

**LIGHTWEIGHT CYBERATTACK INTRUSION DETECTION  
SYSTEM FOR UNMANNED AERIAL VEHICLES USING  
RECURRENT NEURAL NETWORKS**

by

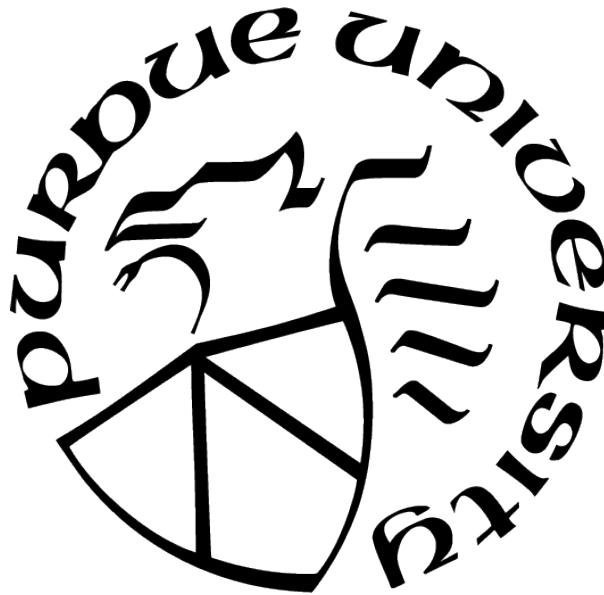
**WEI-CHENG HSU**

**A Thesis**

*Submitted to the Faculty of Purdue University*

*In Partial Fulfillment of the Requirements for the degree of*

**Master of Science in Aeronautics and Astronautics**



School of Aeronautics and Astronautics

West Lafayette, Indiana

August 2021

**THE PURDUE UNIVERSITY GRADUATE SCHOOL  
STATEMENT OF COMMITTEE APPROVAL**

**Dr. Inseok Hwang, Chair**

School of Aeronautics and Astronautics

**Dr. Dengfeng Sun**

School of Aeronautics and Astronautics

**Dr. James Goppert**

School of Aeronautics and Astronautics

**Approved by:**

Dr. Gregory A. Blaisdell

To my parents and brother, for all their love and support.

## ACKNOWLEDGMENTS

I would like to offer my appreciation and thanks to Professor Inseok Hwang, my advisor, and Professor James Goppert for their advice and guidance with both this thesis and my graduate school career. I also want to acknowledge and thank Akshita Gupta and Shreyansh Shethia, who provided invaluable assistance with the experiment and gave me advice throughout my research. Special thanks to my parents and brother for all the encouragement and sacrifices that they have made on my behalf.



# TABLE OF CONTENTS

LIST OF TABLES . . . . .	8
LIST OF FIGURES . . . . .	10
LIST OF SYMBOLS . . . . .	14
ABBREVIATIONS . . . . .	16
ABSTRACT . . . . .	18
1 INTRODUCTION . . . . .	19
1.1 Background and Motivation . . . . .	19
1.2 Related Works . . . . .	20
1.2.1 UAV Vulnerabilities . . . . .	20
Denial of Service (DoS) . . . . .	20
False Data Injection (FDI) . . . . .	22
1.2.2 Intrusion Detection System (IDS) . . . . .	23
Detection Methods . . . . .	24
Challenges . . . . .	26
1.3 Overview . . . . .	27
1.3.1 Contribution . . . . .	27
1.3.2 Thesis Layout . . . . .	27
2 VULNERABILITY AND IDS DESIGN . . . . .	28
2.1 System . . . . .	28
2.1.1 Unmanned Aerial Vehicles (UAVs) . . . . .	28
2.1.2 Autopilot . . . . .	30
PX4 Firmware . . . . .	31
Pixhawk . . . . .	31
2.1.3 Data Link . . . . .	32
WiFi Module . . . . .	32

	MAVLink Protocol . . . . .	32
2.1.4	Control Schemes . . . . .	37
	Control Mode . . . . .	37
	Extended Kalman Filter Estimation . . . . .	39
2.2	Vulnerability . . . . .	41
2.2.1	Assumptions . . . . .	41
2.2.2	DoS—ICMP Flooding . . . . .	42
2.2.3	FDI—False Waypoint Injection . . . . .	43
2.3	Intrusion Detection System (IDS) . . . . .	44
2.3.1	Recurrent Neural Network (RNN) . . . . .	45
	Simple RNN . . . . .	46
	Long-Short Term Memory (LSTM) . . . . .	48
	Gated Recurrent Units (GRUs) . . . . .	50
	Simple Recurrent Units (SRUs) . . . . .	51
2.3.2	Features . . . . .	53
2.3.3	Design . . . . .	60
	Memory . . . . .	60
	Computational Cost . . . . .	64
3	EXPERIMENT CONFIGURATION . . . . .	70
3.1	UAV . . . . .	70
3.2	Autopilot . . . . .	70
3.3	Wi-Fi Module . . . . .	74
3.4	Radio . . . . .	75
3.5	GCS . . . . .	76
3.6	Attacker . . . . .	77
3.7	External Vision System . . . . .	78
3.8	Experiment Sequence . . . . .	79
3.8.1	DoS—ICMP Flooding . . . . .	79
	Hover . . . . .	79

	Cruise . . . . .	79
3.8.2	FDI—False Waypoint Injection . . . . .	79
	Hover . . . . .	80
	Cruise . . . . .	81
4	ANALYSIS . . . . .	82
4.1	DoS—ICMP Flooding . . . . .	82
4.1.1	Normal—No Attack . . . . .	82
4.1.2	Hover . . . . .	87
4.1.3	Cruise . . . . .	93
4.2	FDI—False Waypoint Injection . . . . .	98
4.2.1	Normal—No attack . . . . .	98
4.2.2	Low Frequency . . . . .	103
	Random Waypoints Injection . . . . .	103
	Fixed Waypoint Injection . . . . .	109
4.2.3	High Frequency . . . . .	116
4.3	IDS . . . . .	124
5	CONCLUSION . . . . .	132
5.1	Future Directions . . . . .	133
	REFERENCES . . . . .	134

## LIST OF TABLES

2.1	UAV classification according to the US Department of Defense [46]. . . . .	28
2.2	MAVLink 1 structure content. . . . .	34
2.3	MAVLink 2 signature structure content. . . . .	35
2.4	MAVLink 2 structure content. . . . .	36
2.5	Control modes for multi-copter in PX4 . . . . .	37
2.6	Ratio of attacker data rate successfully forcing UAVs to disarm to legitimate user data rate. . . . .	44
2.7	Selected features. . . . .	54
2.7	Selected features . . . . .	55
2.7	Selected features . . . . .	56
2.7	Selected features . . . . .	57
2.7	Selected features . . . . .	58
2.7	Selected features . . . . .	59
2.8	Features selected using RFE and random forest. . . . .	59
2.9	Number of parameters in a single layer of simple RNN, LSTM, GRU, and SRU. . . . .	61
2.10	Number of parameters in a single layer of simple RNN, LSTM, GRU, and SRU in PyTorch. . . . .	62
2.11	Ratio of final model memory in PyTorch to memory calculated by number of parameters in RNN, LSTM, GRU, and SRU in PyTorch. . . . .	63
2.12	FLOPs for standard arithmetic operators according to [71]. . . . .	64
2.13	FLOPs for single simple RNN layer forward pass. . . . .	65
2.14	FLOPs for single LSTM layer forward pass. . . . .	65
2.15	FLOPs for single GRU layer forward pass. . . . .	66
2.16	FLOPs for single SRU layer forward pass. . . . .	66
2.17	Time for floating point operations on Arduino Uno. . . . .	68
2.18	FLOPs for SRU in various settings and required time for one complete forward pass. . . . .	69
2.19	Selected messages and associated time interval. . . . .	69
3.1	Specifications of the selected UAV platform. . . . .	70

3.2	Holybro Pixhawk 4 technical specifications. . . . .	72
3.3	Holybro Pixhawk 4 mechanical and environment specifications. . . . .	72
3.4	Holybro Pixhawk 4 electrical specifications. . . . .	72
3.5	Holybro Pixhawk 4 connection on Holybro S500 platform used in this study. . .	73
3.6	Specifications of onboard Wi-Fi module. . . . .	75
4.1	RNN IDS performance with 65 features data set and 64 hidden units. . . . .	125
4.2	RNN IDS performance with 32 features data set. . . . .	126
4.3	Threshold methods maximum accuracy. . . . .	128
4.4	Overall performance of RNNs IDS methods. . . . .	130

## LIST OF FIGURES

2.1	MAVLink 1 structure. . . . .	33
2.2	MAVLink 2 signature field structure. . . . .	35
2.3	MAVLink 2 structure. . . . .	36
2.4	Attacker offboard data rate that successfully forces UAVs to disarm vs. legitimate user frequency. . . . .	44
2.5	Simple RNN type 1. . . . .	46
2.6	Simple RNN type 2. . . . .	46
2.7	Simple RNN type 3. . . . .	47
2.8	Single LSTM cell. . . . .	49
2.9	Single GRU cell. . . . .	51
2.10	Single SRU cell. . . . .	53
2.11	Telemetry status rate rx and vehicle local position for hover mission under ICMP flooding attack. (Top): Telemetry status-rate rx; (bottom): local position x and y, (purple): x, (blue): y. . . . .	54
3.1	Holybro S500 UAV platform. . . . .	70
3.2	Holybro Pixhawk 4. . . . .	71
3.3	Holybro Pixhawk 4 connection scheme. . . . .	73
3.4	Holybro Pixhawk 4 connection on Holybro S500 platform used in this study. . .	74
3.5	ESP8266 Wi-Fi module . . . . .	75
3.6	FrSKY TARANIS. . . . .	76
3.7	FrSKY X8R. . . . .	76
3.8	Screen shot of QGroundControl GCS. . . . .	77
3.9	Qualisys motion capture camera setup in Purdue University’s Indoor UAS Research and Test (PURT) Facility; each apex of pyramid shape represents one motion capture camera. . . . .	78
4.1	Hover in no-attack scenario: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . . . .	83
4.2	Hover in no attack scenario: (Left) Cell phone record; (right) MATLAB. . . . .	84
4.3	Cruise in no-attack scenario: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . . . .	85

4.4	Cruise in no-attack scenario: (Left) Cell phone record; (right) MATLAB. . . . .	86
4.5	Hover under 10 seconds of ICMP flooding attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . . . .	88
4.6	Hover under 10 seconds of ICMP flooding attack: (Left) Cell phone record; (right) MATLAB. . . . .	89
4.7	Hover under 30 seconds of ICMP flooding attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . . . .	90
4.8	Hover under 30 seconds of ICMP flooding attack: (Left) Cell phone record; (right) MATLAB. . . . .	91
4.9	Hover trajectory under: (Top) no-attack scenario; (middle) 10 seconds of ICMP flooding attack; (bottom) 30 seconds of ICMP flooding attack. . . . .	92
4.10	Hover trajectories under no attack, 10 seconds of ICMP flooding attack, and 30 seconds of ICMP flooding attack. . . . .	93
4.11	Cruise under 5 seconds of ICMP flooding attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . . . .	95
4.12	Cruise under 5 seconds of ICMP flooding attack: (Left) Cell phone record; (right) MATLAB. . . . .	96
4.13	(Left) Cruise trajectory for no-attack scenario; (right) cruise trajectory under 5 seconds of ICMP flooding attack. . . . .	97
4.14	Cruise trajectory under no attack and 5 seconds of ICMP flooding attack. . . .	97
4.15	Hover in no-attack scenario: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . . . .	99
4.16	Hover in no-attack scenario: (Left) Cell phone record; (right) MATLAB. . . . .	100
4.17	Cruise in no-attack scenario: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . . . .	101
4.18	Cruise in no-attack scenario: (Left) Cell phone record; (Right) MATLAB. . . .	102
4.19	Hover under the low-frequency random setpoints FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . .	104
4.20	Hover under the low-frequency random setpoints FDI attack: (Left) Cell phone record; (right) MATLAB. . . . .	105
4.21	Hover trajectories under no attack and the low-frequency random setpoints FDI attack. . . . .	106
4.22	Cruise under the low-frequency random setpoints FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . .	107

4.23	Cruise under the low-frequency random setpoints FDI attack: (Left) Cell phone record; (right) MATLAB. . . . .	108
4.24	Cruise trajectories under no attack and the low-frequency random setpoints FDI attack. . . . .	109
4.25	Hover under the low-frequency fixed setpoint FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . .	110
4.26	Hover under the low-frequency fixed setpoint FDI attack: (Left) Cell phone record; (right) MATLAB. . . . .	111
4.27	Hover trajectories under no attack and the low-frequency fixed setpoint FDI attack.	112
4.28	Recovery trajectory in hover test after the low-frequency fixed setpoint FDI attack.	112
4.29	Cruise under the low-frequency fixed setpoint FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . .	114
4.30	Cruise under low-frequency fixed setpoint FDI attack: (Left) Cell phone record; (right) MATLAB. . . . .	115
4.31	Cruise trajectory under no attack and the low-frequency fixed setpoint FDI attack.	116
4.32	Cruise trajectory after the low-frequency fixed setpoint FDI attack. . . . .	116
4.33	Hover under the high-frequency fixed setpoint FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . .	118
4.34	Hover under the high-frequency fixed setpoint FDI attack: (Left) Cell phone record; (right) MATLAB. . . . .	119
4.35	Hover trajectories under no attack and the high-frequency fixed setpoint FDI attack.	120
4.36	Hover trajectories under no attack, the low-frequency fixed setpoint attack, and the high-frequency fixed setpoint FDI attack. . . . .	120
4.37	Cruise under the high-frequency fixed setpoint FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment. . .	122
4.38	Cruise under the high-frequency fixed setpoint FDI attack: (Left) Cell phone record; (right) MATLAB. . . . .	123
4.39	Cruise trajectories under no attack and the high-frequency fixed setpoint FDI attack. . . . .	124
4.40	Cruise trajectories under no attack, the low-frequency fixed setpoint FDI attack, and the high-frequency fixed setpoint FDI attack. . . . .	124
4.41	Accuracy performance of RNNs with 65 features (%). . . . .	126
4.42	Accuracy performance of RNNs with 32 features (%). . . . .	127
4.43	Detection accuracy versus the factor of standard deviation used for the threshold.	128



4.44	Simple RNN IDS probability of each attack and classification under ICMP flooding attack. . . . .	131
4.45	Simple RNN IDS probability of each attack and classification under FDI attack.	131

## LIST OF SYMBOLS

$\odot$	Piecewise multiplication
$\sigma$	Sigmoid function
$b$	Bias for hidden unit for simple RNN
$b_c$	Bias for LSTM intermediate cell state
$b_f$	Bias for for LSTM and SRU forget gate
$b_h$	Bias for for GRU intermediate hidden state
$b_i$	Bias for for LSTM input gate
$b_o$	Bias for for LSTM output gate
$b_r$	Bias for for GRU and SRU reset gate
$b_z$	Bias for for GRU update gate
$c$	Bias for output for simple RNN
$c_t$	Cell state for LSTM and SRU at time t
$\hat{c}_t$	Intermediate cell state for LSTM at time t
$f_t$	Forget gate for LSTM and SRU at time t
$h(t)$	Hidden unit for simple RNN at time t
$h_t$	Hidden unit for simple RNN, LSTM,GRU, and SRU at time t
$\hat{h}_t$	Intermediate hidden state for GRU at time t
$i_t$	Input gate for LSTM at time t
$o(t)$	Output at time t, output gate for LSTM at time t
$r_t$	Reset gate for GRU and SRU at time t
$softmax$	Softmax activation function
$tanh$	Hyperbolic tangent function
$U$	Weight matrix between hidden units for simple RNN
$U_c$	Weight matrix between hidden units and intermediate cell state for LSTM
$U_f$	Weight matrix between hidden units and forget gate for LSTM
$U_h$	Weight matrix between hidden units, reset gate and intermediate hidden state for GRU
$U_i$	Weight matrix between hidden units and input gate for LSTM

$U_o$	Weight matrix between hidden units and output gate for LSTM
$U_r$	Weight matrix between hidden units and reset gate for GRU
$U_z$	Weight matrix between hidden units and update gate for GRU
$V$	Weight matrix between hidden unit and output for simple RNN
$v_r$	Weight matrix between cell state and reset gate for SRU
$W$	Weight matrix between input and hidden unit for simple RNN, Weight matrix between input and cell state for SRU
$W_c$	Weight matrix between input and intermediate cell state for LSTM
$W_f$	Weight matrix between input and forget gate for LSTM and SRU
$W_h$	Weight matrix between input and intermediate hidden state for GRU
$W_i$	Weight matrix between input and input gate for LSTM
$W_o$	Weight matrix between input and output gate for LSTM
$W_r$	Weight matrix between input and reset gate for GRU and SRU
$W_z$	Weight matrix between input and update gate for GRU
$\hat{y}_t$	Output after softmax activation for simple RNN at time t
$x(t)$	Input at time t
$z_t$	Update gate for GRU at time t

## ABBREVIATIONS

°C	Degree Celsius
A	Ampere
AGL	Above ground level
cm	Centimeter
CPU	Central processing unit
DoS	Denial of service
EKF	Extended Kalman filter
ESC	Electric speed controller
FDI	False data injection
FLOPS	Floating point operations per second
FLOPs	Floating point operations
ft	Feet
g	Grams
GCS	Ground control station
GPS	Global positioning system
GRU	Gated recurrent units
HITL	Hardware-in-the-loop
Hz	Hertz
ICMP	Internet control message protocol
IDS	Intrusion detection system
in	Inches
IMU	Inertial measurement unit
KB	Kilobytes
kts	Knots
lb	Pounds
LOIC	Low orbit ion cannon
LSTM	Long short-term memory
m	Meter

mm	Millimeter
MSL	Mean sea level
PoD	Ping of death
PX4	PX4 autopilot system
RNN	Recurrent neural network
s	Seconds
SRUs	Simple recurrent units
SITL	Software-in-the-loop
STL	Self-taught learning
SYN	Synchronize
SVM	Support vector machine
TCP	Transmission control protocol
UAVs	Unmanned aerial vehicles
V	Voltage

## ABSTRACT

Unmanned aerial vehicles (UAVs) have gained more attention in recent years because of their ability to execute various missions. However, recent works have identified vulnerabilities in UAV systems that make them more readily prone to cyberattacks. In this work, the vulnerabilities in the communication channel between the UAV and ground control station are exploited to implement cyberattacks, specifically, the denial of service and false data injection attacks. Unlike other related studies that implemented attacks in simulations, we demonstrate the actual implementation of these attacks on a Holybro S500 quadrotor with PX4 autopilot firmware and MAVLink communication protocol.

The goal was to create a lightweight intrusion detection system (IDS) that leverages recurrent neural networks (RNNs) to accurately detect cyberattacks, even when implemented on a resource-constrained platform. Different types of RNNs, including simple RNNs, long short-term memory, gated recurrent units, and simple recurrent units, were trained and tested on actual experimental data. A recursive feature elimination approach was carried out on selected features to remove redundant features and to create a lighter RNN IDS model. We also studied the resource consumption of these RNNs on an Arduino Uno board, the lowest-cost companion computer that can be implemented with PX4 autopilot firmware and Pixhawk autopilot boards. The results show that a simple RNN has the best accuracy while also satisfying the constraints of the selected computer.

# 1. INTRODUCTION

## 1.1 Background and Motivation

With lower operational costs and the maturing of autonomous control in recent years, unmanned aerial vehicles (UAVs) have gained increasing popularity in various fields. UAVs can also reduce risks to human life by remotely executing complex and potentially dangerous tasks without an onboard pilot. Usually, UAVs execute these missions by following the waypoints or reference inputs sent by a ground control station (GCS) to the onboard autopilot system via wireless communication. In return, such a communication channel can obtain periodic status reports about the UAV and, in an emergency, regain control of the UAV from the GCS. These functionalities indicate that continuous communication between UAVs and the GCS is indispensable for the safe execution of missions. Thus, the loss of this communication link can lead to mission delays, loss of control, or collisions.

Because UAVs execute missions in high-frequency and highly dynamic environments, communications must be handled with a fast and lightweight message transmitting protocol. Therefore, the communication between UAVs and the GCS is often not encrypted, as is the case with the MAVlink protocol used in PX4 autopilot firmware. Such a feature helps with facilitating high-frequency communication and ensures the safety of the mission. However, it can also be readily prone to cyberattacks. For example, adversaries can easily disrupt the communication between UAVs and the GCS by launching a denial of service (DoS) attack, such as a jamming or flooding attack, or can take over the UAV using false data injection (FDI) attack by implementing a de-authentication attack or injecting false waypoints. In [1] and [2], the authors demonstrated that the communication channel between UAVs and the GCS can be easily tampered with. Given these dangers, the development of an intrusion detection system (IDS) is important when considering the unique challenges of UAVs.

Furthermore, because an attacker can disrupt the communication channel, an emergency command cannot be executed remotely after the GCS detects the attack. Therefore, in this thesis, we consider the problem of implementing an onboard IDS for UAVs. This goal means we also must address the challenge of the resource-constrained environment on UAVs.

This work aims to develop a data-driven recurrent neural network (RNN)-based lightweight IDS to detect DoS and FDI attacks. At the same time, the onboard constraints pertaining to limited memory size, weight, and power are taken into account in the design of the RNN-based IDS.

## 1.2 Related Works

### 1.2.1 UAV Vulnerabilities

In the past few decades, the number of cyberattacks has increased dramatically, thereby driving more people to identify vulnerabilities and develop detection methods for systems such as power grids [3] and the internet of things [4]. However, it was not until a boom in the use of UAVs in recent years that research identifying UAV systems' vulnerabilities and developing detection schemes has started growing.

A comprehensive study of different UAV vulnerabilities was conducted by Kim et al. [5] which focused on hardware, wireless, and sensor spoofing attacks on a general autopilot architecture. The researchers also presented a hardware-in-the-loop (HITL) analysis of post-attack behavior. In [6], Dahiya et al. explored the different levels of vulnerabilities and their preventive measures. Anis et al. [1] provided an overview of the MAVlink communication protocol, which is mainly used in the common commercial autopilot systems ArduPilot and PX4. The identified security issues with and feasible cyberattacks on the MAVLink protocol pertain to confidentiality and privacy, integrity, availability, and authenticity of messages [1].

### Denial of Service (DoS)

A DoS attack, such as jamming and flooding, aims to disrupt a network or system and make the resources unavailable to intended users [7], [8]. In [8], two principal classes of DoS attacks, logic and flooding attack, were presented. A logic attack such as a ping of death (PoD) uses software flaws to crash or decrease the performance of a remote device. The PoD attack, which is a well-known example of a logic attack, pings the computer with a packet larger than 65535 bytes. Although software updates can help prevent logic attacks, they are still a severe issue today.



The second method, a flooding attack, overwhelms the resources, such as the central processing unit (CPU), memory, or network resources, of the target device by sending many unauthorized requests. Due to the nature of request messages, it is hard to differentiate legitimate requests from unauthorized ones. An example is the SYN-flood attack, in which an attacker transmits a large number of synchronize (SYN) packages to the transmission control protocol (TCP) port of a target device to initiate a connection but does not provide final confirmation. The target spends resources waiting for the connection, which leaves the target unavailable for authorized communications.

Vulnerability surveys of UAVs often group jamming and flooding attacks under communication interruptions or availability attacks [1]. Adversaries can launch a jamming attack using different strategies, such as constant jammer, deceptive jammer, random jammer, or reactive jammer, according to [9]. One of the most common strategies is injecting high-power noise into the frequency that UAVs use to receive information or commands [10]. Jamming attacks often come with global positioning system (GPS) spoofing because one of the methods of achieving spoofing is to disrupt communications with jamming attacks to induce a reconnection [11].

In [12], Vasconcelos et al. launched a flooding attack using a low orbit ion cannon (LOIC), Netwox, and Hping3. The study showed that when an AR.Drone is under a flooding attack, the frame rate of video streaming, one of the critical features for a pilot using a cell phone and camera onboard, is significantly affected. In [13], an internet control message protocol (ICMP) flooding attack using the hping3 tool was implemented on the MAVLink protocol-based 3DR X8+ drone. ICMP flooding works by sending ICMP request messages to the target UAV, requesting a response to the sender. When an adversary sends a large number of ICMP request messages with high frequency, the target system gets overloaded with processing and responding to these messages, and thus, a successful DoS attack is executed. Kwon et al. showed that the inter-reception time of packets in UAVs and the GCS increased by 35 and 10 times, respectively [13].

Jeong et al. [14] demonstrated a DoS attack by implementing three different types of false MAVLink message injection attacks: heartbeat flooding, ping flooding, and request flooding [14]. The heartbeat flooding attack sends heartbeat request messages to be received by the

target UAV, without any trigger from the autopilot. The ping flooding attack requests the target system to transmit ping responses to the sender and is mainly used on communication channels without ICMP support. Request flooding requests the target system send back the specific parameter values of the UAV, thus increasing the computation power being used due to the increased kernel task [15].

## False Data Injection (FDI)

FDI attacks inject a false signal into the original data to compromise the mission or disconnect communication with system components. This attack has been successfully demonstrated on complex systems, such as a power grid monitor system [16], [17].

FDI attacks can be mainly divided into two classes—sensor spoofing and unauthorized command injection. Sensor spoofing injects false sensor data into the onboard autopilot system and causes a UAV to deviate from their original planned trajectory or even collide with a surrounding obstacle. In [18], the authors compromised inertia measurement unit (IMU) sensor data by injecting false data following step and Gaussian functions in the roll, yaw, and pitch rate of a UAV. The GPS sensor is another common target of sensor spoofing attacks. In this form, the onboard GPS module is disconnected from the legitimate GPS signal, connected to an adversarial signal, and then controlled by the attacker [19].

Unlike sensor spoofing attacks, unauthorized command injection attacks inject false information into the autopilot system after a successful man-in-the-middle attack. The authors in [1] showed two methods for achieving unauthorized command injection—skyjacking and radio jacking. Skyjacking uses the lack-of-authentication vulnerability in the MAVLink protocol by sending a de-authentication message to disconnect a UAV from the GCS [20]. Attackers can use off-the-shelf software, such as aireplay-ng, to send such messages to disconnect legitimate users, while the attacker tries to establish a connection to take control of a UAV. Radio jacking also exploits the MAVLink protocol vulnerability to take control of UAVs. The MAVLink communication protocol is used in UAVs to establish NETID to allow the user to control UAVs via telemetry. Once an attacker recognizes the NETID informa-

tion, which can be done with a sniffing attack [21], they can send malicious packets and false information to take control of the UAV.

### 1.2.2 Intrusion Detection System (IDS)

Thanks to the increasing number of computers in use and a booming number of networks, IDSs have drawn a lot of attention since the 1990s [22]–[25]. However, it was not until recently that people began paying attention to the use of an IDS for UAVs.

Depending on the taxonomy used, according to [26], UAV IDSs can be divided into various categories. The first, an information gathering source, indicates the origin of the data used to analyze and detect an anomaly. Sensors, communication links, GCSs, and UAV components are typical sources that can be used to gather data. The chosen deployment strategy, such as ground/network-based or autonomous/UAV host-based, is another way of categorizing different IDSs. The detection state taxonomy indicates whether the detection state is evaluated in real-time or analyzed based on all collected data. In contrast, the response taxonomy differentiates IDSs based on whether it can respond to an anomaly in real time or if the analysis takes place after the period of data collection. In addition, IDSs can be categorized based on what kind of anomaly or adversary it is designed to detect, with typical scenarios including malicious software, modification of information, UAV control takeover, and spoofing. The last and most important type of categorization is based on the mechanism of the IDS, which the following:

- Specification-based
- Signature-based
- Anomaly-based
- Hybrid-based

The next section will further discuss these four detection methods.

## Detection Methods

The specification-based detection method develops rules based on the expected behavior of a UAV and implements the rules for monitoring a UAV's operations [27]–[30]. Mitchell and Chen [29] identified several such rules, including disarming weapons if they are detected outside the target area, using minimum thrust if a UAV is loitering, and stowing landing gear when the UAV is outside an airbase. These rules are transferred to state machines, where attack states indicate any violations of the rules. The state machine, which encompasses 4,443 unsafe states and 165 safe ones for a total of 4,608 states, assigns probabilities of a state transfer and uses binary grading to identify how close a state is to safe behavior. The measure of compliance with specified rules is in proportion to the number of times the UAVs are in safe states. In [30], Mitchell and Chen further extended their work [29] by showing the flexibility of considering new attacks and adaptively adjusting the detection strength to balance the false negative and false positive rates. The disadvantage of a specification-based IDS is that a great number of states need to be specified to accurately capture a UAVs' safe behavior, and it is therefore not practical for current applications.

The signature-based detection method intends to detect and/or identify known attacks by matching an attack to a set of available pre-defined signatures, features, and patterns [31]–[36]. While enjoying a high detection rate with known attacks pattern, signature-based IDS lacks the ability to detect previously unknown attacks patterns. In [32]–[34], Vuong et al. implemented DoS, FDI, and malware attacks on ground robotic vehicles and employed a Snort signature-based IDS [32] and a decision tree-based IDS [33], [34] to detect them. Their results showed that incorporating physical features with network-related features can help increase the attack detection accuracy of an IDS. The researchers claimed that detection latency can be a more critical metric than accuracy for an IDS in a highly dynamic system due to the severe consequences of late detection. Weng [35] used a bank of Kalman filters to detect the actuator intrusions of control systems. Two Kalman filters were used, one of which was the system dynamic under normal conditions and the other was compromised system dynamic. The author then obtained the conditional probability of each state estimation

being treated as true by the bank. The system was under attack when the conditional probability of compromised system dynamic is higher and vice versa.

DoS attacks, command injection attacks, and malware were applied in 4x4 robotic vehicles in [36]. This study compared the performance of different IDSs developed using logistic regression, decision trees, random forest, SVM, neural networks and RNNs. While decision tree outperformed the others with malware detection, and SVM with a radial kernel had the greatest accuracy in detecting DoS attacks, RNN was the best at detecting command injection attacks and overall accuracy.

The anomaly-based detection method aims to detect intrusion by observing failures or abnormal behaviors in a UAV [37]. The filtering [38] or learning mechanism often used to construct an anomaly-based IDS improves the ability to detect unknown attacks that do not have associated signatures in the database. Filtering mechanisms like a Kalman filter [38], [39] require a state-space model of selected features and threshold value settings. Therefore, using an accurate dynamic model and setting precise threshold values becomes a primary challenge for these methods. In [39], a residual calculated from the steady-state Kalman filter was used in the monitoring system. In cases where there is no attack, the residual should have a normal distribution with zero mean and a constant covariance matrix. The compound scalar testing checking residual power is used for hypothesis testing with a user-defined threshold value. Because this method uses a Kalman filter to calculate the residual, an accurate dynamic model is needed.

Conversely, a learning-based method, such as machine learning [14], [40], [41], does not require a thorough understanding of the selected feature's dynamics, and the learning-based method considers the correlation of different features. In particular, learning-based methods that use RNNs can explore the correlation of different features across specific time windows. Various machine learning-based IDSs, such as support vector machine (SVM), reinforcement learning, neural networks, and game theories are applied to UAVs to detect attacks using eavesdropping, jamming, GPS spoofing, and more. [40].

Xiao et al. [41] applied a simple RNN, long short-term memory (LSTM) and gated recurrent units (GRUs) to a UAV's arriving angle for the detection of anomalies caused by GPS spoofing attacks. The authors in [14] presented heartbeat flooding, ping flooding,

and request flooding attacks on the MAVLink protocol. Two monitoring methods were presented by Jeong et al. [14]. The first method directly used LSTM as the IDS and message ID sequences as features to detect whether an attack occurred. The software-in-the-loop (SITL) result for the heartbeat flooding attack showed that it required much more training to achieve comparable accuracy. The second method used LSTM as a predictor of the message ID sequence, training the network with normal case data and setting a threshold value for the accuracy of LSTM prediction. Because the normal case data were used to train the LSTM, the attack-free cases were expected to have high levels of accuracy. In contrast, the other three attack cases would result in less accuracy with message ID sequence prediction. Therefore, the second method can detect anomalies by setting a threshold of average prediction accuracy.

The hybrid detection method fuses two or more approaches to generate robust detection rules and achieve better detection rates [42], [43].

## Challenges

Effectiveness and efficiency are essential features when designing an IDS, especially for highly dynamic and resource-constrained UAV systems [26], [44]. Typically, lowering the consumption of computation and communication resources while decreasing the false detection rate and level of interruption are primary considerations when designing an IDS for UAVs. The authors in [26] illustrated several challenges to achieving these goals.

Detection latency represents how agile the IDS is when an attack is implemented. Most IDSs used in other systems pursue the fastest possible response; however, UAVs need to balance latency with onboard resource consumption to make the IDS practical for real-world implementation. Computational costs and implementation overhead are also significant challenges for constructing these IDSs. While high computational costs normally help increase detection accuracy, they can also lead to a high implementation overhead and large memory consumption, making methods with high computational costs unsuitable for UAV applications. In addition, high computational costs can lead to excess battery consumption, and increasing the size of an IDS can drain the memory space of a system.

In addition to the constraints of the UAV system, accurate attack modeling and a practical threat assessment design are other essential categories to be considered when designing an IDS. Finally, the IDS should detect attacks effectively and respond appropriately to avoid critical damage or further intrusion in a system.

### **1.3 Overview**

#### **1.3.1 Contribution**

Given the challenges just discussed, this thesis presents findings that make the following contributions:

- An investigation of the strengths and sensitivities of UAVs under DoS and FDI attacks.
- An analysis of an IDS constructed with actual experimental data collected from a PX4 autopilot system on a Holybro S500 quadrotor.
- Designed a lightweight IDS with RNN for detecting DoS and FDI attacks.

#### **1.3.2 Thesis Layout**

This thesis is organized as follows. Chapter 2 describes the overall system, attack methods, and IDS design, where Section 2.2 introduces the overall system, Section 2.3 provides a detailed description of attack methods, Section 2.4 introduces the IDS method, and Section 2.5 reviews the IDS design process. This is followed by a review of the experimental setup in Chapter 3, and Chapter 4 concludes with a detailed analysis of the experimental results.

## 2. VULNERABILITY AND IDS DESIGN

This chapter provides an overview of the UAV system and its vulnerabilities, along with the IDS design process. Section 2.1 covers the system overview, including the UAV (Section 2.1.1), the autopilot system (Section 2.1.2), data link (Section 2.1.3), and control schemes (Section 2.1.4). Section 2.2 details the vulnerabilities considered in this work, including assumptions (Section 2.2.1), DoS attacks (Section 2.2.2), and FDI attacks (Section 2.2.3). Section 2.3 presents an overview of the considered IDS (Section 2.3.1) and the factors included in the IDS design (Section 2.3.2).

### 2.1 System

#### 2.1.1 Unmanned Aerial Vehicles (UAVs)

Recent technological advances have pushed robotics toward more automation, especially with regard to UAVs. A UAV is an autonomous or remotely piloted aerial vehicle, including fixed-wing, VTOL, blimp, balloon, and hybrid types [45]. According to the US Department of Defense, UAVs can be categorized into five classes [46].

**Table 2.1.** UAV classification according to the US Department of Defense [46].

UAV group	Size	Maximum gross takeoff weight (lb)	Normal operating altitude (ft)	Speed (kts)
Group 1	Small	0–20	<1200 above ground level (AGL)	100
Group 2	Medium	21–55	<3500 AGL	<250
Group 3	Large	<1320	<18000 mean sea level (MSL)	<250
Group 4	Larger	>1320	<18000 MSL	Any airspeed
Group 5	Largest	>1320	>18000 MSL	Any airspeed



UAVs are commonly used because their variety of sizes and features can be customized for a wide range of applications of UAVs, they can be organized into the following taxonomy [47].

- Remote sensing
  1. Photogrammetric applications
  2. Precision agriculture
  3. Natural resource management
- Industrial inspection
  1. Civil infrastructure
  2. Electric power industry
  3. Wind turbine inspection
  4. Tower/Antenna inspection
  5. Oil/Gas inspection
- Aerial filming and photography
  1. Filmmaking
  2. Real estate
  3. Marketing
  4. News reporting
- Intelligence, surveillance, and reconnaissance and emergency response
  1. Law enforcement
  2. Search and rescue
  3. Communications relay
  4. Signal intelligence

- Atmospheric information collection
  1. Meteorology
  2. Hazardous material detection
  3. Radioactive material detection
- Applications requiring physical interaction with substances, materials, or objects
  1. Aerial chemical applications
  2. Water sampling
  3. Cargo/Package delivery

In this work, we are considering popular, commercial multi-rotor UAVs, which are suitable for a range of missions because of their agility and ability to hold a position in the air. Components commonly on multi-rotor UAVs include the following:

- Autopilot: Autopilot gathers all sensor data and incoming commands from a radio transmitter or GCS to process and implement control of the UAV.
- Body frame: The Body frame provides support for sensors, payloads, power units, and motors to satisfy mission needs.
- Motor and electronic speed controller (ESC): The motor provides control power for a UAV, while the ESC helps autopilot control each motor separately and thus provides 6 degrees of freedom control of a UAV.
- Power Unit: The power unit provides the power for entire UAV and is usually a battery.

### **2.1.2 Autopilot**

The advantage of UAV operation is its ability to execute the pre-defined mission without human intervention. In today's UAV applications, the autopilot system can help achieve this goal. An autopilot system collects the necessary sensor readings and incoming commands through wireless telemetry, including manual radio transmitter inputs or GCS, to calculate and provide the control signal to the onboard control surface.

## PX4 Firmware

PX4 is an open-source autopilot software used in various unmanned vehicles, including UAVs, rovers, and underwater vehicles [48]. To provide stabilization and control, it uses support sensors, including GPS/compass, airspeed, distance, and optical flow [49]. Companion computers such as Raspberry Pi, Arduino, and Nvidia board can help extend the functionality and computational power of PX4 [50]. The firmware is usually flashed onto the Pixhawk-series flight controller board through a USB connection with the QGroundControl desktop app [51].

To prevent critical situations while testing the development code or mission, PX4 provides a simulation feature, which is a quick, easy, and safe way to test changes before trying to fly in the real world. PX4 supports SITL simulation, where the whole flight simulation is in the computer, and HITL simulation, where the simulation is run on a real flight controller board [52], [53]. In addition to any parameters set up on the computer for the PX4 firmware using SITL and simulation firmware on the flight controller board for HITL, simulators such as Gazebo [54] and jMAVSim [55] need to be installed on the development computer. Furthermore, the robot operating system and a general-purpose robotics library need to be installed before starting any PX4-related development, such as a UAV application [56].

## Pixhawk

The PX4 firmware requires that the Pixhawk series flight controller board is running on a UAV or any other autonomous platform. The following are some commonly used Pixhawk series boards, according to [57]

- Holybro Pixhawk 4
- Holybro Pixhawk 4 Mini
- Drotek Pixhawk 3 Pro
- mRo Pixracer
- CUAV Pixhack v3

- Hex Cube Black
- mRo Pixhawk
- Holybro pix32
- Holybro Pixhawk Mini

### 2.1.3 Data Link

An essential factor for achieving autonomous missions is the data link used to transmit information, including the UAV's status and control commands, between the UAV and GCS. In this study, we focused on Wi-Fi communication, which transmits odometry information, such as the 6 DoF states of the UAV, and offboard control commands. Although radio frequency signal control also plays a significant role in controlling a UAV, it is not within the scope of this thesis.

### WiFi Module

Wi-Fi, an 802.11 wireless protocol digital data link, allows devices such as laptops, smart devices, and UAVs to interface with the internet and form a local network. With PX4, Wi-Fi telemetry facilitates MAVLink communication between the Wi-Fi modules onboard a UAV and the GCS through a UDP port [58]. Typically, Wi-Fi supports higher data transmission rates, but it suffers from a much shorter range than radio transmission. In PX4, Wi-Fi telemetry supports a higher data rate, with a 921600 baud rate (bit/s) in parameter settings, than radio telemetry, which has a 57600 default baud rate. In this study, external vision data and offboard control commands were sent via Wi-Fi telemetry.

### MAVLink Protocol

MAVLink is a lightweight message protocol used for communication between a GCS, UAVs, and UAV components. It can support most operating systems, including ARM7, ATmega, dsPic, STM32, Windows, Linux, macOS, Android, and iOS. MAVLink shows ex-

cellent potential because of its lightweight mechanism that fits nicely in limited bandwidth applications. MAVLink version 1 (MAVLink 1), released in 2009, only has 8 bytes of overhead per packet, while MAVLink version 2 (MAVLink 2), introduced in 2017, uses 14 bytes overhead per packet. More missions now require multi-agent collaboration, and MAVLink enables up to 255 systems, including UAVs and the GCS, on the network. In addition to communication between UAVs and the GCS, MAVLink also allows communication between autopilot and onboard sensors.

In MAVLink 1, the smallest packet is an acknowledgment packet without payload, which is 8 bytes in length. Conversely, the largest packet is 263 bytes with full payloads. Figure 2.1 and Table 2.2 show the structure of MAVLink 1 and its contents, respectively.

STX	LEN	SEQ	SYS ID	COMP ID	MSG ID	PAYLOAD	CHECKSUM
-----	-----	-----	--------	---------	--------	---------	----------

**Figure 2.1.** MAVLink 1 structure.

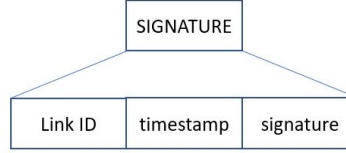
**Table 2.2.** MAVLink 1 structure content.

Acronym	Content	Description
STX	0XFE	Defines the start of the frame; should always be 0XFE, as defined in MAVLink 1.
LEN	0 to 255	Defines the length of a payload.
SEQ	0 to 255	Defines the sequence of packets; used to detect packet loss.
SYS ID	1 to 255	Defines the ID of a system.
COMP ID	0 to 255	Defines which component the system is sending the message to.
MSG ID	0 to 255	Defines the message type.
PAYLOAD	0 to 255 bytes	Contains real information about a message that is dependent on the message type.
CHECK-SUM	2 bytes contents	Contains CKA (1 byte) and CKB (1 byte), which ensure that the sender and receiver have the same message.

MAVLink 2 shares the same fields with MAVLink 1 and also has multiple new fields, including two flags and a signature, to add more functionality. The first type is incompatibility flags, which notify systems of the MAVLink library's features that support the packet. For instance, if the packet is signed and a signature is appended at the end of the message, the 0x01 bit of the incompatibility flag field (C flag = MAVLINK\_IFLAG\_SIGNED) must be set as true. The second type, compatibility flags, does not change the message structure and indicates the flags that a system can ignore if they are not understood. The main difference between incompatibility flags and compatibility flags is that if the system does not understand the incompatibility flags, the system will discard the packet. Conversely, even if a system does not understand a compatibility flag, it can still handle the packet.

Signature fields provide additional security for communications by verifying that a message is from a legitimate source. To use the signature feature, the 0x01 bit of the incompatibility flags must be true, at the end of the message, there must be appended 13 bytes of

the signature field. Figure 2.2 and Table 2.3 provide, respectively, the basic structure of the signature field and a description of its content.



**Figure 2.2.** MAVLink 2 signature field structure.

**Table 2.3.** MAVLink 2 signature structure content.

Acronym	Size	Description
Link ID	8 bits	ID of link regarding which packet is sent.
timestamp	48 bits	Timestamp in 10 microsecond units; since January 1, 2015, this has been GMT.
signature	48 bits	Signature for the packet, obtained by first 48 bits of SHA-256 hash of combination of secret key and complete message without signature data.

The 48-bit signature data in the signature field is the first 48 bits of the SHA-256 hash of the complete message without signature data appended to the secret key. The secret key is 32 bytes of binary data stored on both ends of the MAVLink communication channel. Equation (2.1) is the calculations of signature data, where + indicates concatenation.

$$signature = sha256\_48(secretkey + header + payload + checksum + linkid + timestamp) \quad (2.1)$$

The structure of MAVLink 2 and its content description are provided in, respectively, Figure 2.3 and Table 2.4.

STX	LEN	INC FLAGS	CMP FLAGS	SEQ	SYS ID	COMP ID	MSG ID	PAYLOAD	CHECKSUM	SIGNATURE
-----	-----	--------------	--------------	-----	--------	---------	--------	---------	----------	-----------

**Figure 2.3.** MAVLink 2 structure.

**Table 2.4.** MAVLink 2 structure content.

Acronym	Content	Description
STX	0XFD	Defines the start of the frame; should always be 0XFD, as defined in MAVLink 2.
LEN	0 to 255	Defines the length of payload.
INC FLAGS		Defines the flags that MAVLink needs to be supported.
CMP FLAGS		Defines the flags that MAVLink does not need to understand; flags can be ignored if MAVLink does not support.
SEQ	0 to 255	Defines the sequence of packets; used to detect packet loss.
SYS ID	1 to 255	Defines the ID of a system.
COMP ID	0 to 255	Defines to which component the system is sending the message.
MSG ID	0 to 255	Defines message type.
PAYLOAD	0 to 255 bytes	Contains real information of message dependent on the message type.
CHECKSUM	2 bytes contents	Contains CKA (1 byte) and CKB (1 byte), which can ensure sender and receiver have the same message.
SIGNATURE		Optional. Contains the signature used to ensure the communication is tamper-free.



PX4 uses MAVLink to communicate with the GCS, such as QGroundControl, and extra components onboard such as a companion computer and camera [59]. The set of messages and services are pre-defined for exchanging data and sending commands.

#### 2.1.4 Control Schemes

UAVs can be controlled through two main methods—manual or autonomous control. Manual control requires the user to provide control input through a radio transmitter or joystick. In contrast, autonomous control is conducted through an autopilot function and requires no pilot input. Table 2.5 shows all of the available control modes for multi-copter in PX4 [60].

**Table 2.5.** Control modes for multi-copter in PX4

Manual	Autonomous
Position, Altitude, Manual/Stabilized, Orbit, Acro	Hold, Return, Mission, Takeoff, Land, Follow Me, Offboard

#### Control Mode

- **Position mode:** Position mode, which is considered the safest manual control, uses a roll and pitch stick to control the acceleration level in lateral and longitudinal directions. Differently, the throttle and yaw stick manages the ascend/descend speed and spin of the UAV. When both sticks on the radio transmitter are released and centered, the autopilot system will lock the UAV in a position in 3D space by compensating for any external forces and drift. If the roll and pitch stick is not centered while the throttle and yaw stick is centered, the UAV will start cruising in a straight line at a constant speed.
- **Altitude mode:** Altitude mode also uses a roll and pitch stick to control the acceleration of the left/right and forward/backward directions. The yaw and throttle stick controls the spin rate and manages the ascend/descend speed. When all sticks are re-

leased and centered, the onboard autopilot maintains the UAV at its current altitude. The UAV will start drifting in the horizontal plane if any force is applied and stop until all momentum has dissipated.

- **Manual/Stabilized mode:** The manual mode uses a roll and pitch stick to control the attitude of the UAV and a yaw and throttle stick to manage spin speed and ascend/descend speed. When all sticks are centered, the autopilot will stabilize the UAV and hover in the current position. However, unlike the position and altitude mode, the UAV under manual mode will not maintain position or attitude if drifting occurred.
- **Acro mode:** Acro mode enables users to perform acrobatic maneuvers, such as flips and rolls. The roll, pitch, and yaw sticks manage the speed of attitude rotation in each axis, while the throttle stick sends the signal directly to the output mixer. When all sticks are centered, all rotation stops, but current orientations remain, and the UAV moves in the direction of its current momentum.
- **Orbit mode:** Orbit mode enables the UAV to move in a circle and maintain a yaw direction toward the circle's center. This mode requires GCS, where users can set up the radius and center of orbit. A radio transmitter is optional in this mode, but it can help to change the altitude, radius of the UAV's orbit, speed, and direction. The throttle stick controls speed along the altitude axis, while the roll stick manages the acceleration clockwise/counterclockwise, and the pitch stick manages the radius of the orbit.
- **Hold mode:** Hold mode makes the UAV stop, hover at its current position in three-dimensional (3D) space, and maintain position against external forces or drift. Hold mode can help to pause a mission or regain control of a UAV in an emergency. A predefined switch on a radio transmitter or pause button in a GCS (QGroundControl) can trigger hold mode.
- **Return mode:** Return mode helps the UAV fly a path to a secure location, where the course may follow a mission path or mission landing route, depending on the parameter

setting. A predefined radio transmitter switch or fail-safe activation can trigger this mode. The default behavior of return mode is to cause the UAV to reach a safe altitude, fly to the home position, and land.

- **Mission mode:** Mission mode allows the UAV to execute predefined flight plans that are uploaded to autopilot from the GCS. The predefined flight plan can be configured in GCS, such as the QGroundControl app, by selecting the actions and waypoints.
- **Takeoff mode:** Takeoff mode causes the UAV to ascend to a selected altitude and hover in position.
- **Land mode:** Land mode directs the UAV to land at the location where the mode is activated.
- **Follow me mode:** In follow me mode, the UAV follows the user autonomously by tracking user-provided position setpoints. A portable device provides the setpoints through a QGroundControl app or a MAVSDK app.
- **Offboard mode:** The UAV follows the position, velocity, or attitude setpoints provided by a user-defined script through MAVLink. Typically, this mode is engaged if the user wants to control the UAV from companion computers and GCS.

In this study, position mode was used in ICMP flooding attacks, while offboard mode was used in false waypoint injection.

## Extended Kalman Filter Estimation

Control of UAVs usually involves estimation to provide better position and attitude sensing. PX4 uses the extended Kalman filter (EKF) algorithm in the estimation and control library to estimate the following states [61]:

- Quaternion of UAVs from the north, east, and down local earth frame to the x,y, and z body frame.
- Velocity at the IMU, including north, east, and down in meters per second.

- Position at the IMU, including north, east, and down, in meters.
- IMU delta angle bias estimates, including x, y, and z in radians.
- IMU delta velocity bias estimates, including x, y, and z in meters per second.
- Earth magnetic field components, including north, east, and down directions in gauss.
- Vehicle body frame magnetic field bias, including x, y, and z in gauss.
- Wind velocity, including north and east directions in meters per second.

Though each sensor has its sensing frequency and sampling time, the EKF algorithm can run on a delayed fusion time horizon, enabling the fusion of various sensor measurements at a different time relative to the IMU. The sensor measurements will be stored in the buffer and used at the proper time, which is set by `EKF2_*_DELAY` parameters. Following are the sensors used by the EKF2 algorithm when available and enabled through parameter settings.

- IMU: X, y, and z body axis delta angle and delta velocity measured by IMU at a minimum 100 Hz rate.
- Magnetometer: X, y, and z body axis magnetometer data or external vision system pose at a minimum 5 Hz rate. EKF either combines and transforms the magnetometer data to yaw angle for estimation or directly uses the separate measurements. While the former is more robust but less accurate, the later gives a more precise estimation but performs weakly when anomalies occur.
- Height: Data from either GPS, barometer, range finder, or external vision system at a 5 Hz minimum rate.
- GPS: Position, velocity, and yaw measurements from a GPS/GNSS sensor. The `EKF2_AID_MASK` parameter must change and there must be a passing quality of GPS measurement to enable EKF to take these measurements.
- Range finder: Measurement of distance to the ground.

- Airspeed: Equivalent airspeed can estimate wind velocity and help reduce drift from GPS loss.
- Optical flow: Velocity estimation through downward-facing camera and distance sensor.
- External vision system: Position, velocity, and orientation measurements from external vision system such as Qualisys. By setting the value for EKF2\_AID\_MASK, the user can obtain several combinations of measurements.

For this study, the IMU, magnetometer, and external vision system were used in the EKF2 estimator.

## 2.2 Vulnerability

The MAVLink protocol best serves as the UAVs's communications protocol because of its lightweight mechanism that enables bandwidth and processing power limited applications. However, this benefit comes with the liability that the channel is normally not encrypted, which opens the door for adversaries to infiltrate the system in various ways.

### 2.2.1 Assumptions

Typically, to achieve an advanced attack, some assumptions are made beforehand that basic intrusions can be successfully implemented. In this study, we considered the ICMP flooding of DoS and the false waypoint injection of FDI. We made the following assumptions to achieve these two attacks.

- The IP address of the UAV is known.
- The port number of the UAV is known.
- The connection is in the same network as the Wi-Fi telemetry between the UAV and the GCS.

The first two assumptions can be achieved through sniffing attacks on the specific Wi-Fi modules, such as an ESP8266 [21]. The last assumption is valid because many examples have shown the way to crack a Wi-Fi network [62], [63]. Furthermore, the default Wi-Fi password for the ESP8266 on a PX4 is open to the public [64], which makes it easy for an attacker to connect if the legitimate user does not change the setting of the Wi-Fi module.

### 2.2.2 DoS—ICMP Flooding

ICMP inspects the connection status between a host and client and returns a warning if there are any problems with a packet transfer. A ping command in Windows, macOS, or Linux terminal can send an ICMP message to the target IP address. An ICMP message request packet be sent to the receiver when an ICMP message is transmitted, and the receiver of the request packet will respond to the sender. Suppose the sender transmits a large number of ICMP packets to a single receiver. In that case, the receiver will have to expend numerous resources checking and replying, causing an overload on the receiving system and triggering a DoS.

Tools such as LOIC, Netwox, and hping3 can help launch an ICMP flooding attack, sending a large number of ICMP messages to specific targets. In this study, we used hping3 to execute an ICMP flooding attack. hping3, the newest version of hping, is a command line controlled TCP/IP packet assembler and analyzer [65]. According to [65], hping3 supports the TCP, UDP, ICMP, and RAW-IP protocols and has the following features:

- Firewall testing
- Advanced port scanning
- Network testing
- Manual path maximum transmission unit (MTU) discovery
- Advanced traceroute
- Remote OS fingerprinting

- Remote uptime guessing
- TCP/IP stacks auditing

It also supports various operating systems, including Linux, FreeBSD, NetBSD, OpenBSD, Solaris, macOS, and Windows. In `hping3`, the command `"hping3 -flood"` given with a specified protocol, target IP address, and target port number will cause packets to be sent as rapidly as possible and achieve an ICMP flooding attack.

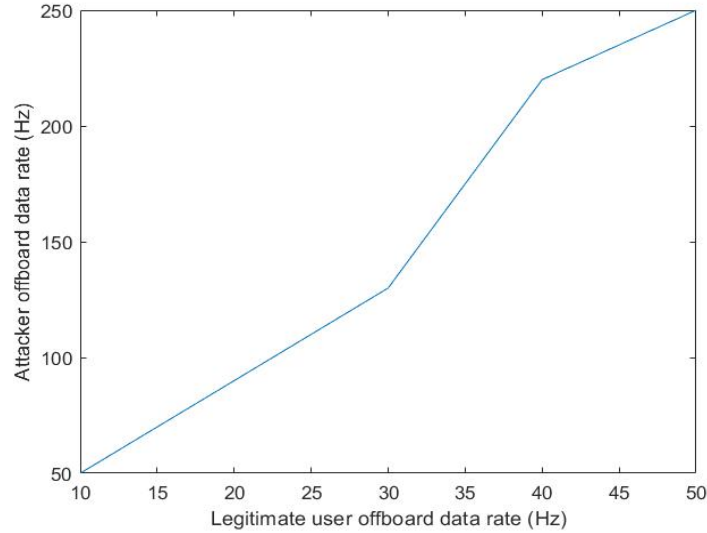
For this research, ICMP flooding was launched against the Wi-Fi module onboard UAVs in the position model controlled by the radio transmitter. The ICMP flooding attack caused delays with external vision position data and failed deliveries of accurate EKF2 estimations. These obstacles can cause UAVs to drift due to increasing errors in the estimator covariance or from velocity changes when the UAV incorrectly assumes it is still in the previous location.

### 2.2.3 FDI—False Waypoint Injection

The false waypoint injection is an FDI that injects an unauthorized waypoint command into the autopilot on a UAV with an offboard script on a malicious GCS. In this study, we assumed an attacker would know the IP address and port number of a target UAV, which allowed us to launch MAVROS using a modified launch script to create a ROS node between the UAV and a malicious GCS. Once a connection is established, the offboard script using different waypoints can be run through the malicious GCS and cause the UAV to drift from its original waypoints without warning the legitimate GCS.

After setting data rate for sending commands in both the legitimate user and attacker offboard script control, we observe that when the ratio of attacker offboard data rate to legitimate user offboard data rate was approximately greater than five, the attacker could force the UAV to land at the location the attacker set in the offboard script. Therefore, we conducted experiments with high-frequency and low-frequency data rates. The results were that at high-frequency data rates, the attack could force the UAV to disarm at a specific location, while with low-frequency data rates, they could not. Figure 2.4 shows the legitimate user frequency and the associated attacker frequency at which the attacker could

successfully force the UAV to land and disarm. Table 2.6 shows the ratios legitimate data rate to attacker data rate at which the attacker could cause the UAV to disarm.



**Figure 2.4.** Attacker offboard data rate that successfully forces UAVs to disarm vs. legitimate user frequency.

**Table 2.6.** Ratio of attacker data rate successfully forcing UAVs to disarm to legitimate user data rate.

Legitimate user data rate (Hz)	Attacker frequency (Hz)	Ratio
10	50	5
20	90	4.5
30	130	4.33
40	220	5.5
50	250	5

### 2.3 Intrusion Detection System (IDS)

An IDS detects a malicious infiltration in a UAV that has an intention to delay a mission, take over a UAV, or destroy a UAV by crashing it. The IDS can also trigger emergency actions such as switching to hold mode or emergency landing once an attack is detected.



For this study, we chose a supervised signature-based detection method over a specification-based or anomaly-based method for several reasons. First, the specification-based detection methods require users to specify a large number of states that accurately represent a UAV’s normal behaviors, and it is thus not practical for today’s complex applications. Second, the anomaly-based detection methods cannot identify specific threats, which helps recognize the type of attack.

Conversely, signature-based methods match the features, patterns, and signatures to detect and identify attacks. In this way, signature-based methods avoid the need to have a large number of rules input, as is the case with specification-based detection methods. In addition, the anomaly-based method has an improved ability to understand the type of attack by matching the associate signature. Understanding the type of attack can help us provide suitable emergency actions. For example, DoS attacks that block the necessary position information for navigation require an emergency landing to avoid unpredictable drift. Differently, FDI attacks that inject false waypoints and cause immediate deviation from the original course need a return-to-home action to avoid a collision.

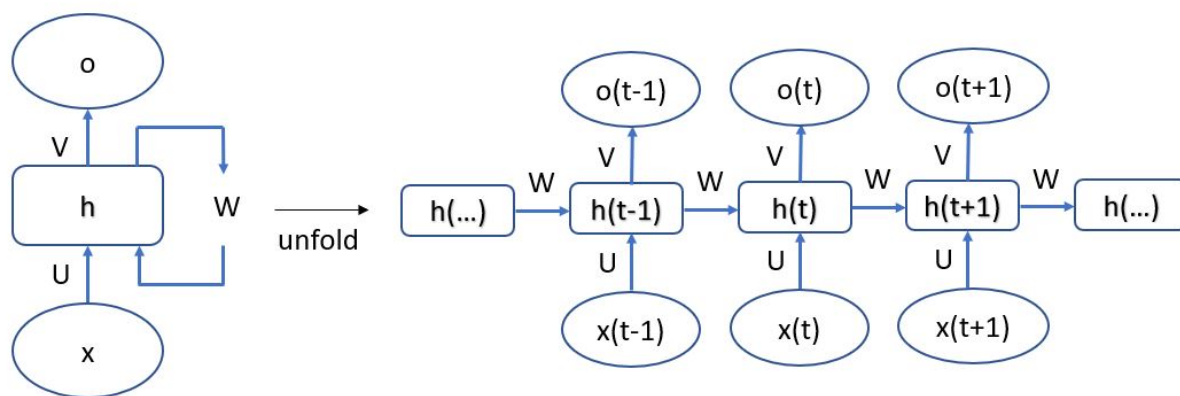
We constructed a signature-based IDS with a learning mechanism to allow the IDS to identify attacks without considering a dynamic model of selected features. The learning mechanism can also find correlations across multiple features and autonomously identify signatures.

### **2.3.1 Recurrent Neural Network (RNN)**

We chose to use an RNN, which is a class of neural networks, in this study rather than other machine learning methods because RNNs have the ability to process time series data and identify long-term temporal correlations. Instead of looking at a single time step, an RNN IDS can detect attacks when, within a certain window of time, selected features are tampered with or show deviations from normal conditions. To identify the best type of RNN for our IDS, we considered a simple RNN, LSTM, GRUs, and simple recurrent units (SRUs).

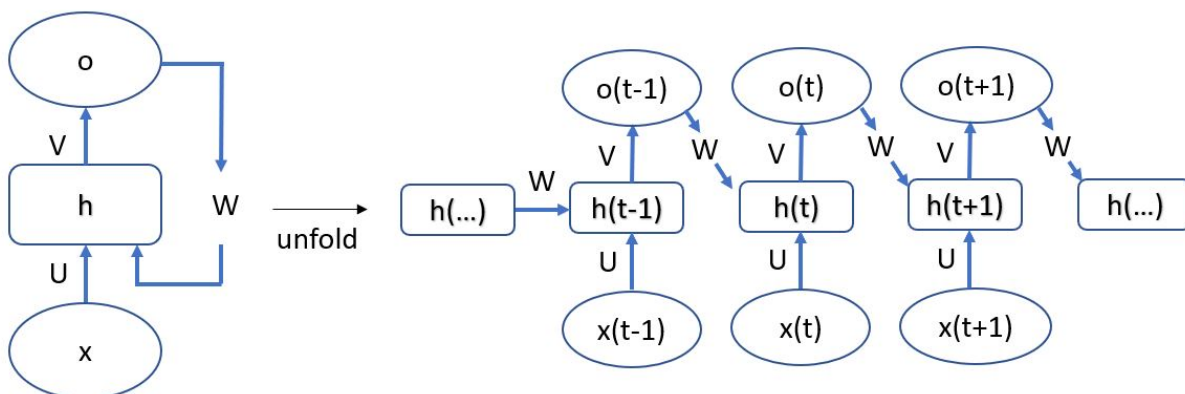
## Simple RNN

Three types of simple RNN are available. The first type of simple RNN, shown in Figure 2.5, creates output at each time step and recurrences between hidden units.  $W$ ,  $U$ , and  $V$  are weight matrices that are required to be learned during training, while  $x(t)$ ,  $h(t)$ , and  $o(t)$  represent input, hidden units, and output at time  $t$ , respectively.



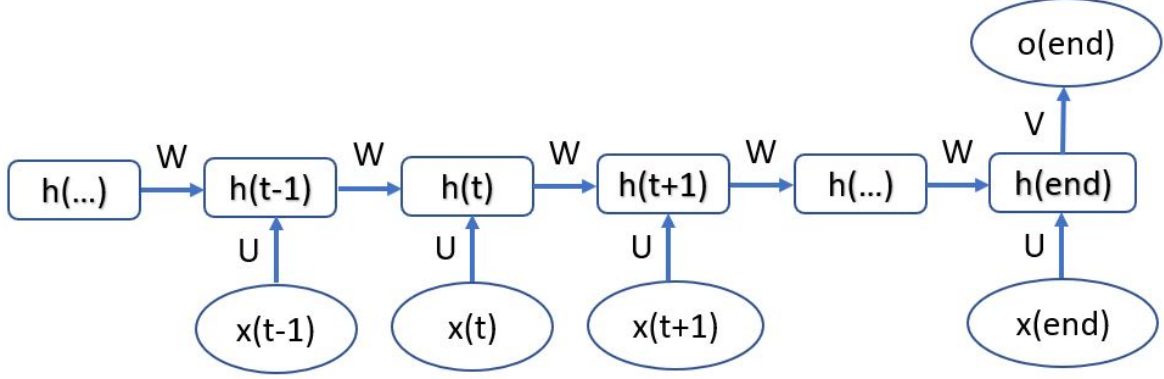
**Figure 2.5.** Simple RNN type 1.

The second type of simple RNN, shown in Figure 2.6, creates output at each time step and recurrences the output to the next hidden unit.



**Figure 2.6.** Simple RNN type 2.

The third type of simple RNN, shown in Figure 2.7, creates recurrences between hidden units, but only outputs after going through an entire sequence.



**Figure 2.7.** Simple RNN type 3.

Equations (2.2) through (2.4) are the basic structure of one recurrence of a simple RNN. In addition to the  $W$ ,  $U$ , and  $V$  weight matrices,  $b$  and  $c$  are bias vectors that also need to be learned during the training phase. Equation (2.2) takes current input  $x_t$  and previous hidden unit  $h_{t-1}$  to calculate the current hidden unit  $h_t$ . In the second type of simple RNN, the  $Uh_{t-1}$  term in Equation (2.2) is replaced by  $Uo_{t-1}$ , which due to the structure of this simple RNN type, is the recurrent output used to calculate the next hidden unit. Equation (2.3) calculates the output from the hidden unit, and Equation (2.4) uses the softmax function to transfer outputs and obtain probabilities of the output. In the third type, simple RNN will not calculate  $o_t$  and  $y_t$  until the last step because this type of RNN outputs only after an entire data length has passed.

$$h_t = \tanh(Wx_t + Uh_{t-1} + b), \quad (2.2)$$

$$o_t = Vh_t + c, \quad (2.3)$$

$$\hat{y}_t = \text{softmax}(o_t), \quad (2.4)$$

It is noteworthy that with this last type of simple RNN, the hidden units are often considered to be a direct output, and therefore, Equations (2.3) and (2.4) are not required.

For this work, we selected the third type of simple RNN with the hidden unit as output, which goes through a sequence of selected features to decide, without using Equation (2.3) and (2.4), whether an attack happened.

## Long-Short Term Memory (LSTM)

LSTM, developed in 1997 by Hochreiter and Schmidhuber [66], has shown high levels of accuracy in multiple application domains. In a simple RNN, the vanishing gradient problem is encountered during the training phase with a gradient-learning based method and backpropagation. LSTM introduces three gates—an input, an output, and a forget gate—to address this issue, controls the amount of information passed over time, and avoids the vanishing gradient problem. Equations (2.5) through (2.10) and Figure 2.8 form the basic structure of a single LSTM cell, which can in larger quantities, further construct a layer.  $W_f$ ,  $W_i$ ,  $W_o$ ,  $W_c$ ,  $U_f$ ,  $U_i$ ,  $U_o$ , and  $U_c$  are weighted matrices, and  $b_f$ ,  $b_i$ ,  $b_o$ , and  $b_c$  are vectors that must be learned during training.

Equations (2.5), (2.6), and (2.7) represent, respectively, the forget gate, the input gate, and the output gate. These three equations determine, respectively, the amount of information to forget from the last cell state  $c_{t-1}$ , keep from the current input  $x_t$ , and output to a current hidden state  $h_t$ . This gating mechanism greatly aids LSTM to memorize important information over long time frames.

Equation (2.8) serves as the intermediate cell state that will then be multiplied by the input gate in an element-wisely manner in Equation (2.9). Equation (2.9) represents the current cell state, which is the weighted sum of the forget gate,  $f_t$ , times the previous cell state,  $c_{t-1}$ , and the input gate,  $i_t$ , times the intermediate cell state,  $\hat{c}_t$ . Equation (2.10) calculates the current hidden unit by element-wise multiplication of the output gate,  $o_t$ , and the current cell state under the tangent hyperbolic function,  $\tanh(c_t)$ .

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \quad (2.5)$$

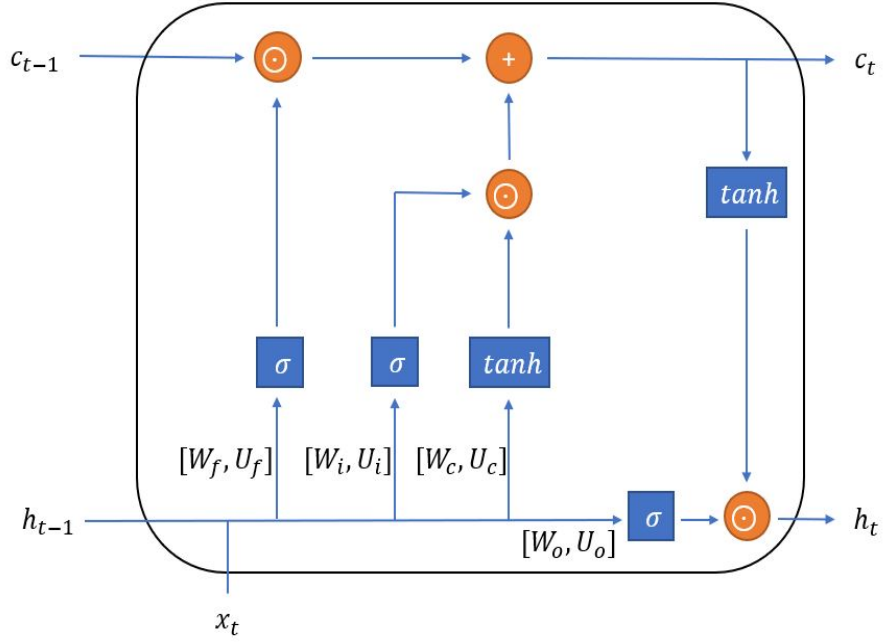
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \quad (2.6)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \quad (2.7)$$

$$\hat{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \quad (2.8)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t, \quad (2.9)$$

$$h_t = o_t \odot \tanh(c_t). \quad (2.10)$$



**Figure 2.8.** Single LSTM cell.

## Gated Recurrent Units (GRUs)

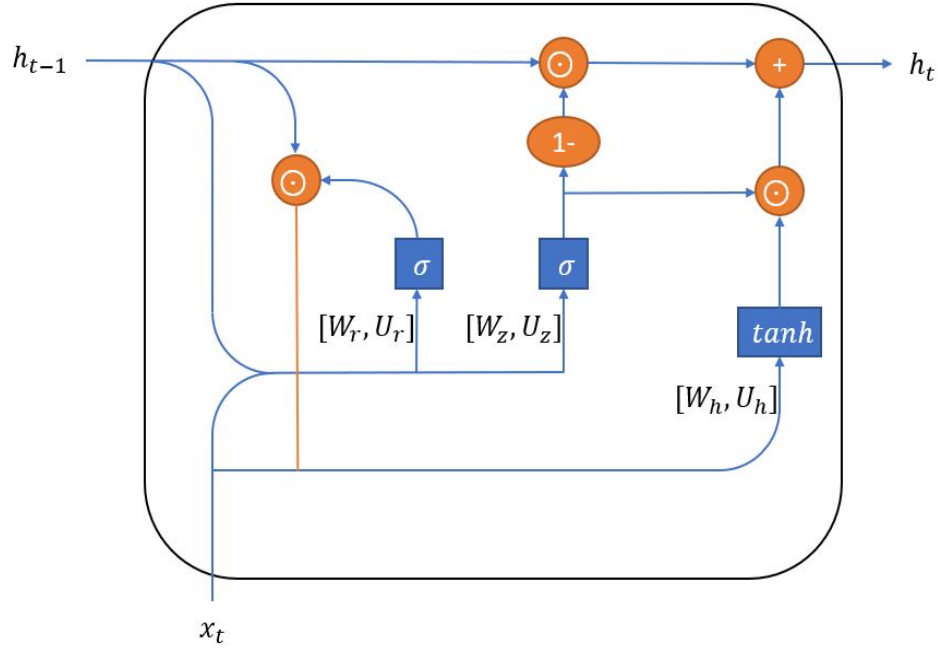
GRUs, developed in 2014 by Kyunghyun Cho et al. [67], reduce the required number of gates down to two from the LSTM. Equations (2.11) through (2.14) and Figure 2.9 form a single GRU cell, which can in number further construct a layer.  $W_z$ ,  $W_r$ ,  $W_h$ ,  $U_z$ ,  $U_r$ , and  $U_h$  are weighted matrices that, together with the bias vectors  $b_z$ ,  $b_r$ , and  $b_h$  must be learned during training. Equations (2.11) and (2.12) represent an update gate and a reset gate, respectively. The update gate, Equation (2.11), determines the amount of information to forget from the previous hidden unit,  $h_{t-1}$ , and to keep from the intermediate hidden unit,  $\hat{h}_t$ . Equation (2.12) represents the reset gate, which determines the amount of information to take into account from the previous hidden unit,  $h_{t-1}$ , for the intermediate hidden unit,  $\hat{h}_t$ . Equation (2.13) serves as the intermediate state, which will be multiplied by the update gate in an element-wise fashion in Equation (2.14). The current hidden state is calculated in Equation (2.14) by the weighted sum of the previous hidden state,  $h_{t-1}$ , and the intermediate hidden state,  $\hat{h}_t$ , with respect to the update gate, Equation (2.11).

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \quad (2.11)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r), \quad (2.12)$$

$$\hat{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h), \quad (2.13)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t. \quad (2.14)$$



**Figure 2.9.** Single GRU cell.

### Simple Recurrent Units (SRUs)

An SRU is a type of RNN that requires fewer parameters than more popular RNNs such as LSTM and GRU, thereby sufficiently speeding up computations and reducing memory requirements. While LSTM uses three gates and GRU uses two gates to control the amount of information passed over time, the SRU structure uses light recurrence and highway connection to handle the information over time, which decreases the number of parameters required.

The light recurrence captures sequential information by reading the input from the last step and computing the state vector. Although LSTM and GRU also contain such components, SRU surpasses these two methods by parallelizing the computation and substituting matrix multiplication with point-wise multiplication. Highway connections facilitate gradient-based training for deep networks, meaning they allow gradients to propagate directly. Equations (2.15) through (2.18) and Figure 2.10 show the structure of a single SRU

cell, which can further construct a layer in numbers.  $W$ ,  $W_f$ , and  $W_r$  are the weight matrices and  $v_f$ ,  $v_r$ ,  $b_f$ , and  $b_r$  are the vectors that must be learned during training.

Equations (2.15) and (2.16) correspond to the light recurrence step, while Equations (2.17) and (2.18) correspond to the highway connections. The forget gate  $f_t$  in Equation (2.15) controls the information flow with input vector  $x_t$  and previous state vector  $c_{t-1}$ . The current state vector  $c_t$  is computed in Equation (2.16) by adaptively averaging the previous state vector  $c_{t-1}$  and current information  $Wx_t$  according to forget gate  $f_t$ . The key advantage in Equations (2.15) and (2.16) is the point-wise multiplication. In LSTM and GRU, the previous state vector  $c_{t-1}$  is multiplied by a weight matrix in the forget gate  $f_t$ . This means all of the dimensions of the forget gate  $f_t$  and current state vector  $c_t$  depend on all of the entries of the previous state vector  $c_{t-1}$ . Therefore, the following computation cannot be completed until the matrix multiplication of the previous state vector  $c_{t-1}$  is fully computed. In SRU, point-wise multiplication makes all dimensions of  $v_f \odot c_{t-1}$  independent, thus parallelizing and accelerating the computation. In addition, SRU reduces the number of parameters needed by substituting the weight matrix with the vector  $v_f$ . The reset gate, Equation (2.17), adaptively averages the input vector  $x_t$  and current state vector  $c_t$  in Equation (2.18). The second term in Equation (2.18),  $(1 - r_t) \odot x_t$ , allows the gradient to propagate directly to the previous layer and thus facilitates gradient-based learning.

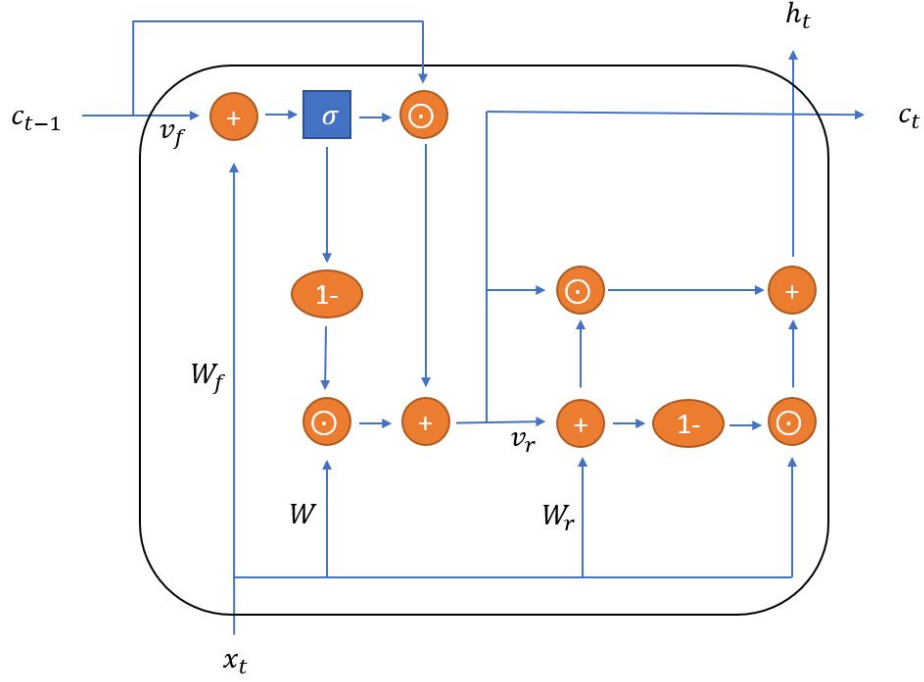
$$f_t = \sigma(W_f x_t + v_f \odot c_{t-1} + b_f), \quad (2.15)$$

$$c_t = f_t \odot c_{t-1} + (1 - f_t) \odot (Wx_t), \quad (2.16)$$

$$r_t = \sigma(W_r x_t + v_r \odot c_t + b_r), \quad (2.17)$$

$$h_t = r_t \odot c_t + (1 - r_t) \odot x_t. \quad (2.18)$$





**Figure 2.10.** Single SRU cell.

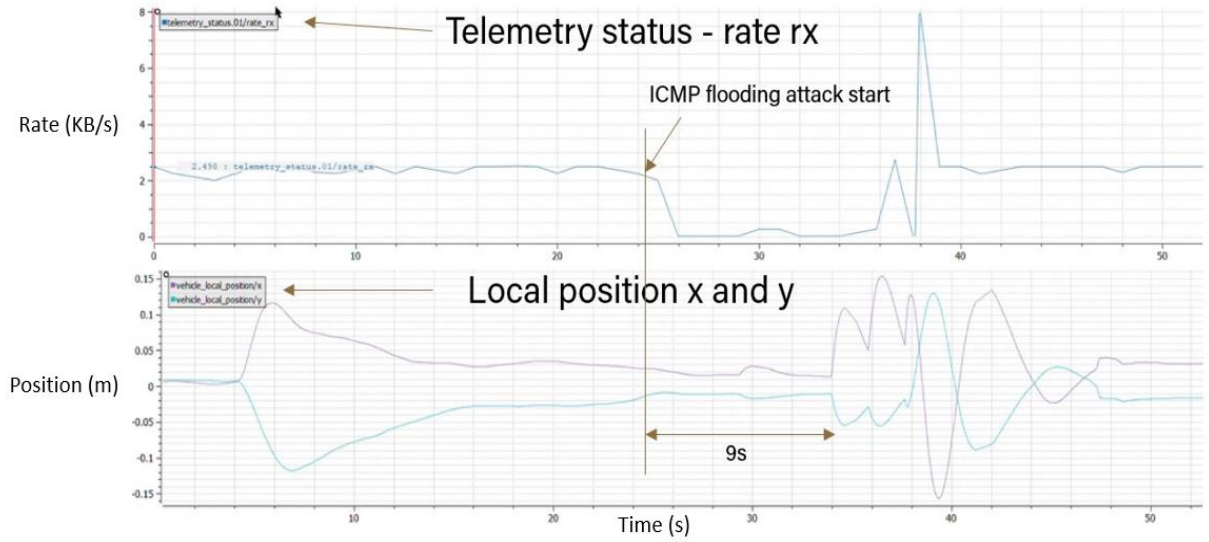
According to [68], even with a smaller number of parameters, SRUs deliver better results than LSTM and convolutional neural networks (CNNs) on natural language processing (NLP) tasks. Furthermore, SRUs achieved a fivefold to ninefold improvement in speed over NVIDIA CUDA deep neural network (cuDNN)-LSTM in the classification and question-answer data set.

Recently, a new variant of SRU, called SRU++ [69], was developed by replacing the linear projection of input (i.e., batched multiplication between the input vector  $x_t$ ,  $W$ ,  $W_f$ , and  $W_r$ ) with a self-attention module. According to [69], SRU++ further decreases the number of parameters needed in the SRU cell and increases the computation speed while delivering better results compared to other existing models.

### 2.3.2 Features

A complete set of training features obtained from the PX4 logs will increase the number of parameters in the IDS model, causing the IDS model's memory size and computational power to grow. Therefore, it is necessary to reduce the number of parameters. Instead of directly

looking into vehicle dynamics, which usually respond later than other parameters when an attack occurs, we selected 65 features from 11 messages related to sensor bias, estimator error, covariance, and so on. Figure 2.11 shows the telemetry rate rx and vehicle local position for hover mission under ICMP flooding attack. While rate rx responds immediately after the attack is launched, the UAV's position, representing vehicle dynamics, does not show any effects until 9 seconds after the attack occurs.



**Figure 2.11.** Telemetry status rate rx and vehicle local position for hover mission under ICMP flooding attack. (Top): Telemetry status-rate rx; (bottom): local position x and y, (purple): x, (blue): y.

The following table shows the complete list of selected features.

**Table 2.7.** Selected features.

Message	Name	Description
cpuload	load	Processor load from 0 to 1.
cpuload	ram usage	RAM usage from 0 to 1.
ekf2 timestamp	visual odometry timestamp rel	Relative timestamp of the odometry input by ekf2.

**Table 2.7.** Selected features

Message	Name	Description
estimator innovation test ratios	ev_hpos[0]	Horizontal external vision position axis 1 innovation test ratio.
estimator innovation test ratios	ev_hpos[1]	Horizontal external vision position axis 2 innovation test ratio.
estimator innovation test ratios	ev_vpos	Vertical external vision position x innovation test ratio.
estimator innovation test ratios	ev_hvel[0]	Horizontal external vision velocity in axis 1 innovation test ratio.
estimator innovation test ratios	ev_hvel[1]	Horizontal external vision velocity in axis 2 innovation test ratio.
estimator innovation test ratios	ev_vvel	Vertical external vision velocity innovation test ratio.
estimator innovation test ratios	heading	Heading innovation test ratio.
estimator innovation variances	ev_hpos[0]	Horizontal external vision position axis 1 innovation variance.
estimator innovation variances	ev_hpos[1]	Horizontal external vision position axis 2 innovation variance.
estimator innovation variances	ev_vpos	Vertical external vision position innovation variance.
estimator innovation variances	ev_hvel[0]	Horizontal external vision velocity in axis 1 innovation variance.
estimator innovation variances	ev_hvel[1]	Horizontal external vision velocity in axis 2 innovation variance.
estimator innovation variances	ev_vvel	Vertical external vision velocity innovation variance.

**Table 2.7.** Selected features

Message	Name	Description
estimator innovation variances	heading	Heading innovation variance.
estimator innovations	ev_hpos[0]	Horizontal external vision position axis 1 innovation.
estimator innovations	ev_hpos[1]	Horizontal external vision position axis 2 innovation.
estimator innovations	ev_vpos	Vertical external vision position innovation.
estimator innovations	ev_hvel[0]	Horizontal external vision velocity in axis 1 innovation.
estimator innovations	ev_hvel[1]	Horizontal external vision velocity in axis 2 innovation.
estimator innovations	ev_vvel	Vertical external vision velocity innovation.
estimator innovations	heading	Heading innovation.
estimator sensor bias	gyro bias[0]	Gyroscope in-run bias in body frame axis 1.
estimator sensor bias	gyro bias[1]	Gyroscope in-run bias in body frame axis 2.
estimator sensor bias	gyro bias[2]	Gyroscope in-run bias in body frame axis 3.
estimator sensor bias	accel bias[0]	Accelerometer in-run bias in body frame axis 1.
estimator sensor bias	accel bias[1]	Accelerometer in-run bias in body frame axis 2.

**Table 2.7.** Selected features

Message	Name	Description
estimator sensor bias	accel bias[2]	Accelerometer in-run bias in body frame axis 3.
estimator status	covariances[0]	Quaternion 1 variance.
estimator status	covariances[1]	Quaternion 2 variance.
estimator status	covariances[2]	Quaternion 3 variance.
estimator status	covariances[3]	Quaternion 4 variance.
estimator status	covariances[4]	Velocity in north direction variance.
estimator status	covariances[5]	Velocity in east direction variance.
estimator status	covariances[6]	Velocity in down direction variance.
estimator status	covariances[7]	Position in north direction variance.
estimator status	covariances[8]	Position in east direction variance.
estimator status	covariances[9]	Position in down direction variance.
estimator status	covariances[10]	IMU delta angle bias in x direction.
estimator status	covariances[11]	IMU delta angle bias in y direction.
estimator status	covariances[12]	IMU delta angle bias in z direction.
estimator status	covariances[13]	IMU delta velocity bias in x direction.
estimator status	covariances[14]	IMU delta velocity bias in y direction.
estimator status	covariances[15]	IMU delta velocity bias in z direction.
estimator status	pos test ratio	Ratio of the largest horizontal position innovation component to the innovation test limit.
estimator status	pos horiz accuracy	1-Sigma estimated horizontal position accuracy relative to the estimator's origin.
estimator status	pos vert accuracy	1-Sigma estimated vertical position accuracy relative to the estimator's origin.

**Table 2.7.** Selected features

Message	Name	Description
estimator status	mag test ratio	Ratio of the largest magnetometer innovation component to the innovation test limit.
estimator status	vibe[0]	IMU vibration metric—gyroscope delta angle coning metric.
estimator status	vibe[1]	IMU vibration metric—gyroscope high-frequency vibe.
estimator status	vibe[2]	IMU vibration metric—accelerometer high-frequency vibe.
rate ctrl status	rollspeed	Roll speed integrator.
rate ctrl status	pitchspeed	Pitch speed integrator.
rate ctrl status	yawspeed	Yaw speed integrator.
telemetry status	rate rx	Telemetry receiving rate.
telemetry status	rate tx	Telemetry transmitting rate.
vehicle imu status	accel vibration metric	High-frequency vibration level in the IMU accelerometer data.
vehicle imu status	gyro vibration metric	High-frequency vibration level in IMU gyroscope data.
vehicle imu status	gyro coning vibration	Level of coning vibration in the IMU delta angle.
vehicle local position	eph	Standard deviation of horizontal position error.
vehicle local position	epv	Standard deviation of vertical position error.
vehicle local position	evh	Standard deviation of horizontal velocity error.

**Table 2.7.** Selected features

Message	Name	Description
vehicle local position	evv	Standard deviation of vertical velocity error.

We used recursive feature elimination (RFE) to find suitable representatives for detecting attacks and decreasing the number of selected features. RFE is a systematic feature selection method that can help in choosing a subset of features when necessary. RFE will recursively fit the selected model with a smaller set of features by eliminating the weakest feature (or features) in each iteration until the target number of features is reached. The following table shows the 32 features selected by RFE using the random forest model.

**Table 2.8.** Features selected using RFE and random forest.

Message	Features
cpuload	load, ram usage
estimator innovation test ratios	ev_hpos[1], ev_hvel[1]
estimator innovation variances	ev_hpos[1]
estimator sensor bias	gyro bias[0], gyro bias[2], accel bias[0], accel bias[2]
estimator status	covariances[0], covariances[1], covariances[2], covariances[4], covariances[5], covariances[6], covariances[7], covariances[8], covariances[9], covariances[10], covariances[11], pos horiz accuracy, vibe[0], vibe[1], vibe[2]
integ rate ctrl status	yawspeed
telemetry status	rate rx
vehicle imu status	accel vibration metric, gyro vibration metric, gyro coning vibration
vehicle local position	eph, evh, evv

### 2.3.3 Design

A neural network IDS typically requires a high level of computational power and sizeable onboard memory to store the model onboard. Traditional IDS fields, such as the internet of things and smart power grids have enough power and memory space, so they rarely face such disadvantages. However, high level of computational power and sizeable onboard memory become a critical concern for UAV applications because they are typically size, weight, and power-constrained with the software and hardware onboard.

#### Memory

The memory required for a neural network IDS can be determined by the number of parameters in the neural network model. Let  $n$  represent the number of hidden units in one layer of RNNs and  $m$  represent the input dimension. Table 2.9 compares the number of parameters used in a layer of simple RNN, LSTM, GRU, and SRU. We do not consider Equations (2.3) and (2.4) in this table.



**Table 2.9.** Number of parameters in a single layer of simple RNN, LSTM, GRU, and SRU.

RNN	Parameters	Total parameters
Simple RNN	$W \in n \times m, U \in n \times n, b \in n$	$n^2 + nm + n$
LSTM	$W_f \in n \times m, U_f \in n \times n, b_f \in n$ $W_i \in n \times m, U_i \in n \times n, b_i \in n$ $W_o \in n \times m, U_o \in n \times n, b_o \in n$ $W_c \in n \times m, U_c \in n \times n, b_c \in n$	$4 \times (n^2 + nm + n)$
GRU	$W_z \in n \times m, U_z \in n \times n, b_z \in n$ $W_r \in n \times m, U_r \in n \times n, b_r \in n$ $W_h \in n \times m, U_h \in n \times n, b_h \in n$	$3 \times (n^2 + nm + n)$
SRU	$W_f \in n \times m, v_f \in n, b_z \in n$ $W \in n \times m$ $W_r \in n \times m, v_r \in n, b_r \in n$	$3nm + 4n$

Based on the machine learning library, the number of parameters might vary a little due to algorithms. In PyTorch, the machine learning library developed by Facebook's AI Research lab, the bias vectors for input and hidden unit in simple RNN, LSTM, and GRU are separate. Therefore, we needed to add additional  $n$  parameters to each equation with bias vectors. Table 2.10 compares the number of parameters used in a layer of simple RNN, LSTM, GRU, and SRU in PyTorch. We do not consider Equations (2.3) and (2.4) in this table.

**Table 2.10.** Number of parameters in a single layer of simple RNN, LSTM, GRU, and SRU in PyTorch.

RNN	Parameters	Total parameters
Simple RNN	$W \in n \times m, U \in n \times n, b \text{ for } Wx_t \in n,$ $b \text{ for } Uh_{t-1} \in n$	$n^2 + nm + 2n$
LSTM	$W_f \in n \times m, U_f \in n \times n, b \text{ for } W_fx_t \in n,$ $b \text{ for } U_fh_{t-1} \in n$ $W_i \in n \times m, U_i \in n \times n, b \text{ for } W_ix_t \in n,$ $b \text{ for } U_ih_{t-1} \in n$ $W_o \in n \times m, U_o \in n \times n, b \text{ for } W_ox_t \in n,$ $b \text{ for } U_oh_{t-1} \in n$ $W_c \in n \times m, U_c \in n \times n, b \text{ for } W_cx_t \in n,$ $b \text{ for } U_ch_{t-1} \in n$	$4 \times (n^2 + nm + 2n)$
GRU	$W_z \in n \times m, U_z \in n \times n, b \text{ for } W_zx_t \in n,$ $b \text{ for } U_zh_{t-1} \in n$ $W_r \in n \times m, U_r \in n \times n, b \text{ for } W_rx_t \in n,$ $b \text{ for } U_rh_{t-1} \in n$ $W_h \in n \times m, U_h \in n \times n, b \text{ for } W_hx_t \in n,$ $b \text{ for } U_h(r_t \odot h_{t-1}) \in n$	$3 \times (n^2 + nm + 2n)$
SRU	$W_f \in n \times m, v_f \in n, b_z \in n$ $W \in n \times m$ $W_r \in n \times m, v_r \in n, b_r \in n$	$3nm + 4n$

Typically, after the last layer of RNNs, the classification problem will use a single dense layer to shrink the number of hidden units from the last layer to the number of classes to get classification probabilities. Thus, an additional  $(n+1) \times \text{output dimension}$  (i.e., *number of classes*) parameters need to be added to each model. The  $n \times \text{output dimension}$  (i.e., *number of classes*) parameters are weights and  $1 \times \text{output dimension}$  (i.e., *number of classes*) parameters are bias. Equations (2.19), (2.20), (2.21), and (2.22) show the final total parameters for  $k$  layers of RNN with a dense layer attached in PyTorch. Where  $n$  is the number of hidden units in

one layer,  $m$  is the number of inputs in a single time step, and  $o$  is the number of the final outputs, meaning number of classes,

$$\text{simple RNN total parameters} = k \times (n^2 + nm + 2n) + o \times (n + 1) \quad (2.19)$$

$$\text{LSTM total parameters} = k \times (4 \times (n^2 + nm + 2n)) + o \times (n + 1) \quad (2.20)$$

$$\text{GRU total parameters} = k \times (3 \times (n^2 + nm + 2n)) + o \times (n + 1) \quad (2.21)$$

$$\text{SRU total parameters} = k \times (3nm + 4n) + o \times (n + 1) \quad (2.22)$$

After calculating the total parameters, we can calculate how big the model is in bytes according to parameter variable type. In this work, we used the 32-bit floating-point numbers that occupy 4 bytes per number. However, the actual final model memory might be larger than the memory calculated by the number of parameters due to additional information the model needs to carry. Table 2.11 shows the ratio of the final RNN model's memory to the memory calculated by the number of parameters in PyTorch.

**Table 2.11.** Ratio of final model memory in PyTorch to memory calculated by number of parameters in RNN, LSTM, GRU, and SRU in PyTorch.

<b>RNN</b>	<b>Ratio</b>
Simple RNN	1.76×
LSTM	1.25×
GRU	1.31×
SRU	1.44×

In addition to model memory, some libraries and software required to running an IDS also need to account for onboard memory usage. For instance, the torch library for PyTorch

can take 169 MB of space onboard. It should be noted that memory for storing temporary feature data, which has *number of features*  $\times$  *number of timestep* parameters, should also be considered in total memory. Equation (2.23) shows that total memory usage should be less than the free space available onboard.

$$\text{memory of (Final IDS model+software+library+temporary feature data)} \leq \text{onboard free space} \quad (2.23)$$

## Computational Cost

The computational cost of an RNN can be calculated by the number of floating-point operations (FLOPs) [70]. Table 2.12 is the standard arithmetic operators with their FLOPs according to [71] (p. 116), which can be used to analyze the number of FLOPs required for each RNN model.

**Table 2.12.** FLOPs for standard arithmetic operators according to [71].

Arithmetic operation	Complexity
Addition (+)	1 FLOP
Subtraction (−)	1 FLOP
Multiplication ( $\times$ )	1 FLOP
Division ( $\div$ )	4 FLOPs
Exponential ( <i>exp</i> )	8 FLOPs

Where  $m$  is the input size and  $n$  is the hidden dimension, Tables 2.13 through 2.16 show the number of FLOPs for each standard arithmetic operation in a single forward pass of simple RNN, LSTM, GRU, and SRU.

**Table 2.13.** FLOPs for single simple RNN layer forward pass.

Arithmetic operation	FLOPs for simple RNN
Addition (+)	$n^2 + nm$ (without bias vector) $n^2 + nm + n$ (with bias vector) $n^2 + nm + 2n$ (PyTorch with two bias vector)
Subtraction (−)	$n$
Multiplication ( $\times$ )	$n^2 + nm + n$
Division ( $\div$ )	$4 \times n$
Exponential ( <i>exp</i> )	$8 \times n$
Total	$2n^2 + 2nm + 14n$ (without bias vector) $2n^2 + 2nm + 15n$ (with bias vector) $2n^2 + 2nm + 16n$ (PyTorch with two bias vectors)

**Table 2.14.** FLOPs for single LSTM layer forward pass.

Arithmetic operation	FLOPs for LSTM
Addition (+)	$4n^2 + 4nm + 2n$ (without bias vector) $4n^2 + 4nm + 6n$ (with bias vector) $4n^2 + 4nm + 10n$ (PyTorch with two bias vectors)
Subtraction (−)	$2n$
Multiplication ( $\times$ )	$4n^2 + 4nm + 7n$
Division ( $\div$ )	$4 \times 5n$
Exponential ( <i>exp</i> )	$8 \times 5n$
Total	$8n^2 + 8nm + 71n$ (without bias vector) $8n^2 + 8nm + 75n$ (with bias vector) $8n^2 + 8nm + 79n$ (PyTorch with two bias vectors)

**Table 2.15.** FLOPs for single GRU layer forward pass.

Arithmetic operation	FLOPs for GRU
Addition (+)	$3n^2 + 3nm$ (without bias vector) $3n^2 + 3nm + 3n$ (with bias vector) $3n^2 + 3nm + 6n$ (PyTorch with two bias vectors)
Subtraction (−)	$2n$
Multiplication ( $\times$ )	$3n^2 + 3nm + 6n$
Division ( $\div$ )	$4 \times 3n$
Exponential ( <i>exp</i> )	$8 \times 3n$
Total	$6n^2 + 6nm + 44n$ (without bias vector) $6n^2 + 6nm + 47n$ (with bias vector) $6n^2 + 6nm + 50n$ (PyTorch with two bias vectors)

**Table 2.16.** FLOPs for single SRU layer forward pass.

Arithmetic operation	FLOPs for SRU
Addition (+)	$3nm + 2n$ (without bias vector) $3nm + 4n$ (with bias vector)
Subtraction (−)	$2n$
Multiplication ( $\times$ )	$3nm + 8n$
Division ( $\div$ )	$4 \times 2n$
Exponential ( <i>exp</i> )	$8 \times 2n$
Total	$6nm + 36n$ (without bias vector) $6nm + 38n$ (with bias vector)

In one complete forward pass of the RNN model, each layer will pass the number of time steps of an instance. Therefore, we need to multiply FLOPs by the number of time steps before the number of layers. In addition, the dense layer that is last for the classification problem also takes  $(n + 1) \times \text{number of classes}$  FLOPs. The final equations calculating the

number of FLOPs for a complete forward pass of simple RNN, LSTM, GRU, and SRU IDSs in PyTorch with bias terms are shown in Equations (2.24) through (2.27).

*simple RNN total FLOPs =*

$$((2n^2 + 2nm + 16n) \times \text{number of timesteps}) \times \text{number of layers} + (n + 1) \times \text{number of classes} \quad (2.24)$$

*LSTM total FLOPs =*

$$((8n^2 + 8nm + 79n) \times \text{number of timesteps}) \times \text{number of layers} + (n + 1) \times \text{number of classes} \quad (2.25)$$

*GRU total FLOPs =*

$$((6n^2 + 6nm + 50n) \times \text{number of timesteps}) \times \text{number of layers} + (n + 1) \times \text{number of classes} \quad (2.26)$$

*SRU total FLOPs =*

$$((6nm + 38n) \times \text{number of timesteps}) \times \text{number of layers} + (n + 1) \times \text{number of classes} \quad (2.27)$$

In resources-constrained applications, it is essential to ensure that the RNN IDS model can run a complete forward pass with a specific frequency for time-based detection or within a specific time frame for event-based detection. For this study, we chose to measure FLOPS performance with the RNN IDS using the Arduino Uno, the lowest-cost companion computer that can be implemented with PX4 firmware and Pixhawk autopilots as the target system. The time required for Arduino Uno to complete specific arithmetic operations, according to

[72] and [73], are provided in Table 2.17. In addition, the FLOPS can be calculated with Equation (2.28) for each arithmetic operation, which are also shown in Table 2.17.

$FLOPS =$

$$1/(10^{-6} \times \text{nanoseconds per operations} \times \text{number of FLOPs for single arithmetic operation}) \quad (2.28)$$

**Table 2.17.** Time for floating point operations on Arduino Uno.

Arithmetic operation	Nanoseconds per operations	FLOPS
Addition (+)	9.09	110011.00
Subtraction (−)	9.19	108813.93
Multiplication (×)	9.69	103199.17
Division (÷)	30.82	129785.85

We chose to use an event-based IDS mechanism that requires a complete forward pass of IDS once the selected message arrives. Thus, selecting an appropriate message with enough of a time interval for the IDS to run a complete forward pass became critical. Equation (2.29) would need to be observed in order for the IDS to complete a complete forward pass; that is, deciding whether the attack happened within the selected message interval.

$$\frac{\text{FLOPs of one complete forward pass for IDS model}}{\text{FLOPS of selected computer performance}} \leq \text{Time interval of selected message} \quad (2.29)$$

According to Equation (2.27), the FLOPs required to run an SRU model complete forward pass with difference settings and the associated time required if Arduino Uno FLOPS equals 103199.17 is used are shown in Table 2.18. All settings in Table 2.18 have the number of timesteps set at 8, the number of layers at 1, and the number of classes at 3.



**Table 2.18.** FLOPs for SRU in various settings and required time for one complete forward pass.

Input size (number of features)	Hidden unit size	FLOPs	Time required for one forward pass (s)
128	64	412867	4.00
65	64	219331	2.13
32	16	29491	0.29

Table 2.19 shows the messages we chose and their associated time intervals. To be implementable onboard a UAV with Arduino Uno, the time required for one forward pass had to be less than the minimum time interval of the message, and we therefore selected the cpuload message to be the event. This meant that the IDS should run a complete forward pass and decide whether an attack happened once a new cpuload message arrived.

**Table 2.19.** Selected messages and associated time interval.

Message	Mean (s)	Standard deviation (s)	Max (s)	Min (s)
cpuload	0.5625	0.2085	1.5009	0.4989
estimator innovation test ratios	0.0112	0.0216	0.7612	0.0099
estimator innovation variances	0.0112	0.0241	0.7814	0.0050
estimator innovations	0.0112	0.0241	0.7814	0.0050
estimator sensor bias	0.0112	0.0241	0.7814	0.0050
estimator status	0.0112	0.0241	0.7814	0.0050
rate ctrl status	0.0039	0.0143	0.7702	0.0012
telemetry status	0.5745	0.3797	2.0020	0.0098
vehicle imu status	0.1132	0.0994	1.3023	0.1000
vehicle local position	0.0113	0.0244	0.7814	0.0050

### 3. EXPERIMENT CONFIGURATION

#### 3.1 UAV

The UAV platform selected for this study is the Holybro S500 quadrotor shown in Figure 3.1, which is one of the most commonly used UAV platforms. It is supported by the PX4 airframe [48] and operated by the PX4 autopilot system. Table 3.1 provides the specifications of the Holybro S500.



**Figure 3.1.** Holybro S500 UAV platform.

**Table 3.1.** Specifications of the selected UAV platform.

UAV model	Type	Span	Length	Wheelbase	Weight
Holybro S500	Quadrotor	15.07 in/383 mm	15.16 in/385 mm	18.90 in/48 mm	2.06 lb/935 g

#### 3.2 Autopilot

For the autopilot used in this study, the Holybro Pixhawk 4, shown in Figure 3.2, was selected because of its various supported ports and newest onboard processor.



**Figure 3.2.** Holybro Pixhawk 4.

The following are the specifications of the Holybro Pixhawk 4.

**Interface:**

- 8–16 pulse width modulation (PWM) servo outputs
- 3 dedicated PWM/capture inputs on the flight management unit (FMU)
- Dedicated radio-controlled (R/C) input for combined pulse position modulation (CPPM)
- Dedicated R/C input for Spektrum/digital system multiplexer (DSM) and serial bus (S.Bus) with analog/PWM received signal strength indication (RSSI) input
- Dedicated S.Bus output
- 5 general purpose serial ports
- 3 inter-integrated circuit (I2C) ports
- 4 serial peripheral interface (SPI) buses

- 2 controller area network (CAN) buses for dual CAN with serial ESC
- Analog input for voltage and current of 2 batteries

**Table 3.2.** Holybro Pixhawk 4 technical specifications.

<b>Main FMU Processor</b>	<b>IO Processor</b>	<b>Onboard Sensor</b>	<b>GPS</b>
STM32F765 <ul style="list-style-type: none"> <li>• 32 Bit Arm Cortex-M7</li> <li>• 216 MHz</li> <li>• 2 MB memory</li> <li>• 512 KB RAM</li> </ul>	STM32F100 <ul style="list-style-type: none"> <li>• 32 Bit Arm Cortex-M3</li> <li>• 24 MHz</li> <li>• 8 KB SRAM</li> </ul>	<ul style="list-style-type: none"> <li>• Accel/Gyro: ICM-20689</li> <li>• Accel/Gyro: BMI055</li> <li>• Mag: IST8310</li> <li>• Barometer: MS5611</li> </ul>	<ul style="list-style-type: none"> <li>• ublox Neo-M8N GPS/GLONASS receiver</li> <li>• Magnetometer IST8310</li> </ul>

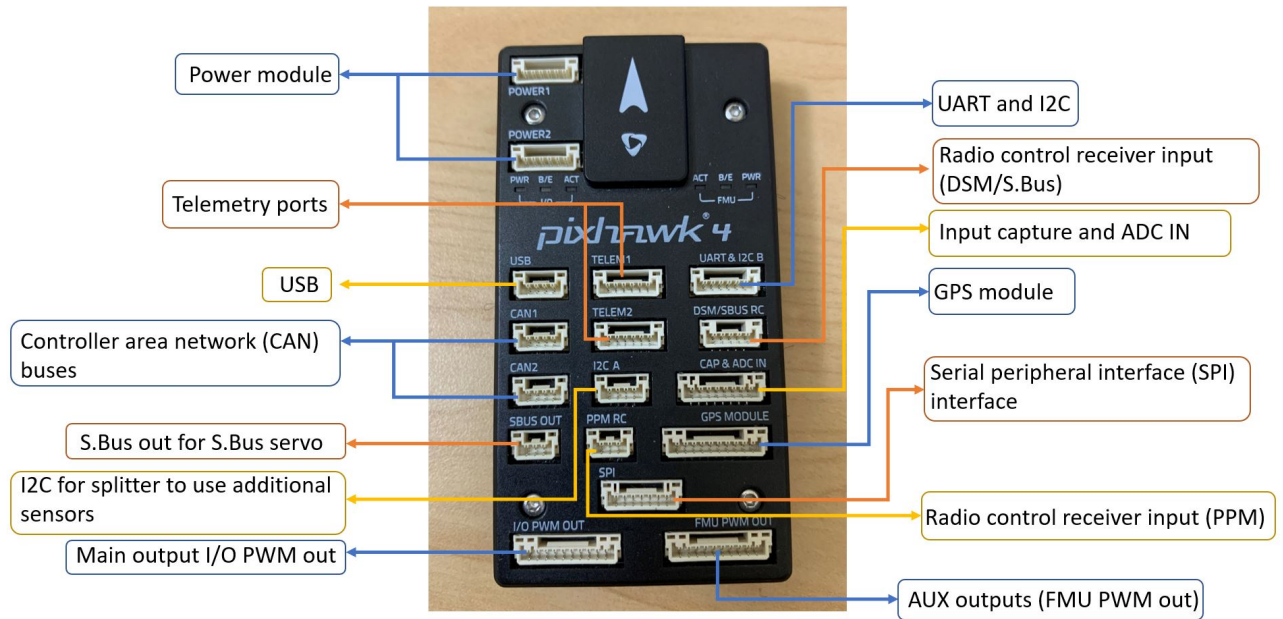
**Table 3.3.** Holybro Pixhawk 4 mechanical and environment specifications.

<b>Dimension</b>	<b>Weight</b>	<b>Operating temperature</b>	<b>Storage temperature</b>
44 x 84 x 12 mm	15.8 g	-40–85 °C	-40–85 °C

**Table 3.4.** Holybro Pixhawk 4 electrical specifications.

<b>Power module output</b>	<b>Max input voltage</b>	<b>Max current sensing</b>	<b>USB power input</b>	<b>Servo rail input</b>
4.9–5.5 V	6V	120A	4.75–5.25V	0–36V

Figure 3.3 shows all the connection ports on the Holybro Pixhawk 4.



**Figure 3.3.** Holybro Pixhawk 4 connection scheme.

Table 3.5 and Figure 3.4 provide details about the connections used in this work.

**Table 3.5.** Holybro Pixhawk 4 connection on Holybro S500 platform used in this study.

Port	Description
POWER1	Connection to 4S 4000mAh Battery.
TELEM1	Connection to ESP8266 Wi-Fi module for Wi-Fi telemetry.
DSM/SBUS RC	Connection FrSKY X8R radio receiver for RC control and emergency control takeover.
GPS module	Connection to GPS sensor for compass reading.
I/O PWM out	Connection to ESC and motors for control.

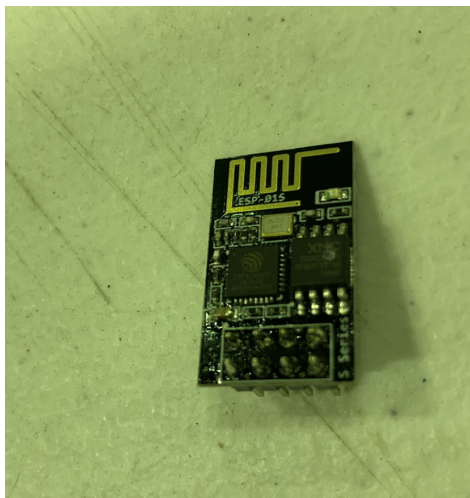


**Figure 3.4.** Holybro Pixhawk 4 connection on Holybro S500 platform used in this study.

### 3.3 Wi-Fi Module

The onboard Wi-Fi module used in this study was the ESP8266 [74], shown in Figure 3.5, which is a popular low-cost Wi-Fi chip supported by any Pixhawk series controller board [64]. Table 3.6 gives the specifications of the ESP8266.





**Figure 3.5.** ESP8266 Wi-Fi module

**Table 3.6.** Specifications of onboard Wi-Fi module.

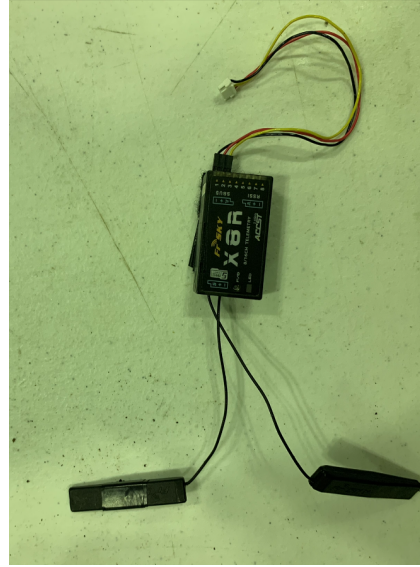
Wi-Fi Model	Type	Power	Length	Width	Weight
ESP8266	IEEE 802.11 b/g/n Wi-Fi	3.3 V	0.98 in/24.89 mm	0.59 in/14.99 mm	0.00625 lb/2.83 g

### 3.4 Radio

A radio transmitter is required for the position mode in the ICMP flooding and the emergency control takeovers in both attack scenarios. The radio transmitter we used was a FrSKY TARANIS, shown in Figure 3.6, and its associated receiver onboard the Holybro S500 was the FrSKY X8R, shown in Figure 3.7.



**Figure 3.6.** FrSKY TARANIS.



**Figure 3.7.** FrSKY X8R.

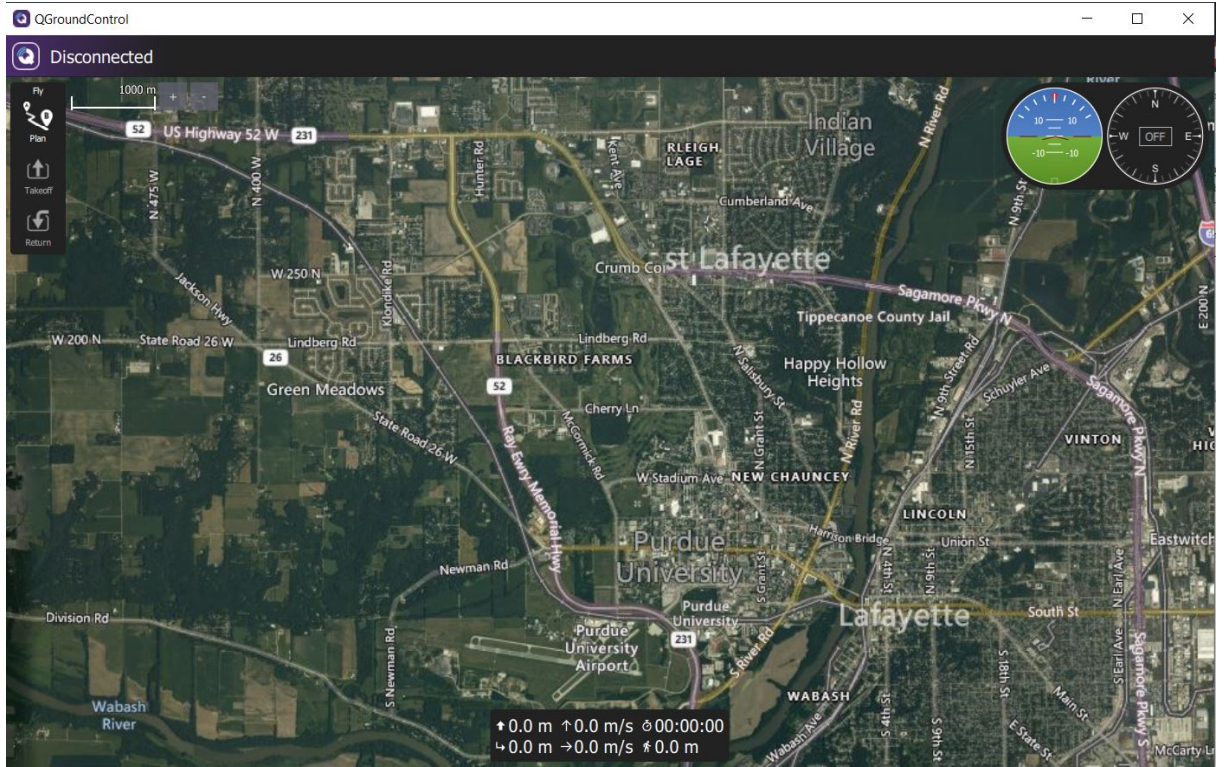
### 3.5 GCS

The following are the specifications of the computer that acted as the GCS.

- Model: Intel NUC
- Processor: Intel *Core* i7-7567U @ 3.5 GHz
- Graphics: Mesa Intel Iris Plus Graphics 650
- Memory (RAM): 31.3 GB
- Disk Capacity: 250.1 GB
- OS: Ubuntu 20.04.2 LTS 64 bit

QGroundControl, shown in Figure 3.8, was selected as the GCS application used to monitor the UAV's status during experiments. The offboard mode experiments were conducted using offboard Python scripts in a terminal window.





**Figure 3.8.** Screen shot of QGroundControl GCS.

### 3.6 Attacker

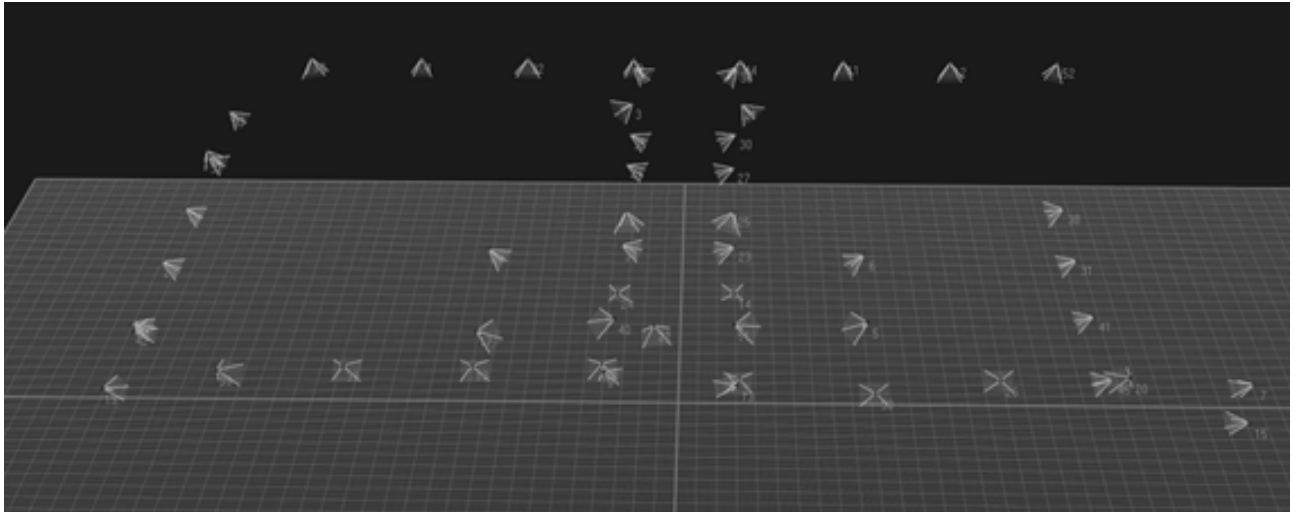
The following are the specifications of the computer that acted as the attacker.

- Model: Intel NUC
- Processor: Intel *Core* i7-7567U @ 3.5 GHz
- Graphics: Mesa Intel Iris Plus Graphics 650
- Memory (RAM): 31.2 GB
- Disk Capacity: 250.1 GB
- OS: Ubuntu 20.04.1 LTS 64 bit

The ICMP flooding attack tests ran a shell script that contained the `hping3` flooding command to launch the attack. The FDI false waypoint injection tests executed the malicious offboard Python script in a terminal window.

### 3.7 External Vision System

The external vision system was used in this work to facilitate indoor experiments. An external vision system can provide 6 DoF information on the UAV to the GCS and autopilot for the EKF estimator and UAV navigation. The experiments conducted in this study were completed in Purdue University's Indoor UAS Research and Test Facility (PURT) in order to collect actual experimental data. The PURT facility is equipped with 53 Qualisys motion capture (MoCap) cameras that can accurately track the position and orientation of UAVs with around 1.5 mm tracking error. Figure 3.9 shows the PURT Qualisys MoCap system setup.



**Figure 3.9.** Qualisys motion capture camera setup in Purdue University's Indoor UAS Research and Test (PURT) Facility; each apex of pyramid shape represents one motion capture camera.

The following are the specifications of the computer that ran the Qualisys motion capture system:

- Model: Qualisys QTM
- Processor: Intel *Core* i9-9900K @ 3.6 GHz
- Memory (RAM): 32 GB (31.8 GB usable)
- OS: Windows 10 Pro 64 bit

## 3.8 Experiment Sequence

### 3.8.1 DoS—ICMP Flooding

#### Hover

1. 10 seconds for takeoff and hovering at certain heights.
2. 10 seconds hover as the no-attack data.
3. Launch DoS attack for 10 seconds, as the attack data.
4. R/C control takeover and land.

#### Cruise

- Attack tests
  1. 15 seconds for takeoff, hovering at certain heights, and start cruising (at around 13 seconds).
  2. Launch DoS attack for 5 seconds as the attack data.
  3. R/C control takeover and land.
- Normal tests
  1. 10 seconds for takeoff and hovering at certain heights.
  2. 10 seconds for cruising as the no-attack data.
  3. Land.

### 3.8.2 FDI—False Waypoint Injection

We conducted the FDI-false waypoint injection tests with both low-frequency and high-frequency settings. The legitimate user offboard script data rate was set between 10 Hz and 50 Hz. In a low-frequency setting, the ratio of attacker data rate to legitimate user data rate was below 5. The low-frequency tests aimed to delay the mission or cause a collision. In addition, random setpoints and fixed setpoint were used in the low-frequency settings.

Random setpoints injects a random setpoint in every command packet sent, while fixed setpoint set an attacker-desired stationary point target within the complete attack sequence. Conversely, the high-frequency setting involved attempting to land the drone in the attacker's desired location. The attacker data rate was greater than 5 times the legitimate user data rate, as detailed in Figure 2.4 and Table 2.6.

## Hover

1. 10 seconds to takeoff and hovering at certain heights.
2. 10 seconds hovering as no-attack data.
3. While the original offboard script was running hover command, FDI attack was launched as attack data.
  - High frequency: 10 seconds to the attacker-defined waypoint, then 5s to force the UAV to land and disarm.
  - Low frequency with random setpoints: 15 seconds of random waypoints was given to the UAV.
  - Low frequency with a fixed setpoint: 10 seconds to the attacker-defined waypoint, then 5 seconds to force the UAV to land and disarm.
4. Landing.
  - High frequency: If not disarm by the attacker, 5 seconds to original waypoint, hovering, and then 10 seconds to land.
  - Low frequency with random setpoints: 5 seconds to original waypoint, hovering, and then 10 seconds to land.
  - Low frequency with a fixed setpoint: If not disarm by the attacker, 5 seconds to original waypoint, hovering, and then 10 seconds to land.

## Cruise

- Attack tests:
  1. 10 seconds to takeoff and hovering at certain heights.
  2. While the original offboard script was running cruise, the FDI attack launched as attack data.
    - High frequency: 10 seconds to attacker-defined waypoint, and then 5 seconds to force the UAV to land and disarm.
    - Low frequency with random setpoints: 10 seconds of random waypoints was given to the UAV.
    - Low frequency with a fixed setpoint: 10 seconds to attacker-defined waypoint, and then 5 seconds to force the UAV to land and disarm.
  3. Landing.
    - High frequency: If not disarm by the attacker, takeover by the original script and 5 seconds to land where original script targeted.
    - Low frequency with random setpoints: 10 seconds to land.
    - Low frequency with a fixed setpoint: If not disarm by the attacker, takeover by the original script and 5 seconds to land where original script targeted.
- Normal tests.
  1. 10 seconds to takeoff and hovering at certain heights.
  2. 20 seconds for cruising as no-attack data.
  3. Land.

## 4. ANALYSIS

To better demonstrate our experiments, we routed the real-time Qualisys MoCap system information to the gazebo Abu Dhabi simulation environment. This allowed us to illustrate the effects of the DoS and FDI attacks by showing the figures under these two environments with the cell phone video recordings and PX4 log odometry data visualized by MATLAB.

### 4.1 DoS—ICMP Flooding

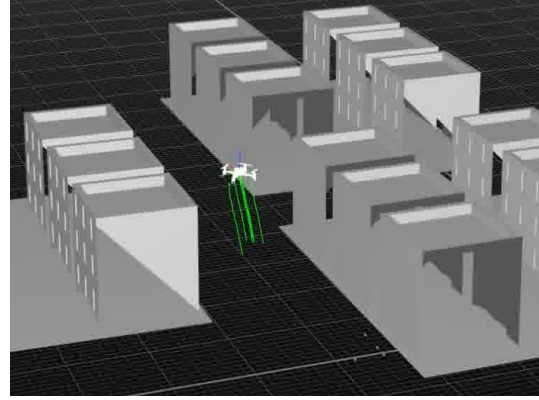
#### 4.1.1 Normal—No Attack

Figures 4.1 and 4.2 show the hover test under no attack. We held the Holybro S500 in a 3D location for the hover mission via the position mode. Figures 4.3 and 4.4 show the cruise test under no attack. The cruise mission for the UAV involved cruising straight at a constant speed.

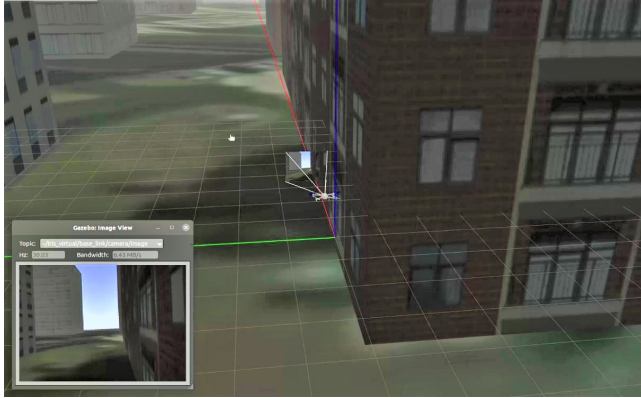




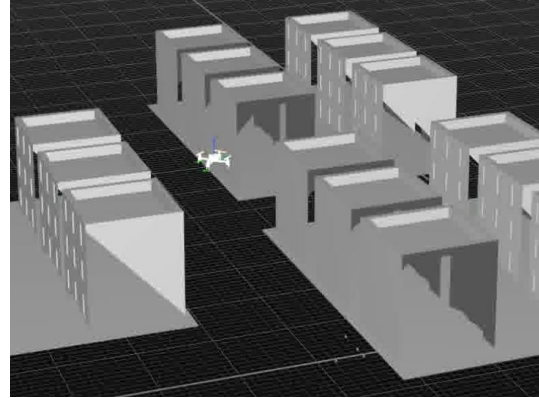
(a) Takeoff.



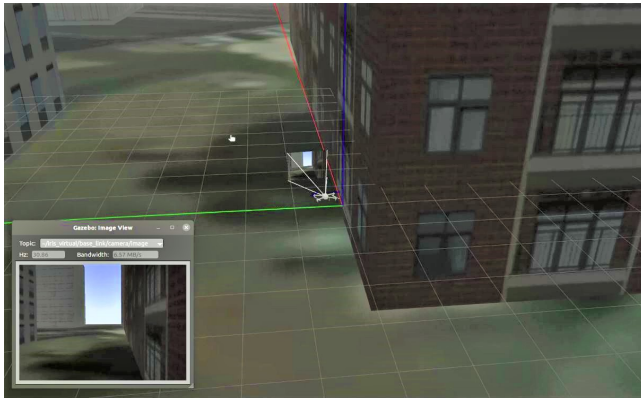
(b) Takeoff.



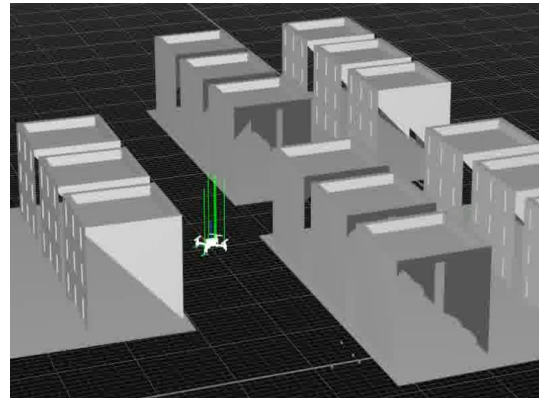
(c) Hover.



(d) Hover.

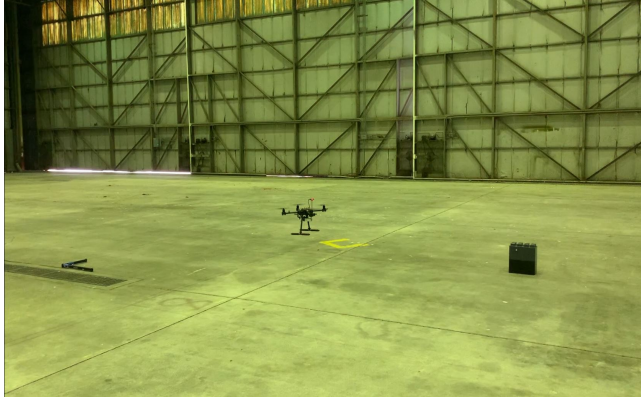


(e) Landing.

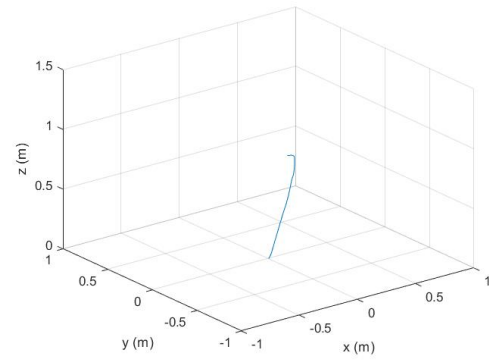


(f) Landing.

**Figure 4.1.** Hover in no-attack scenario: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



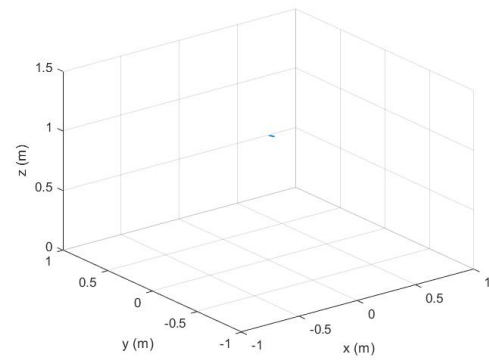
(a) Takeoff.



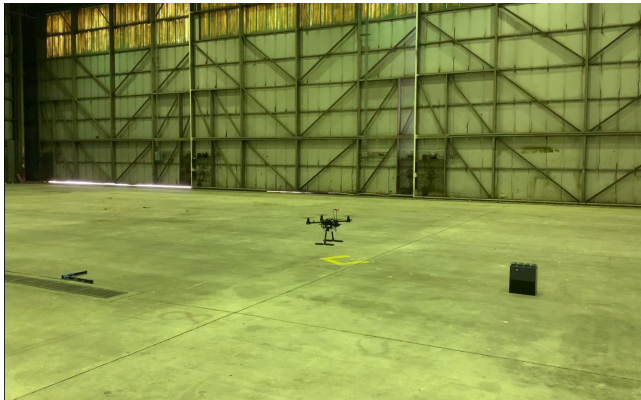
(b) Takeoff.



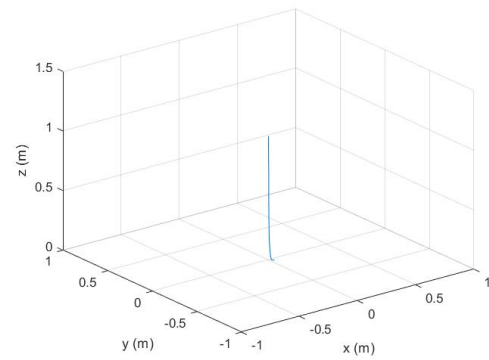
(c) Hover.



(d) Hover.



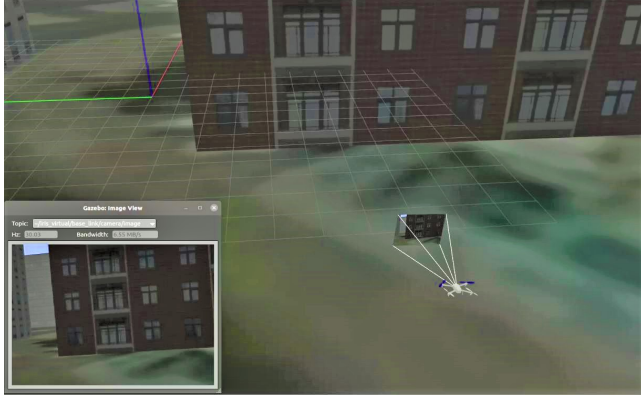
(e) Landing.



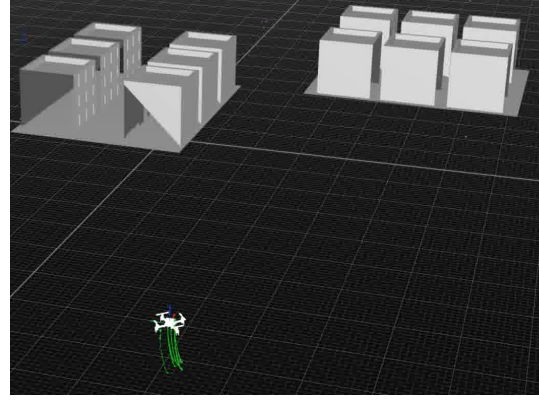
(f) Landing.

**Figure 4.2.** Hover in no attack scenario: (Left) Cell phone record; (right) MATLAB.

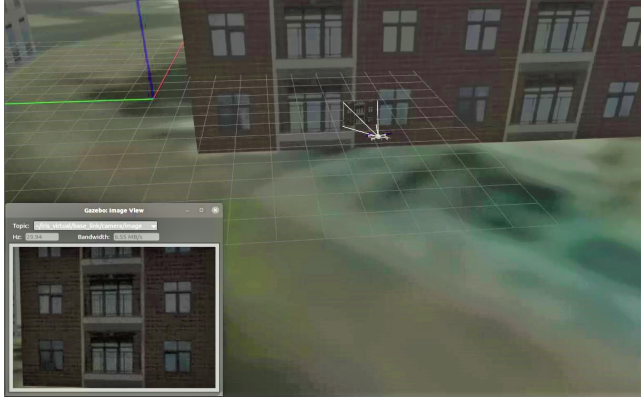




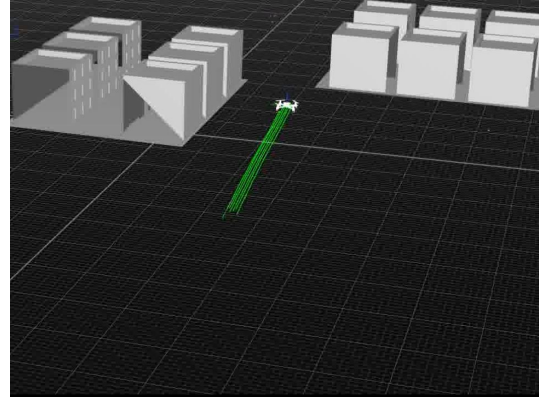
(a) Takeoff.



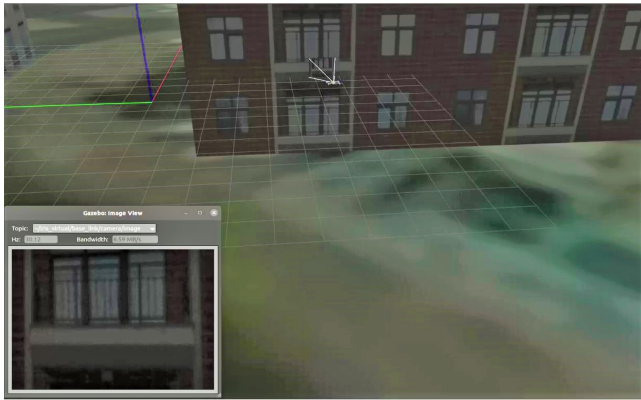
(b) Takeoff.



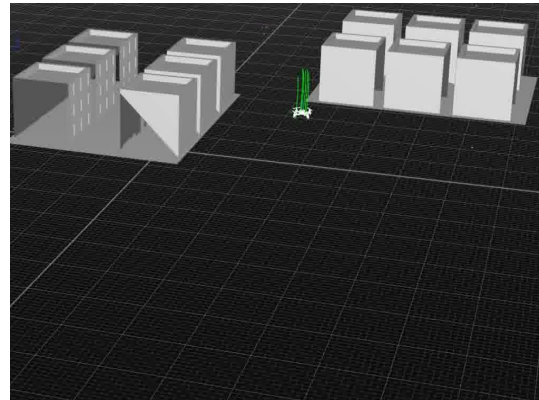
(c) Cruise.



(d) Cruise.



(e) Landing.

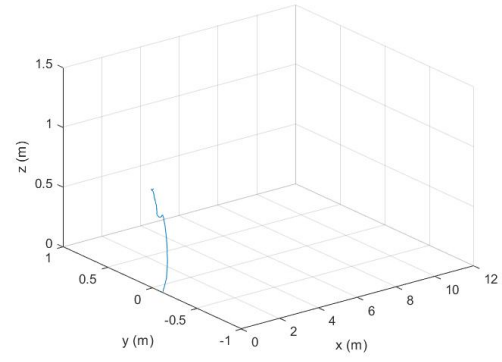


(f) Landing.

**Figure 4.3.** Cruise in no-attack scenario: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



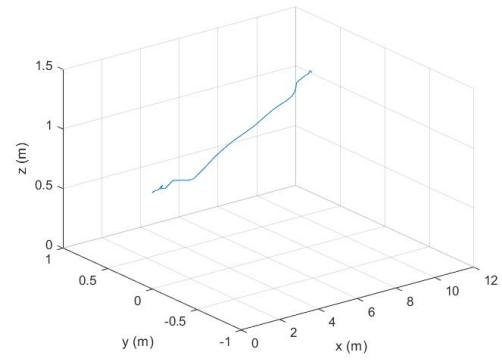
(a) Takeoff.



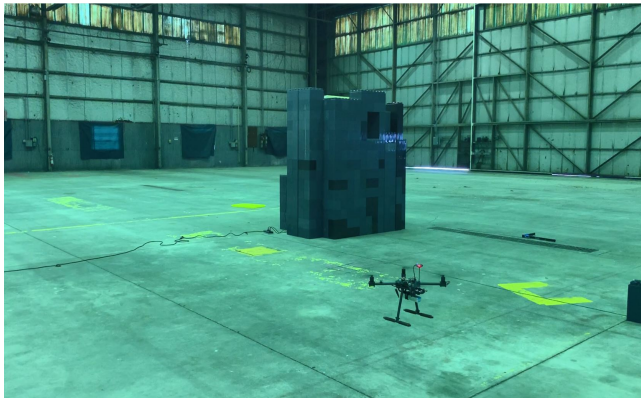
(b) Takeoff.



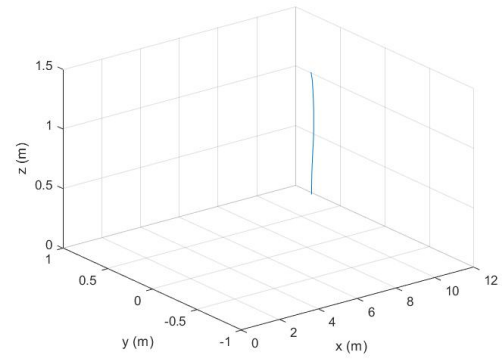
(c) Cruise.



(d) Cruise.



(e) Landing.



(f) Landing.

**Figure 4.4.** Cruise in no-attack scenario: (Left) Cell phone record; (right) MATLAB.

### 4.1.2 Hover

Figures 4.5 and 4.6 demonstrate the hover mission under 10 seconds of the ICMP flooding attack. To show the critical reactions of the UAV under a longer attack, Figures 4.7 and 4.8 demonstrate the hover mission under 30 seconds of the ICMP flooding attack. It should be noted that it was not safe to perform an ICMP flooding attack for longer than 30 seconds in the PURT indoor environment because the UAV could crash into the boundary of the test field.

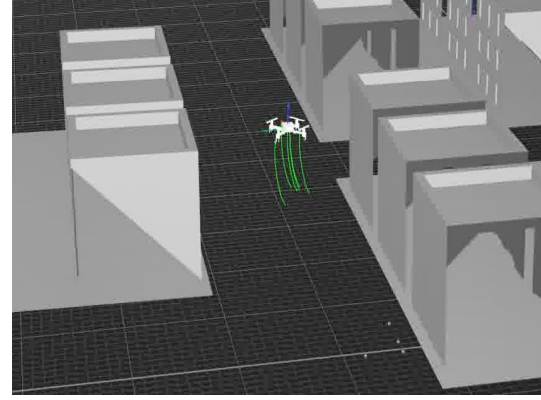
The trajectories shown in Figure 4.5d and Figure 4.6d for 10 second ICMP flooding attack and in Figure 4.7f and Figure 4.8f for 30 second ICMP flooding attack reveal large deviations from the hover location. Figures 4.5d and 4.6d show the deviations when the UAV should have been hovering at origin. Figures 4.7e and 4.7f show the UAV crashing into the simulated virtual building in the gazebo and the Qualisys MoCap system under 30 seconds of the ICMP flooding attack.

Figures 4.9 and 4.10 present the comparison of trajectories under no attack and in the two attack cases. The deviations from the hover location were caused by unavailable, delayed or slow transmitted external vision system information, such as odometry messages. Because of the ICMP flooding attack on the WiFi module, the odometry messages being transmitted through the Wi-Fi channel began to send at a very low frequency or became unavailable. As the EKF2 continuously either received the odometry messages at a low frequency or received them only after a delay of a couple of seconds, the UAV position estimation became unstable, and the UAV started drifting.





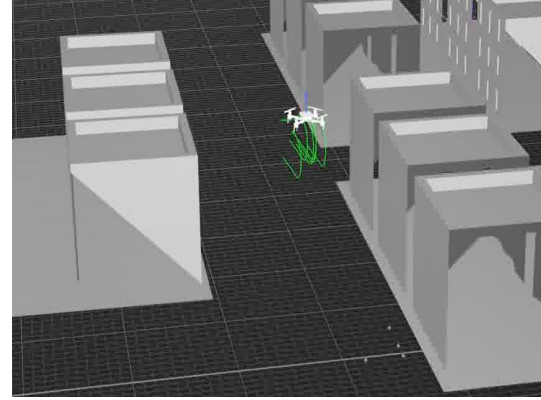
(a) Takeoff.



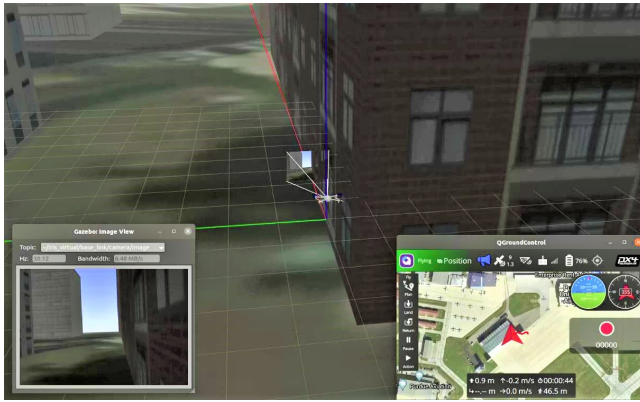
(b) Takeoff.



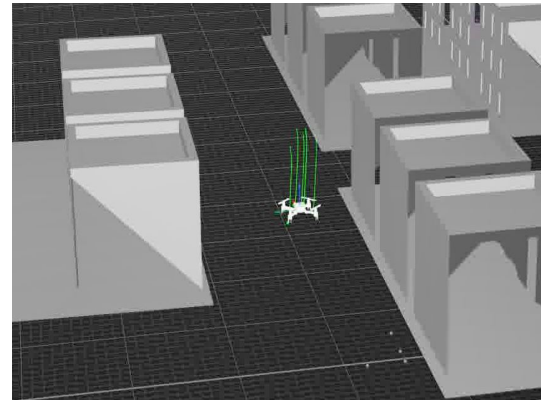
(c) Hover under 10 seconds of ICMP flooding at-tack



(d) Hover under 10 seconds of ICMP flooding at-tack

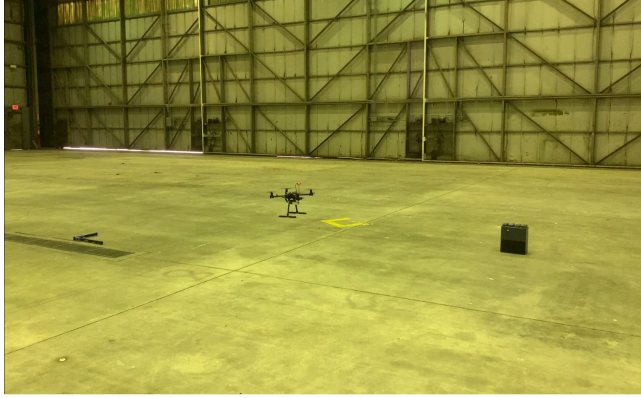


(e) Landing.

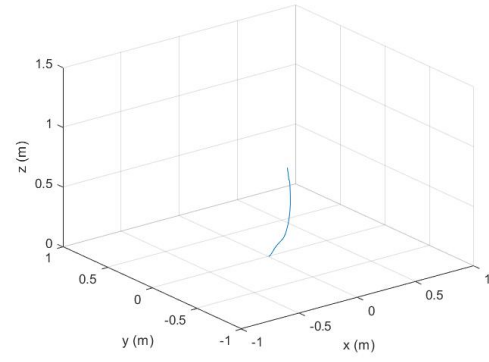


(f) Landing.

**Figure 4.5.** Hover under 10 seconds of ICMP flooding attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



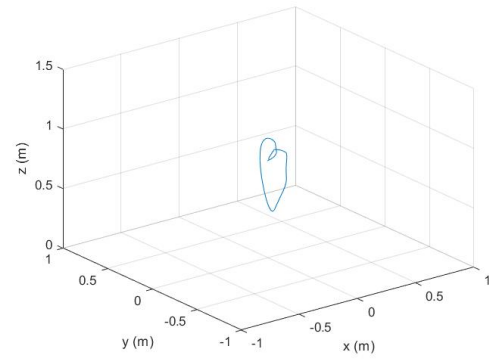
(a) Takeoff.



(b) Takeoff.



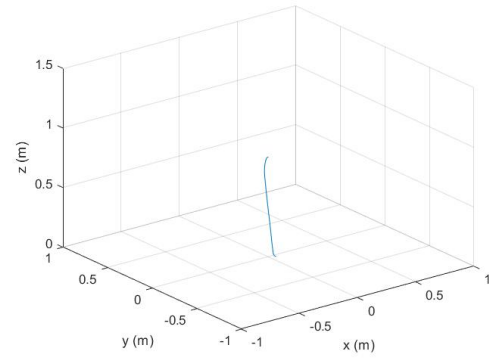
(c) Hover under 10 seconds of ICMP flooding at-tack.



(d) Hover under 10 seconds of ICMP flooding at-tack.



(e) Landing.



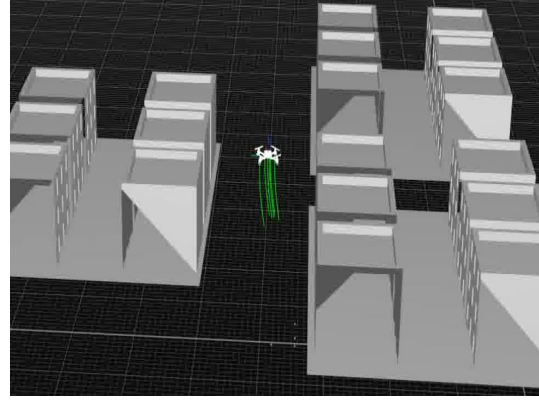
(f) Landing.

**Figure 4.6.** Hover under 10 seconds of ICMP flooding attack: (Left) Cell phone record; (right) MATLAB.





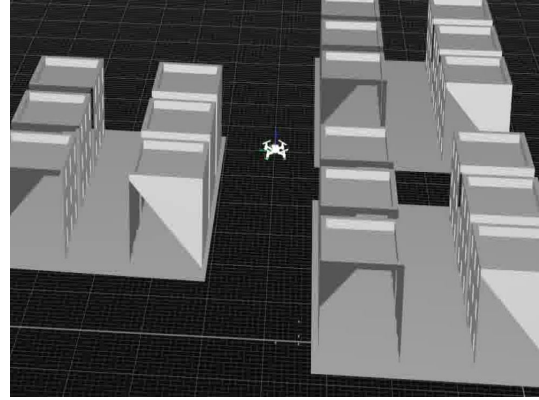
(a) Takeoff.



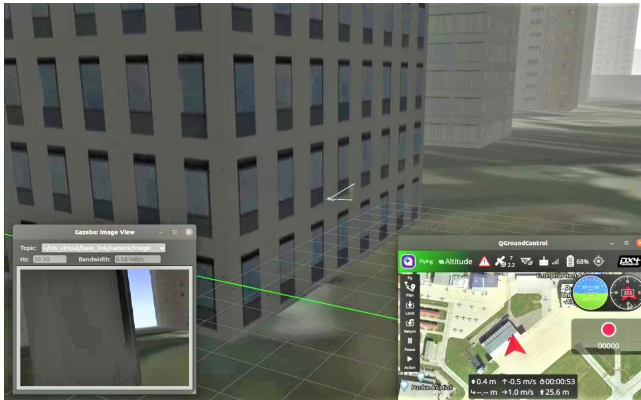
(b) Takeoff.



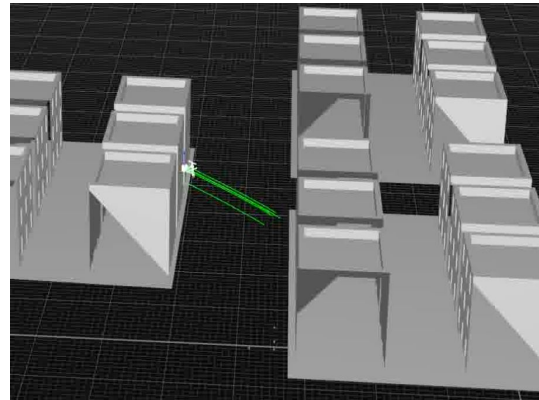
(c) Hover.



(d) Hover.

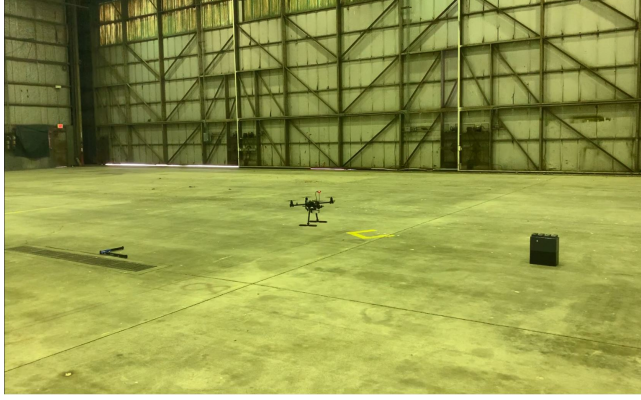


(e) Hover under 30 seconds of ICMP flooding attack.

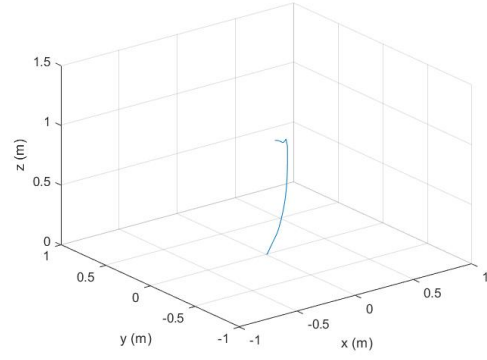


(f) Hover under 30 seconds of ICMP flooding attack.

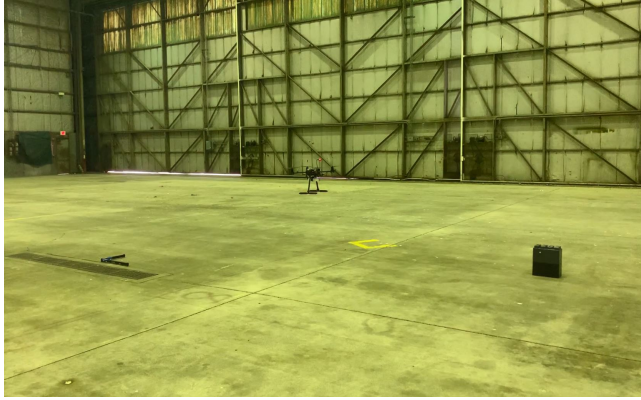
**Figure 4.7.** Hover under 30 seconds of ICMP flooding attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



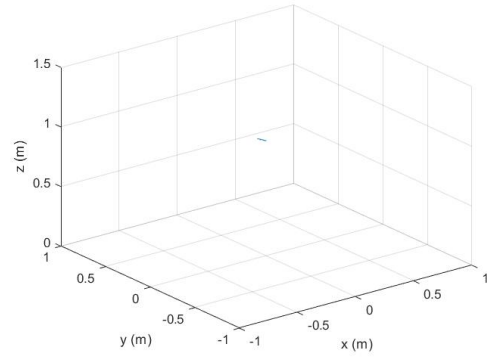
(a) Takeoff.



(b) Takeoff.



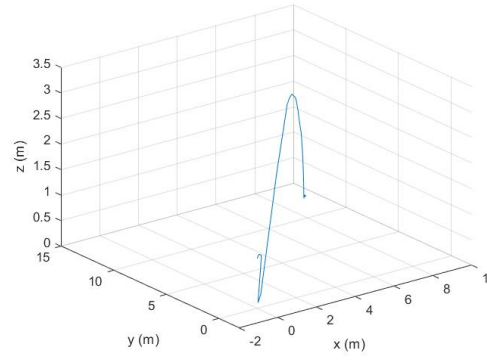
(c) Hover.



(d) Hover.

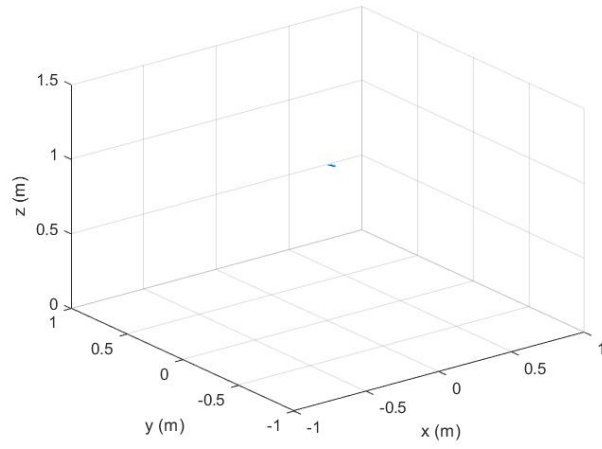


(e) Hover under 30 second of ICMP flooding attack.

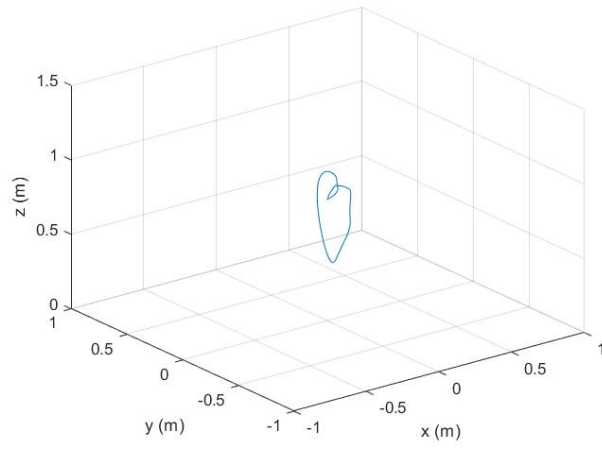


(f) Hover under 30 second of ICMP flooding attack.

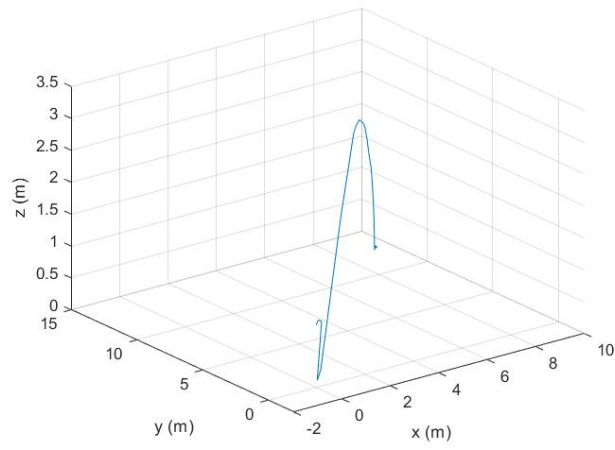
**Figure 4.8.** Hover under 30 seconds of ICMP flooding attack: (Left) Cell phone record; (right) MATLAB.



(a) No attack.



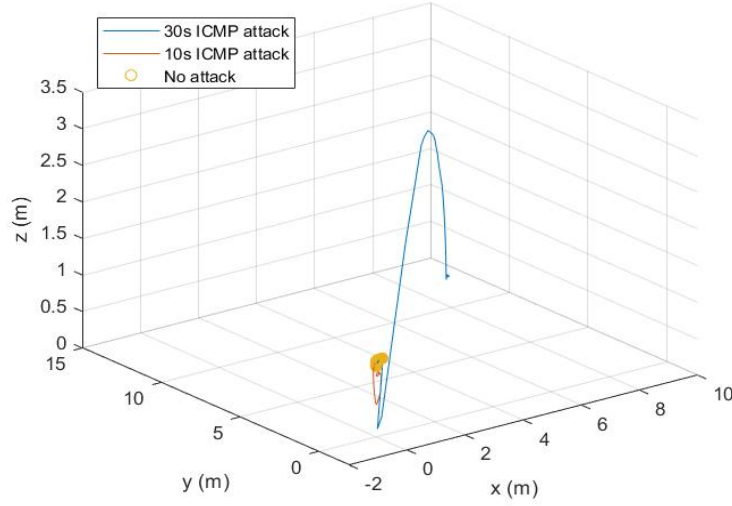
(b) Hover under 10 second of ICMP flooding attack.



(c) Hover under 30 second of ICMP flooding attack.

**Figure 4.9.** Hover trajectory under: (Top) no-attack scenario; (middle) 10 seconds of ICMP flooding attack; (bottom) 30 seconds of ICMP flooding attack.





**Figure 4.10.** Hover trajectories under no attack, 10 seconds of ICMP flooding attack, and 30 seconds of ICMP flooding attack.

### 4.1.3 Cruise

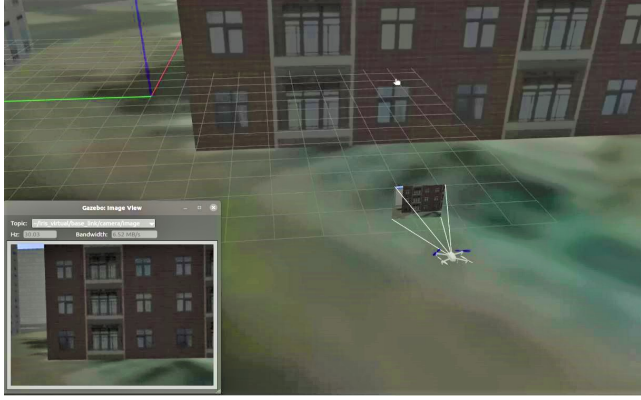
Figures 4.11 and 4.12 demonstrate the cruise mission under 5 seconds of the ICMP flooding attack. It should be noted that it was unsafe to perform the ICMP flooding attack for a cruise mission longer than 5 seconds in the PURT indoor environment because the UAV could crash into the boundary of the test field.

The trajectories shown in Figures 4.11d and 4.12d are the first reaction of the UAV under the ICMP flooding attack during the cruise mission. We observed that the UAV pitched forward and increased speed within 1 to 2 seconds of the attack being launched. This behavior is caused by unavailable or delayed odometry messages in the Wi-Fi channel. When the UAV did not receive the odometry messages, it was assumed that the current position was the prior one. Therefore, the UAV increases speed to reach what was believed to be the expected location.

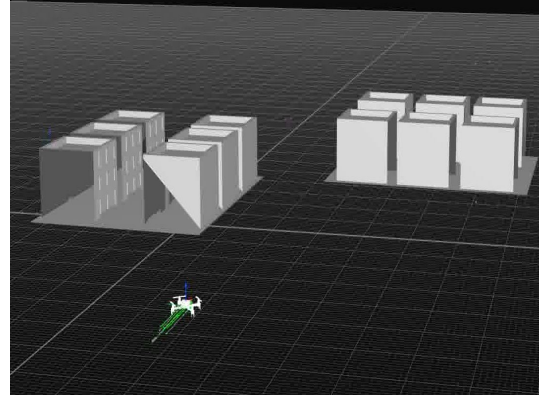
The trajectories shown in Figures 4.11f and 4.12f are the later reaction of the UAV under the ICMP flooding attack during the cruise mission. We observed that the UAV began drifting from its original course. Similar to the hover case, the delay or unavailable

odometry messages caused position estimation to be unstable, and therefore, the UAV began drifting.

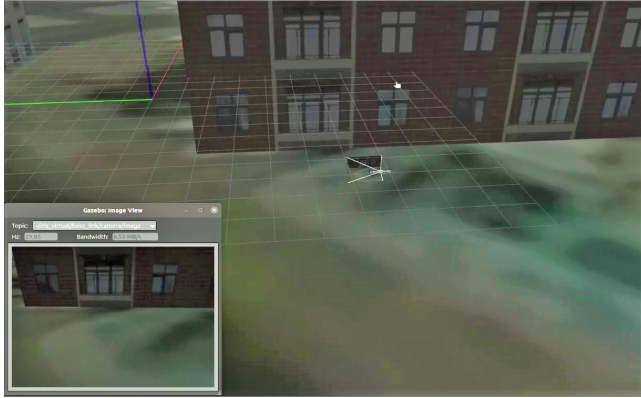
Figure 4.14 presents a comparison of the complete trajectories in the cases of no attack and the ICMP flooding attacks. A substantial difference can be seen.



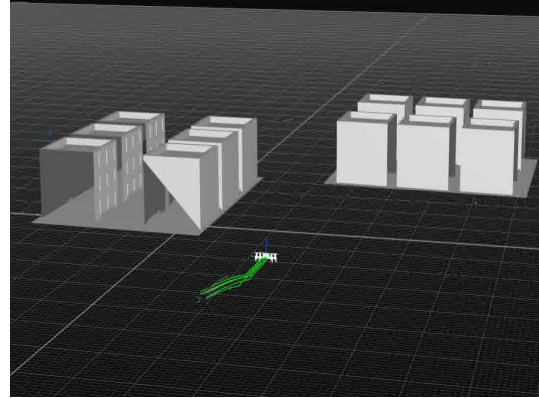
(a) Takeoff and start cruising.



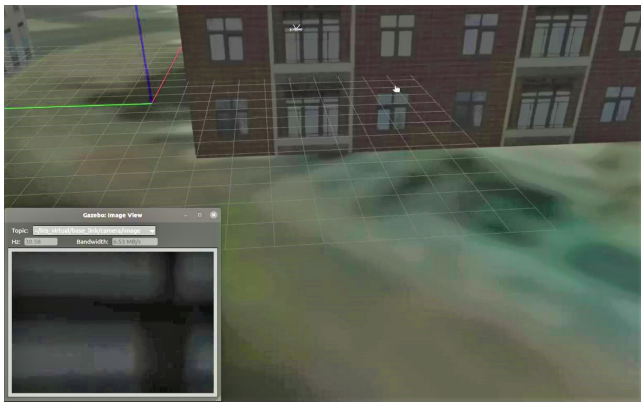
(b) Takeoff and start cruising.



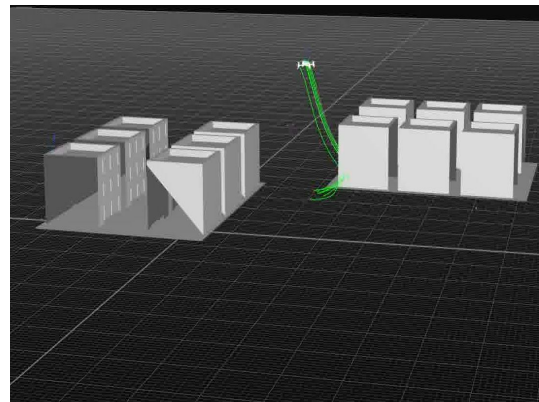
(c) Cruise under 5 seconds of ICMP flooding at-tack.



(d) Cruise under 5 seconds of ICMP flooding at-tack.



(e) Cruise under 5 seconds of ICMP flooding at-tack.

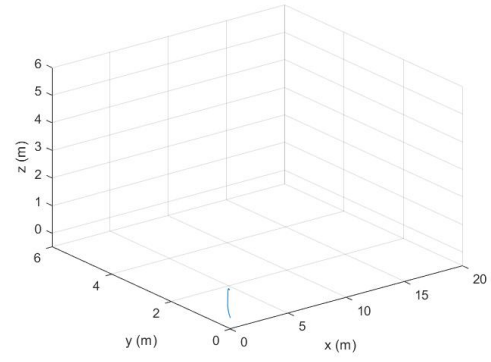


(f) Cruise under 5 seconds of ICMP flooding at-tack.

**Figure 4.11.** Cruise under 5 seconds of ICMP flooding attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



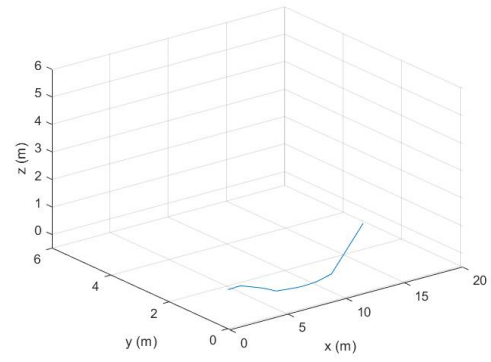
(a) Takeoff and start cruising.



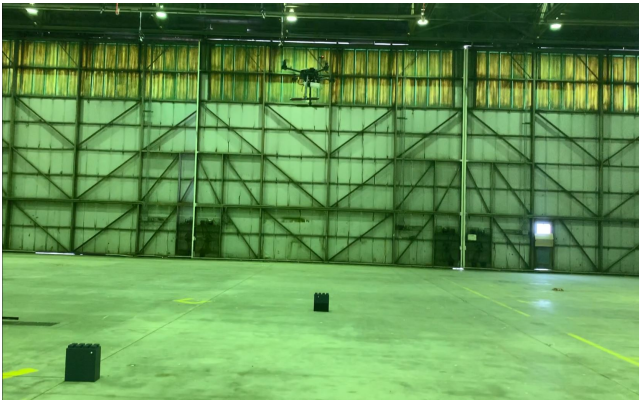
(b) Takeoff and start cruising.



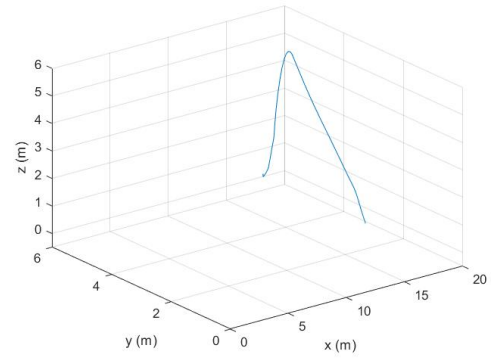
(c) Cruise under 5 seconds of ICMP flooding at-tack.



(d) Cruise under 5 seconds of ICMP flooding at-tack.

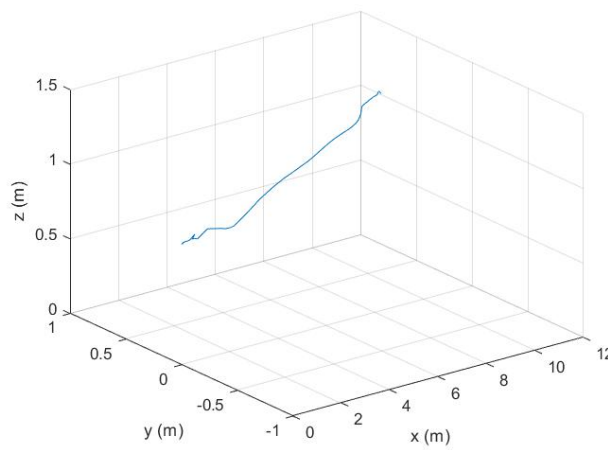


(e) Cruise under 5 seconds of ICMP flooding at-tack.

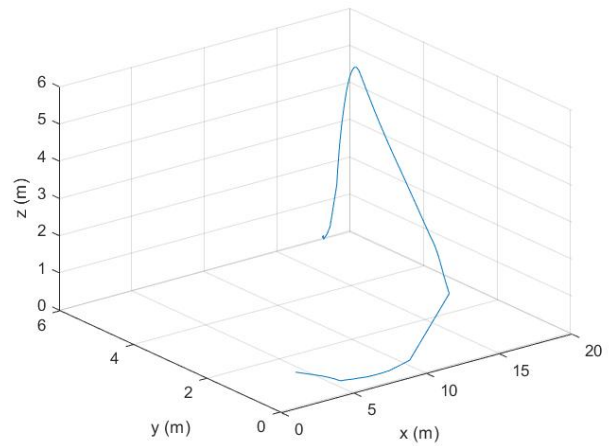


(f) Cruise under 5 seconds of ICMP flooding at-tack.

**Figure 4.12.** Cruise under 5 seconds of ICMP flooding attack: (Left) Cell phone record; (right) MATLAB.

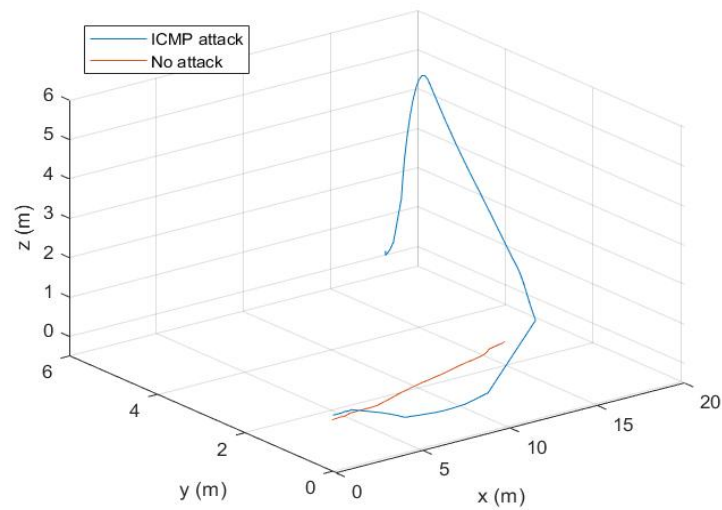


(a) No attack.



(b) Cruise under 5 seconds of ICMP flooding attack.

**Figure 4.13.** (Left) Cruise trajectory for no-attack scenario; (right) cruise trajectory under 5 seconds of ICMP flooding attack.



**Figure 4.14.** Cruise trajectory under no attack and 5 seconds of ICMP flooding attack.

## 4.2 FDI—False Waypoint Injection

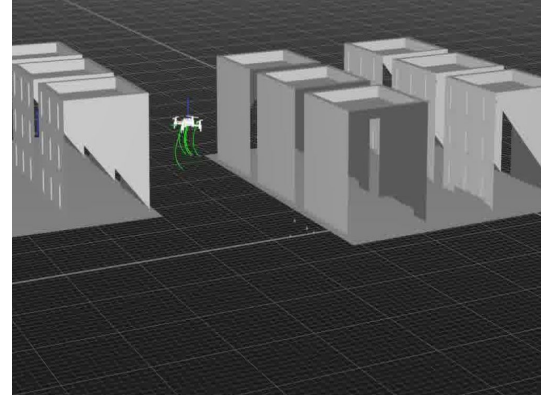
### 4.2.1 Normal—No attack

Figures 4.15 and 4.16 show the hover test under no attack. We held the Holybro S500 at the origin at a height equal to 0.7 meters. Figures 4.17 and 4.18 show the cruise test under no attack. The cruise mission for the UAV involved cruising straight from the origin at a height equal to 0.7 meters to the point where  $x$ ,  $y$ , and  $z$  equaled 8, 0, and 0.7 meters, respectively.





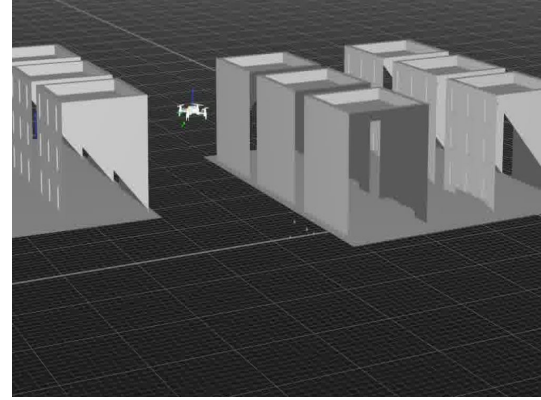
(a) Takeoff.



(b) Takeoff.



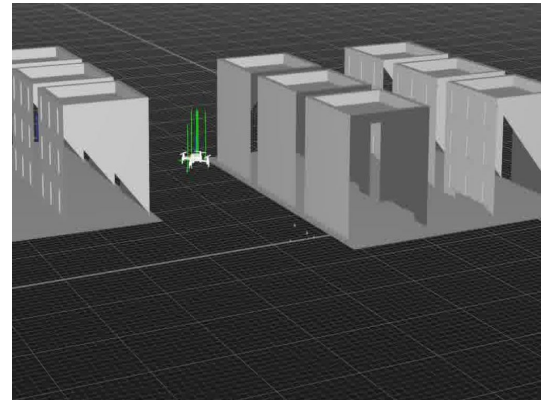
(c) Hover.



(d) Hover.



(e) Landing.

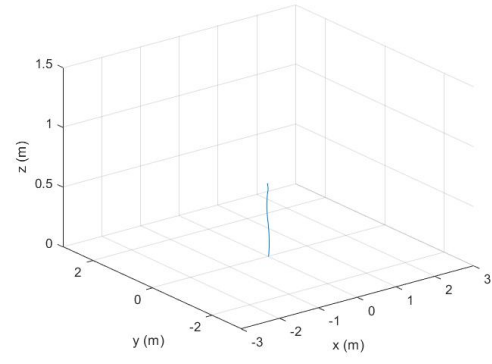


(f) Landing.

**Figure 4.15.** Hover in no-attack scenario: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



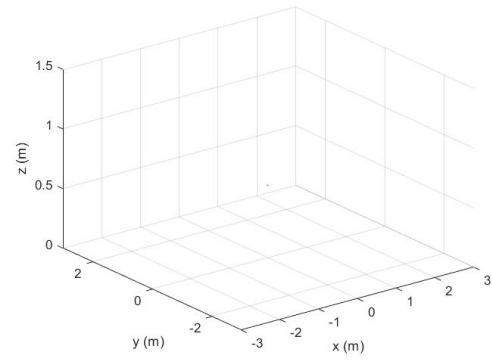
(a) Takeoff.



(b) Takeoff.



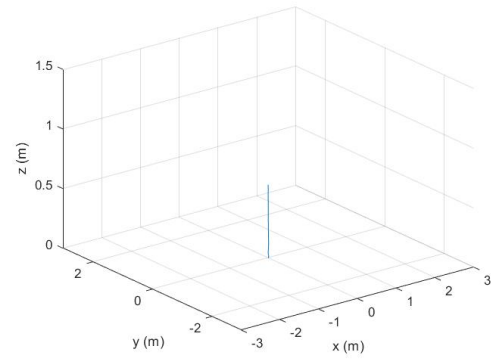
(c) Hover.



(d) Hover.



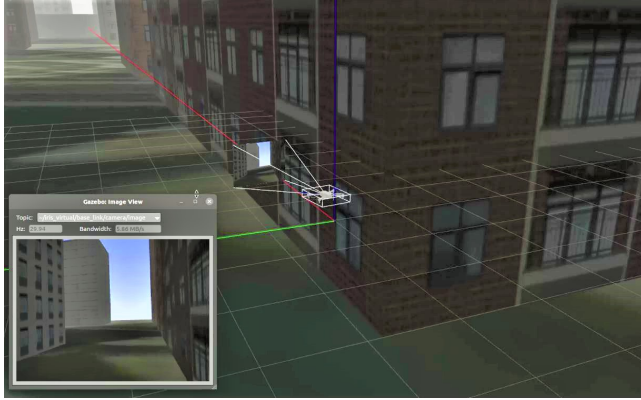
(e) Landing.



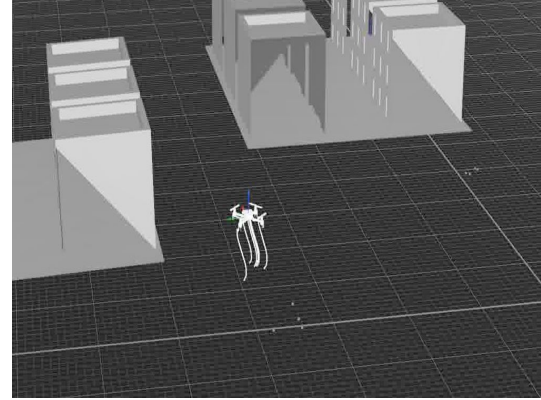
(f) Landing.

**Figure 4.16.** Hover in no-attack scenario: (Left) Cell phone record; (right) MATLAB.

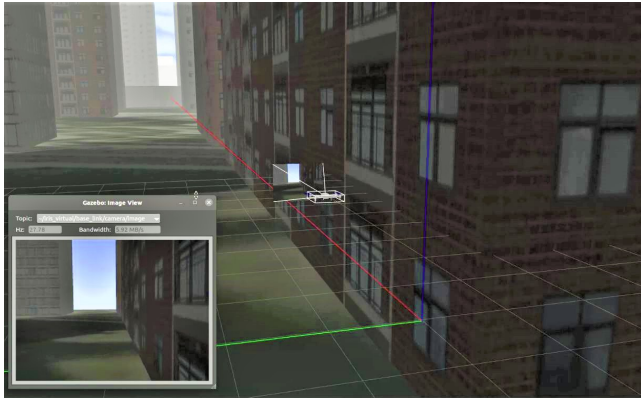




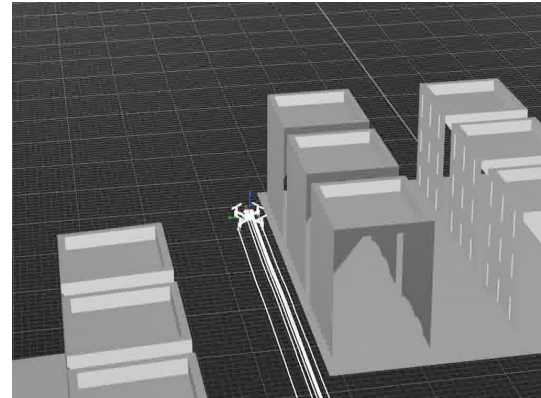
(a) Takeoff.



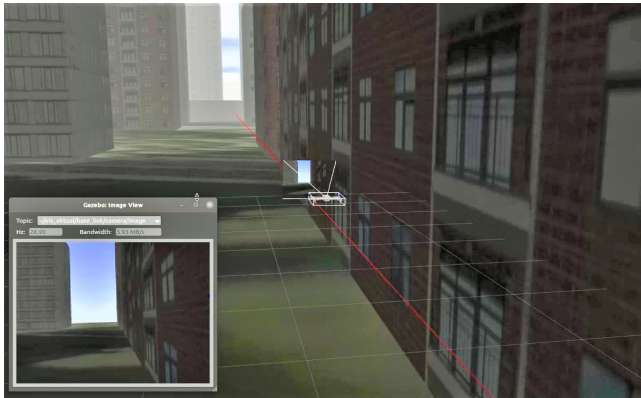
(b) Takeoff.



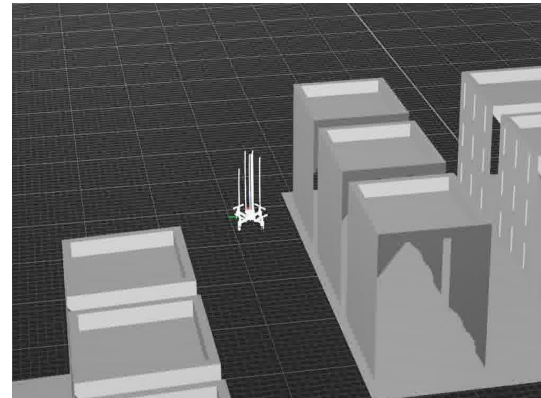
(c) Cruise.



(d) Cruise.



(e) Landing.

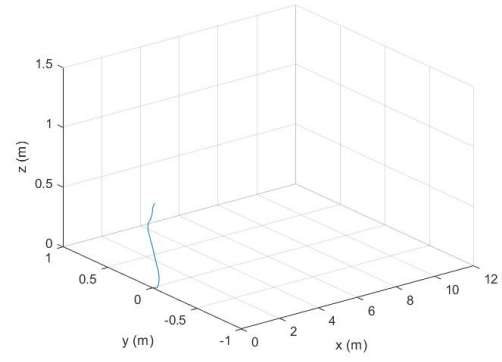


(f) Landing.

**Figure 4.17.** Cruise in no-attack scenario: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



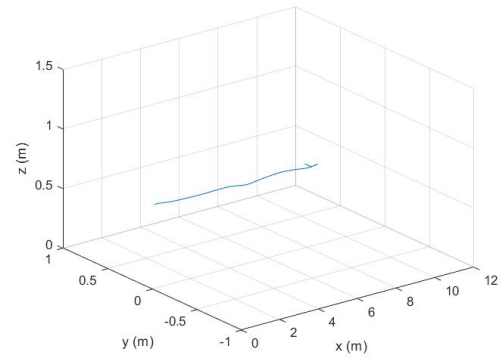
(a) Takeoff.



(b) Takeoff.



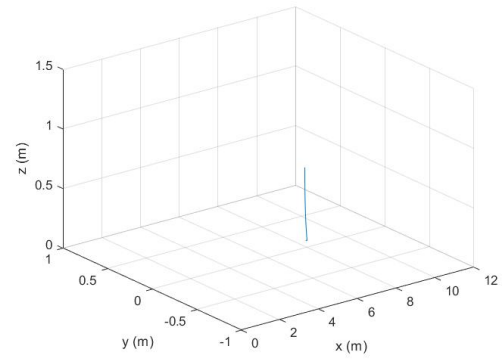
(c) Cruise.



(d) Cruise.



(e) Landing.



(f) Landing.

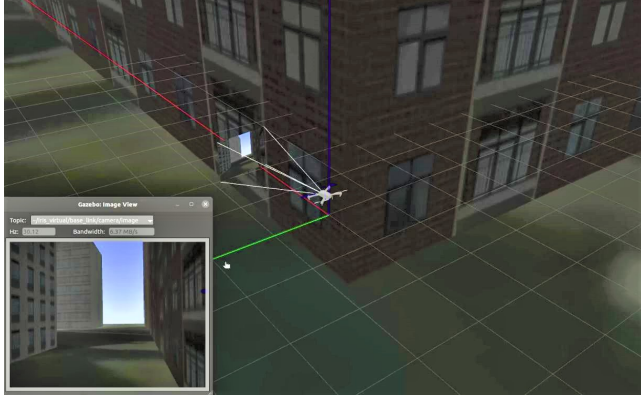
**Figure 4.18.** Cruise in no-attack scenario: (Left) Cell phone record; (Right) MATLAB.

### 4.2.2 Low Frequency

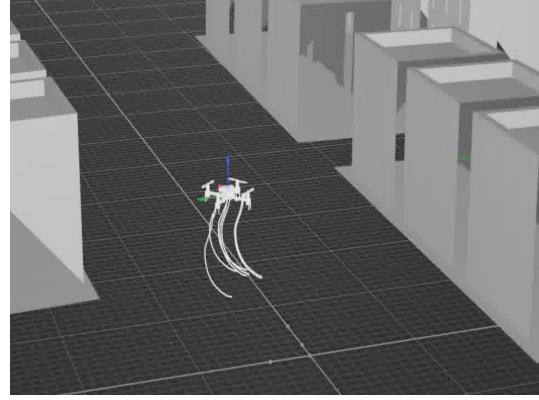
#### Random Waypoints Injection

Figures 4.19 and 4.20 show the hover mission under the low-frequency random setpoints FDI attack. The attacker injected random setpoints within  $[-2\text{m}, 2\text{m}]$  in  $x$  and  $y$  and  $[0.5\text{m}, 1.5\text{m}]$  in  $z$  to delay or cause the UAV to drift away from the hover location. Figure 4.21 compares the hover trajectories under no attack and under the low-frequency random setpoints FDI attack. It was observed that, during the attack, the UAV drifted about  $0.25\text{m}$  from the hover location and showed unstable poses. The drift distance from the origin increased if the attacker injected more distant waypoints and had extended attack time.

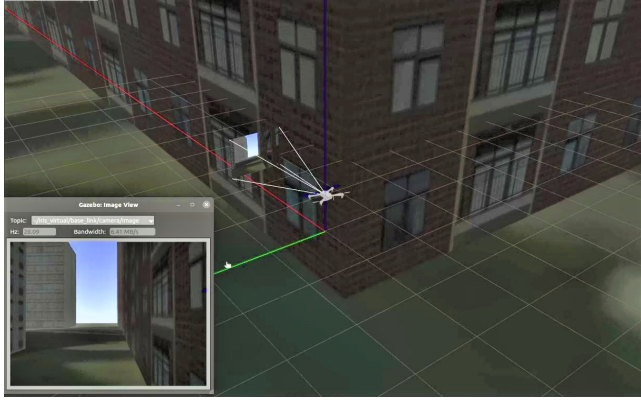




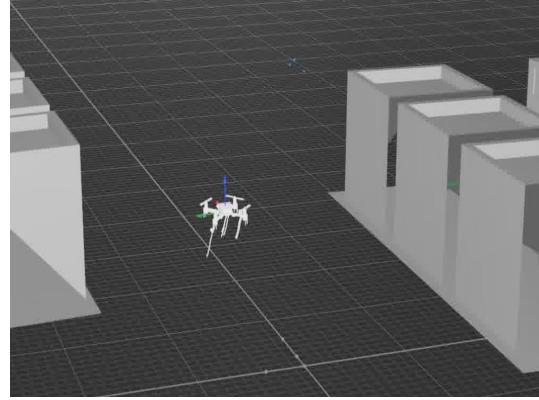
(a) Takeoff.



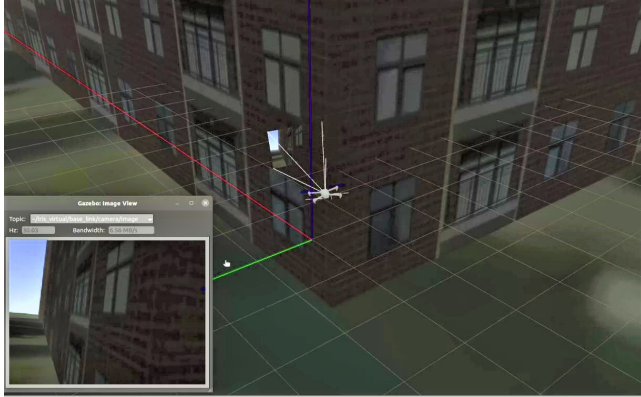
(b) Takeoff.



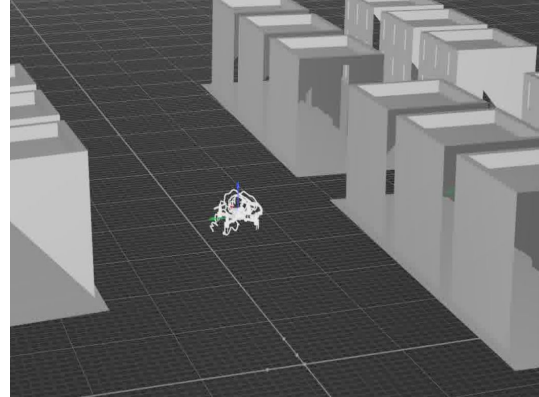
(c) Hover.



(d) Hover.



(e) Hover under the low-frequency random setpoints FDI attack.

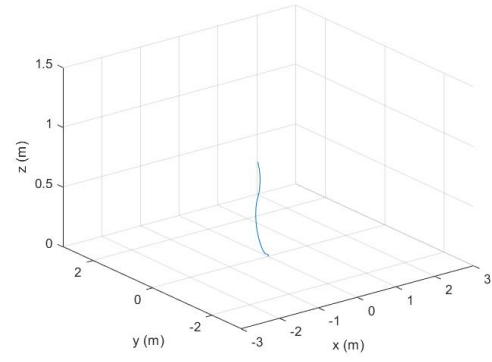


(f) Hover under the low-frequency random setpoints FDI attack.

**Figure 4.19.** Hover under the low-frequency random setpoints FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



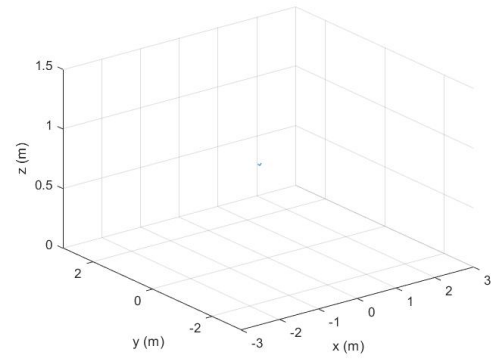
(a) Takeoff.



(b) Takeoff.



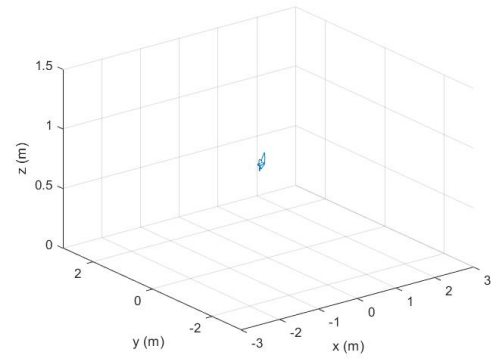
(c) Hover.



(d) Hover.

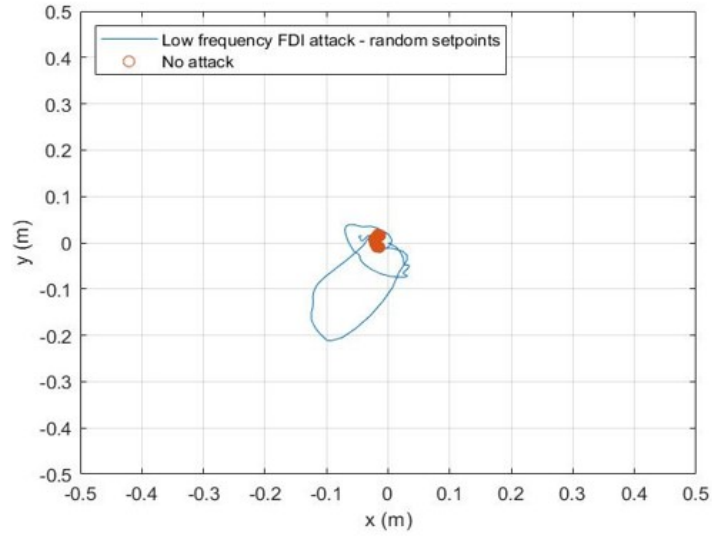


(e) Hover under the low-frequency random setpoints FDI attack.



(f) Hover under the low-frequency random setpoints FDI attack.

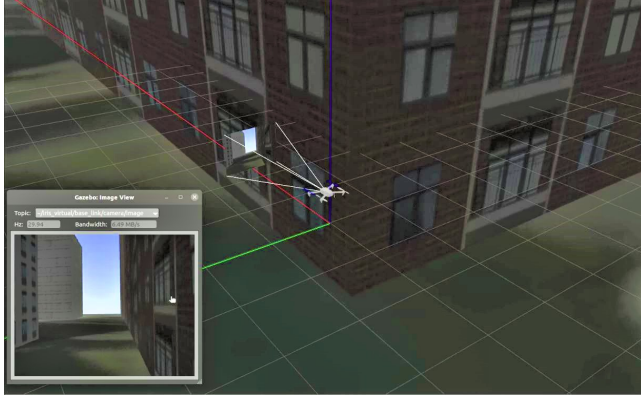
**Figure 4.20.** Hover under the low-frequency random setpoints FDI attack: (Left) Cell phone record; (right) MATLAB.



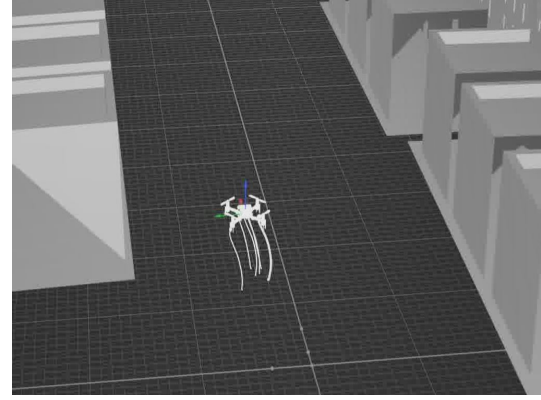
**Figure 4.21.** Hover trajectories under no attack and the low-frequency random setpoints FDI attack.

Figures 4.22 and 4.23 show the cruise mission under the low-frequency random setpoints FDI attack. The attacker injected random setpoints within  $[0\text{m}, +8\text{m}]$  in  $x$ ,  $[-8\text{m}, 8\text{m}]$  in  $y$ , and  $[0.5\text{m}, 1.5\text{m}]$  in  $z$  to delay or drive the UAV away from the cruise course. Figure 4.24 compares the cruise trajectories under no attack and the low-frequency random setpoints FDI attack. The UAV deviated significantly from the expected cruise trajectory, particularly in the  $y$ -axis. The  $y$  setpoint was a constant value in the legitimate user offboard script for this cruise test, which means there should be no momentum on the  $y$ -axis. Therefore it is easy to perturb if any false waypoint is injected into the  $y$ -axis.

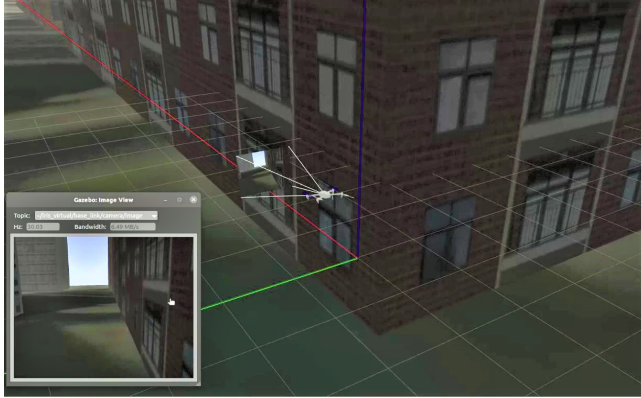




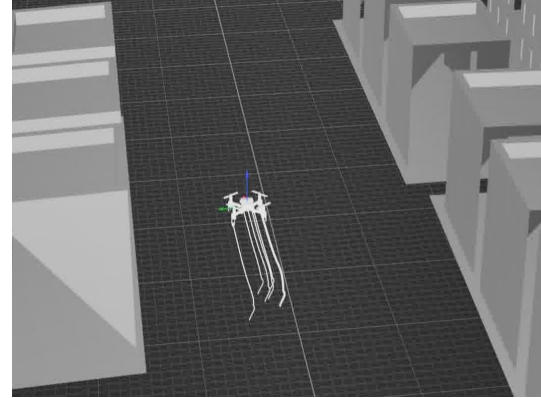
(a) Takeoff.



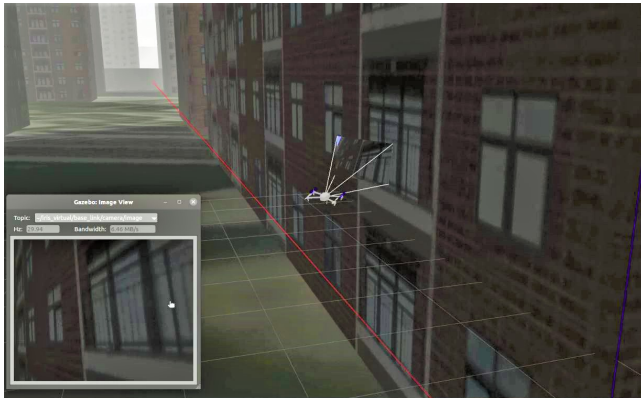
(b) Takeoff.



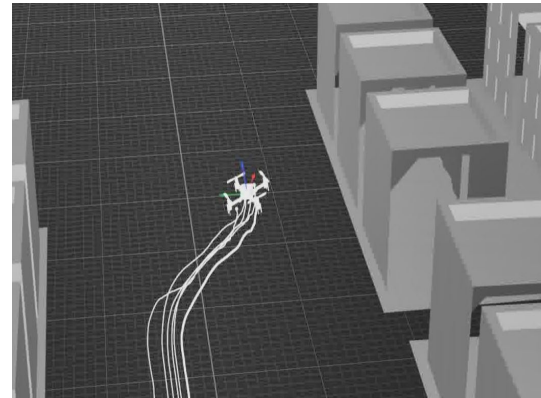
(c) Cruise.



(d) Cruise.



(e) Cruise under the low-frequency random setpoints FDI attack.

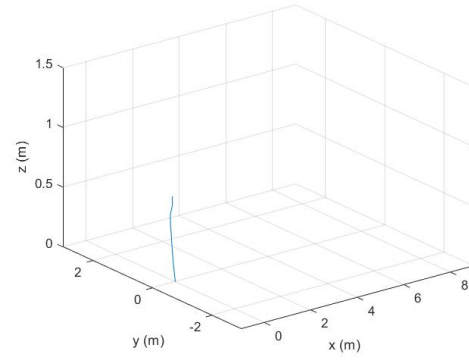


(f) Cruise under the low-frequency random setpoints FDI attack.

**Figure 4.22.** Cruise under the low-frequency random setpoints FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



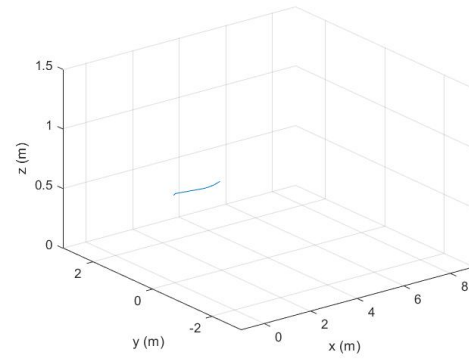
(a) Takeoff.



(b) Takeoff.



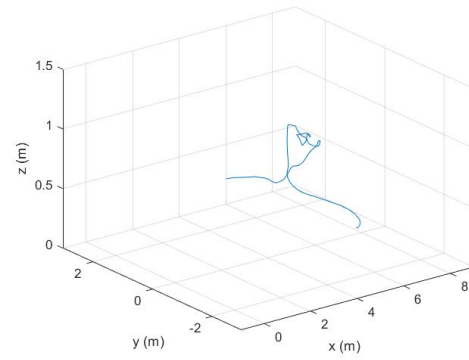
(c) Cruise.



(d) Cruise.



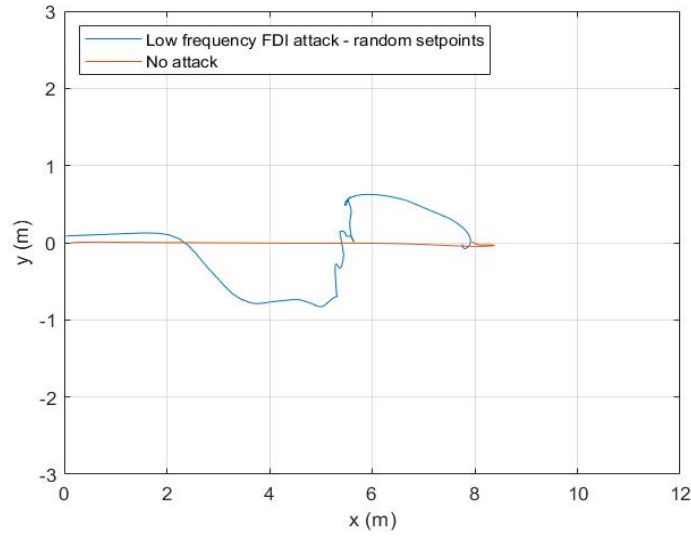
(e) Cruise under the low-frequency random set-points FDI attack.



(f) Cruise under the low-frequency random set-points FDI attack.

**Figure 4.23.** Cruise under the low-frequency random setpoints FDI attack: (Left) Cell phone record; (right) MATLAB.

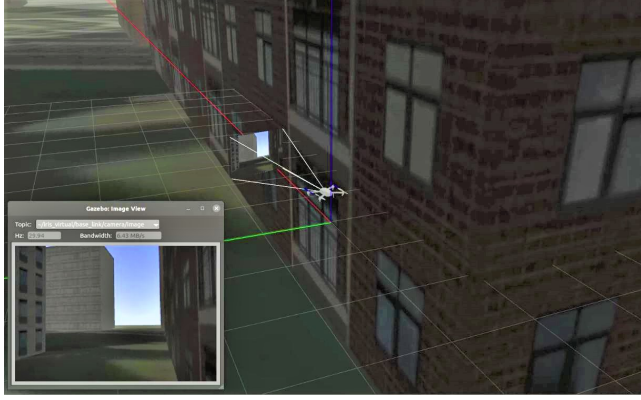




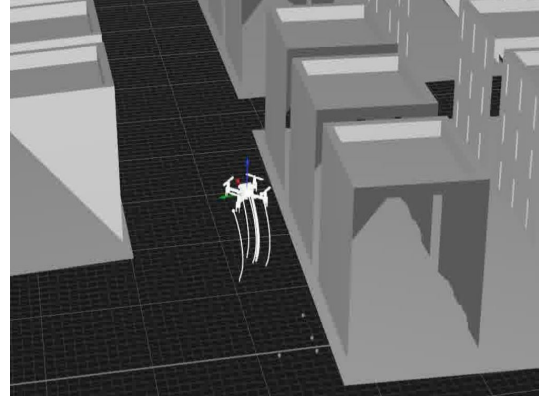
**Figure 4.24.** Cruise trajectories under no attack and the low-frequency random setpoints FDI attack.

### Fixed Waypoint Injection

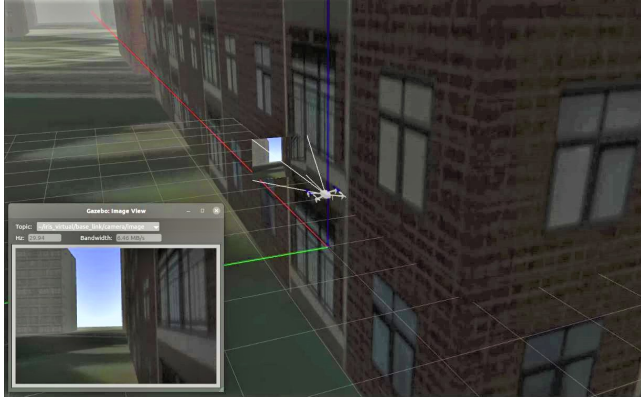
Figures 4.25 and 4.26 show the hover mission under the low-frequency fixed point FDI attack. The attacker aimed to take the UAV to  $x = 2\text{m}$  and  $y = -2\text{m}$ . Figure 4.27 compares the hover trajectories under no attack and the low-frequency fixed point FDI attack. The trajectory under attack deviated from the hover point as the UAV tried reach the attacker's desired location. However, because the frequency of the attack was not rapid enough, the UAV was not precisely at the attacker's desired location. Additionally, because of the attack's low-frequency setting, the UAV could not be landed and disarmed. Figure 4.28 shows the recovery trajectory of an attacker trying to land the UAV at the attacker-desired location, but the attacker was unable to disarm the UAV, and therefore, the UAV was recovered by the legitimate user.



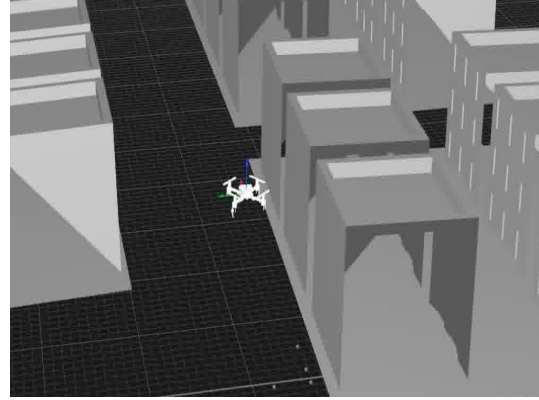
(a) Takeoff.



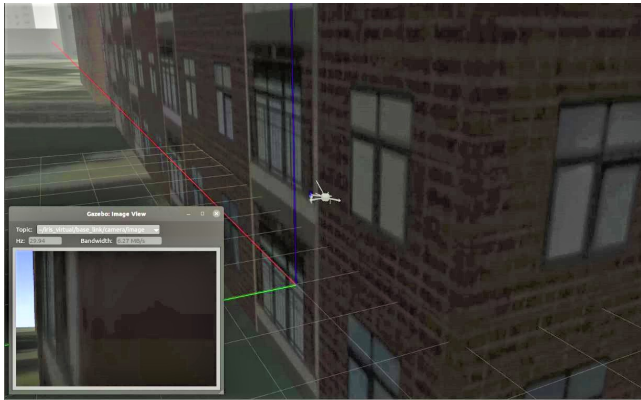
(b) Takeoff.



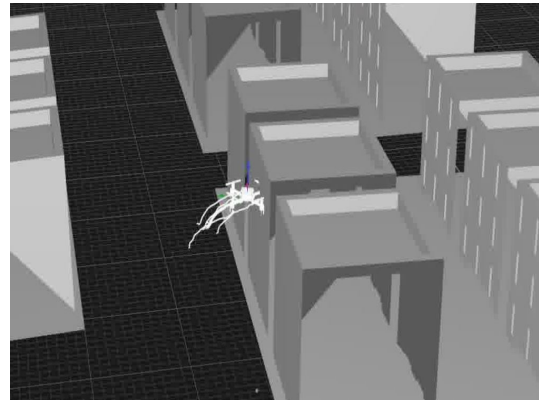
(c) Hover.



(d) Hover.



(e) Hover under the low-frequency fixed setpoint FDI attack.

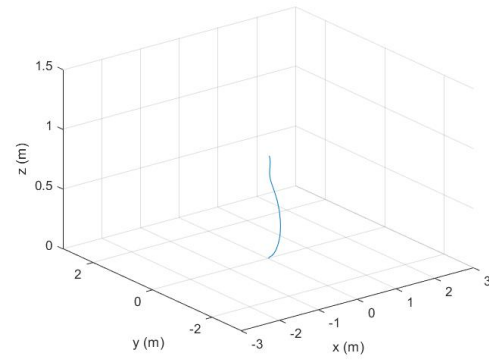


(f) Hover under the low-frequency fixed setpoint FDI attack.

**Figure 4.25.** Hover under the low-frequency fixed setpoint FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



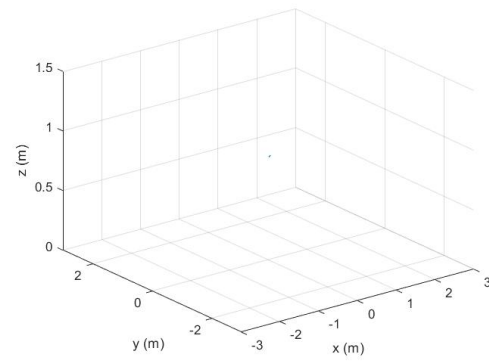
(a) Takeoff.



(b) Takeoff.



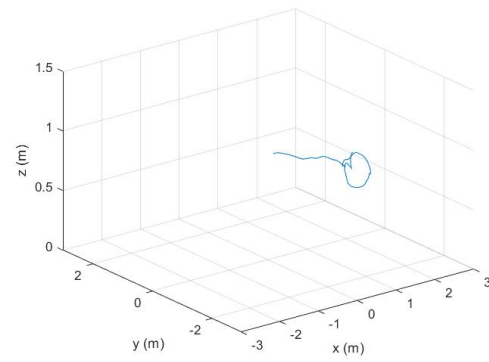
(c) Hover.



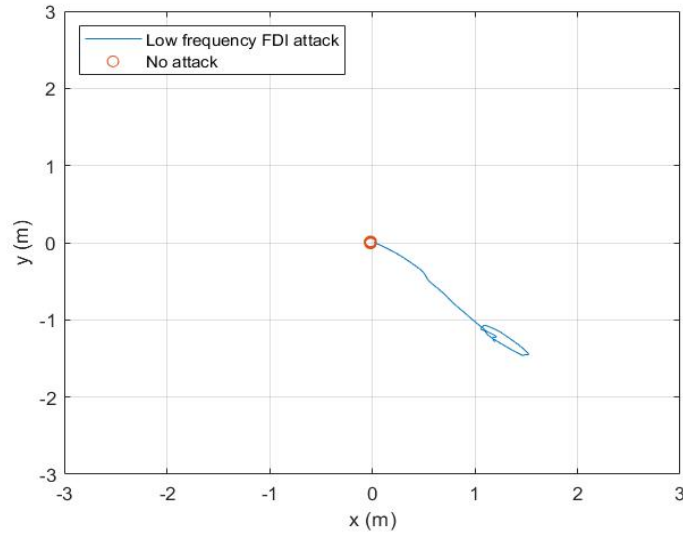
(d) Hover.



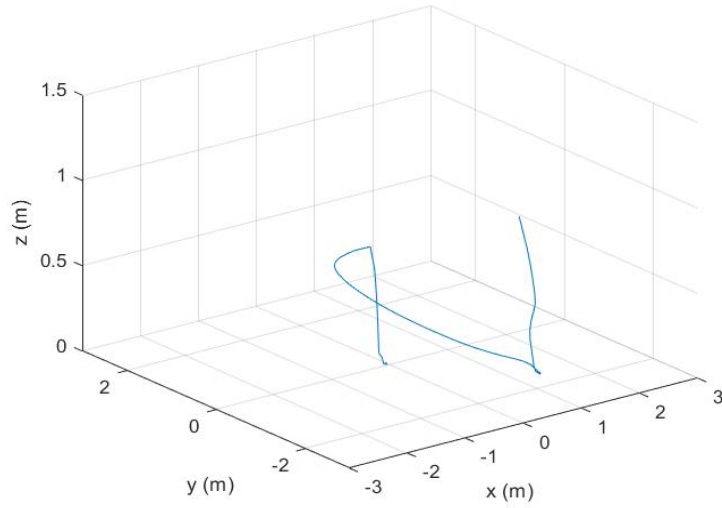
(e) Hover under the low-frequency fixed setpoint FDI attack. (f) Hover under the low-frequency fixed setpoint FDI attack.



**Figure 4.26.** Hover under the low-frequency fixed setpoint FDI attack: (Left) Cell phone record; (right) MATLAB.



**Figure 4.27.** Hover trajectories under no attack and the low-frequency fixed setpoint FDI attack.

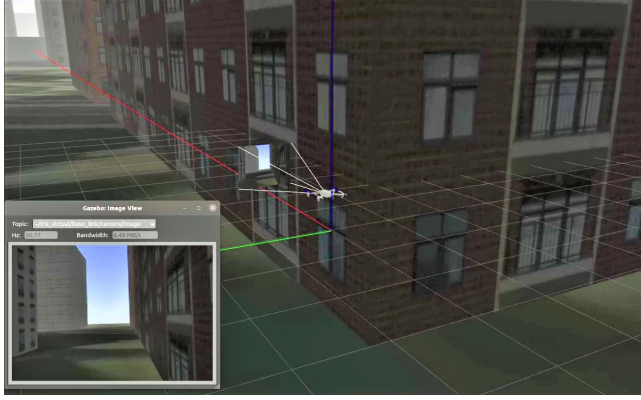


**Figure 4.28.** Recovery trajectory in hover test after the low-frequency fixed setpoint FDI attack.

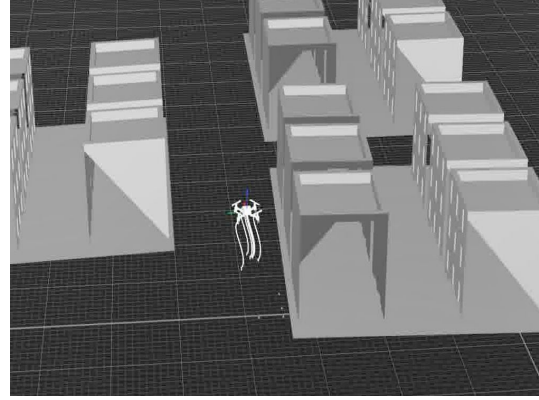
Figures 4.29 and 4.30 show the cruise mission under the low-frequency fixed point FDI attack. The attacker aimed to take the UAV to  $x = 6\text{m}$  and  $y = -2\text{m}$ . Figure 4.31 compares the cruise trajectory under no attack and the fixed point FDI attack. The trajectory under the attack deviated from the original straight-line course as the UAV tried to reach the

attacker's desired location. However, because the frequency of attacks was slow, the UAV was not precisely at the attacker's desired location. Additionally, as with the low-frequency fixed point FDI attack in the hover case, the UAV could not be disarmed even when the attacker successfully drove the UAV to the ground. In the Figure [4.32](#), the trajectory close to the ground was the legitimate offboard script trying to recover the UAV and return it to the original target point.

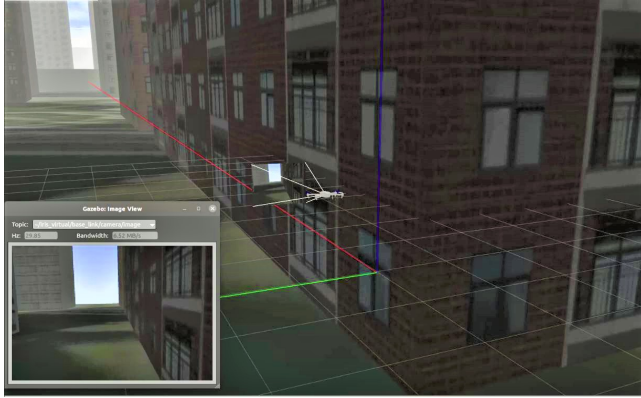




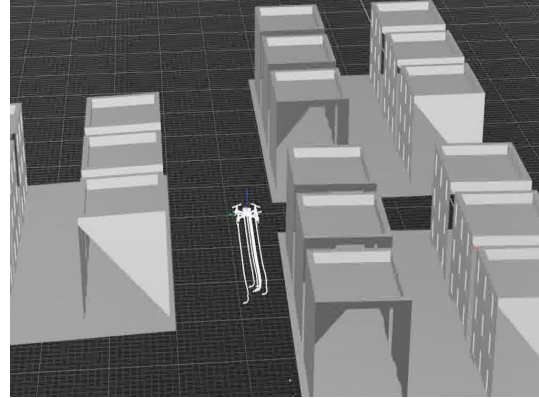
(a) Takeoff.



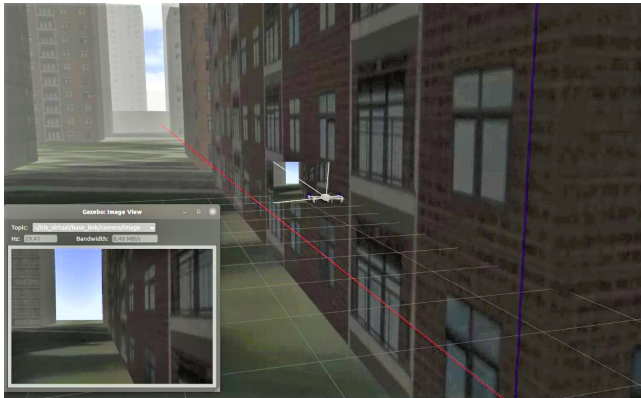
(b) Takeoff.



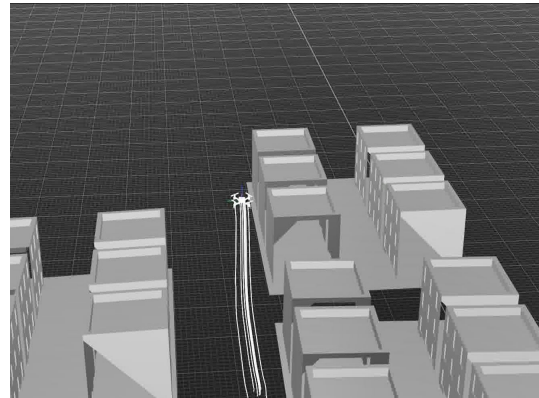
(c) Cruise.



(d) Cruise.



(e) Cruise under the low-frequency fixed setpoint FDI attack.

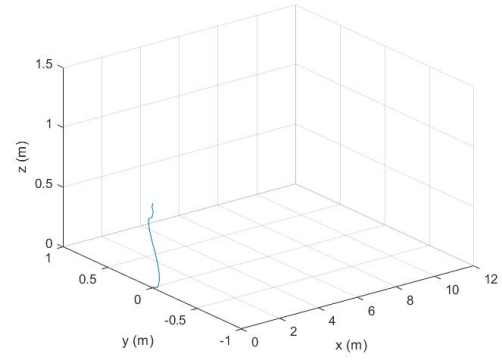


(f) Cruise under the low-frequency fixed setpoint FDI attack.

**Figure 4.29.** Cruise under the low-frequency fixed setpoint FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



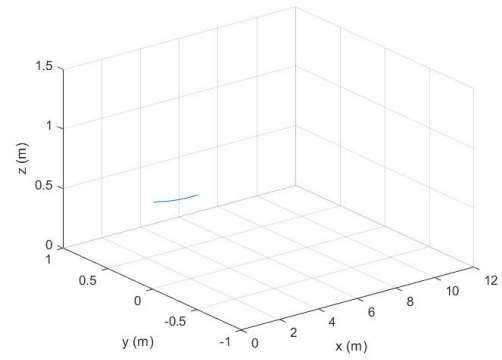
(a) Takeoff.



(b) Takeoff.



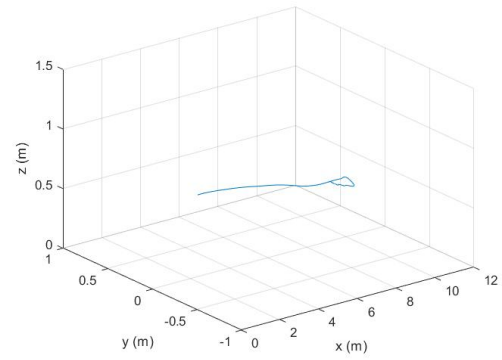
(c) Cruise.



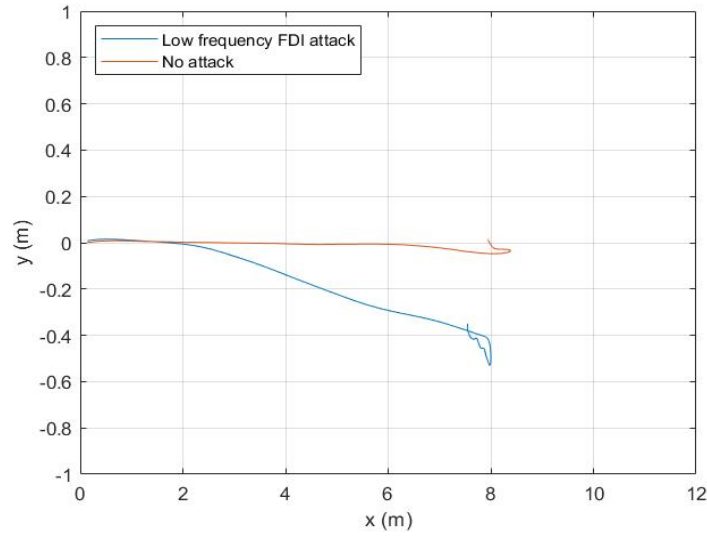
(d) Cruise.



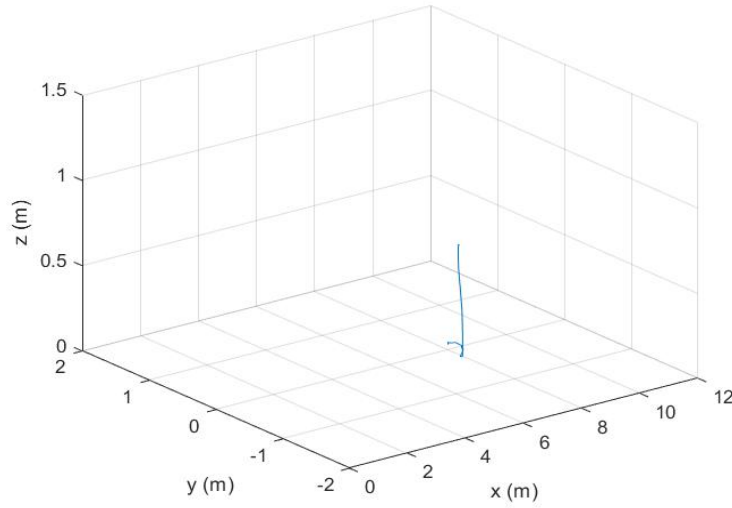
(e) Cruise under the low-frequency fixed setpoint FDI attack. (f) Cruise under the low-frequency fixed setpoint FDI attack.



**Figure 4.30.** Cruise under low-frequency fixed setpoint FDI attack: (Left) Cell phone record; (right) MATLAB.



**Figure 4.31.** Cruise trajectory under no attack and the low-frequency fixed setpoint FDI attack.



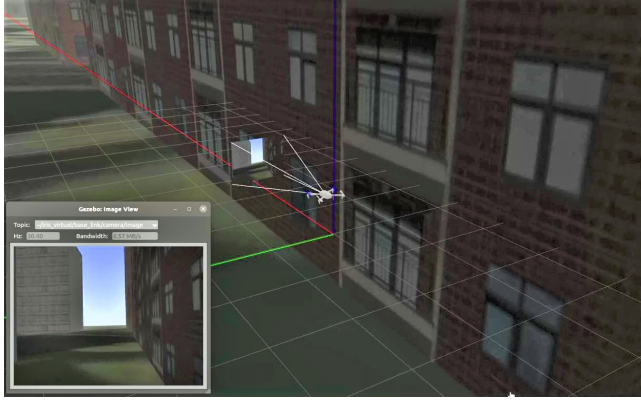
**Figure 4.32.** Cruise trajectory after the low-frequency fixed setpoint FDI attack.

### 4.2.3 High Frequency

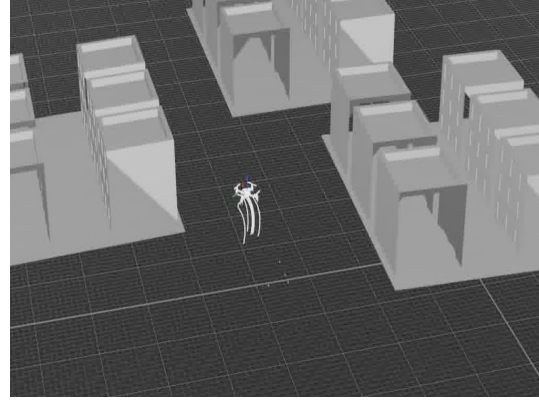
In high-frequency tests, the attacker was able to take control and disarm the UAV at the attacker's desired location. We used a fixed setpoint to force the UAV to move to a malicious waypoint and then force it to land and disarm. Figures 4.33 and 4.34 show the hover mission



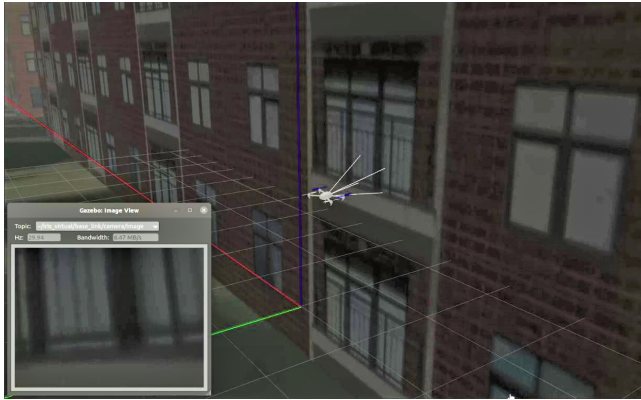
under high-frequency fixed point FDI attack. The attacker aimed to take the UAV to  $x = 2\text{m}$  and  $y = -2\text{m}$ . Figure 4.35 compares the hover trajectories under no attack and the high-frequency fixed point FDI attack. The trajectory under the attack deviated from the hover point and the UAV was successfully directed to the attacker's desired location. In Figure 4.36, we can observe that the advantage of sending at a higher frequency is that it allows the attacker to take the UAV further toward the attacker's desired location.



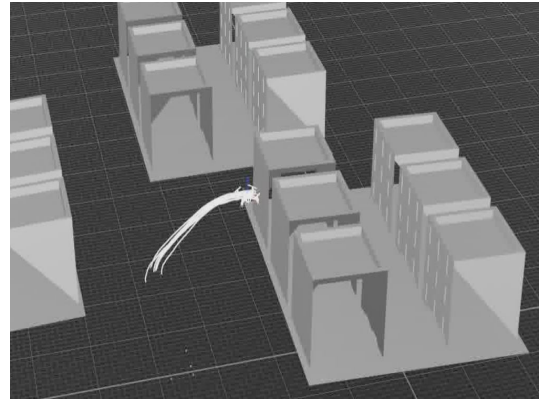
(a) Takeoff and hover.



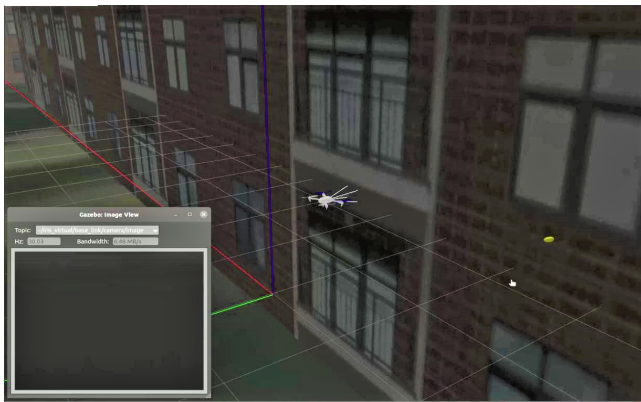
(b) Takeoff and hover.



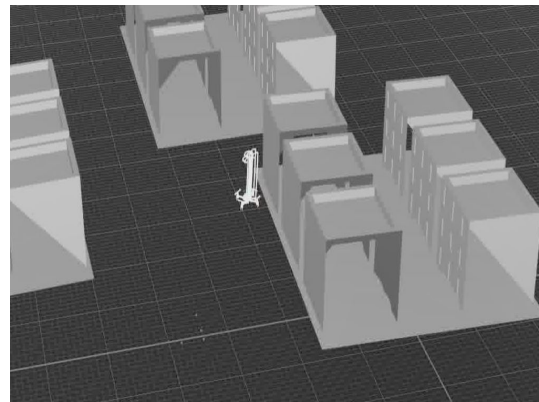
(c) Hover under the high-frequency fixed setpoint FDI attack.



(d) Hover under the high-frequency fixed setpoint FDI attack.



(e) Landing by the high-frequency fixed setpoint FDI attack.

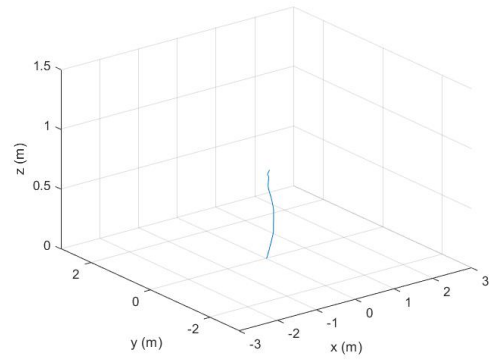


(f) Landing by the high-frequency fixed setpoint FDI attack.

**Figure 4.33.** Hover under the high-frequency fixed setpoint FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



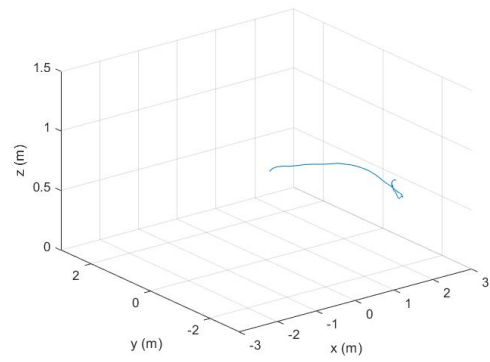
(a) Takeoff and hover.



(b) Takeoff and hover.



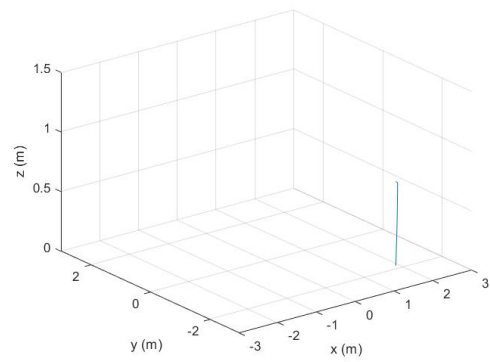
(c) Hover under the high-frequency fixed setpoint FDI attack.



(d) Hover under the high-frequency fixed setpoint FDI attack.

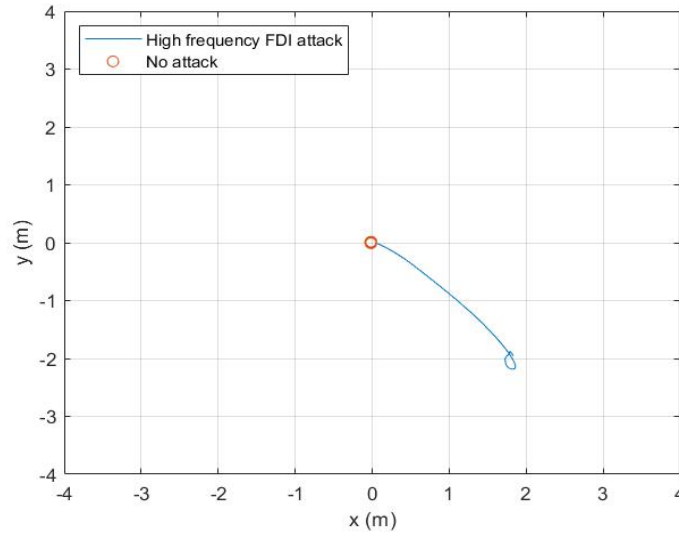


(e) Landing by the high-frequency fixed setpoint FDI attack.

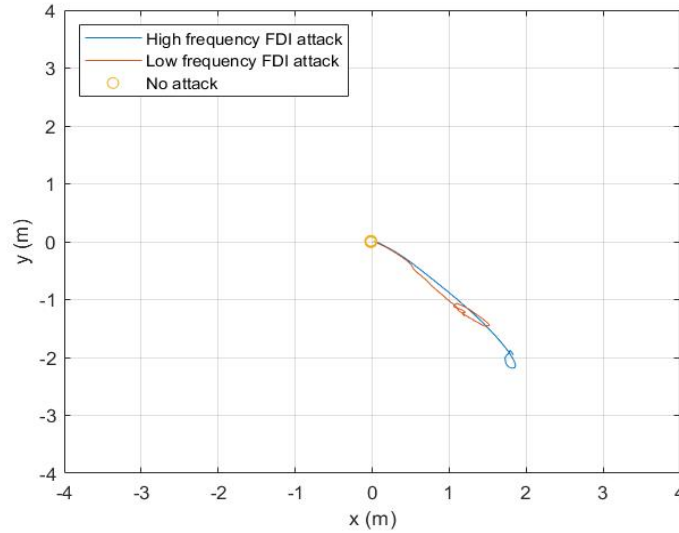


(f) Landing by the high-frequency fixed setpoint FDI attack.

**Figure 4.34.** Hover under the high-frequency fixed setpoint FDI attack: (Left) Cell phone record; (right) MATLAB.



**Figure 4.35.** Hover trajectories under no attack and the high frequency fixed setpoint FDI attack.

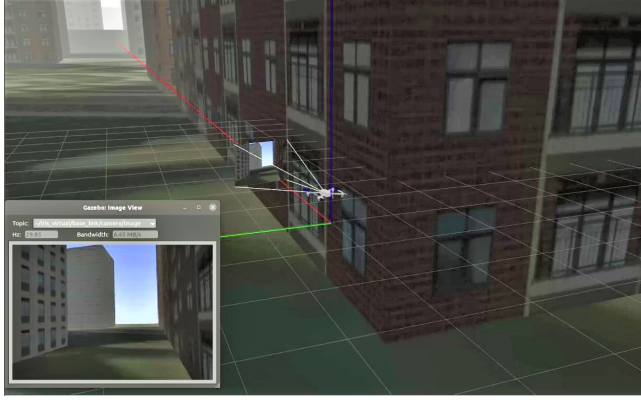


**Figure 4.36.** Hover trajectories under no attack, the low-frequency fixed setpoint attack, and the high-frequency fixed setpoint FDI attack.

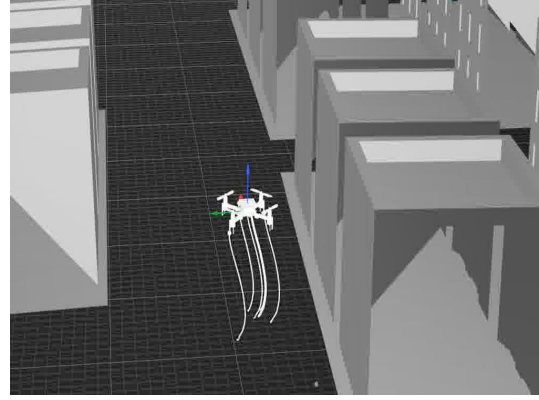
Figures 4.29 and 4.30 show the cruise mission under the high-frequency fixed point FDI attack. The attacker intended to take the UAV to  $x = 6\text{m}$  and  $y = -2\text{m}$ , then land and disarm it. Figure 4.31 compares the cruise trajectories under no attack and the fixed point

FDI attack. The trajectory under the attack deviated from the original straight-line course, and the UAV was successfully guided to the attacker's desired location. In Figure 4.40, we can see that when the attack was at a higher frequency, it was more likely that the attacker could drive the UAV toward the attacker's desired location.

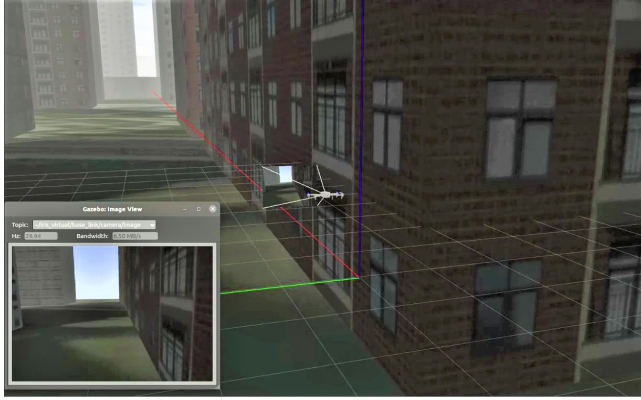




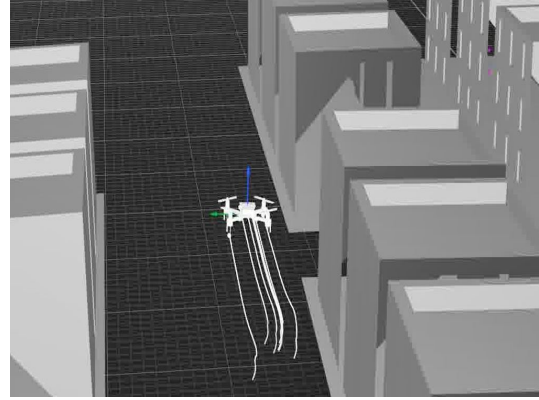
(a) Takeoff.



(b) Takeoff.



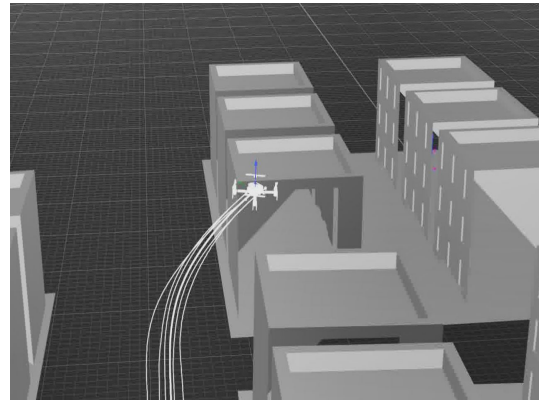
(c) Cruise.



(d) Cruise.



(e) Cruise under the high-frequency fixed setpoint FDI attack.

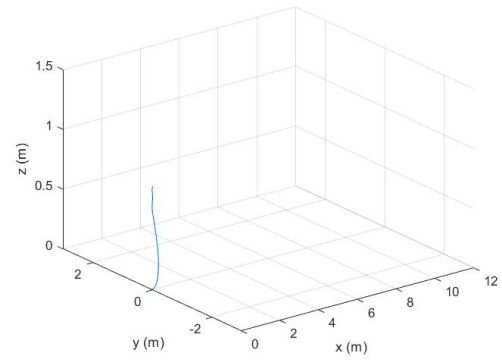


(f) Cruise under the high-frequency fixed setpoint FDI attack.

**Figure 4.37.** Cruise under the high-frequency fixed setpoint FDI attack: (Left) Gazebo Abu Dhabi simulation environment; (right) Qualisys MoCap system environment.



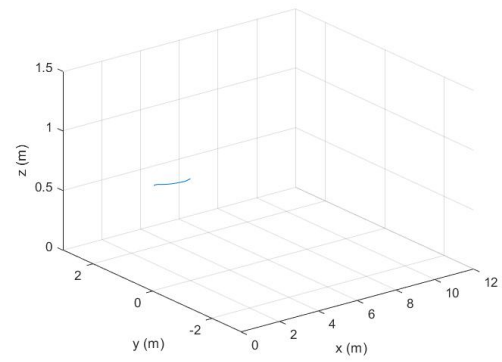
(a) Takeoff.



(b) Takeoff.



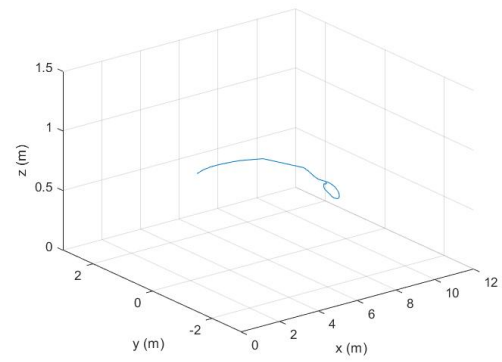
(c) Cruise.



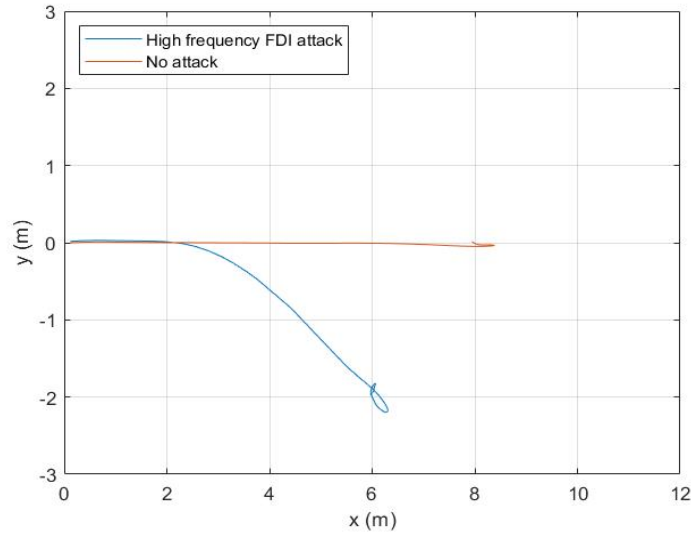
(d) Cruise.



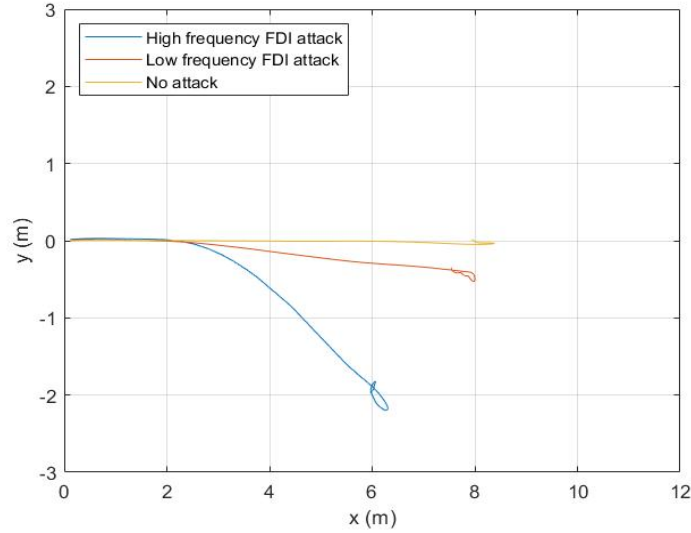
(e) Cruise under the high-frequency fixed setpoint FDI attack. (f) Cruise under the high-frequency fixed setpoint FDI attack.



**Figure 4.38.** Cruise under the high-frequency fixed setpoint FDI attack: (Left) Cell phone record; (right) MATLAB.



**Figure 4.39.** Cruise trajectories under no attack and the high-frequency fixed setpoint FDI attack.



**Figure 4.40.** Cruise trajectories under no attack, the low-frequency fixed setpoint FDI attack, and the high-frequency fixed setpoint FDI attack.

### 4.3 IDS

We have collected 4,739 no-attack, 1,002 ICMP flooding attack, and 2,476 FDI attack samples and randomly separated 80% of the data into a training set and the remaining 20%



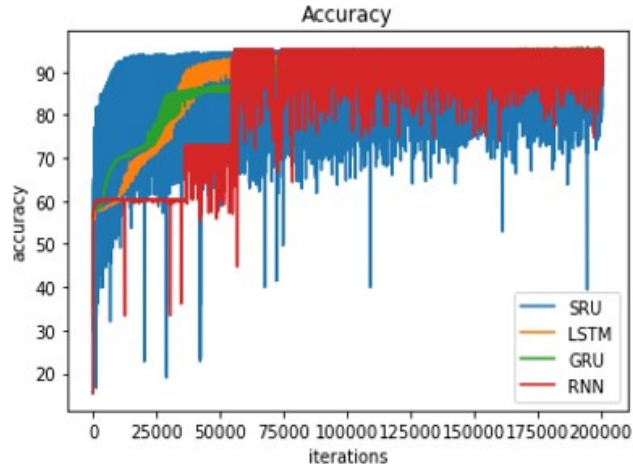
into a testing set. Table 4.1 and Figure 4.41 show the performance and resource consumption of the simple RNN, LSTM, GRU, and SRU with 65 features, hidden units size equal to 64, 1 layer, and 3 output classes.

To create a lighter RNN IDS model, we used RFE to eliminate unnecessary features and thereby decreased the memory and number of FLOPs required per forward pass. RFE also identified and differentiated the subsets of characteristic features for different attack scenarios from the original set of 65 features to further reduce resource consumption and avoid using redundant features. Table 4.2 and Figure 4.42 show the performance and resource consumption of the simple RNN, LSTM, GRU, and SRU with 32 out of 65 features selected by the RFE, with a random forest model, hidden units size equal to 16, 1 layer, and 3 output classes. The RNNs were trained using the Adam optimizer, with a learning rate equal to 0.0001 for simple RNN and SRU and 0.00001 for LSTM and GRU.

While in the case of 65 input features, the accuracy was around 95% for all of the models, with 32 input features, the simple RNN achieved the greatest accuracy, with the lowest number of parameters and FLOPs required for a complete forward pass. Despite having the fewest parameters, simple RNN had the greatest accuracy because the number of time steps required for this problem was low enough that the simple RNN did not have a vanishing gradient problem. Conversely, with more parameters than simple RNN, LSTM and GRU ran into the overfitting problem, which caused them to have less accuracy in testing data. More trainable parameters had increased the model complexity, and thus tightly fitting the model to the training data, which caused overfitting.

**Table 4.1.** RNN IDS performance with 65 features data set and 64 hidden units.

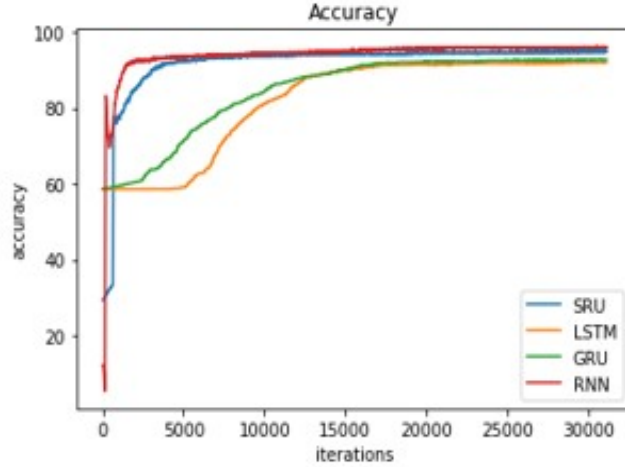
<b>RNN</b>	<b>Accuracy, %</b>	<b>Parameters</b>	<b>Memory (KB)</b>	<b>FLOPs per complete forward pass</b>
Simple RNN	95.3771	8579	59	140483
LSTM	95.0730	33731	162.79	569027
GRU	95.4380	25347	128.40	422083
SRU	95.2555	12931	72	219331



**Figure 4.41.** Accuracy performance of RNNs with 65 features (%).

**Table 4.2.** RNN IDS performance with 32 features data set.

RNN	Accuracy, %	Parameters	Memory (KB)	FLOPs per complete forward pass
Simple RNN	96.2895	851	5.864	14387
LSTM	92.0925	3251	15.69	59315
GRU	92.9440	2451	12.42	43315
SRU	95.1338	1651	9.206	29491



**Figure 4.42.** Accuracy performance of RNNs with 32 features (%).

We also compared the performance of the RNN-based IDSs with other filtering methods by setting threshold values for specific features. In the control and estimation field, Kalman filter parameters are often used as the threshold feature. However, computer science researchers typically consider network properties to be the important features. Here, for the features, we chose estimator velocity covariances in the north, east, and down directions (message: estimator status, features: covariances[4], covariances[5], covariances[6]) and telemetry receiving rate (message: telemetry status, feature: rate rx).

The threshold value was set by two methods—maximum and minimum test and mean and standard deviation. The maximum velocity covariance values in each directions standard case were set as the maximum and minimum test threshold values. Any value above the threshold was treated as an attack. In contrast, the maximum and minimum values of rate rx were the threshold boundaries. If the rate rx was above the maximum value or below the minimum value, it was classified as an attack.

The second method used the mean and standard deviation of the features to set the threshold values, also known as sigma threshold, so that the mean and standard deviation of velocity covariance in each direction and rate rx was calculated as the normal cases. Then, we set the threshold value at the mean value plus a factor of the standard deviation for each velocity covariance. Any value above the threshold was classified as an attack. For the rate rx, the upper bound of the threshold value was the mean value plus a factor of standard

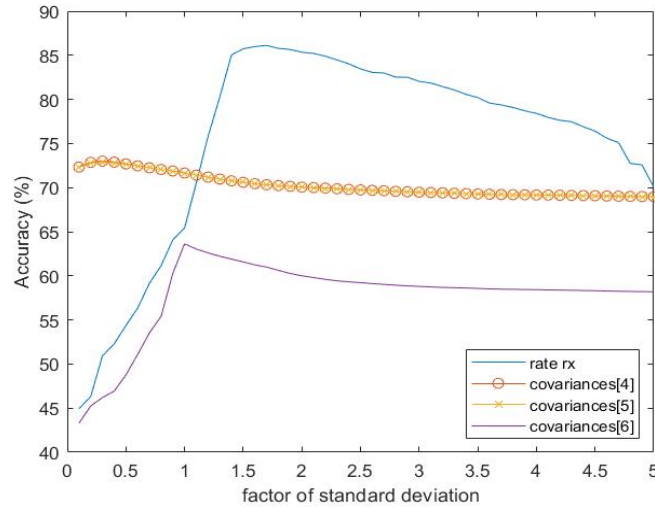
deviation. Similarly, the lower bound was the mean value minus a factor of the standard deviation. If the rate rx was higher than the upper bound threshold value or lower than the lower bound threshold value, we classified it as an attack.

The following table shows the maximum accuracy of each threshold method with the selected features. Figure 4.43 shows the accuracy versus the factor of standard deviation used for threshold value setting. We confirm that the accuracy of the threshold methods was much lower than with any of the RNN methods. In addition, threshold method was not able to identify the type of attack.

**Table 4.3.** Threshold methods maximum accuracy.

Feature	Max and Min value	Mean and standard deviation
Covariances [4]	65.11%	73.00% at ratio of std = 0.24
Covariances [5]	64.96%	72.90% at ratio of std = 0.27
Covariances [6]	58.07%	63.62% at ratio of std = 0.96
Rate rx	81.87%	86.21% at ratio of std = 1.66

ratio of std = ratio of standard deviation.



**Figure 4.43.** Detection accuracy versus the factor of standard deviation used for the threshold.

Considering the resource-constrained characteristic of UAVs, the resource consumption with different IDSs also needs to be evaluated and compared. If the user wants to put the IDS model on the computer's flash memory, the memory limit needs to be met. For example, Arduino Uno only has 32 KB of flash memory, which means that only the RNN IDS constructed with 32 input features and 16 hidden units, with a time step window of 8 and 3 output classes can fit the board, according to Tables 4.1 and 4.2. Similarly, the FLOPs required for one complete forward pass is another critical category to be examined. Using Equation (2.29) and 0.4989 seconds as the time interval of the selected cpuload message, we could calculate the maximum FLOPs of one complete forward pass for the IDS models with the FLOPS of the selected computer. Here, we used Arduino Uno as an example. According to Table 2.17, we selected 103199.17 FLOPS to calculate the maximum FLOPs of one complete forward pass for the IDS model. The following equation shows that 51486.066 was the max number of FLOPs that the IDS can use for a full forward pass on Arduino Uno.

$$\text{FLOPs of one complete forward pass for IDS model} \leq 103199.17 \times 0.4989 = 51486.066 \quad (4.1)$$

According to Tables 4.1 and 4.2, only the simple RNN, GRU, and SRU with a 32 features data set, 16 hidden units, 1 layer, and 3 classes could meet the requirements of Arduino Uno. Obviously, one could choose to add a more powerful computer to satisfy the computation needs of a larger model. However, doing so also increases power consumption.

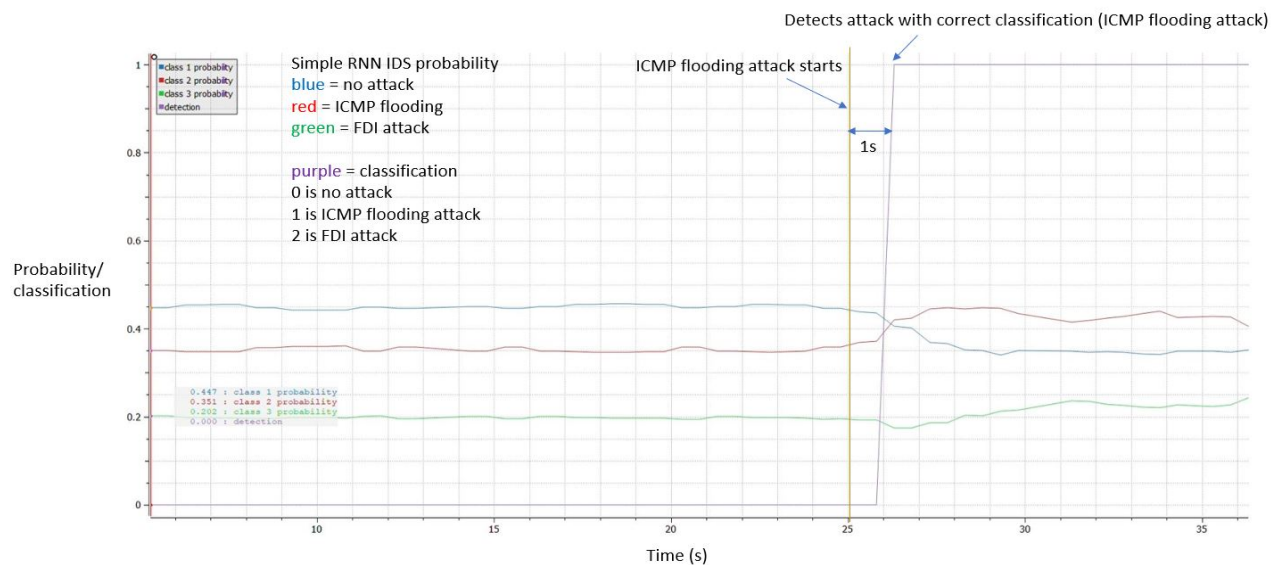
Considering both memory and computational cost constrains, Table 4.4 shows that only the simple RNN, GRU, and SRU with 32 features in the data set and 16 hidden units are implementable on the target computer. Among these three options, the simple RNN has highest accuracy rate with the least memory and fewest FLOPS required. Thus, we conclude that simple RNN is the best choice among RNNs and the selected threshold methods for detecting an attack onboard limited-resources platforms.

**Table 4.4.** Overall performance of RNNs IDS methods.

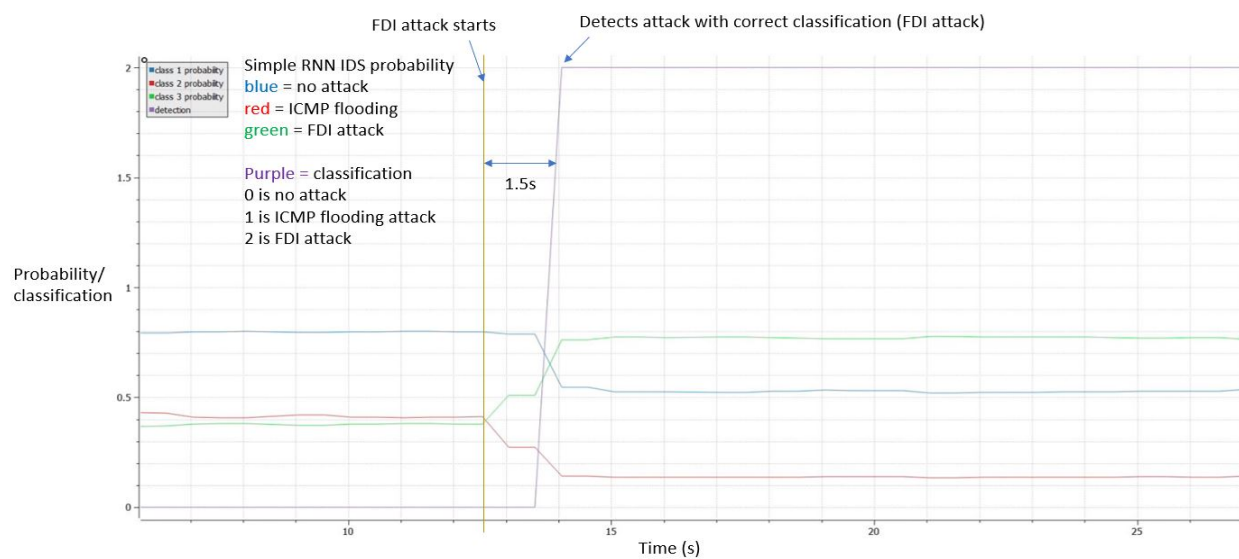
IDS model	Number of features	Hidden units	Accuracy, %	Memory (KB)	FLOPs per complete forward pass
Simple RNN	65	64	95.3771	59	140483
LSTM	65	64	95.0730	162.79	569027
GRU	65	64	95.4380	128.40	422083
SRU	65	64	95.2555	72	219331
Simple RNN	32	16	96.2895	5.864	14387
LSTM	32	16	92.0925	15.69	59315
GRU	32	16	92.9440	12.42	43315
SRU	32	16	95.1338	9.206	29491
Arduino Uno constraint				32	51486.066

Red = Parameters that does not meet the constrains of selected computer.

To verify that our proposed Simple RNN IDS can detect and identify the type of attack, we injected ICMP flooding attack and FDI attack into the final IDS model. Figures 4.44 and 4.45 show the probability of normal condition and each type of attack under, respectively, ICMP flooding attack and FDI attack. Additionally, the classification decision based on these probabilities is also shown in both figures. The results show that simple RNN IDS can detect the attacks in 1 to 1.5 seconds after the attacks occurred and correctly classify the attack type.



**Figure 4.44.** Simple RNN IDS probability of each attack and classification under ICMP flooding attack.



**Figure 4.45.** Simple RNN IDS probability of each attack and classification under FDI attack.

## 5. CONCLUSION

This work explored the vulnerability in the communication channel between an unmanned aerial vehicle (UAV) and a ground control station (GCS), and conducted real-world experiments to demonstrate the implementation of different cyberattacks, and developed a lightweight IDS using a recurrent neural network (RNN). The overall system details and attack strategies were presented in this paper, along with thorough explanation of the design of the RNN structure and the challenges of deploying an intrusion detection system (IDS) onboard a UAV. Various parameters were proposed for testing whether the simple RNN, long short-term memory (LSTM), gated recurrent units (GRU) or simple recurrent units (SRU)-based IDSs are suitable for the selected resources-constrained system. Unlike other works, extensive actual experiments were conducted for this thesis to demonstrate the attacks and collect the training data for IDS models.

The experiments implemented attacks on a Holybro S500 quadrotor with PX4 autopilot firmware and MAVLink protocol. It was shown that the UAV was in fact vulnerable to an Internet control message protocol (ICMP) flooding attack, resulting in a denial of service (DoS) attack and a false waypoint injection attack once the attacker knew the network parameters of the UAV and GCS. Deviations from the normal mission trajectory and loss of control, such as the sudden acceleration of the Holybro S500 quadrotor, were observed during the experiments.

The simple RNN, GRU, and SRU were tested for suitability of implementation on the Arduino Uno board, the lowest-cost companion that can be implemented with the PX4, by checking the memory size of the IDS model and number of floating point operations (FLOPs) required for a complete forward pass. Among RNNs, the simple RNN achieved the highest rate of attack detection accuracy at 96 %, with the lowest memory size of 5.864 kilobytes. The number of FLOPs required for one complete forward pass of the simple RNN was 14,387. In addition, RNNs outperformed the threshold value based method that used the covariance of velocity in each direction and telemetry receiving rate as features. Thus, we confirmed that the simple RNN is our best choice of IDS for resources-constrained platforms.



## 5.1 Future Directions

Future extensions of this work can include the following:

- Conduct more experiments with various attack methods to gather more data in the training data set, thus making the IDS robust for various malicious actions.
- Instead of using the default PX4 features, create custom features that can help the IDS detect attacks more accurately.
- Online learning for an IDS model to perform updates and corrections to the model.
- Try different approaches for designing IDSs for different attacks. It may be found that one method is more accurate for detecting a particular type of attack.

## REFERENCES

- [1] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, “Micro air vehicle link (mavlink) in a nutshell: A survey,” *IEEE Access*, vol. 7, pp. 87 658–87 680, 2019. DOI: [10.1109/ACCESS.2019.2924410](https://doi.org/10.1109/ACCESS.2019.2924410).
- [2] A. Allouch, O. Cheikhrouhou, A. Koubâa, M. Khalgui, and T. Abbes, “Mavsec: Securing the mavlink protocol for ardupilot/px4 unmanned aerial systems,” in *2019 15th International Wireless Communications Mobile Computing Conference (IWCMC)*, 2019, pp. 621–628. DOI: [10.1109/IWCMC.2019.8766667](https://doi.org/10.1109/IWCMC.2019.8766667).
- [3] C.-C. Sun, A. Hahn, and C.-C. Liu, “Cyber security of a power grid: State-of-the-art,” *International Journal of Electrical Power & Energy Systems*, vol. 99, pp. 45–56, 2018.
- [4] M. Abomhara and G. M. K ien, “Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks,” *Journal of Cyber Security and Mobility*, pp. 65–88, 2015.
- [5] A. Kim, B. Wampler, J. Goppert, I. Hwang, and H. Aldridge, “Cyber attack vulnerabilities analysis for unmanned aerial vehicles,” in *Infotech@Aerospace*, 2012.
- [6] S. Dahiya and M. Garg, “Unmanned aerial vehicles: Vulnerability to cyber attacks,” in *Proceedings of UASG 2019*, K. Jain, K. Khoshelham, X. Zhu, and A. Tiwari, Eds., Cham: Springer International Publishing, 2020, pp. 201–211, ISBN: 978-3-030-37393-1.
- [7] L. Liang, K. Zheng, Q. Sheng, and X. Huang, “A denial of service attack method for an iot system,” in *2016 8th International Conference on Information Technology in Medicine and Education (ITME)*, 2016, pp. 360–364. DOI: [10.1109/ITME.2016.0087](https://doi.org/10.1109/ITME.2016.0087).
- [8] D. Moore, G. M. Voelker, and S. Savage, “Inferring internet denial-of-service activity,” in *10th USENIX Security Symposium (USENIX Security 01)*, Washington, D.C.: USENIX Association, Aug. 2001. [Online]. Available: <https://www.usenix.org/conference/10th-usenix-security-symposium/inferring-internet-denial-service-activity>.
- [9] S. Vadlamani, B. Eksioglu, H. Medal, and A. Nandi, “Jamming attacks on wireless networks: A taxonomic survey,” *International Journal of Production Economics*, vol. 172, pp. 76–94, 2016, ISSN: 0925-5273. DOI: <https://doi.org/10.1016/j.ijpe.2015.11.008>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S092552731500451X>.
- [10] B. Van den Bergh and S. Pollin, “Keeping uavs under control during gps jamming,” *IEEE Systems Journal*, vol. 13, no. 2, pp. 2010–2021, 2019. DOI: [10.1109/JSYST.2018.2882769](https://doi.org/10.1109/JSYST.2018.2882769).

- [11] M. L. Psiaki and T. E. Humphreys, “Gnss spoofing and detection,” *Proceedings of the IEEE*, vol. 104, no. 6, pp. 1258–1270, 2016.
- [12] G. Vasconcelos, G. Carrijo, R. Miani, J. Souza, and V. Guizilini, “The impact of dos attacks on the ar.drone 2.0,” in *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, 2016, pp. 127–132. DOI: [10.1109/LARS-SBR.2016.28](https://doi.org/10.1109/LARS-SBR.2016.28).
- [13] Y.-M. Kwon, J. Yu, B.-M. Cho, Y. Eun, and K.-J. Park, “Empirical analysis of mavlink protocol vulnerability for attacking unmanned aerial vehicles,” *IEEE Access*, vol. 6, pp. 43 203–43 212, 2018.
- [14] S. Jeong, E. Park, K. Seo, J. D. Yoo, and H. Kim, “Muvids: False mavlink injection attack detection in communication for unmanned vehicles,” 2021.
- [15] Pymavlink, *Pymavlink*, <https://www.ardubus.com/developers/pymavlink.html>, 2021.
- [16] Y. Liu, P. Ning, and M. K. Reiter, “False data injection attacks against state estimation in electric power grids,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, pp. 1–33, 2011.
- [17] M. A. Rahman and H. Mohsenian-Rad, “False data injection attacks against nonlinear state estimation in smart power grids,” in *2013 IEEE Power & Energy Society General Meeting*, IEEE, 2013, pp. 1–5.
- [18] A. Abbaspour, K. K. Yen, S. Noei, and A. Sargolzaei, “Detection of fault data injection attack on UAV using adaptive neural network,” *Procedia computer science*, vol. 95, pp. 193–200, 2016.
- [19] S. M. Giray, “Anatomy of unmanned aerial vehicle hijacking with signal spoofing,” in *2013 6th International Conference on Recent Advances in Space Technologies (RAST)*, 2013, pp. 795–800. DOI: [10.1109/RAST.2013.6581320](https://doi.org/10.1109/RAST.2013.6581320).
- [20] SpacehuhnTech, *Esp8266 deauther version 2*, [https://github.com/SpacehuhnTech/esp8266\\_deauther](https://github.com/SpacehuhnTech/esp8266_deauther), Feb. 2021.
- [21] pulkin, *Esp8266 packet injection/sniffer example*, <https://github.com/pulkin/esp8266-injection-example>, Jan. 2020.
- [22] T. F. Lunt, “A survey of intrusion detection techniques,” *Computers & Security*, vol. 12, no. 4, pp. 405–418, 1993.
- [23] S. Axelsson, “Research in intrusion-detection systems: A survey,” Technical report 98–17. Department of Computer Engineering, Chalmers ..., Tech. Rep., 1998.

- [24] F. Sabahi and A. Movaghar, “Intrusion detection: A survey,” in *2008 Third International Conference on Systems and Networks Communications*, IEEE, 2008, pp. 23–26.
- [25] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, “Intrusion detection system: A comprehensive review,” *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.
- [26] G. Choudhary, V. Sharma, I. You, K. Yim, R. Chen, and J.-H. Cho, “Intrusion detection systems for networked unmanned aerial vehicles: A survey,” in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, IEEE, 2018, pp. 560–565.
- [27] C.-Y. Tseng, P. Balasubramanyam, C. Ko, R. Limprasittiporn, J. Rowe, and K. Levitt, “A specification-based intrusion detection system for aodv,” in *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, 2003, pp. 125–134.
- [28] P. Uppuluri and R. Sekar, “Experiences with specification-based intrusion detection,” in *Recent Advances in Intrusion Detection*, 2001.
- [29] R. Mitchell and I.-R. Chen, “Specification based intrusion detection for unmanned aircraft systems,” in *Proceedings of the first ACM MobiHoc workshop on Airborne Networks and Communications*, 2012, pp. 31–36.
- [30] R. Mitchell and R. Chen, “Adaptive intrusion detection of malicious unmanned air vehicles using behavior rule specifications,” *IEEE transactions on systems, man, and cybernetics: systems*, vol. 44, no. 5, pp. 593–604, 2013.
- [31] V. Vaidya, *Dynamic signature inspection-based network intrusion detection*, US Patent 6,279,113, Aug. 2001.
- [32] T. Vuong, A. Filippoupolitis, G. Loukas, and D. Gan, “Physical indicators of cyber attacks against a rescue robot,” in *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*, 2014, pp. 338–343. DOI: [10.1109/PerComW.2014.6815228](https://doi.org/10.1109/PerComW.2014.6815228).
- [33] T. P. Vuong, G. Loukas, and D. Gan, “Performance evaluation of cyber-physical intrusion detection on a robotic vehicle,” in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, 2015, pp. 2106–2113. DOI: [10.1109/CIT/IUCC/DASC/PICOM.2015.313](https://doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.313).

- [34] T. P. Vuong, G. Loukas, D. Gan, and A. Bezemskij, "Decision tree-based detection of denial of service and command injection attacks on robotic vehicles," in *2015 IEEE International Workshop on Information Forensics and Security (WIFS)*, 2015, pp. 1–6. DOI: [10.1109/WIFS.2015.7368559](https://doi.org/10.1109/WIFS.2015.7368559).
- [35] Y. O. Weng, "Detection and characterization of actuator attacks using kalman filter estimation," PhD thesis, Marquette University, 2018.
- [36] G. Loukas, T. Vuong, R. Heartfield, G. Sakellari, Y. Yoon, and D. Gan, "Cloud-based cyber-physical intrusion detection for vehicles using deep learning," *Ieee Access*, vol. 6, pp. 3491–3508, 2017.
- [37] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.
- [38] K. Manandhar, X. Cao, F. Hu, and Y. Liu, "Detection of faults and attacks including false data injection attack in smart grid using kalman filter," *IEEE transactions on control of network systems*, vol. 1, no. 4, pp. 370–379, 2014.
- [39] C. Kwon, "Cyber attack analysis on cyber-physical systems: Detectability, severity, and attenuation strategy," English, Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2021-05-18, PhD thesis, 2013, p. 95, ISBN: 978-1-303-61553-5. [Online]. Available: <https://www.proquest.com/dissertations-theses/cyber-attack-analysis-on-physical-systems/docview/1490789215/se-2?accountid=13360>.
- [40] H. Bangui and B. Buhnova, "Recent advances in machine-learning driven intrusion detection in transportation: Survey," *Procedia Computer Science*, vol. 184, pp. 877–886, 2021, The 12th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops, ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2021.04.014>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050921007894>.
- [41] K. Xiao, J. Zhao, Y. He, C. Li, and W. Cheng, "Abnormal behavior detection scheme of uav using recurrent neural networks," *IEEE Access*, vol. 7, pp. 110 293–110 305, 2019. DOI: [10.1109/ACCESS.2019.2934188](https://doi.org/10.1109/ACCESS.2019.2934188).
- [42] M. A. Aydın, A. H. Zaim, and K. G. Ceylan, "A hybrid intrusion detection system design for computer network security," *Computers Electrical Engineering*, vol. 35, no. 3, pp. 517–526, 2009, ISSN: 0045-7906. DOI: <https://doi.org/10.1016/j.compeleceng.2008.12.005>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0045790609000020>.

- [43] S. Pan, T. Morris, and U. Adhikari, “Developing a hybrid intrusion detection system using data mining for power systems,” *IEEE Transactions on Smart Grid*, vol. 6, no. 6, pp. 3104–3113, 2015. DOI: [10.1109/TSG.2015.2409775](https://doi.org/10.1109/TSG.2015.2409775).
- [44] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, “Anomaly-based network intrusion detection: Techniques, systems and challenges,” *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.
- [45] D. M. Marshall, R. K. Barnhart, E. Shappee, and M. Most, *Introduction to unmanned aircraft systems*. CRC Press, Taylor amp; Francis Group, 2016.
- [46] D. of Defense, *Unmanned aircraft system airspace integration plan*, <https://info.publicintelligence.net/DoD-UAS-AirspaceIntegration.pdf>, Mar. 2011.
- [47] D. Marshall, R. Barnhart, E. Shappee, and M. Most, “Uas applications,” in *Introduction to unmanned aircraft systems*. CRC Press, Taylor amp; Francis Group, 2016.
- [48] PX4, *Airframes reference*, [https://docs.px4.io/master/en/airframes/airframe\\_reference.html](https://docs.px4.io/master/en/airframes/airframe_reference.html), Jun. 2021.
- [49] PX4, *Sensors*, [https://docs.px4.io/v1.9.0/en/getting\\_started/sensor\\_selection.html](https://docs.px4.io/v1.9.0/en/getting_started/sensor_selection.html), Oct. 2020.
- [50] PX4, *Companion computer for pixhawk series*, [https://docs.px4.io/master/en/companion\\_computer/pixhawk\\_companion.html](https://docs.px4.io/master/en/companion_computer/pixhawk_companion.html), Jan. 2021.
- [51] PX4autopilot, *Loading firmware*, <https://docs.px4.io/master/en/config/firmware.html>, Jun. 2021.
- [52] PX4, *Simulation*, <https://docs.px4.io/master/en/simulation/>, Jun. 2021.
- [53] PX4, *Hardware in the loop simulation (hitl)*, <https://docs.px4.io/master/en/simulation/hitl.html>, Jun. 2021.
- [54] Gazebo, *Gazebo*, <http://gazebo.org>.
- [55] jMAVSim, *Jmavsim*, <https://github.com/PX4/jMAVSim>, May 2021.
- [56] PX4, *Ros (robot operating system)*, <https://docs.px4.io/master/en/ros/>, Mar. 2021.
- [57] PX4, *Pixhawk series*, [https://docs.px4.io/master/en/flight\\_controller/pixhawk\\_series.html](https://docs.px4.io/master/en/flight_controller/pixhawk_series.html), Jun. 2021.

- [58] PX4, *Wifi telemetry radio*, [https://docs.px4.io/v1.9.0/en/telemetry/telemetry\\_wifi.html](https://docs.px4.io/v1.9.0/en/telemetry/telemetry_wifi.html), Oct. 2020.
- [59] PX4, *Mavlink messaging*, <https://docs.px4.io/master/en/middleware/mavlink.html>, Dec. 2020.
- [60] PX4, *Px4 flight modes overview*, [https://docs.px4.io/master/en/flight\\_modes/position\\_mc.html](https://docs.px4.io/master/en/flight_modes/position_mc.html), Jun. 2021.
- [61] PX4, *Using the ecl ekf*, [https://docs.px4.io/master/en/advanced\\_config/tuning\\_the\\_ecl\\_ekf.html](https://docs.px4.io/master/en/advanced_config/tuning_the_ecl_ekf.html), Jun. 2021.
- [62] RETIA, *Generate crackable wi-fi handshakes with an esp8266-based test network*, <https://null-byte.wonderhowto.com/how-to/generate-crackable-wi-fi-handshakes-with-esp8266-based-test-network-0236794/>, Feb. 2021.
- [63] skickar, *Esp8266wpa2handshake*, <https://github.com/skickar/Esp8266Wpa2Handshake>, Jan. 2020.
- [64] PX4, *Esp8266 wifi module*, [https://docs.px4.io/master/en/telemetry/esp8266\\_wifi\\_module.html](https://docs.px4.io/master/en/telemetry/esp8266_wifi_module.html), Apr. 2021.
- [65] S. Sanfilippo, *Hping*, <http://www.hping.org>.
- [66] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [67] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.
- [68] T. Lei, Y. Zhang, S. I. Wang, H. Dai, and Y. Artzi, “Simple recurrent units for highly parallelizable recurrence,” *arXiv preprint arXiv:1709.02755*, 2017.
- [69] T. Lei, “When attention meets fast recurrence: Training language models with reduced compute,” *CoRR*, vol. abs/2102.12459, 2021. arXiv: [2102.12459](https://arxiv.org/abs/2102.12459). [Online]. Available: <https://arxiv.org/abs/2102.12459>.
- [70] A. Nisar, J. A. Sue, and J. Teich, “Performance comparison between machine learnings based lte downlink grant predictors,” in *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, The Steering Committee of The World Congress in Computer Science, Computer ..., 2019, pp. 226–232.

- [71] X. Wu, *Performance evaluation, prediction and visualization of parallel systems*. Springer Science & Business Media, 2012, vol. 4.
- [72] danial-heinrich, *Arduino speedtest*, <https://github.com/daniel-heinrich/hmbd/blob/master/Arduino/speedtest/speedtest.ino/>, Mar. 2018.
- [73] Daniel, *Speed comparison for arduino uno/nano, due, teensy 3.5 and esp32*, <https://hmbd.wordpress.com/2016/08/24/speed-comparisons-for-arduino-unonano-and-due/>, Aug. 2016.
- [74] *Esp8266 technical reference*, May 2021. [Online]. Available: [https://www.espressif.com/sites/default/files/documentation/esp8266-technical\\_reference\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf).