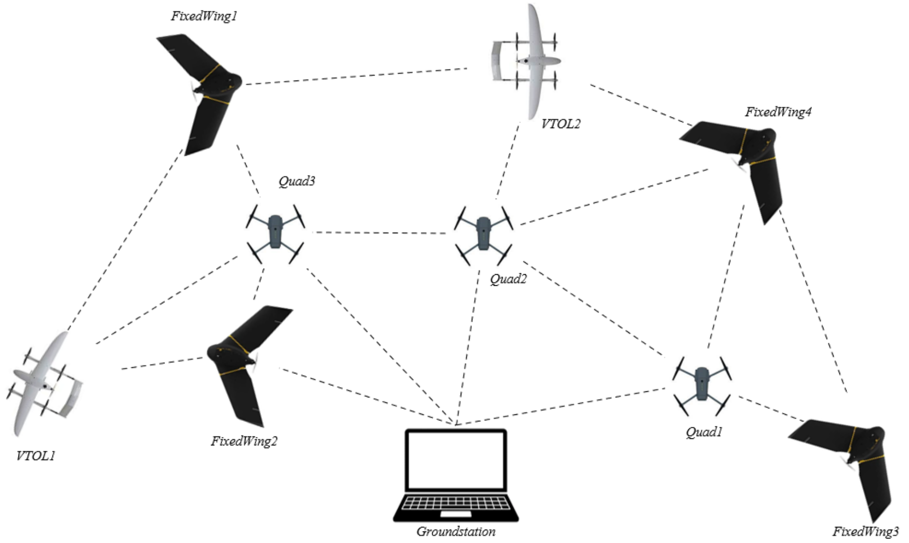


Figure 1.2: A visualization of a flat mesh network. In a flat mesh network, all devices have the same role allowing them to send their own data along the network and act as a router for other data.

cooperative search algorithm using multiple UAVs leveraging ROS for the interface with controls and telemetry and ROS2 as the communication middleware.

1.3.3 Aerial Gas Detection Methods

Existing studies approach the problem of aerial gas detection and sampling using hovering vehicles such as quad-copters [1, 3] or hexa-copters [24, 30]. These studies note limitations or trade-offs in measurement accuracy and flight time due to larger copters that can fly longer having a larger impact on the local wind environment. Fixed-wing UAVs have an advantage in this regard for wind measurements as they are able to use simpler sensors, such as 5-hole probes, that can more easily be placed to avoid the atmospheric influence of the aircraft [32]. Fixed-wing UAVs also benefit from being faster and more efficient which increases the area coverage of each UAV.

Exploiting the high correlation between gas and wind measurements allows for simplified simulation and mapping of gas in mobile robotics. This shows that a multivariate Gaussian joint estimation is accurate in simulating the fluid-dynamics generated ground truth [9]. A similar approach to this Gaussian joint estimation is used in this thesis. The problem of trajectory planning to take wind measurements within this map has been formulated as an optimization of information theoretic and has been evaluated to be performant in comparison to other methods such as Kullback-Leibler divergence, random walk, and fixed trajectories [2, 7, 14]. However these strategies focus on maximizing immediate rewards with points of high information gain that lead to sub-optimal trajectories over the course of a complete path.

1.3.4 POMDPs for Aerial Mapping and Exploration

Partially observable Markov decision processes (POMDPs) offer a more complete formulation of the information theoretic approach to map exploration. Map exploration uses the Bellman equation and measurement expectations to inform the total future potential reward of taking certain actions out to an infinite time horizon [29]. However, POMDPs can quickly become computation intractable with complex environments. Approximate solutions to POMDP models have been applied to fixed-wing UAV map exploration problems. One such example uses state space discretization and a direct asynchronous graph search to simplify computation [22]. The method still requires high online computation and does not scale to multiple UAVs. A more scalable approach to the problem uses deep reinforcement learning (DRL) specifically using deep Q-networks (DQN) to learn the value function associated to the POMDP iteratively through training [16]. The method requires high offline computation during training and needs to retrain the DQN for each possible quantity of UAVs.

Copyright© Joshua A. Ashley, 2022.

Chapter 2 Designing a Data Aggregation System

2.1 System Design

2.1.1 Overview

The flow of data from the sensor to the ground station is the primary goal of automation in this work. Firstly, the data is collected from the sensor which is a very sensor-specific operation and requires a sensor interface be developed for each type of sensor used. Once the sensor interface extracts the data, that sensor data is then published over the ROS2 DDS as messages. Messages are serialized according to the process outlined in section 2.1.4 and sent over the DigiMesh network as the ROS2 DDS cannot be asserted over the entire network due to bandwidth and latency constraints. Instead highly compressed messages are transmitted with specifications of how to decompress them present on both the planes and ground station. The ground station receives and deserializes these messages back into their sensor data. Once deserialized, the sensor data is formatted into a database entry with the associated tags to relate it to its respective sensor type. The entry is inserted into a local time-series database that is optimized for asynchronous modifications and accesses of time series data. These asynchronous accesses are how ground-based applications are able to access sensor data in real time or any relevant historical sensor data.

2.1.2 Sensor Abstraction

In ROS2, self-contained processes called nodes are connected though a data distributions service (DDS) using a message passing schema. These nodes provide the ability to interface with this DDS with interruptable callbacks for receiving data (subscription) and continuous loops for collecting data (publication). On top of these nodes

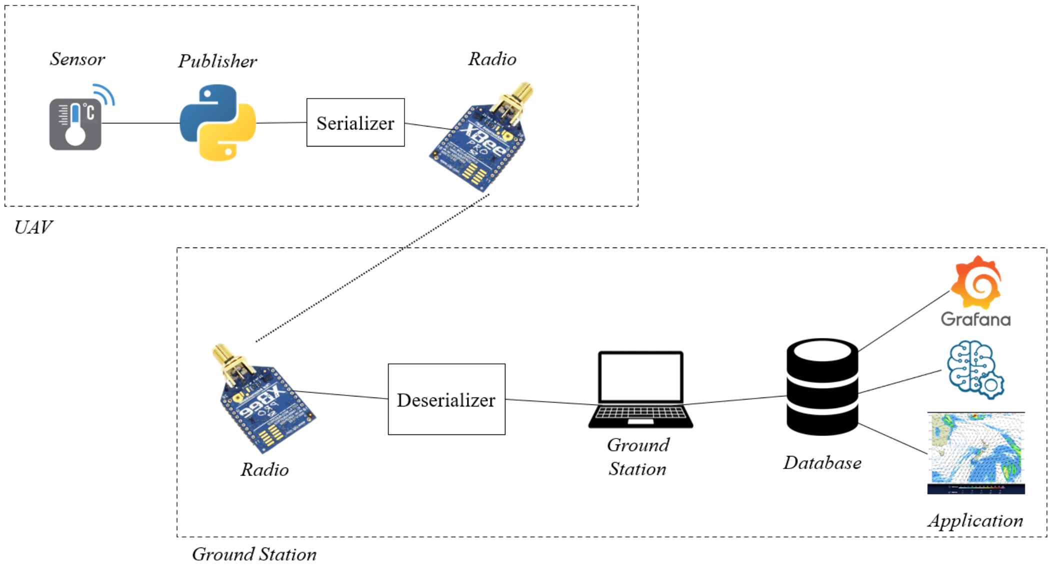


Figure 2.1: A visualization the data flow from a sensor to the ground station and ground applications through the described system.

an abstraction called sensor nodes was created to generically handle the message serialization for transmitting and categorization/tagging for database entry.

The creator of a sensor node writes a program very similar in form to a simple publisher [26] with the primary difference of inheriting from a class 'Sensor' instead of the ROS2 class 'Node'. The intermediate nodes recognize publishers produced by the sensor node and creates a specification for each publisher that details how to serialize each sub-component of messages from the publisher and assigns names and tags used to categorize the resulting message in the database.

Both the UAV and ground station share a parsing function that can generate a strictly-ordered specification of how to serialize a ROS2 message type. The sensor specifications are constructed by associating ROS2 message primitives (i.e. int8, float32, etc.) with Python *'struct'* format codes. The parsing function iterates through each attribute of the ROS2 message accumulating format codes as it discovers these primitives. This process creates the most compact serialization of a ROS2 message without compression.

2.1.3 High-Level Protocol

In order to accommodate automatic serialization of sensor information at scale a high-level protocol was designed to encode sensor data with minimal bandwidth. The protocol exploits the fact that each node has the same ROS2 environment. Upon startup of the system, the first messages across the network are produced by the UAVs which send specifications for each of their sensor types. These specifications have three parts: the first character is '0' which is a reserved sensor code for specifying sensor messages, then there is the sensor code corresponding to the specified sensor type, and finally the import path for the sensor's ROS message type. These messages are parsed by a function that can determine deserialization rules for a given sensor purely from the import path of its respective ROS message and therefore the only information that needs to be transmitted is the import path.

The sensor specification is sent to the ground station and once it is successfully parsed it sends an acknowledgement to the UAV. The underlying DigiMesh protocol allows for the functionality of being able to address a message to a specific device. All UAVs transmit specifically to the ground station device and acknowledgements from the ground station go back to the UAV that sent the message. The message is the same '0' code followed by the sensor code being acknowledged. The allows the UAV to confirm that a sensor's data is ready to be sent over network and can provide information for debugging. Once the sensor specification has be sent and acknowledged, sensor information of that type can start to be transmitted over the network. Each sensor message starts with the corresponding sensor code of that sensor type followed by the GPS time that the sensor data was recorded. The final part of the packet is the serialized sensor data itself. For the purposes of tagging the data descriptively, a dedicated sensor code is given to allow the UAV to transmit its machine name. The ground station can then associate an Xbee device address to a specific UAV in real time.

Table 2.1: Messages from UAV to ground station

Example	**Description**
0,4,environ_msgs.Pth,pth_msg	Sensor code 4 is a pressure, temperature, humidity (PTH) message
2,plane1	This UAV's hostname is 'plane1'
4,\x02\x88\x91\x0b\xfb\x1a \xb4\xd8A\x00\ x00\xa0A\x9a \x99\x99A\xcd\xcc\x9cAff \xaaB\x00\ x00\xa0A	Sensor data for sensor code 4

Table 2.2: Messages from ground station to UAV

Example	**Description**
0,4	Acknowledge sensor code 4
1,4	Unknown sensor code 4
3	Unknown plane hostname

In addition to this core set of messages, an additional two codes have been added in the event that there is a miscommunication or reset of the ground station. The first code occurs when the ground station does not recognize sensor information coming from a UAV. In this event the ground station will send an 'unknown sensor' message with the respective code to the UAV and the UAV will resend that sensor message type. An example of this message coming from the GSU when it cannot recognize sensor code 4 is shown here. There is also an 'unknown hostname' message that the GSU can send to a UAV to request the re-sending of that UAV's hostname. The message is simply to send the 'unknown hostname' code to that respective UAV. Tables 2.1.3 and 2.1.3 show examples of these messages as they would be sent from either the UAV to the ground station or vice versa.

2.1.4 Message Serialization

ROS2 contains a small set of primitive data types that all other ROS2 messages are built atop. Aside from *'string'*, which is dynamically allocated with pointers, primitives have a static memory size and therefore the memory size of each primitive can be known at the time of message definition. ROS2 also enables the ability to create static and dynamic arrays of primitives within messages, the prior retaining the same feature of static memory size as the primitives. These primitives and arrays of primitives form the basis of parsing ROS2 messages into a highly compact serialized form. At the start of the program, each ROS2 message type in use by sensors on the system is recursively parsed until all terminal nodes in the recursion tree are primitive types. These terminal nodes are compiled into a list ordered by their depth-first occurrence in the ROS2 message tree. The resulting list of primitive types is used by the system to serialize every message of the corresponding type as it is queued to be transmitted.

There are several constraints on the parsed message in order to ensure that the message can be sent through the network. Firstly, only statically-sized message attributes can be used in serialization. All dynamic arrays and strings are removed in parsing and, if no message data remains, the message is logged locally instead of transmitted. This ensures the parsed packet will always be smaller than the maximum allowed packet size. It also allows deserialization to work from payload byte positions rather than needing extra delimiters and termination bytes to parse serialized packets. Second, the parsed message cannot be greater than the maximum payload size which is determined by the radio and protocol. DigiMesh and Digi devices permit 255 byte payloads, with the floating-point timestamp consuming 8 bytes and the high-level protocol described in subsection 2.1.3 requiring only 2 additional bytes for transmitting sensor data. If the primitive list for a message type exceeds this maximum size, a truncated version of the list is transmitted with the full list being logged locally. Finally, if the ROS2 message type contains the attribute *DONT_TRANSMIT*, that message type is logged locally and not transmitted as a way to control the network from the message definitions. An example of a serialized ROS2 message using this method is shown in Fig. 2.2.

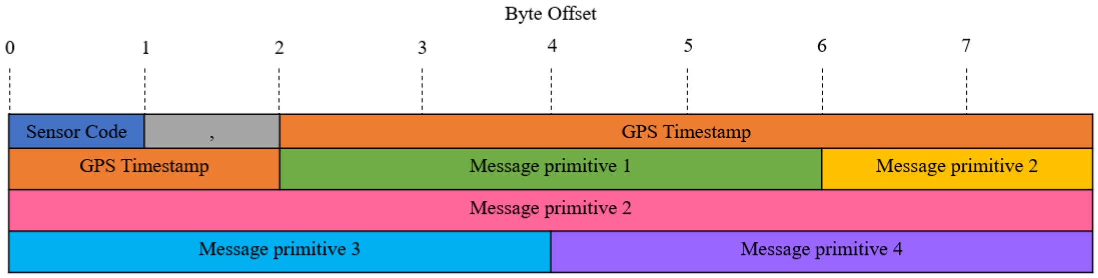


Figure 2.2: The packet layout for an example ROS2 message type following the serialization method described in this section. The packet header consists of a sensor code and GPS time which consume 10 bytes followed by the byte-indexed sensor data.

2.1.5 Network Setup

The network for the devices are of two parts. The first part is enabled by the ROS2 DDS which is transparent over UDP/IP communication and is enabled on the platforms though 802.11 WiFi. This transparency allows UAV-to-UAV communication as long as they are in range of one another. An application example of this feature can be seen in formation flying where high bandwidth, low latency telemetry communication within the formation is required [13].

The second part of the network is the longer range, higher latency DigiMesh network that enables ground-to-plane communication. DigiMesh is a proprietary mesh networking system designed by Digi Internation Inc. that is particularly effective in flying adhoc networks (FANETs) due to its ability to re-route the network during flight. Ground-to-plane communication allows the development of central sensor-driven planning algorithms like the system that will be described in Chapter 3 of this thesis.

In addition to these two networks, the UAVs have the ability to store data locally as necessary. This can be specified in the creation of a type of message that does not need to be transmitted or it is automatically decided based on prohibitively the large data sizes present in particular messages. For example, the packet serializer takes into account the size, in bytes, that a serialized message from a specification will be and if this size is larger than the maximum DigiMesh packet size it will opt to log that data instead of transmitting to preserve bandwidth. With this policy, sensor data such as video capture or high-frequency wind measurements would be automatically logged locally.

2.1.6 GPS Timing

In order to accurately relate sensor information between one another, GPS time is leveraged as a universal time stamping mechanism for all sensors. A ROS2 publisher was created to distribute the GPS time throughout the system as it was made available. It does so by recording a timestamp from the GPS every time it receives an interrupt from the GPS's pulse-per-second (PPS) signal over the Raspberry Pi's general purpose inputs and outputs (GPIO). This node publishes a time reference message that relates a system time t to the acquired GPS time t_g. These time references are used to linearly interpolate system time to GPS time, so that as sensor data is recorded in system time it can be accurately be converted into GPS time [19]. Equation 2.1 shows the interpolation equation to determine GPS time for a new system time and two GPS points. t_s is the system time the sample was taken, t_1 and t_2 are the system time of the two GPS samples and t_{g1} and t_{g2} are the GPS times of the two GPS samples. Finally, t_{gs} is the interpolated GPS time that the sample was taken.

$$t_{gs} = t_{g1} * \frac{t_s - t_1}{t_2 - t_1} * t_{g2} - t_{g1} \tag{2.1}$$

2.2 Design Validation

The primary metrics to evaluate this design has to do with packet consistency especially under a significant system load. The experiments conducted were therefore designed to test the variability of the packet delay and dropout under a variety of flight scenarios.

Packet delay was measured from the sensor GPS time, to the GPS time that it's corresponding database entry was created which would accurately represent the packet's end-to-end delay. Maximum packet delay was set within the radio configuration to be 10 s and would indicate a packet dropout. Different flight scenarios were used to test their effect on these two metrics: packet delay and packet dropout rate.

2.2.1 Experiment Setup

Two experiments were conducted to test the relationship between distance and packet consistency. Fig. 2.3 shows the assembled hardware for these experiments. A UAV was flown to different distances from the ground station (located at (38.12N, 84.498E)) while taking lateral passes at incremental distances. The first experiment was conducted only with a ground station emulating a point-to-point network, and the second experiment had another node in the mesh located at (38.126N, 84.486E). This experiment was conducted twice over the course of development to test new features and improvements in the design. For completeness, both test results are included in Table 2.3 with the most recent experiment labelled as 'new'.

Additional experiments were conducted with the system as incremental improvements and feature additions were made. One such test involved 4 UAV devices active on the ground simultaneously. Each UAV transmitted GPS information and some of the UAVs had additional sensors which included PTH probes and a high data-rate analog-to-digital (ADC) converter to test the system's capability to log data over the network and locally simultaneously.

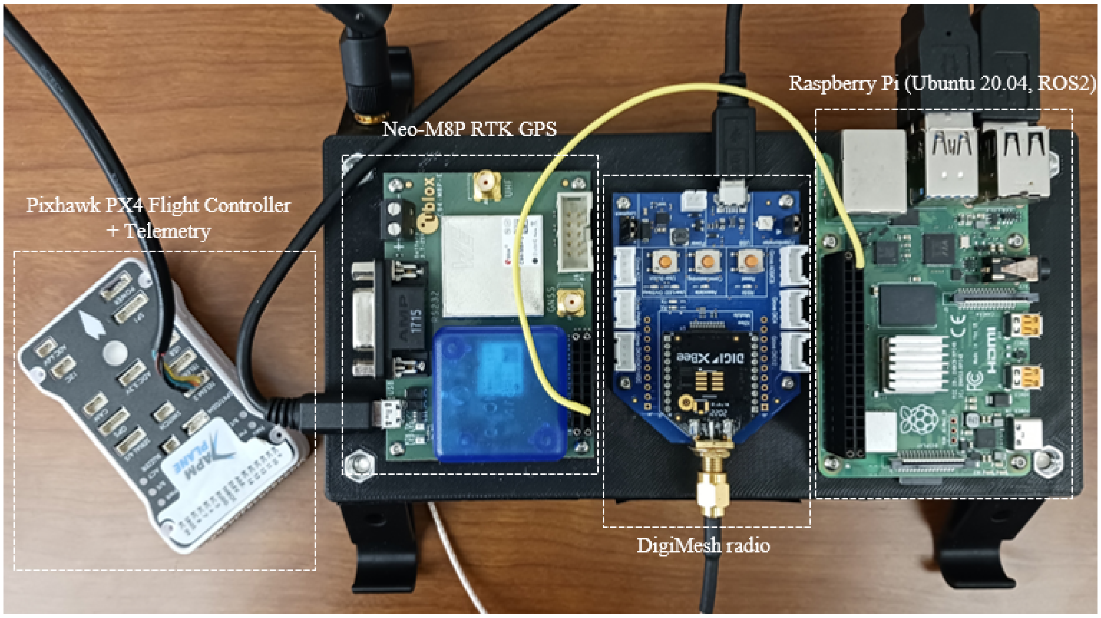

Raspberry Pi (Ubuntu 20.04, ROS2)
Neo-M8P RTK GPS
Pixhawk PX4 Flight Controller
+ Telemetry
DigiMesh radio

Figure 2.3: The computer hardware that is put on the UAVs during flight is shown. All sensor data is aggregated to the Raspberry Pi and then transmitted over the DigiMesh network. The Neo-M8P RTK GPS is used to associate all sensor readings to a GPS time and location with high accuracy.

2.2.2 Results

As can be seen in Fig. 2.4, the addition of one other node drastically increased the packet consistency of the system as it made each node more resistant to occlusions and propagation loss over long distances. This can be seen by the substantial drop in packet loss shown in in Table 2.3. In the most recent experiment median packet delay fell due to increases in efficiency of the protocol and serialization pipeline. The network did experience marginally more packet loss likely due to an increased load on the network. The use of more sensors and an extended time in the air meant that the total packets transmitted was much more for the most recent test.

2.2.3 Conclusion

The objective of this work was to create a distributed system capable of aggregating sensor data to a ground station in real time. This is a requirement of the larger project

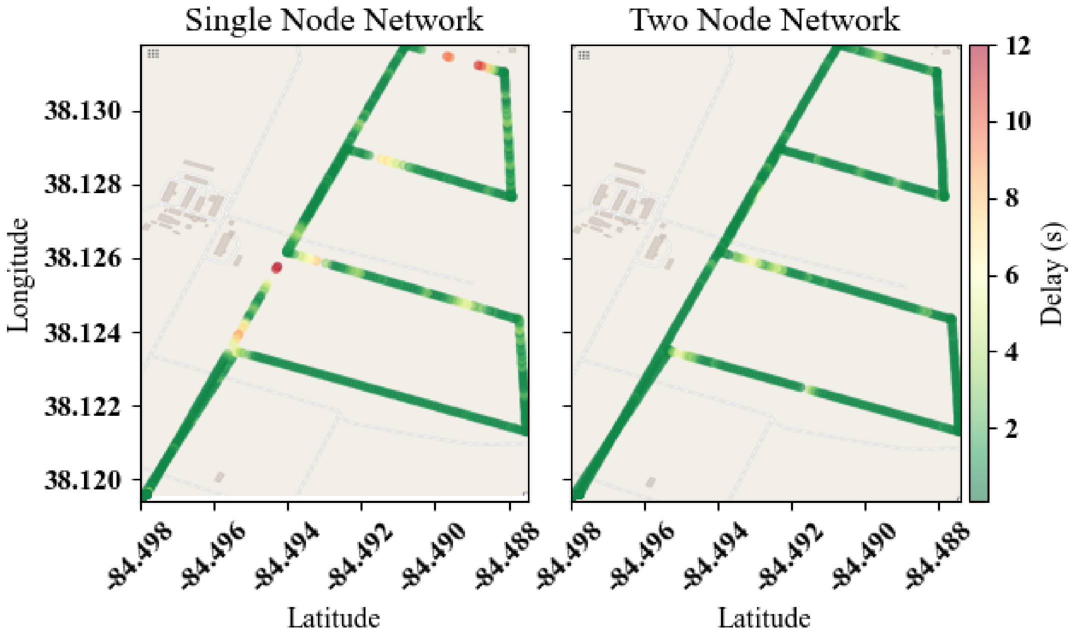


Figure 2.4: Maps of the packet delay from the first two experiments. The position of the points correspond to the recorded position of the UAV when that packet information was recorded and the color of the points indicate the delay.

Table 2.3: Packet delay metrics for one and two UAVs

	Single Node Network	Two Node Network	Two Node Network (new)
Packet Drop Rate (%)	7.3	0.8	2
Mean Delay (s)	1.62	1.16	1.12
1st Quartile (s)	0.90	0.894	0.36
Median Delay (s)	0.91	0.90	0.48
3rd Quartile (s)	1.576	0.916	1.41
Total Packets	647	680	1785

as the gas simulation must be computed centrally with access to sensor information from all UAVs. It was decided to design a system underlying ROS2 and DigiMesh to make it robust and easily-extensible. Multiple experiments were carried out both in the air and on the ground to validate the different system features such as hybrid networked and local data logging, full sensor modularity, mesh networking, etc. The experiments showed each of these features to be functional and that the system as a whole has the potential to fulfill the gas detection project requirements.

Future work on the *UAVMesh* platform should be primarily focused on the addition of new sensors, radios, and applications as the core architecture is tested and complete. Additional stress tests of the concept with more drones, sensors, and bandwidth would also provide interesting insight to the efficacy of the system at scale. Work on the project is tracked through its Github page and corresponding documentation.

Copyright© Joshua A. Ashley, 2022.

Chapter 3 Creating a Sensor-driven Multi-UAV Trajectory Planning Algorithm

3.1 Problem Model

3.1.1 Overview

The tested data aggregation system described in Chapter 2 allows for the creation of a centralized sensor-driven trajectory planning system. The requirements of this system is that it needs to be able to make trajectory decisions for each UAV. It needs to do so based on current partial knowledge of the environment with the objective of maximizing future information gain about the local atmospheric environment. The following section will describe how this system can be modelled in such a way that it can be optimized through DRL.

The input to the system is a map M relating wind vectors (u, v) to spatial points (x, y) generated by the WRF model from recorded sensor data. WRF generates this information using complex atmospheric simulation. The belief space, B, quantifies the prediction of the WRF model based on known measurement accuracies and their role in the model's prediction. The aircraft model establishes the observer's interaction with its environment and creates the action space. The POMDP model creates the relationship between the environment states and UAV actions, $\mathcal{M}(s, a)$, as well as the associated rewards. The rewards in exploration relate to the change in belief, $c_{(x,y)}$, of our environment, known as information gain I_Z. Fig. 3.1 shows how each model is connected for the entire system and how the positive feedback system is used to inform future decisions. The details of each model will be described in the following subsections.

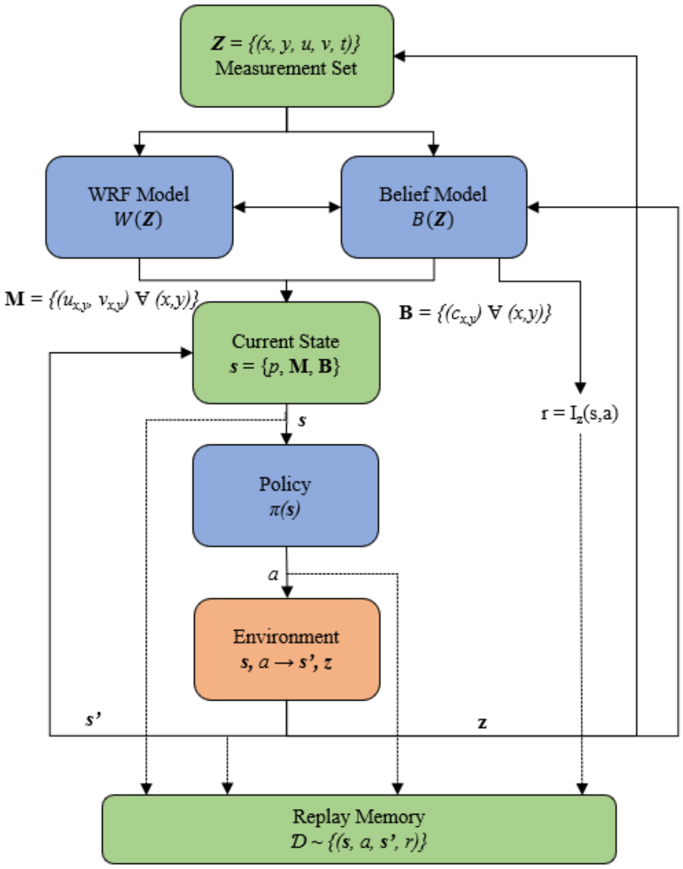


Figure 3.1: An overview of the data relationship of the each component model is shown. This overview shows the positive feedback relationship between new actions taken by the policy and information gain that increases model accuracy.

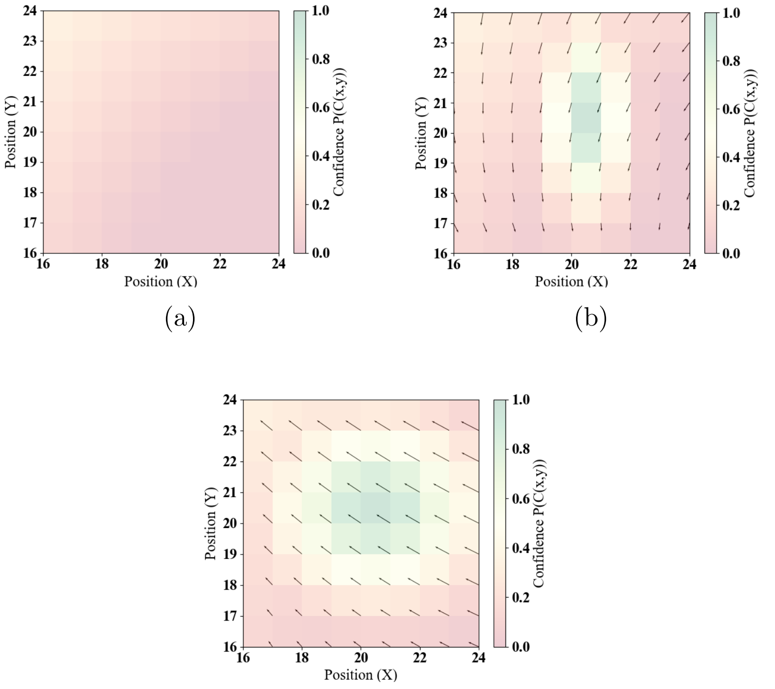
3.1.2 WRF Model

When modelling contaminant transport in a given space, measuring and simulating wind behavior jointly increases model accuracy [9]. Given a grid of spatial points $M = \{(x, y)\}$ such that $x, y \in \mathbb{Z}$ and $0 \leq x \leq x_{max}$, $0 \leq y \leq y_{max}$ where x_{max} and y_{max} are determined by the bounds and resolution of the map. A wind vector, $(u, v) \in \mathbb{R}^2$, must be assigned to each point to qualify the movement of particles though this space. This relationship between a (u, v) vector and (x, y) point produces a vector field otherwise known by the WRF model as a flow field. The UAVs are able to take partially-accurate measurements $Z = \{(x, y, u, v, t)\}$ of wind vectors at a given point in this space and time t. The WRF model therefore interpolates and extrapolates the measured information from the plane to create a complete, but approximate, map of the local weather environment $\mathcal{W}(m, Z) \to (u, v) \; \forall \; m \in M$. This approximation is qualified by the belief model in the subsequent subsection and is the basis for the trajectory planning throughout the thesis.

3.1.3 Belief Model

The belief model is based on the way the WRF model makes predictions. The observer gives measurements z with a low-but-present inaccuracy at specific points within M. The WRF model must then extrapolate forward through M using z and therefore compounding this inaccuracy with distance from z. This inaccuracy results in the prediction confidence decaying exponentially from the point of the measurement.

Taking a measurement has a radial effect of the surrounding value as there is a spatial relationship between measurements. Therefore the area around the point of measurement in the belief map will have its confidence increased upon the measurement being taken. To establish a relationship between the measurement and resulting impact on the belief map, the radial effect is determined by a Gaussian distribution whose mean is the point of observation and variance is proportional to the measured wind vector $\sigma_x \propto u$, $\sigma_y \propto v$. This results in larger wind vectors having larger impact on the map in the respective direction of the the wind vector itself. This is to provide the simulation with a feedback mechanism where certain measurements are seen as having higher map impact than others which will also be true in the more complex case. Fig. 3.2 visualizes how the measurement of u and v at a given point can affect the belief in different ways. The belief confidence is shown as a color grid where red is lower confidence and green is higher confidence. The wind vectors are shown by the arrows. Both measurements start from the same belief map shown in Fig. 3.2(a). The impact each measurement has on the belief map are different in shape due to the wind vectors pointing in different directions between each measurement. Fig. 3.2(b) mainly impacts in the y direction since the wind is mostly pointing in the y direction and Fig. 3.2(c) impacts both the x and y confidences since the wind vector is pointing roughly equally in both.



(a) (b)

(c)

Figure 3.2: A visualization of how measurements within the flowfield affect the belief space in different ways is shown. The values within the belief map are color coded with higher confidences being assigned green and lower confidences being assigned red. The arrows represent the direction and magnitude of the wind vector for each cell. An image of the belief map before, Fig. 3.2(a) and after measurements from within two different flow fields is made at point $x = 20$, $y = 20$ is shown, Fig. 3.2(b) and Fig. 3.2(c).

The key to measuring information gain from the belief space is to quantify the entropy of a given belief space and compare the change in entropy from taking an action in a given state. A confidence, $c_{(x,y)}$, is assigned to each point in M creating the set $B = \{c_{(x,y)} \ \forall \ (x, y)\}$ which describes the confidence that the simulation has correctly predicted the measurement at each point. Therefore, event $C_{(x,y)}$ is the event that the simulation has predicted correctly at the location of (x, y), thus

$P(C_{(x,y)} = correct) = c_{(x,y)}$ and $P(C_{(x,y)} = incorrect) = 1 - c_{(x,y)}$. The entropy of the probability distribution of a discrete event is the summation of entropy at each possibility

$$H_P(C_{(x,y)}) = -[c_{(x,y)} * log(c_{(x,y)}) + (1 - c_{(x,y)}) * log(1 - c_{(x,y)})] \,. \tag{3.1}$$

The observer is capable of taking a measurement $z = (x, y, u, v)$ that includes its position and wind vector at the current state of the observer. The goal of information gain is to find out what the change in entropy would be if a measurement z were taken, which is the event Z. The change in entropy can be obtained using the principle of mutual information

$$\begin{aligned} &E_Z[\Delta H(C_{(x,y)}|Z)] = H_p(C_{(x,y)}) + \int P(Z|s,a)* \\ &[P(C_{(x,y)}|Z) * log(P(C_{(x,y)}|Z))+ \\ &(1 - P(C_{(x,y)}|Z)) * log(1 - P(C_{(x,y)}|Z))]dz, \end{aligned} \tag{3.2}$$

where $P(C_{(x,y)}|Z)$ is the effect the measurement has on the confidence at the point (x, y) which was establish earlier as the Gaussian distribution characterised by the measurement $P(C_{(x,y)}|Z) = \mathcal{N}((z_x, z_y), (z_u, z_v))$ [29]. In practice, if a measurement gives a lower $c_{(x,y)}$ than was previously there it does not replace the prior value in the belief map. Since each measurement is used by the simulation to extrapolate all other states in the space, the total information gain must account for all (x, y) in M at each measurement

$$I_Z(s, a) = \sum_{x,y \in M} E_Z[\Delta H(C_{(x,y)}|Z)]. \tag{3.3}$$

Note that observation Z is a measurement done when the observer is at the next state s', and therefore state s, and action a, are variables in the information gain function.

3.1.4 Aircraft Model

The fixed-wing aircraft model is taken from a simple, two-dimensional case [6]. The aircraft has the ability to control its roll angle ϕ which turns the plane. The UAV also has a constant velocity v_p since this can be handled by inner-loop control [34]. With these constraints, the plane's state can therefore be described with three variables: position in 2D space x, y and heading or yaw ψ. The state-action relationship comes from the UAV's dynamics model

$$\begin{aligned} \dot{x} &= v_p \cos(\psi) \\ \dot{y} &= v_p \sin(\psi) \\ \dot{\psi} &= \frac{g}{v_p} \tan(\phi), \end{aligned} \tag{3.4}$$

where g is the acceleration due to gravity.

3.1.5 POMDP Model

A POMDP associates state-action pairs with the set of possible next states and reward that results in taking the action in that state $\mathcal{M}(s, a) \rightarrow (s', r)$. Traditionally, in an POMDP, there is a probability of transition between states $P(s'|s, a)$ that is non-zero for multiple different next states, and is therefore probabilistic. This allows an POMDP to take into account external environmental effects that could affect state (i.e. wind gusts moving the UAV). However, the solution model does not change for a probabilistic vs. deterministic versions of the POMDP and therefore the simple simulation uses deterministic behavior.

The state s for this problem has two parts. The first part is the observer state which consists of three variables, two describing spatial position (x_p, y_p), and one describing orientation ψ_p of the plane forming the vector $s_p = (x_p, y_p, \psi_p)$. For the second part of the state, each point in map M is given wind vector (u, v) and measurement confidence c. Therefore a vector of three values describing the u, v, and c is given for each spatial point in the map forming the map state $s_m = \{(u_m, v_m, c_m) \mid m \ \forall \ M\}$. Thus the state is completely defined by $s = (s_m, s_p)$.

The action space is the control allowed for a single UAV. Referring to the previous subsection, the aircraft model in 2D allows for a single control of roll angle ϕ. The model allows for continuous control and is bound by a maximum roll of the aircraft which is 40 degrees.

The primary component of the reward function is the information gain function I_Z described in the belief model. The second component of the reward function pertains to the simulation's finishing states. If the plane goes out of bounds, the run is terminated and the reward that is given is negative. If the plane stays in bounds for the duration of the flight it is rewarded

$$r(s,a) = \begin{cases} 20 & \text{time expired} \\ -20 & \text{out of bounds} \cdot \\ I_Z(s,a) & \text{otherwise} \end{cases} \tag{3.5}$$

3.2 Methodology

3.2.1 Using Soft Actor Critic to Approximate the Optimal Solution of a POMDP

The Bellman equation describes the solution to a POMDP given an infinite-discounted time horizon requiring infinite computation

$$Q(s,a) = r(s,a) + \lambda \int P(s'|s,a) * \underset{a'}{max}(Q'(s',a'))ds'. \tag{3.6}$$

where λ is the discount factor for each subsequent future state. This equation poses a multitude of issues for an analytical solution including a recurrent future term in Q' and an integral over an continuous variable in s. Certain simplifications and constraints make it such that the Q-function is able to be reasonably approximated using reinforcement learning [28]. For instance, the determination of future states and actions can be derived from the policy and simulation iteratively instead of analyzing all of the state and action space.

The type of reinforcement learning used in this algorithm to approximate a tractable solution to the Bellman equation is soft actor-critic (SAC), which trains three networks [12]. One network learns the optimal policy $\pi_\theta(s)$ for a given state where θ are the policy network parameters. The other two networks estimate the Q-function $Q_{\phi_1}(s,a)$ and $Q_{\phi_2}(s,a)$ where ϕ_1 and ϕ_2 are the Q-network parameters. Training iterations for the networks consist of regression over data sets D of recorded information called experience replay. In experience replay, states-action pairs taken in the simulator are recorded as sets of state, next state, action, reward, and a boolean denoting whether this is a terminal state (s, s', a, r, d). Training of the DQNs works very similarly to standalone DQN applications. Mean-squared Bellman error (MSBE) is used as the basis of the loss function

$$\mathcal{L}(D) = \underset{(s,s',a,r,d)\sim D}{E}[(Q(s,a) - (r + \lambda \underset{a'}{max}(Q'(s',a'))))^2]. \tag{3.7}$$

In training the DQNs, SAC utilizes the strategies target Q-networks, Polyak averaging [4], and dual-clipped Q-networks [12], to improve network stability and avoid local minima in the the Q-value estimation. These strategies result in the final loss function for the DQNs as

$$\mathcal{L}(\phi_i, D) = \underset{(s,a,s',r,d)\sim D}{E}[(Q_{\phi_i} - y(r,s',d))^2] \tag{3.8}$$

where

$$y(r,s',d) = r + \lambda(1-d)\underset{i=1,2}{min} Q_{\phi_{i,targ}}(s', \pi(s')) \tag{3.9}$$

The policy network attempts to select the optimal action for any given state. In SAC, this network is a stochastic policy $\pi_\theta(s) = \mathcal{N}(\mu_a, \sigma_a)$ that is entropy regularized using its loss function. Entropy regularization incentivizes exploration by rewarding a higher entropy stochastic policy. This entropy term means that the policy is optimized according to the loss

$$\mathcal{L}(\theta, D) = -\underset{\substack{s\sim D\\ a\sim\pi}}{E}[Q_\phi(s,a) - \alpha log(\pi_\theta(a|s))]. \tag{3.10}$$

Fig. 3.3 shows the data flow of the replay memory components through the SAC algorithm during training. Parts of the replay memory are used to generate loss values to update the Q-network and policy network. The target Q-network is updated intermittently from the main Q-network parameters.

3.2.2 Neural Network Architecture and Training

SAC was used to train the following network model. The network has the same hidden layers for both policy and Q-networks while only the inputs and outputs change. It is non-sequential and processes the UAV state using a deep, densely-connected network while it processes the belief image using convolutional layers. It then combines both 'sides' of the network using dense layers to produce a single set of outputs. Fig. 3.4 describes the structure of the networks used in this study.

Training was done using on a NVIDIA Volta GPU using pytorch for 300,000-1,200,000 training iterations. The optimizer chosen is the first-order, gradient-based optimizer Adam [18]. Table 3.1 shows the hyperparameters used for training the networks. If any hyperparameter is excluded from this list, the default supplied by pytorch and Adam was used. Training on this configuration takes approximately 48 hours to complete 1,000,000 training iterations.

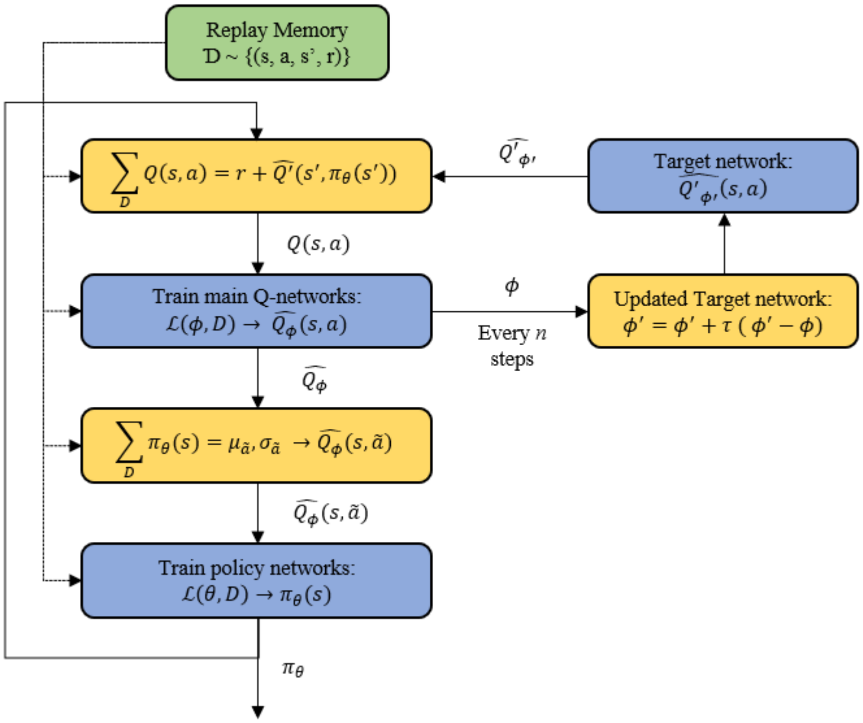

Replay Memory
D ~ {(s, a, s', r)}
$\sum_D Q(s,a) = r + \widehat{Q'}(s', \pi_\theta(s'))$
$\widehat{Q'}_{\phi'}$
Target network:
$\widehat{Q'}_{\phi'}(s,a)$
$Q(s,a)$
Train main Q-networks:
$\mathcal{L}(\phi, D) \rightarrow \widehat{Q_\phi}(s,a)$
ϕ
Every n steps
Updated Target network:
$\phi' = \phi' + \tau(\phi' - \phi)$
$\widehat{Q_\phi}$
$\sum_D \pi_\theta(s) = \mu_{\tilde{a}}, \sigma_{\tilde{a}} \rightarrow \widehat{Q_\phi}(s,\tilde{a})$
$\widehat{Q_\phi}(s,\tilde{a})$
Train policy networks:
$\mathcal{L}(\theta, D) \rightarrow \pi_\theta(s)$
π_θ

Figure 3.3: The data-flow diagram for a training iteration in SAC. Replay memory is the input data which consists of a representative data set of the POMDP. Yellow boxes indicate the parameter update functions with the blue boxes showing the network that receives those updates.

Table 3.1: Training Hyperparameters

λ	Bellman discount factor	0.95
τ	Polyak average rate	0.007
lr	Learning rate	0.00005
α	Entropy weight	0.1
Batch size	# of elements to train on	256
U	Updates per step	0.2
n	Steps to update target Q-network	20

3.2.3 Reducing State Dimensionality by Accounting for Environment Dynamics

Although SAC and the described network is able to train effectively with the state space describing the entire map at each point, a large portion of that information may

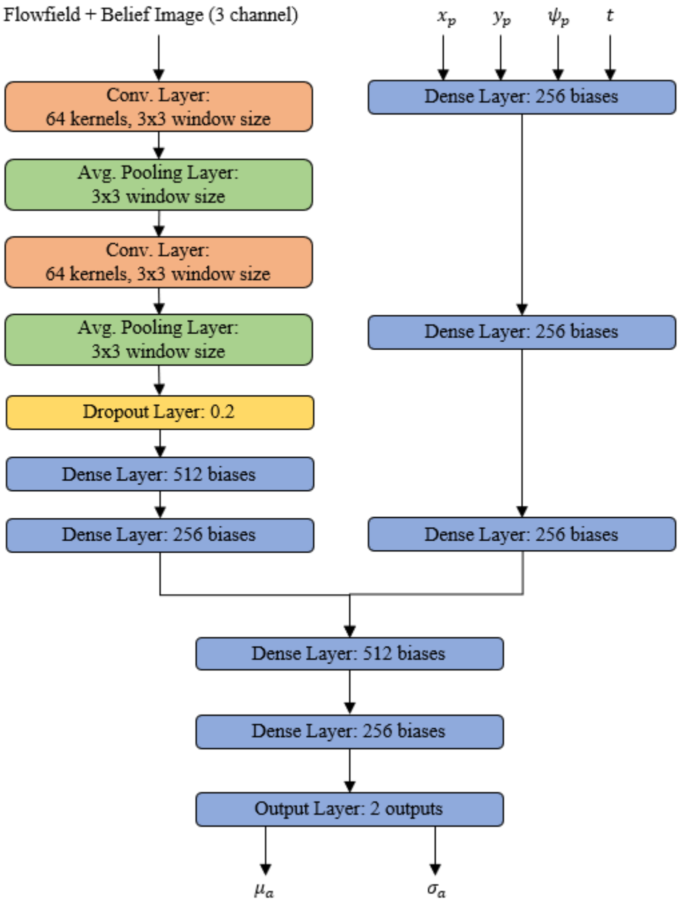


Figure 3.4: The structure of the neural networks that are trained by SAC is shown. Both the DQNs and policy networks use this structure with only a different output layer. The output layer shown describes the stochastic policy output which is a distribution $\mathcal{N}(\mu_a, \sigma_a)$.

not be useful to the UAV even when planning long horizons. This is because of the relationship between the Q-value and the spatial distance of map points to the UAV in its current position. The UAV is only able to move at a finite speed within the map and therefore the points in the map that are closer to the UAV will occur in fewer state transitions of the POMDP yielding a higher potential value due to the discount factor. For example, if you have a point in the map whose reward is estimated to be 20 and it would take 10 state transitions in order to get to that point, the maximum possible impact of that reward on the current true Q-value would be $20 * \lambda^{10}$ or approximately 11.9 for a discount factor of 0.95. While if the value of 20 only took 5 state transitions in order to get to that point, the maximum impact would be $20 * \lambda^5$ or approximately 15.4 for the same discount factor. Taken to the extreme, as the distance from the current UAV position increases, the maximum possible impact on the Q-value of a given point in the map asymptotically approaches zero.

In setting a threshold, ϵ, for how small of a potential impact the algorithm considers, a window size that is significantly smaller than the total map size can be used as input to the network. Equation 3.11 uses the speed of the plane in meters per second, v_p, the conversion of time to steps in the POMDP, dt, the current discount factor, λ, and this threshold ϵ to determine what the relevant map radius is

$$radius = \log_{\lambda}(\epsilon) * v * dt. \tag{3.11}$$

Using the tested simulation parameters of $v = 2$, $dt = 0.2$, $\lambda = 0.95$, and $\epsilon = 0.05$ yields a window radius of 23. This significantly reduces the window size from 100 x 50 cells to 46 x 46 cells while only compromising a total possible 5% impact on the accuracy of the approximated Q-value.

As shown in Fig. 3.5, during training and verification using the windowed method a slice of the current total map state will be given to the network based on the UAV's position and calculated window size. This also makes the network input independent to map size and shape since the windows are purely dependent on simulation parameters. Therefore the network policy would only need to be retrained for new UAV dynamics or model parameters, which is the case regardless of the windowed method, but would not need to be retrained for new areas or flight boundaries.

3.2.4 Extension to Multiple UAVs

Using the above method of SAC to train an agent for trajectory planning of one UAV, extension to multiple UAVs produces the challenge of an explosion in action space

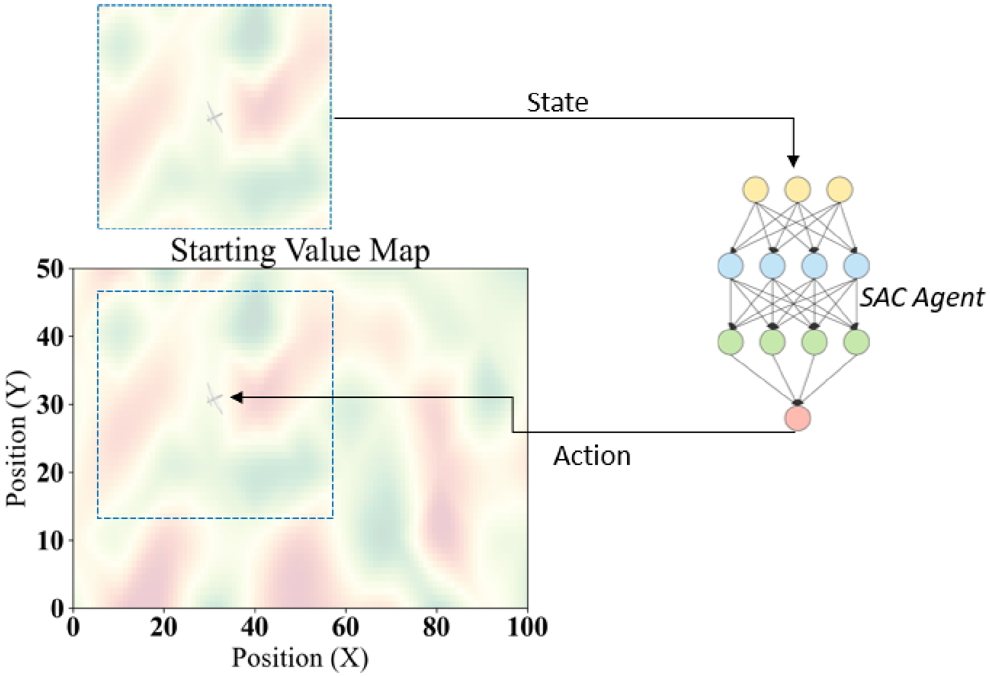


Figure 3.5: An example of the total state being sliced into a window based on the current location of the UAV. This slice is what is given to the trained agent which produces an action from that windowed input.

as one increases the available control decisions. Given that each UAV can make an independent action from one another at any given time step, the combination of potential actions scales polynomially with an increase in UAVs. A turn-taking strategy is proposed to allow the UAVs to act separately from one another which increases scalability and improves transfer learning given that in any case the model must only look at one UAV at a time. As the UAVs move through the environment, they impact the belief map. This impact can be extrapolated forward in time with some confidence using the current expectation values as detailed in (3.3). In turn-taking, the policy network is used to plan a UAV's trajectory forward in time an amount of steps, called the horizon, and the impact to the belief map is recorded. This impact will be shared and used as the global belief map for all other UAVs. As new measurements are actually obtained, the original belief map will be updated to represent reality and the planning will repeat with this new map. Therefore, each

UAV sees the plan of the other UAVs for those time steps through the belief map before making a plan of its own. No UAV will be allowed to plan farther in the future than the current segment of time such that one UAV does not get too far ahead of any other. The turn order of the UAVs is decided by the current maximal Q-value from the Q-network in the set of UAVs still to go in the current turn. Given that the Q-value is accurate, this means that the UAVs with the most potential gain in the following steps are allowed to take precedent over UAVs with less.

3.3 Results

3.3.1 Model Verification Process

During and after training the models undergo periodic verification. This verification is done by running 100 validation episodes of the simulation and recording the reward and crash rate for each. A crash is defined as the UAV going outside of the map boundary and can be mitigated by an inner-loop controller during real flight. The episodes are generated with random starting states including placement of the UAVs and map values. The map components, u, v and c, that form M and B are generated by randomly sampling a sparse grid of points and then doing cubic interpolation between each point to achieve the desired map resolution. This makes for a realistic atmospheric environment as the result is a continuous and differentiable map of points. Additionally, this yields a much larger sample space than any available dataset as well as allowing explicit adjustments to emulate the scale and resolution the real-world application will work within.

To validate the proposed algorithm, a greedy policy and fixed-trajectory policy were chosen for comparison. Additionally, policies generated with and without windowed input will be tested on simulations with and without turn-taking to cross-examine the impact of both of these additions. The greedy policy only takes into account the immediate, highest-reward action. The fixed-trajectory policy generates a 'lawnmower' pattern from the start point of the UAV. All policies were tested on one, two, and three UAVs in the simulation to show the increase in reward with scaling.

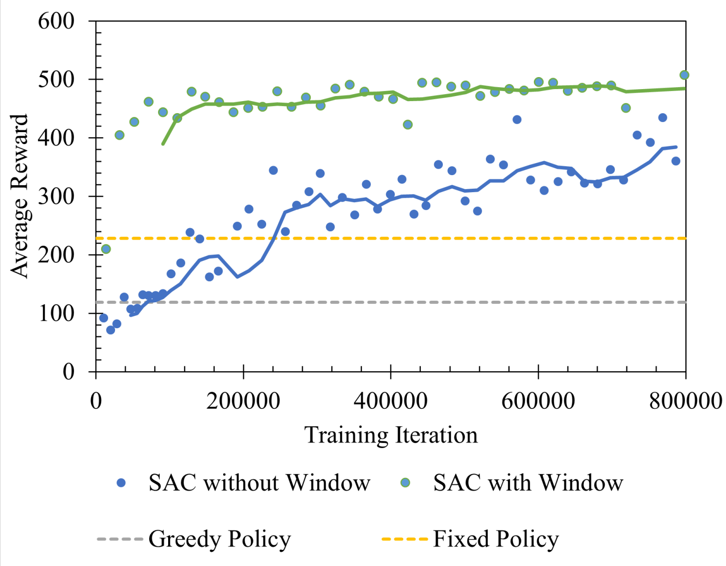


Figure 3.6: Training progress is made over 1,200,000 iterations. Reward is taken over an average of 50 verification runs. The yellow and grey dotted line shows the baseline average reward of the lawnmower and greedy policies respectively.

Fig. 3.6 shows the verification rewards as the agents receive more training updates. The green and blue points in the figure represent the average verification reward for a network trained using a windowed state space and a network trained using a complete state space, respectively, while their corresponding lines represent a moving average to visualize trends during training. The dashed yellow and grey lines show the average reward of the fixed and greedy policies respectively to indicate when in each trained policy surpasses these benchmarks. As seen in Fig. 3.6 the majority of reward gains from training the network without a windowed state space happen within the first 200,000 training iterations. After this point, the algorithm continues to increase the agent's average reward but the primary gains are from metrics other than reward. The network with a windowed state space reaches near a maximum reward within 100,000 training iterations.

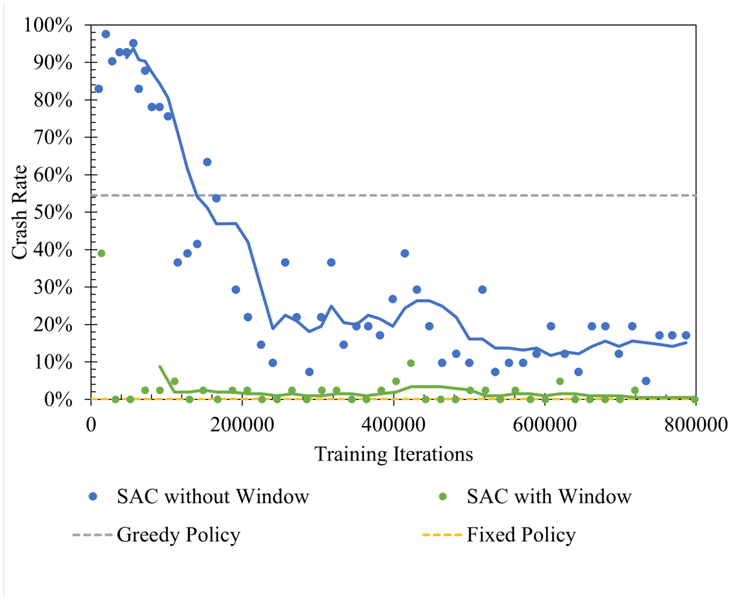


Figure 3.7: Training progress is made over 800,000 iterations. A 'crash' is qualified by the UAV going out of the boundary of the map. The yellow and grey dotted line shows the baseline average reward of the lawnmower and greedy policies respectively.

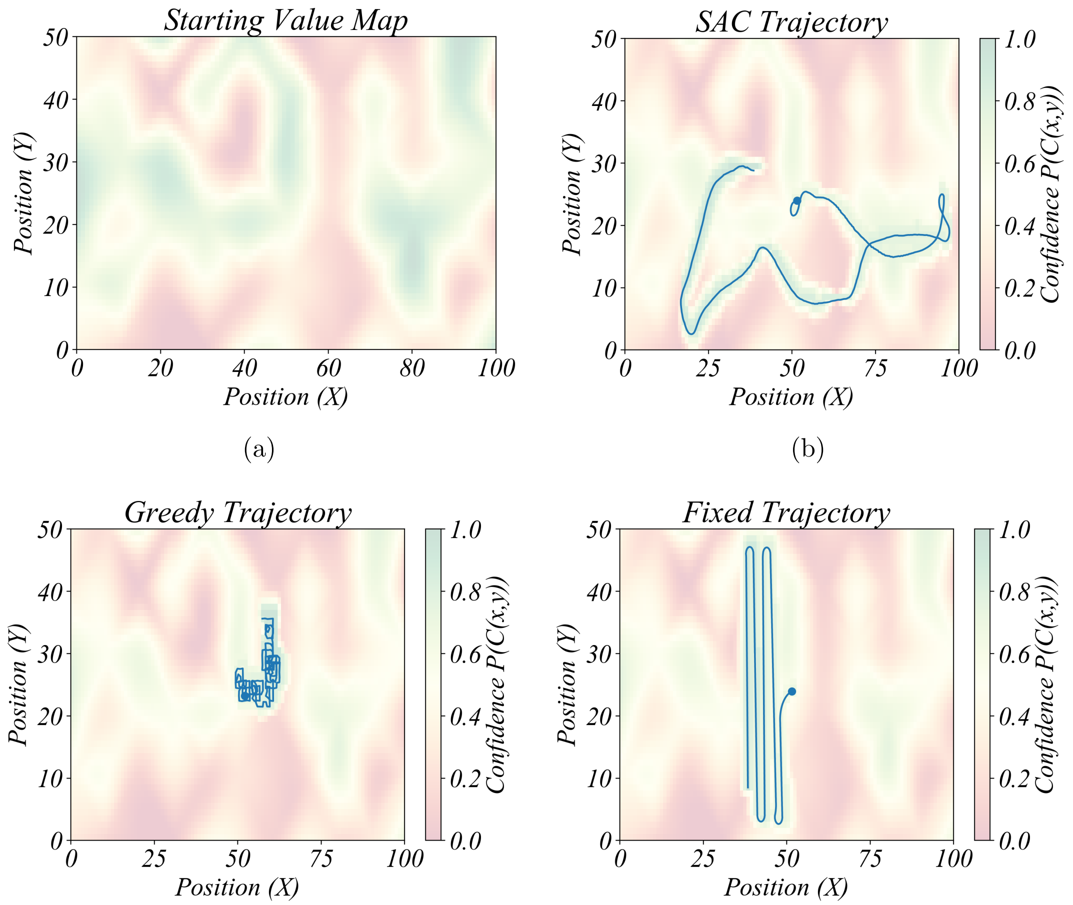
Fig. 3.7 shows the crash rate over the verification runs as the agents receive more training updates. The green and blue points in the figure represent the average verification reward for a network trained using a windowed state space and a network trained using a complete state space, respectively. The dashed yellow and grey lines show the crash rate of the fixed and greedy policies respectively to indicate when in each trained policy surpasses these benchmarks. The crash rate of the fixed trajectories is 0 in all cases due to the trajectory being hard-coded not to crash. The network trained with complete state space progressively decreases its crash rate throughout the entirety of training. The crash rate of the network with a windowed state space drops to 0 within 100,000 training iterations and mostly stays close to 0 throughout training.

3.3.2 Simulation Comparison for One UAV

Fig. 3.8 shows an example of UAV trajectories taken by each policy. Fig. 3.8(a) shows the starting confidence map each policy was provided. Fig. 3.8(b), 3.8(c), and 3.8(d) show the trajectories taken by the trained agent, greedy policy, and fixed-trajectory policy, respectively and their resulting impact on the confidence map. The confidence map is shown with low-confidence areas colored red and high-confidence areas colored green. The wind vectors are omitted from the visualization to reduce clutter. Intuitively, as the UAVs visit a location, the confidence at and near that location increases. As the UAVs spend more time away from a location, the confidence decreases as any measurement from that area becomes more irrelevant for a dynamic environment.

As shown in Fig. 3.8(b), the trained policy is able to take into account long-term rewards available on the map which allows the trajectories to more often converge on global maxima of value. The greedy policy, as shown in Fig. 3.8(c), is unable to take this into account and will often get stuck in local maxima which it will quickly exploit and be left with a high long-term penalty. The fixed trajectory policy, as shown in 3.8(d), is simply a maximization of area coverage and therefore does not utilize the confidence map. The fixed trajectory gets the benefit of a nearly non-existent crash rate but the disadvantage of low average value exploitation per unit time.

For instance, when the agent is nearby a small, local reward but could take a penalty in order to find larger, lower-confidence areas it will take the penalty while the greedy case would stay trapped in the small, local reward. This behavior can be exemplified in Fig. 3.8(b) as the agent reaches the low-confidence region between $x \in [20, 35]$, $y \in [25, 35]$ while the greedy case does not. The fixed trajectory encounters good and bad value by circumstance of the random environment, and therefore can see high penalty when the regions generated around the UAV's starting point have high confidence. For instance the fixed trajectory in Fig. 3.8(d) has the UAV spending a large amount of time in the high-confidence region $x \in [45, 55]$, $y \in [20, 40]$.



(a) (b) (c) (d)

Figure 3.8: The generated policy by the proposed algorithm produces a trajectory that is compared to the other policies in this figure. Fig. 3.8(a) shows the starting value map. Fig. 3.8(b) shows the trajectory made by the agent produced by SAC. Fig. 3.8(c) shows the greedy policy trajectory. Fig. 3.8(d) shows the fixed trajectory.

3.3.3 Simulation Comparison for Multiple UAVs

The main metric when looking at the scalability of each respective policy is to look at how the average reward increases with the addition of more UAVs. In an ideal circumstance, the addition of a another UAV would proportionally increase the total reward earned in the simulation up until the number of UAVs are able to completely exploit the information gain in the map. Fig. 3.9 shows a segmented bar chart of the mean and standard deviation for the average reward of each policy at 1, 2, and 3 UAVs over 100 runs each. As shown in Fig. 3.9, each method gets close to this scaling for the increase in UAVs into the environment. As more reward is accrued by the UAVs, less information and therefore potential future reward becomes available. Fig 3.10 shows 3 UAVs taking trajectories in the same map over the same period of time. Each UAV trajectory is differentiated by the colors blue, green, and orange. Fig. 3.10(b) shows that the UAVs will take trajectories that avoid visiting the same areas as other UAVs in quick succession. When put in similar starting states, such as the orange and blue trajectories, the UAVs will take different trajectory decisions at key points that allow them to acquire their own reward. This effect is shown on Fig. 3.10(b) in the bottom right quadrant of the map when the UAVs have overlapping trajectories to start but diverge quickly in order to obtain a wider-spread of information. The orange trajectory goes near the bottom of the y axis, the blue trajectory goes above it, and the green trajectory attempts to avoid the quadrant altogether. While there is inevitably overlap between the trajectories due to the map size, the UAVs succeed in minimizing overlap at most points in the trajectory. This results in the learned policy acquiring over 30% more reward per UAV than than the fixed policy for the agents not using a windowed input.

Fig. 3.9 also shows the comparison in of the agents with combinations of with and without turn-taking and with and without a windowed input. The impact of turn-taking is relatively small, but highly repeatable. The turn-taking strategy is hypothesized to have an increased impact as the problem scales to more UAVs which is corroborated by the data shown in this figure. For the case of three UAVs, a 5% increase with the turn-taking strategy as opposed to without is observed for both the windowed and not windowed input agents. The windowed input however has a very large impact on the collected reward by the agent as it attributes to a 50% to 60% increase in reward per UAV for 1, 2, and 3 UAVs. This validates that reducing the state space complexity at the sacrifice of marginal reward potential can be very beneficial to the success of the policy for this application.

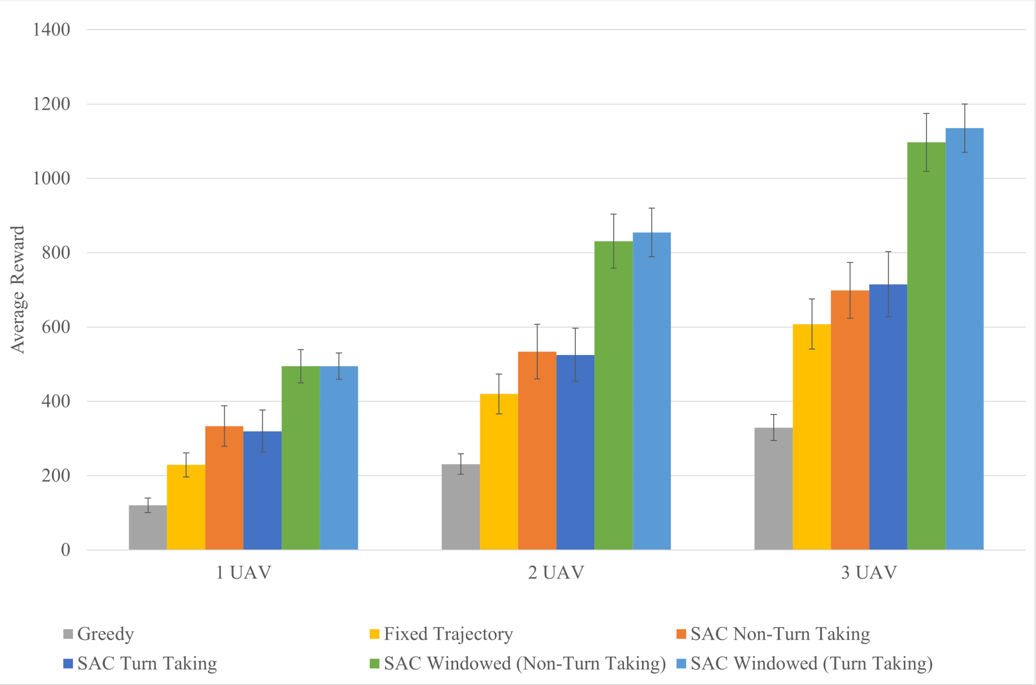
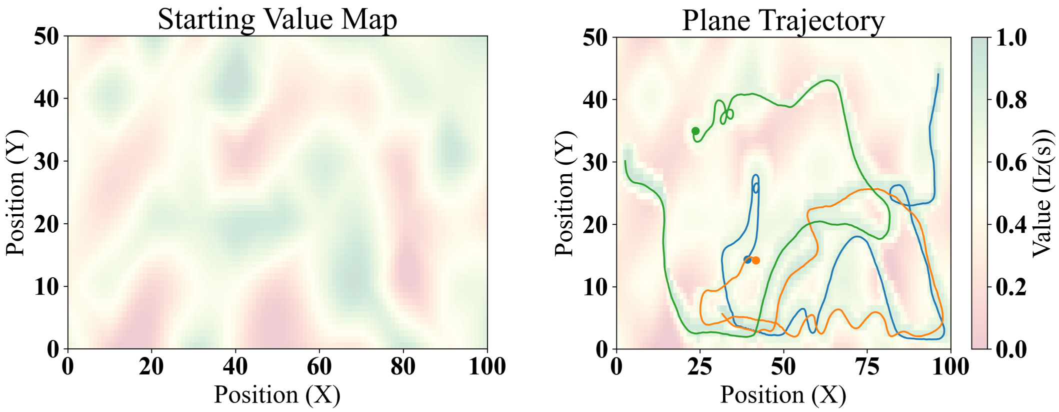


Figure 3.9: The different policies' reward are compared over 100 runs for 1, 2, and 3 UAVs flying simultaneously. This shows a clustered column chart of the mean and standard deviation for the reward across these 100 runs for each.

3.4 Conclusion

Using a probabilistic model that collaborates with the behavior of the WRF model gives much more information about the surrounding weather environment than any sparse set of measurements would give on its own. This drastically reduces the amount of work and time the UAVs need to create an accurate map. The turn-taking strategy of the UAVs means that the reinforcement learning and action space is agnostic to the amount of UAVs in the environment, increasing the scalability of the problem. Given a sufficiently accurate simulation of the future effect that each UAV will have on the map's accuracy, this should produce similar results to taking all actions in parallel while improving tractability. Additionally using a subset, or window, of



(a)

(b)

Figure 3.10: Example trajectories made by multiple UAVs using the turn-taking strategy on the trained policy is shown. Fig. 3.10(a) is the starting value map that is given to the policy as input, and fig. 3.10(b) is the resulting total trajectories planned by the policy and the resulting impact on the map.

the current complete map state based on spatial relevance to the UAV can lead to a drastic decrease in state space complexity and corresponding increases in overall reward. These results provide promise for expansion of the algorithm to more complex scenarios as the project progresses closer to real flight. Future work will involve addressing the challenges of integrating and testing this algorithm onto the described implementation from Chapter 2. This process will likely require an increasingly complex atmospheric simulation and subsequent probabilistic model based on the current work of the UKY DART project as well as a more complex UAV dynamics model of flight to increase accuracy of the policy between simulation and reality.

Copyright© Joshua A. Ashley, 2022.

Chapter 4 Conclusion

4.1 Summary

The objective of this work was to describe the contributions made towards designing real-time, sensor-driven flight system that can efficiently and accurately map surrounding atmospheric environment. The first contribution, detailed in Chapter 2, was to design a real-time sensor data aggregation system to enable UAVs to communicate important information for atmospheric simulation during flight. The system was built using a state-of-the-art mesh networking protocol, DigiMesh, and an interprocess data distribution system (DDS) ROS2. Automation and abstractions were designed to make communication with the ground station as efficient over the network as possible while still allow for ease of development when introducing new sensors. Over the course of multiple experiments, the system was able to meet the latency and distance requirements of the project while also introducing a variety of new features and sensors. The second contribution, detailed in 3, was the creation of a sensor-driven trajectory planning algorithm for optimizing information gain of the atmospheric simulation over the course of the trajectory. This planning algorithm was modelled as a robotic exploration problem with a POMDP that could be optimized using a DRL agent. In order to introduce multiple UAV's a turn-taking strategy during verification. This allowed the UAVs to plan ahead a certain distance in time such that each UAV could plan sequentially and not require retraining of the DRL agent with the addition of more UAVs. The state space of POMDP was drastically reduced in training the DRL agent by taking into account the spatial relevance of points to the UAV's position. The DRL agent in conjunction with these two enhancements yield a result that was 80%-115% more performant than a fixed trajectory focused on area coverage for all scenarios of 1, 2, and 3 UAVs. These results shows the efficacy of the algorithm for expansion to more complex simulations that bridge the gap between simulation and real flight.

4.2 Future Work

Future work on both systems pertain roughly to scaling the designs as the overall project proceeds closer to real flights. The data aggregation platform will likely require integration of new sensors, radios, and end applications but the core architecture of the design is tested and complete. Additional stress tests of the system using fixed-wing drones, more sensors, and higher bandwidth are planned and will provide interesting insight into the efficacy of the system at scale. Work on the project is tracked through its Github page and corresponding documentation. Future work on the sensor-driven flight planning algorithm must also address the challenges of effective implementation in the real scenario. This will likely require increasingly complex atmospheric simulation with using the WRF model directly and more complex UAV dynamics that better represent the UAVs that will be used in the project.

Copyright© Joshua A. Ashley, 2022.

Bibliography

[1] L. Bing, M. Qing-Hao, W. Jia-Ying, S. Biao, and W. Ying. Three-dimensional gas distribution mapping with a micro-drone. In *2015 34th Chinese Control Conference (CCC)*, pages 6011–6015. IEEE, 2015.

[2] J. R. Bourne, M. N. Goodell, X. He, J. A. Steiner, and K. K. Leang. Decentralized multi-agent information-theoretic control for target estimation and localization: Finding gas leaks. *The International Journal of Robotics Research*, 39(13):1525–1548, 2020.

[3] J. Burgués, V. Hernández, A. J. Lilienthal, and S. Marco. Smelling nano aerial vehicle for gas source localization and mapping. *Sensors*, 19(3):478, 2019.

[4] S. Dankwa and W. Zheng. Twin-delayed DDPG: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, pages 1–5, 2019.

[5] S. Devarakonda, P. Sevusu, H. Liu, R. Liu, L. Iftode, and B. Nath. Real-time air quality monitoring through mobile sensing in metropolitan areas. In *Proceedings of the 2nd ACM SIGKDD international workshop on urban computing*, pages 1–8, 2013.

[6] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[7] C. Ercolani, L. Tang, A. A. Humne, and A. Martinoli. Clustering and informative path planning for 3D gas distribution mapping: Algorithms and performance evaluation. *IEEE Robotics and Automation Letters*, 7(2):5310–5317, 2022.

[8] S. Favot. Health officials: Porter ranch gas leak may cause long-term health effects. *Los Angeles Daily News*, 2015.

[9] A. Gongora, J. Monroy, and J. Gonzalez-Jimenez. Joint estimation of gas and wind maps for fast-response applications. *Applied Mathematical Modelling*, 87:655–674, 2020.

[10] S. Gourdji, V. Yadav, A. Karion, K. Mueller, S. Conley, T. Ryerson, T. Nehrkorn, and E. Kort. Reducing errors in aircraft atmospheric inversion estimates of point-source emissions: the aliso canyon natural gas leak as a natural tracer experiment. *Environmental Research Letters*, 13(4):045003, 2018.

[11] L. Gupta, R. Jain, and G. Vaszkun. Survey of important issues in uav communication networks. *IEEE Communications Surveys & Tutorials*, 18(2):1123–1152, 2015.

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[13] C. Heintz and J. B. Hoagg. Formation control for fixed-wing uavs modeled with extended unicycle dynamics that include attitude kinematics on so(m) and speed constraints. *Proceedings of American Control Conference.*

[14] M. Hutchinson, C. Liu, P. Thomas, and W.-H. Chen. Unmanned aerial vehicle-based hazardous materials response: Information-theoretic hazardous source search and reconstruction. *IEEE Robotics Automation Magazine*, 27(3):108–119, 2020.

[15] M. Jerrett. Community monitoring around the porter ranch natural gas leak disaster. In *ISEE Conference Abstracts*, 2016.

[16] K. D. Julian and M. J. Kochenderfer. Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning. *Journal of Guidance, Control, and Dynamics*, 42(8):1768–1778, 2019.

[17] A. Khalifeh, H. Salah, S. Alouneh, A. Al-Assaf, and K. Darabkh. Performance evaluation of digimesh and zigbee wireless mesh networks. In *2018 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pages 1–6, 2018.

[18] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] K. Y. Koo, D. Hester, and S. Kim. Time synchronization for wireless sensors using low-cost gps module and arduino. *Frontiers in Built Environment*, 4:82, 2019.

[20] N. Melnikova, J. Wu, P. Ruiz, and M. F. Orr. National toxic substances incidents program—nine states, 2010–2014. *MMWR Surveillance Summaries*, 69(2):1, 2020.

[21] M. Pan, C. Chen, X. Yin, and Z. Huang. Uavs-aided emergency environmental monitoring in infrastructure-less areas: Lora mesh networking approach. *IEEE Internet of Things Journal*, PP:1–1, 07 2021.

[22] L. Paull, C. Thibault, A. Nagaty, M. Seto, and H. Li. Sensor-driven area coverage for an autonomous fixed-wing unmanned aerial vehicle. *IEEE Transactions on Cybernetics*, 44(9):1605–1618, 2014.

[23] J. P. Queralta, Y. Xianjia, L. Qingqing, and T. Westerlund. Towards large-scale scalable mav swarms with ros2 and uwb-based situated communication. *arXiv preprint arXiv:2104.09274*, 2021.

[24] M. Rossi and D. Brunelli. Gas sensing on unmanned vehicles: Challenges and opportunities. *2017 New Generation of CAS (NGCAS)*, pages 117–120, 2017.

[25] I. Senocak, N. W. Hengartner, M. B. Short, and W. B. Daniel. Stochastic event reconstruction of atmospheric contaminant dispersion using Bayesian inference. *Atmospheric Environment*, 42(33):7718–7727, 2008.

[26] M. Steven, F. Tully, B. Gerkey, L. Chris, and W. William. Robotoperatingsystem2:design,architecture,andusesinthewild. *ScienceRobotics*, 7(66):eabm6074, 2022.

[27] S. Subchan, B. A. White, A. Tsourdos, M. Shanmugavel, and R. Żbikowski. Dubins path planning of multiple UAVs for tracking contaminant cloud. *IFAC Proceedings Volumes*, 41(2):5718–5723, 2008.

[28] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

[29] S. Thrun, W. Burgard, and D. Fox. Probalistic robotics. *Kybernetes*, 2006.

[30] T. F. Villa, F. Salimi, K. Morton, L. Morawska, and F. Gonzalez. Development and validation of a UAV based system for air pollution measurements. *Sensors*, 16(12):2202, 2016.

[31] B. A. Visan. *DigiMesh Reliability in High Speed Vehicular Applications.* PhD thesis, Purdue University, 2017.

[32] B. M. Witte, R. F. Singler, and S. C. C. Bailey. Development of an unmanned aerial vehicle for the measurement of turbulence in the atmospheric boundary layer. *Atmosphere*, 8(10), 2017.

[33] G. Xu, Z. Yang, W. Lu, and L. Zhang. Decentralized multi-uav cooperative search based on ros1 and ros2. In *International Conference on Autonomous Unmanned Systems*, pages 2427–2435. Springer, 2021.

[34] G. Yang and V. Kapila. Optimal path planning for unmanned air vehicles with kinematic and tactical constraints. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 2, pages 1301–1306. IEEE, 2002.

Vita

Joshua Ashley

Place of Birth:

- Louisville, KY

Education:

- University of Kentucky, Lexington, KY
  M.S. in Electrical Engineering, Dec. 2022
  in progress

- University of Kentucky, Lexington, KY
  B.S. in Computer Engineering, Dec. 2020
  magna cum laude

Professional Positions:

- Graduate Research Assistant, University of Kentucky Spring 2020–Fall 2022

Honors

- Kentucky Excellence Scholarship, University of Kentucky

- Provost Scholarship, University of Kentucky

Publications & Preprints:

- J. Qin, J. Ashley and B. Xie, ”Hand Gesture Recognition Based on a Nonconvex Regularization,” 2021 IEEE International Conference on Mechatronics and Automation (ICMA), 2021