SECURITY INVESTIGATION OF DRONE CONTROL ALGORITHMS

A DISSERTATION SUBMITTED TO THE
GRADUATE DIVISION OF THE
UNIVERSITY OF HAWAI‘I AT MĀNOA
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

ELECTRICAL ENGINEERING

JULY 2021

By

Wenxin Chen

Dissertation Committee:

Yingfei Dong, Chairperson
Gürdal Arslan
Galen Sasaki
Yao Zheng
Edoardo Biagioni

We certify that we have read this dissertation and that, in our opinion, it is satisfactory in scope and quality as a dissertation for the degree of Doctor of Philosophy in Electrical Engineering .

DISSERTATION
COMMITTEE

_____

Chairperson

_____

_____

_____

_____

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Yingfei Dong, for his guidance, patience, and help throughout my research and study at Department of Electrical Engineering, University of Hawai‘i.

I also would like to thank the members of my dissertation committee: Dr. Gürdal Arslan, Dr. Edoardo Biagioni, Dr. Galen Sasaki and Dr. Yao Zheng. Thanks for your advice and support.

In addition, I would like to thank Dr. Zhenhai Duan and Mr. Jianqiu Cao. They have collaborated with me in my dissertation research project.

Finally, my deep and sincere gratitude to my family for their continuous and unparalleled love, help and support.

ABSTRACT

While more and more autonomous vehicles and devices are deployed in our society, the security of these systems has raised serious concerns. Although many efforts have focused on their performance and reliability, more systematic research on their security becomes urgent and critical. Therefore, we explore this direction and select consumer drones as our subjects because we can access open-source drone systems (i.e., ArduPilot systems) such that we are able to conduct in-depth investigations of their control algorithms in both theory and practice.

As consumer drones have been abused in many incidents, protecting critical assets from consumer drone invasions has become increasingly challenging. While existing methods can interrupt an invading drone, none of them is able to accurately guide it to a desired location for safe handling. By exploiting the weaknesses identified in common state estimation methods and navigation algorithms of drones, and utilizing existing sensor attacking tools, in this research, we develop generic methods to compromise drone state estimation and position control in order to make malicious drones deviate from their targets. In general, an autonomous drone can be attacked at three levels: its onboard sensors, its state estimation, and its navigation algorithms.

Our first focus is to accurately manipulate a drone's state estimation by utilizing existing sensor attack tools. We propose several *False Data Injection (FDI)* attacks to quantitatively control the EKF-based estimation of 2-dimensional horizontal position, altitude, and magnetic states, and conduct comprehensive analyses on the proposed attacks. Our simulation results show the effectiveness of such attacks. We also propose countermeasures to deal with such attacks.

Furthermore, we focus on the navigation algorithms and develop the *Drone Position Manipulation (DPM)* attack based on the ability of precisely attacking drone sensors and

state estimation. DPM is able to accurately manipulate a drone's physical position and help us guide an invading drone away from its target to a redirected destination. In addition, we formally analyze the feasible range of redirected destinations for a given target. The proposed attacks are validated on the popular ArduPilot flight control system to show its effectiveness. This unique method of exploiting the entire stack of sensing, state estimation, and navigation control together enables the quantitative manipulation of flight paths, different from all existing methods. We also discuss countermeasures to deal with such attacks and illustrate potential solutions.

Because the weaknesses of common control algorithms investigated here are popular in many autonomous systems, the proposed attacks may also pose serious threats to the security of these systems. Utilizing different resources available on these autonomous systems, we are further investigating unique vulnerabilities and countermeasures in these environments while ensuring system performance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

In recent years, we have witnessed many new applications of consumer drones [69], including search-and-rescue, aerial imaging, environmental monitoring, infrastructure inspection, and package delivery. Unfortunately, consumer drones have also been abused in many incidents [44]. For example, a drone crashed into the White House grounds in 2015 [58]; malicious drones disturbed 1000+ flights with 140,000+ passengers around the UK's second largest airport (Gatwick) for three days during 2018 Christmas season; coalition forces faced many attacks by low-cost consumer drones in Iraq and Afganistan. Therefore, it is urgent to develop effective drone countermeasures, especially when more and more such autonomous devices are emerging in this IoT age. Note that we focus on consumer drones because of their broad deployment, and we do not consider high-end mission-critical (military) drones here because they have different resources and requirements. In the following, when we mention "drones", we mean "consumer drones".

Existing drone countermeasures are mostly developed by industry using direct physical methods, such as jamming a drone's radio control channels to trigger a drone into its fail-safe mode (e.g., landing when its control channel is lost), or shooting it down with a projectile. While such brute-force physical methods work well in many cases, they have serious limitations, e.g., not handling collateral damages well. If a drone carries a malicious payload, we certainly do not want it to land in a protected area. The best solution for this case is to guide the drone away to a designated area for safe handling. While several projects [20, 34, 48, 66] have demonstrated the feasibility of such attack, none of them is able to achieve concrete quantitative control, which is the focus of this research. Furthermore, as more robotic vehicles are developed for new applications, many similar security issues become serious concerns. In this research, we propose to systematically address

such issues by investigating the entire control stack to achieve accurate quantitative control for specific concerns, such as redirecting a drone.

Ideally, we like to gain complete control of an invading drone, e.g., by taking over its control channel (by cracking its encryption) or hacking into its control software. While several countermeasures have been developed to exploit specific drone settings, they required to compromise drone software, sensors, or communication channels [20, 25, 35, 45, 56, 59], which are difficult to achieve in practice. While these methods may work well on weak systems with known vulnerabilities, we cannot solely rely on such methods to counter drones, because the vulnerabilities may be easily patched.

Therefore, different from these methods, we will focus on a new challenge in this research: *we would like to accurately control a drone without depending on compromising its software or hardware.* To achieve this goal, we propose a holistic approach to explore the entire stack of sensing, state estimation, and navigation together, as presented in the following.

First, almost all consumer drones are dependent on guidance inputs, such as civil GPSs, IMUs, magnetometers, barometers, etc. However, the designs of these sensors focus more on the performance and cost rather than security-related issues. Recently, researchers have identified the vulnerabilities of some sensors and proves that they can be easily spoofed [34, 61, 66], which we name as the first-level attack. In this research, we assume that we can manipulate the measurements of GPSs, magnetometers, or barometers remotely. In particular, for GPS, we can utilize existing software-defined radio (SDR) tools to spoof the signals. Although anecdotes on military GPS spoofing attacks have been reported (such as the capture of US Sentinel drone by Iran [52]), the details have never been revealed. So, we consider these attacks on military drones are beyond the scope of this research; our focus is the civil GPS system on consumer drones. For MEMS magnetometers and barometers, though no one has showed their abilities of precisely sensor reading manipulation remotely

for now, however, state-of-the-art work has proved the possibility of IMU measurement manipulation. Due to the similar structures among these sensors, we believe these spoofing approaches can also be applied for the modification of the barometer and magnetometer readings. However, this is out of the scope in this research and will be our future work. Here we just assume to have the ability to accurately modify the barometer or magnetometer readings remotely.

Second, assume we can identify the type and model of an invading drone [9, 24, 37], we can then find out its state estimation and control algorithms. A reliable control system is critical to the successful operation of unmanned or autonomous vehicles such as drones. The normal operation of control system of a drone relies on accurate information of critical state variables (such as position, velocity, and attitude). However, the measurements of sensors often contains inaccuracies and uncertainties. To solve this problem, state estimation is introduced. State estimation is a process of providing relatively accurate estimations of unknown states based on series of measurements containing noise and other inaccuracies. It is an essential component of an (integrated) control system [31]. Kalman Filter (KF) and its variants such as extended Kalman Filter (EKF) are the most widely used state estimation algorithms employed by drone control systems. In recent few years, several researchers have tried different estimation methods for estimations of a specific state (altitude) in drone control systems, including First-order Low-pass Filter [70], MLE based on accurate and detailed error models [76] and so on. For details, please refer to Chapter 4.2.1. In our research, we aim at the vulnerabilities in these state estimation schemes in drone control systems as our first attack target.

Moreover, based on the position estimation, a drone control system usually uses an automatic navigation controller to manage its physical position in real time such that it follows a given flight path, and automatically guides it to a desired destination. Therefore, we further investigate drone navigation algorithms to figure out generic methods to accurately

manipulate drones' physical positions, which is our second target.

As these state estimation and navigation algorithms are designed mainly for control without considering security concerns, we have carefully analyzed them and identified their guidance inputs as the attack surface. Therefore, we focused on carefully constructing spoofed GPS inputs to exploit both state estimation and navigation algorithms for manipulating a drone's state estimation and corresponding physical states.

First, assume existing tools can help us accurately manipulate a drone's sensor measurements remotely; we first propose several *False Data Injection (FDI)* attacks (also denoted as the second-level attack) to quantitatively control the EKF-based estimation of 2D horizontal position, altitude, and magnetic states, and conduct comprehensive analyses on the proposed attacks.

Furthermore, based on the ability of precisely modifying a drone's sensor measurements and state estimation, we develop the *Drone Position Manipulation (DPM)* attack (also denoted as the third-level attack), which is able to accurately manipulate a drone's physical position and help us guide an invading drone away from its target to a redirected destination. In addition, we formally analyze the feasible range of redirected destinations for a given target. The proposed attacks are validated on one of the most popular open-source flight control systems, ArduPilot [1, 54] to show its effectiveness. Such a holistic solution allows us to integrate the vulnerabilities at three levels together and achieve accurate quantitative physical position control.

Although spoofing GPS to attack drones had been exploited in two other projects [34, 48], neither of them exploited the entire control stack as in this research. Their methods indeed gained some control of the drone but did not achieve as the accurate control as the proposed second-level FDI attacks and third-level DPM attacks.

In summary, our novel contributions in this research include: (1) We develop a theoretical model to help us achieve accurate manipulation of a drone's state estimation and phys-

ical position for specific concerns. In comparison, existing methods were able to disrupt a drone's mission but did not define a clear model or achieve quantitative control. (2) The proposed attacks exploit the entire stack of sensing, state estimation, and navigation control, while existing methods mostly focused on one or two layers. (3) The proposed attacks are validated on ArduPilot, arguably the most popular open-source flight control system (compared to Paparazzi [50] and OpenPilot [49]), to show their effectiveness; while existing methods are mostly evaluated on theoretical platforms. (4) The proposed attacks are not required to compromise the software or hardware of a drone as some existing methods required. (5) Because the state estimation and navigation control algorithms on consumer drones are also broadly used in many other autonomous systems, the proposed attacks may also pose serious threats to the security of these systems.

The remainder of this research is organized as follows. We will first discuss related work in Chapter 2. In Chapter 3, we will define the attack model, and provide background knowledge related to required tools for the proposed attacks and drone control systems. We will then present the second-level False Data Injection (FDI) attack in Chapter 4 and the third-level Drone Position Manipulation (DPM) attack in Chapter 5. We will conclude this research and discuss future research in Chapter 6.

CHAPTER 2
RELATED WORK

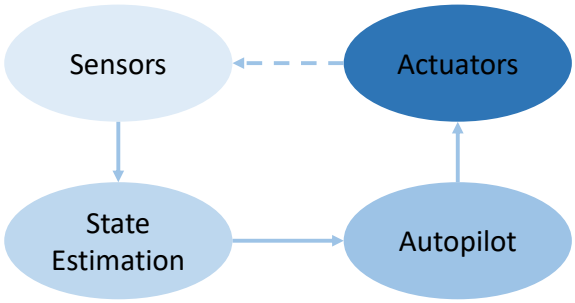


Figure 2.1: Common Drone Control Loop.

We will discuss the work related to our research in this chapter. As the flow chart of an autopilot system in Figure 2.1 shows, attacking a drone may happen at three levels on its onboard sensors [34, 61, 65, 66], its state estimation scheme [14, 15, 18, 30, 40], or its navigation algorithms [16, 48].

(1) **First-level attacks.** A consumer drone often uses an Inertial Measurement Unit (IMU) with MEMS sensors (e.g., rate-gyroscopes, accelerometers, or magnetometers) to measure three-dimension angular velocities, accelerations, and magnetic readings, respectively. Based on these measurements, a drone can estimate its system states including position, velocity, and attitude [28, 60, 74]. Different from mission-critical systems (such as military drones), consumer drones are usually equipped with low-end sensors with limited protection to reduce costs, which leaves many opportunities for hardware or software attacks.

Hardware attacks include selectively jamming GPS and radio control channels [8, 62, 64], or compromising drone hardware components (such as MEMS sensors) via acoustic approaches to disturb its normal operation [34, 61, 65, 66]. In particular, a high-accuracy covert spoofer was built by manipulating the signal delays in the physical layer to spoof GPS signals arriving at a drone GPS receiver [34]. The spoofer measures relevant delays

to the receiver within a few nanoseconds, and compensates for these delays by generating a slightly advanced version of the official GPS signals that the spoofer receives. Then, it gradually increases power to win the signal acquisition on the receiver over the official signals. This covert GPS spoofer is a pioneer work that can also help us implement our attack to manipulate GPS signals at the physical layer. [34] also defined a simple drone control model of EKF estimators and built their own drone simulator. Although their idea of spoofing GPS is similar to our approach, their work mostly focused on the manipulation in the physical layer, while our work focuses on state estimation and navigation control. We can use their physical layer solution for covert spoofing; our project is complementary to their project. There are a few key differences at the higher level: First, [34] built their own simple state estimator for analysis, while we exploit the mature state estimation schemes on an open-source system broadly used on many commercial drones, which makes our method more practical. Second, [34] did not consider the navigation control such that they do not have accurate control where the victim drone will fly to, while we exploit the navigation control and are able to accurately control the position of a drone.

(2) **Attacks on State Estimation.**

Attacks against state estimation have been examined in various control systems. In [40], Liu et al. first proposed an FDI attack against the state estimation in power grids. In particular, their schemes exploited the nature of state estimation and anomaly detection in power grid to successfully inject malicious values into certain states while bypassing the anomaly detector. However, the state estimation in a power system is usually simplified from a AC model to a DC model and focuses on the static components of the system; it is very different from the one we performed in drone systems. Following [40], several projects focused on false data injection attacks against static state estimation in power systems [10, 11, 36, 51, 57, 75].

Compared with static state estimation, dynamic state estimation can achieve estimations

for a series of real-time states such as rotor angles in power grids or on drones. There are also great efforts on dynamic state estimation and anomaly detection techniques in power systems [19, 30, 41, 46, 67]. which usually use the Kalman filter to estimate the states. However, only little efforts have been paid on FDI attacks against dynamic state estimation. In [47], the authors presented limited analytical results on KF-based dynamic state estimation in the presence of FDI attack. However, their methods require restricted assumptions about the system models (such as LTI system), which cannot be applied to the nonlinear EKF-based state estimation model in Chapter 3.2.3. In addition, they only give analytical results under constant injections on measurements, while injections in our attack schemes can vary. These approaches cannot be directly applied to the problems considered in our research.

(3) **Attack Drone Navigation Controls.** By analyzing the practical navigation code of ArduPilot, we identified weaknesses in the most popular path-following algorithms [63] as introduced in Section 3.2. In this research, we utilize the vulnerability in the state estimation and exploit the weakness of navigation to accurately manipulate a drone's position in order to guide it to a redirected destination. By carefully literature reviewing, we have found one project with similar attack goals. In [48], Noh et al. focused their attack on the navigation control on an earlier version ArduPilot 3.3. In particular, Noh et al. developed an adaptive GPS spoofing strategy to hijack a drone's position control. However, [48] could only achieve flight direction manipulation, with a non-negligible angular error magnitudes (around 9°). In contrast, we develop complete algorithms to manipulate its position with very small errors, and we also determined the feasible range of the redirected destination relative to an original destination. In summary, the key difference between our method with [48] is: while they can achieve some control over a target drone, they cannot achieve quantitative control as the proposed DPM.

(4) **Other Attacks against Drones.** Several software attacks mostly aim at the vul-

nerabilities of drone firmware [20, 25, 35, 59], such as insecure communication channels or software. For example, in [20], the authors developed several stealthy attacks against Robotic Vehicles (RVs) including drones to disrupt their position control without being detected. However, their attacks are depending on compromising RVs' firmware, which is impractical in real life. In addition, their attacks can only cause them to malfunction rather than quantitatively control their positions. However, the requirement of compromising the software or communication channels is out of scope of this research. A few other projects [8, 62, 64] showed the feasibility of various attacks to disrupt a drone's mission, but none of them is able to achieve as accurate position manipulation as the DPM. To the best of our knowledge, this research is the first to explore the entire stack of sensing, state estimation, and navigation control.

CHAPTER 3
BACKGROUND AND RESEARCH OUTLINE

While traditional control algorithms can handle most common system errors and ensure the smooth control of the system, very little research has been conducted in understanding their vulnerabilities under malicious attacks. Therefore, we focus our research on this direction and investigate the potential issues in these common control algorithms. In particular, we would like to explore how to mislead the consumer drones by compromising their sensors, state estimation, and navigation algorithms. To address this research problem, we first define the attack model. Then we provide background knowledge related to required tools for the proposed attacks and drone control system, including sensor measurement manipulation tools, popular state estimation and bad data detection techniques (especially the Kalman filter (KF) and its variant extended Kalman filter (EKF)), as well as the common navigation algorithm on drones. Finally we present the outline of the research.

3.1 Attack Model

To investigate the proposed problem, we first define the following attack model. As shown in Figure 3.1, we set up a restricted area around a critical asset to protect it from drone invasions. When we detect a drone invasion event using existing tools such as radars, we need to redirect it away from the asset (its target destination) to a redirected destination for safe handling, e.g., flying into a blast containment chamber. Because we can usually recognize an invading drone with existing approaches (via radar, radio or traffic profiling, or image processing [9,24,37]), we are able to identify its sensors (e.g., GPS) and firmware (including its state estimation and navigation algorithms). Assume that we have obtained the same model of drone and analyzed it ahead of time; we would like to propose a method to change its flight path towards a redirected destination. To achieve this goal, we need

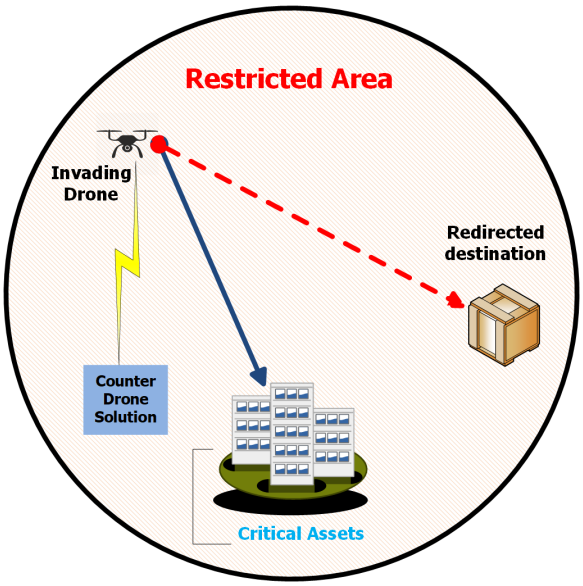


Figure 3.1: Restricted Area around a critical asset.

to carefully exploit the entire sensing, state estimation, and navigation control stack to achieve quantitative control of state estimation and eventually the physical position of a drone. Here the states we focus on include 2D horizontal position, altitude and magnetic ones. In this attack, we do not depend on compromising its software or hardware as some existing methods required.

Without loss of generality, to simplify our model, we assume that an invading drone is on autopilot, because of two reasons: First, because the drone operator usually does not want to expose itself, it has to turn off the control channel to avoid being triangulated based on its control signals. Second, because the control channel can be easily disrupted by the defense, the operator cannot depend on the channel to control the drone in the restricted area. So, autopilot is a natural choice.

To achieve the attack goal, we first introduce sensor manipulation attacks against barometer and magnetometer; then we focus on the tools of GPS spoofing (denoted as first-level attack). With the help of sensor manipulation techniques, we propose FDI attacks on state estimation to quantitatively manipulate the horizontal (2D) position, altitude, and magnetic states (denoted as second-level attack). Finally, utilizing these attacks, and by identifying

the vulnerabilities of the common navigation algorithms, we design attack schemes to manipulate the 2D physical positions of a drone. We have considered the attack on altitude and heading (related to magnetic states), and will investigate these direction in our future work.

3.2 Related Background

Here we introduce the background related to essential tools for proposed attacks and drone control system in detail.

3.2.1 Control Loop

As shown in Figure 2.1, the control loop of a drone contains four steps. Starting with sensor measurements, the system estimates related states and then passes the states to its navigation algorithms to determine how to adjust actuators for real-time control. Such a loop is usually completed in a fixed period, e.g., the default state update on ArduPilot is set to every 10 ms when IMU sensors generate new readings. We will introduce the sensor spoofing methods, the most popular state estimation algorithms, and the most common navigation algorithm in the following.

3.2.2 Sensor measurement manipulation

In the proposed research, we consider several attacks on different sensors, including barometer, magnetometer, and GPS. We will first introduce different existing sensor manipulation attacks for barometer and magnetometer; then we will focus on the tools of GPS spoofing.

3.2.2.1 MEMS Barometer and Magnetometer measurement manipulation

In this research, we propose some attacks on altitude and magnetic state estimation. For these attacks, we assume that, after a drone has entered the no-fly zone, we are able to manipulate the readings of MEMS barometer and magnetometer. We note that this is a relatively strong assumption. However, there have been significant progresses in the area of MEMS sensor reading manipulation recently. In particular, to maximize the detection sensitivity, the MEMS micro-structure is designed to be oscillated at a resonant frequency. However, this operation principle also brings an intrinsic vulnerability that the sensor will be susceptible to external vibrations (e.g., sound noise) around the resonant frequency [13, 22, 23]. In [61], the authors show that attackers can compromise the MEMS gyroscopes of a drone, by generating crafted noise with the same resonant frequency to degrade the performance of the gyroscopes. In [66], the authors makes further progresses, by providing acoustic injection attack schemes on MEMS accelerometers. Due to the similar structures among these sensors, we believe these attack approaches can also be applied for the modification of the barometer and magnetometer readings, and the further developments of our attacks on state estimation.

3.2.2.2 GPS Spoofing

To attack the 2D horizontal position estimation and physical position of a drone, we plan to craft well-designed fake GPS inputs with GPS spoofing tools. While a GPS message includes position, velocity, and other auxiliary data, we focus on the position and velocity data in this research, as other parameters usually have little impacts on the drone navigation algorithms examined here. We selected GPS spoofing as our attack method based on the following reasons. Because the 2D position estimation and navigation control of a drone depends on many dynamic factors, we have tried to identify which factors are the most

influential in such a complicated system. Through analyzing the 2D position estimation algorithms of ArduPilot, we identified two main factors among common drone sensors: GPS inputs and IMU measurements. When operated in the active GPS mode, the state estimator typically considers non-GPS navigation sensors as assistances to GPS data. This is also confirmed as a common solution in related work [34]. We have also verified this via both simulations and GPS spoofing on real ArduPilot drones. We have found that GPS inputs dominate the IMU measurements on an ArduPilot drone [12].

While we assume we can use existing tools to achieve covert GPS spoofing such as [34], we already conducted overt GPS spoofing on a SkyViper GPS drone for testing, whose firmware is a variant of ArduPilot [6]. We overpowered the civilian GPS signals using a BladeRF A9 card to transmit shifted GPS signals from a moderate distance [12]. The drone gained a 3D fix on the spoofed GPS signals for its position control. We have verified this by directly reading the GPS raw inputs from the drone via its MAVLink interface [42]. In our SITL simulations for drone physical position manipulation, we switched the GPS input of a simulated drone for a covert spoofing. In particular, we let the SITL drone take MAVLink GPS_INPUT messages [43] as its GPS input, in the same way as it receives GPS messages from a GPS-capable device. We then built a mission control program with the DroneKit Developer tools [26] to obtain drone real-time states, craft GPS_INPUT messages per GPS cycle, and send them to the drone via MAVProxy [7], same as a common ground control station (QGroundControl [53]) communicates with a real drone. With this method, we can test our attacks on state estimation and navigation control, which are the focus of this research.

3.2.3 Common State Estimation Algorithms

As sensor measurements may have errors, the navigation system deals with such errors using state estimation methods. A consumer drone conducts state estimation based on sen-

sor readings from accelerometers, gyroscopes, magnetometers, GPS signals, and barometers. Such state estimation methods enables low-cost sensors on drones to achieve required performance and robustness such that: faulty sensor measurements can be detected; the impacts of sensor noises and errors can be reduced; complementary sensing modalities can be combined (inertial, vision, air pressure, magnetic, etc.) to achieve robust estimations.

While state estimation methods are critical to the reliable and accurate estimation of vehicle states using low cost sensors, it is still the weakest link in the system reliability. Especially, most common state estimation methods emphasize performance and reliability, and do not consider malicious cyber attacks. State estimation mistakes can result in unexpected mission changes and loss of control [33, 54].

Extended Kalman Filters (EKF) and its variants are the most commonly-used state estimation methods in current systems [29, 60, 68, 72], providing fairly accurate state estimations. A few enhanced methods are proposed for critical altitude estimations based on other sensors (e.g., using images [27], differential GPS, or ultrasonic sensors [39]). However, because they usually need more resources that are not available on low-end consumer drones, we focus on the common EKF-based state estimation in this work. In the following, we will introduce how EKF works.

3.2.3.1 Kalman Filter (KF) and Extended Kalman Filters (EKF)

Before presenting EKF, we will first show its basic version - Kalman Filter (KF). The KF and its variants are the most popular estimation algorithms used in navigation systems. They estimate states based on time-series data. For example, for a flying drone, it needs to know its precise position, velocity and other movement states at each time point to navigate. Otherwise, inaccurate movement information may lead the drone to fall to the ground or hit against a tree. The on-board MEMS sensors can provide information about these movement states, but all contain various types of noises and other uncertainties. However, the position

and velocity states are actually correlated: we can predict the current position based on the position and velocity in a previous time slot. This kind of relevance can give us more information. By using this extra information, KFs reduce the inaccuracy within the raw data and provide a better estimation of these states.

KFs work in an iterative fashion to estimate dynamic system states based on both the new sensor measurement data and the predicted system states. In a KF, the system state model is commonly defined in two iterative steps [60, 72]: *Prediction* step and *Update* step. In the Prediction step, the KF predicts the estimation of current state based on its relationship with previous states. In the Update step, the KF updates the state with the current measurement based on the relationship between the indirect state and observable measurements. The updated state is a weighted average between the predicted state and the measurement, where the weight (Kalman gain) is calculated based on the covariance, a measure of the estimated uncertainty of the prediction. The whole procedure is executed in each time interval.

As an example, let us consider a drone that moves with a constant velocity in a one-dimensional space. In this system, we define the state (variable) vector as:

$$\mathbf{x}^{True}(t) = \begin{bmatrix} d(t) \\ v(t) \end{bmatrix}, \tag{3.1}$$

where $d(t)$ and $v(t)$ are the position and velocity at time t, respectively. Assuming the system states are updated at fixed time intervals of length T, then, $\mathbf{x}^{True}(t)$ satisfies the following equations:

$$\begin{aligned} d(t) &= d(t-1) + v(t-1)T + w_d(t), \\ v(t) &= v(t-1) + w_v(t), \end{aligned} \tag{3.2}$$

which can be represented in a matrix form:

$$\begin{bmatrix} d(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d(t-1) \\ v(t-1) \end{bmatrix} + \begin{bmatrix} w_d(t) \\ w_v(t) \end{bmatrix}, \tag{3.3}$$

or

$$\mathbf{x}^{True}(t) = \mathbf{A}\mathbf{x}^{True}(t-1) + \mathbf{w}(t), \tag{3.4}$$

where $\mathbf{A}$ is commonly referred to as the state transition matrix, and $\mathbf{w}(t)$ is the process noise following a zero mean normal distribution. The measurement vector $\mathbf{z}(t)$ has the following equation:

$$\mathbf{z}(t) = \mathbf{H}\mathbf{x}^{True}(t) + \mathbf{v}(t), \tag{3.5}$$

where $\mathbf{H}$ is the observation (or measurement) matrix, which maps system states to sensor measurements; $\mathbf{v}(t)$ is the observation noise, and also follows a zero mean normal distribution. With the above notations, the KF can be defined in the following two steps (here we do not consider control signal $\mathbf{u}(t-1)$):

Prediction:

$$\begin{aligned} \mathbf{x}^{-}(t) &= \mathbf{A}\mathbf{x}(t-1), \\ \mathbf{P}^{-}(t) &= \mathbf{A}\mathbf{P}(t-1)\mathbf{A}^{T} + \mathbf{Q}(t-1), \end{aligned} \tag{3.6}$$

Update:

$$\begin{aligned} \Delta(t) &= \mathbf{z}(t) - \mathbf{H}(t)\mathbf{x}^{-}(t), \\ \mathbf{K}(t) &= \mathbf{P}^{-}(t)\mathbf{H}^{T}(t)(\mathbf{H}(t)\mathbf{P}^{-}(t)\mathbf{H}^{T}(t) + \mathbf{R}(t))^{-1}, \\ \mathbf{x}(t) &= \mathbf{x}^{-}(t) + \mathbf{K}(t)\Delta(t), \\ \mathbf{P}(t) &= (\mathbf{I} - \mathbf{K}(t)\mathbf{H}(t))\mathbf{P}^{-}(t), \end{aligned} \tag{3.7}$$

where the superscript $^{-}$ indicates the corresponding vector or variable is predicted based on the estimated system states in the last time interval. In essence, in the Prediction step, the

KF predicts the state $\mathbf{x}(t)$ based on the estimation of $\mathbf{x}(t-1)$, while in the Update step, $\mathbf{x}(t)$ is updated based on the predicted value and the new measurement data $\mathbf{z}(t)$. The Kalman gain, $\mathbf{K}(t)$, assigns different weights for $\mathbf{x}(t)$ and $\mathbf{z}(t)$, to correct the state estimation. $\mathbf{P}(t)$ is the covariance matrix. $\mathbf{Q}(t)$ and $\mathbf{R}(t)$ are the process and measurement noise covariances, respectively, where $\mathbf{w}(t) \sim \mathcal{N}(0, \mathbf{Q}(t))$, and $\mathbf{v}(t) \sim \mathcal{N}(0, \mathbf{R}(t))$. In practice, $\mathbf{Q}(t)$ and $\mathbf{R}(t)$ are tuned for desired performance. KF is an iterative process which is executed whenever a new $\mathbf{z}(t)$ is available.

KFs are mostly applied to linear systems. To deal with a non-linear system, an EKF partitions the curve of differential functions into many small successive segments. As long as these segments are sufficiently small, they can be approximated by linear segments, which can then be processed by the KF. In an EKF, the system model can be rewritten as

$$\begin{aligned} \mathbf{x}^{True}(t) &= f(\mathbf{x}^{True}(t-1)) + \mathbf{w}(t), \\ \mathbf{z}(t) &= h(\mathbf{x}^{True}(t)) + \mathbf{v}(t), \end{aligned} \tag{3.8}$$

where the state transition matrix $\mathbf{A}$ is replaced by a state-transition function f, and the observation matrix $\mathbf{H}$ is replaced by a sensor function h. Then the prediction and update steps of EKF are generalized as:

Prediction:

$$\begin{aligned} \mathbf{x}^{-}(t) &= f(\mathbf{x}(t-1)), \\ \mathbf{P}^{-}(t) &= \mathbf{F}(t-1)\mathbf{P}(t-1)\mathbf{F}^{T}(t-1) + \mathbf{Q}(t-1), \end{aligned} \tag{3.9}$$

Update:

$$\begin{aligned} \Delta(t) &= \mathbf{z}(t) - h(\mathbf{x}^{-}(t)), \\ \mathbf{K}(t) &= \mathbf{P}^{-}(t)\mathbf{H}^{T}(t)(\mathbf{H}(t)\mathbf{P}^{-}(t)\mathbf{H}^{T}(t) + \mathbf{R}(t))^{-1}, \\ \mathbf{x}(t) &= \mathbf{x}^{-}(t) + \mathbf{K}(t)\Delta(t), \\ \mathbf{P}(t) &= (\mathbf{I} - \mathbf{K}(t)\mathbf{H}(t))\mathbf{P}^{-}(t), \end{aligned} \tag{3.10}$$

where $\mathbf{F}$ is the Jacobian matrix of the state transition function f, which contains the first derivative of the state transition function with respect to each state. Similarly, $\mathbf{H}$ is the Jacobian matrix of the sensor function h. $\mathbf{P}(t)$, $\mathbf{Q}(t)$ and $\mathbf{R}(t)$ have the same meanings as the ones in KF.

In practice, the implementation of EKF used in ArduPilot is a little different from the above. In particular, it does not estimate the whole 24 states in one time; instead, it estimates the states one by one. Accordingly, the variables in eq. 3.9 and 3.10 are a little different, e.g., for the estimation of each body magnetic field bias state $MagX$, $MagY$ and $MagZ$, $\mathbf{K}_{MAG}$ and $\mathbf{H}_{MAG}$ are 24×1 and 1×24 vector, respectively. However, this implementation is equivalent to the EKF model introduced in eq. 3.9 and 3.10.

Among the 24 states for drone control on ArduPilot 3.6, we focus on the 2D position, altitude, and magnetic estimation in this research. We will introduce the details of EKF-based state estimation for these states in the next chapter. The EKF on ArduPilot uses a common anomaly detection algorithm to determine if a sensor measurement is acceptable. We will also introduce the details of the anomaly detection algorithm in the next chapter.

For estimations of a specific state, namely altitude, other estimation techniques are also investigated, including First-order Low-pass Filters [70], MLE based on accurate and detailed error models [76] and so on. We will introduce them in Chapter 4.2.1.

3.2.4 Common Navigation Algorithm

The default waypoint navigation of ArduPilot is the *linear-track-based algorithm*, which is a Vector-Field (VF) based algorithm and also the most commonly-used path-following algorithm on drones, more accurate than others [63]. Although a drone's control loop adjusts quickly, it may still drifts away slightly from its scheduled flight track, due to various factors (e.g., wind disturbances). In order to fly along the scheduled flight track, a drone needs to constantly adjust its positions in small time intervals, e.g., the default fast-loop on

ArduPilot is 2.5 ms.

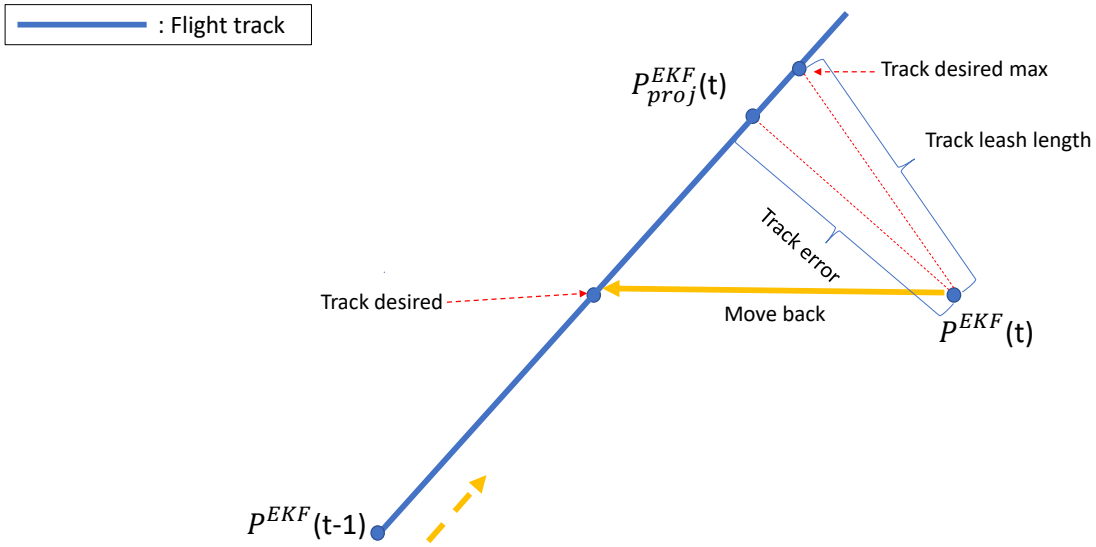


Figure 3.2: The linear-track-based algorithm.

Figure 3.2 shows how the drone should move back to the flight track under the navigation algorithm in ArduPilot. Assume a drone is scheduled to fly along the track from bottom left to upper right. In each time step, the system will calculate a $track\_desired$ position for the drone based on its position estimation at previous time step and the scheduled flight velocity. At time $(t-1)$, its position estimation is $P^{EKF}(t-1)$. At time t, the drone finds itself drifting away from the track and at the position $P^{EKF}(t)$. Here we call the distance between $P^{EKF}(t)$ and its projection on the track $P^{EKF}_{proj}(t)$ as the $track\_error$. Now the algorithm uses the $track\_leash\_length$ to determine how the drone should fly back to the original track, where the $track\_leash\_length$ is determined based on the velocity, acceleration, and current position of the drone. Specifically, the algorithm first determines a position on the track called $track\_desired\_max$. If $track\_leash\_length$ is larger than the $track\_error$, then $track\_desired\_max$ is the position on the track that has the distance of $track\_leash\_length$ from the current position $P^{EKF}(t)$ (there are two positions on the track that have the distance of $track\_leash\_length$ from the current position; here we just

call the one closer to the destination as the $track\_desired\_max$). If $track\_leash\_length$ is less than or equal to the $track\_error$, then the $track\_desired\_max$ is the projection of the current position $P^{EKF}(t)$ on the track. Now with $track\_desired\_max$, the algorithm can determine which position on the track the drone should fly back to: if $track\_desired\_max$ is closer to the destination than $track\_desired$, then the drone will fly back to $track\_desired$; otherwise, the drone will fly back to $track\_desired\_max$. In the figure, since the $track\_desired\_max$ is closer to the destination than $track\_desired$, the drone will fly back to the $track\_desired$.

3.3 Outline of the Research

By exploiting the weaknesses identified in common state estimation methods and navigation algorithms of drones, and with the help of existing sensor measurement spoofing tools, in this research, we develop generic methods to compromise drone position control in order to make malicious drones deviate from their targets. In particular, we consider attacking an autonomous drone in three phases: attacking its onboard sensors, attacking its state estimation, and attacking its navigation algorithms.

Assume existing tools can help us accurately manipulate a drone's sensor reading remotely; in Chapter 4 we propose several *False Data Injection (FDI) attacks* to quantitatively control the EKF-based estimation of 2-dimensional horizontal position, altitude and magnetic states. In particular, we design two state estimation attacks against each type of state: a maximum FDI attack that can maximize the deviation of state estimation, and a generic FDI attack that can achieve accurate manipulation of the state estimation. We conduct comprehensive analyses on the proposed attacks, and evaluate the attacks with simulations on the popular ArduPilot flight control system. We also investigate attacks on other altitude estimation methods.

In Chapter 5, based on the ability of precisely modifying a drone's sensor measurements and state estimation, we develop the *Drone Position Manipulation (DPM)* attack, which is able to accurately manipulate a drone's physical position and help us guide an invading drone away from its target to a redirected destination. Utilizing available GPS spoofing tools, we carefully craft the spoofed inputs to the flight control algorithms based on the redirected destination and other parameters. Furthermore, we formally analyze the feasible range of redirected destinations for a given target. The challenge here is: because the maximum spoofing range in a cycle is limited by a bad data detection threshold and the physical limitation of its GPS receiver, we have to carefully determine the spoofing signals within proper ranges in order to shift the drone position as much as we can, without triggering GPS-failure alarms. The proposed attack is validated on SITL simulations of ArduPilot to show its effectiveness in practical settings.

This unique method of exploiting the entire stack of sensing, state estimation, and navigation control together enables the quantitative manipulation of flight paths, different from all existing methods. Many other research problems still need to be investigated in the future, such as accurate manipulation of physical altitude and heading (related to magnetic states), or comprehensive countermeasures to stop the proposed attacks. We will introduce the details of our attacks in the following chapters.

CHAPTER 4
MANIPULATION OF DRONES' STATE ESTIMATION

In this chapter, we will present the second-level FDI attack on state estimation, including horizontal position, altitude, and magnetic state estimation in detail. We will first present the proposed attack schemes. Then based on the characteristics of the attack results, we design some countermeasures to these attacks. This work lays a solid theoretical foundation for the third-level DPM attack presented in Chapter 5.

4.1 Proposed Attacks for Horizontal (2D) Position Estimation

In this section, we focus on designing general methods to compromise a drone's 2-dimensional horizontal position estimation (simplified as 2D position estimation in the next). A drone's 2D position estimation provides information on its real-time horizontal location, which is fundamental to its navigation control. Inaccurate 2D position estimation may lead a drone to drift away from its original flight track, resulting in mission failure or even a crash. Here our attack goal is to maximize the deviation of its position estimation, or just precisely reset the estimation to a specific value, by feeding well-designed "manipulated" data to drone position estimation algorithms. In the attack schemes, we assume that we can access the real-time EKF-based estimated state and several related parameters for theoretical analysis.

4.1.1 Attacks on EKF-based 2D Position Estimation

EKF in ArduPilot estimates the 2D position and altitude separately. The model for 2D position estimation is described in Section 3.2.3, where

$$\begin{aligned} f(\mathbf{x}(t-1)) &= \mathbf{x}(t-1), \\ h(\mathbf{x}^{-}(t)) &= \mathbf{x}^{-}(t). \end{aligned} \tag{4.1}$$

Therefore **F** and **H** here are the identity matrix.

The EKF on ArduPilot uses a common anomaly detection algorithm to determine if a sensor measurement is acceptable. For 2D position measurements, it checks if the following condition is true:

$$inn_N{}^2 + inn_E{}^2 \leq (var_N^{inn} + var_E^{inn}) \cdot \tau, \tag{4.2}$$

Here τ is a pre-set threshold, $inn_i = \mathbf{z}_i(t) - \mathbf{x}_i^{-}(t)$ is the difference between a prediction and a measurement (a.k.a., the innovation) for i= N, E (for the two dimensions North and East), var_N^{inn} and var_E^{inn} are the variances of inn_N and inn_E, respectively. The values of var_N^{inn} and var_E^{inn} are calculated based on the covariance matrix of the EKF and the GPS position accuracy information; they can be regarded as constants in a steady-state system, based on our observations and many previous works such as [3,30,32,34,55,71,72]. Since the drone is in a steady state, τ and $(var_N^{inn} + var_E^{inn})$ can be regarded as constant. Then, assume we only attacking one direction N or E (in this case, the innovation of the position in other direction will be close to 0), eq. (4.2) can be simplified as the following:

$$\left|\mathbf{z}_i(t) - \mathbf{x}_i^{-}(t)\right| \leq \lambda, \tag{4.3}$$

For i= N, E. where $\lambda = \sqrt{(var_N^{inn} + var_E^{inn}) \cdot \tau}$ is a constant parameter. With the above settings, our goal is to maximize the deviation of state estimations without being detected.

We call this attack a *maximum False Data Injection (FDI) attack*. As one realization of this attack, the attacker can make the compromised estimation as large as possible.

To achieve this attack, in every time step t, the malicious measurements $\mathbf{z'}_i(t)$ is set as follows:

$$\mathbf{z'}_i(t) = \mathbf{x}_i^-(t) + \lambda. \tag{4.4}$$

This is a basic attack with a specific goal. In the following, we will analyze the attack effects. Based on the analysis, we develop the second attack – *generic FDI attack*.

Since it has reached the steady state, $\mathbf{K}(i)$ has converged to a constant. Then, in the maximum FDI attack, according to eq. (3.6),(3.7), (4.1) and (4.4), we have

$$\mathbf{x}_i(t) - \mathbf{x}_i(t-1) = \mathbf{K}_i \cdot \lambda \tag{4.5}$$

Since $\mathbf{K}_i \cdot \lambda$ can be regarded as constant, $\mathbf{x}_i(t)$ will increase **linearly**. Therefore

$$\mathbf{x}_i(t) = \mathbf{x}_i(t_0) + (t - t_0) \cdot \mathbf{K}_i \cdot \lambda, \tag{4.6}$$

where t_0 is the starting time of the attack.

With this result, we can achieve a more general purpose attack – *generic FDI attack*: an attacker usually has a specific goal, e.g., misguide a drone to a specific location. To achieve such a goal, it may want to make a drone drift a little at a time, without being detected. More specifically, in this attack, assume the attacker starts performing the attack at time t_0. Our goal is to make the state value $\mathbf{x}(t_0 + n)$ be a specific value ξ at time $t_0 + n$, after n time cycles.

Following the previous analysis, we know that after n time cycles, the compromised position estimation will be in the range:

$$[\mathbf{x}_i(t_0) - n\mathbf{K}_i\lambda, \mathbf{x}_i(t_0) + n\mathbf{K}_i\lambda]. \tag{4.7}$$

To set $\mathbf{x}_i(t_0+n)$ to be a specific value ξ at time t_0+n, a simple way is to divide $\xi - \mathbf{x}_i(t_0)$ into n equal small values $\frac{\xi - \mathbf{x}_i(t_0)}{n}$ and allocate it to each attack cycle. Based on the previous analysis, to achieve the attack goal, we set the malicious measurement as:

$$\mathbf{z'}_i(t) = \mathbf{x}_i^-(t) + \frac{\xi - \mathbf{x}_i(t_0)}{n\mathbf{K}_i}, \tag{4.8}$$

for $t = t_0+1, t_0+2, \ldots, t_0+n$. We note that, using this method, an attack cannot be achieved if the attack requires ξ to be outside the range $[\mathbf{x}_i(t_0) - n\mathbf{K}_i\lambda, \mathbf{x}_i(t_0) + n\mathbf{K}_i\lambda]$.

If we would like to manipulate the 2D position estimation in any direction, we just need to decompose the target position estimation ξ into North and East sub-components $\xi(N)$ and $\xi(E)$, then apply the above generic FDI attack respectively. What calls for special attention is that such ξ should not alert the bad data detector.

4.1.2 Attack with Coarse Manipulation of GPS

In the above attack schemes, we assume that we are able to manipulate the GPS data precisely so as to achieve the desired attack objectives. However, such a requirement may be too costly or unrealistic. Fortunately, it is still possible to perform the attacks even with coarse manipulation of the GPS data. Though in practice, a noise is generally normally distributed, to simplify the analysis, here we assume that a desired value $\mathbf{z}'_i(t)$ is always with a noise which is uniformly distributed between $[-b, b]$, where b is a constant parameter. We acknowledge that this is a strong assumption, but here our goal is to introduce the general idea of how to reduce the impact of GPS manipulation noise. We will investigate the attack with coarse manipulation under normal distributed noise in the future. In addition, we still require that the manipulation will not trigger the anomaly detector.

Generic FDI attack In our generic FDI attack, our goal is to set $\mathbf{x}_i(t_0 + n)$ to be ξ at time $t_0 + n$. Since coarse manipulation has noise, we need to make compensations for the "coarse" compromised estimation at the next time cycle. In particular, if the actual malicious measurement at time cycle $t_0 + 1$ is $\mathbf{x}_i^-(t_0 + 1) + \frac{\xi - \mathbf{x}_i(t_0)}{n\mathbf{K}_i} - \mathbf{r}_{t_0+1}(i)$, where $\mathbf{r}_{t_0+1}(i)$ is a random noise added to the desired measurement data, then the injected measurement at time cycle $t_0 + 2$ should be set to $\mathbf{x}_i^-(t_0 + 2) + \frac{\xi - \mathbf{x}_i(t_0)}{n\mathbf{K}_i} + \mathbf{r}_i(t_0 + 1)$, which compensates the noise $\mathbf{r}_i(t_0 + 1)$ from the previous time cycle, but may bring another noise $\mathbf{r}_i(t_0 + 2)$. Similarly, the measurement at time cycle $t_0 + 3$ will compensate the noise brought at cycle $t_0 + 2$. Generally speaking, to perform the generic FDI attack under coarse manipulation, the compromised measurement at time t should fill up the gap caused by the noise in the previous time cycle: $\mathbf{z'}_i(t) = \mathbf{x}_i^-(t) + \frac{\xi - \mathbf{x}_i(t_0)}{n\mathbf{K}_i} - \mathbf{r}_i(t - 1)$. In this way, the noise contained in the estimation at time cycle $t_0 + n$ will be at most $b\mathbf{K}_i$. Note that the malicious value at each time cycle cannot be set very large in order to avoid alerting the anomaly detector, which means all values should be within $[\mathbf{x}_i^-(t) - \lambda + b, \mathbf{x}_i^-(t) + \lambda - b]$ when attacking one direction.

Maximum FDI attack For the maximum FDI attack with the noisy manipulation of GPS data, we cannot directly apply the method outlined above, with which the magnitude of injected value could be large enough to trigger the anomaly detector. To perform the maximum FDI attack under noisy manipulation of GPS data, a straightforward way is to set the malicious measurement $\mathbf{z'}_i(t)$ to be $\mathbf{x}_i^-(t) + \lambda - b$ instead of $\mathbf{x}_i^-(t) + \lambda$. However, the cost of noisy manipulation is that the magnitude of the compromised estimation will be reduced.

4.1.3 Countermeasures

As discussed in the above, the anomaly detection algorithm in ArduPilot cannot detect the proposed FDI attacks on the EKF-based 2D position estimation. Another advanced detection method - chi-squared test, has been widely applied to defend against FDI attacks on KF-based estimation. This type of detector may work well against the proposed FDI attacks. However, it usually requires significant computational overheads and makes it impractical to be applied in simple drone flight control systems. In this subsection, we propose a novel detector that can effectively detect the proposed FDI attacks against EKF-based 2D position estimation with low computational overheads, which can then prevent the attacks on autopilot algorithms.

The new detector is designed based on the statistical characteristics among the innovations in different time steps. In particular, we know that the innovation Δ_k in time step k follows a Gaussian distribution:

$$\Delta_i(t) = \mathbf{z}_i(t) - h(\mathbf{x}_i^-(t)) \sim N(0, \mathbf{R}_{i,i}(t)), \tag{4.9}$$

where $\mathbf{R}(t)$ is the measurement covariance matrix (the innovation $\Delta(t)$ under the FDI attacks, however, usually does not follow this distribution). In addition, the innovations in different time steps can be regarded as independent. Then by the Chebyshev's Inequality, for each dimension in $\Delta(t)$, we have:

$$P(\left|\frac{\sum_{j=1}^{t}\Delta_i(j)}{t}\right| < \epsilon) \geq 1 - \frac{\mathbf{R}_{i,i}(t)}{t^2 \cdot \epsilon^2}, \tag{4.10}$$

for i=N, E (which corresponds to two dimensions of the innovation vector $\Delta(t)$, respectively). If we let ϵ be $\frac{\tau_{pd}}{t}$, then eq. (4.10) becomes:

$$P(\left|\frac{\sum_{j=1}^{t}\Delta_i(j)}{t}\right| < \frac{\tau_{pd}}{t}) \geq 1 - \frac{\mathbf{R}_{i,i}(t)}{\tau_{pd}^2}, \tag{4.11}$$

where τ_{pd} is a threshold. As long as τ_{pd} is large enough, $P(\left|\frac{\sum_{j=1}^{t}\Delta_i(j)}{t}\right| < \frac{\tau_{pd}}{t}) \approx 1$. Based on this observation, we let the proposed detector check whether the following statement is true:

$$\left|\frac{\sum_{j=1}^{t}\Delta_i(j)}{t}\right| < \frac{\tau_{pd}}{t}. \tag{4.12}$$

To implement this detector, in each time step in the estimation procedure, we need to calculate an additional variable - the mean of the innovation $\bar{\Delta}_k$ as:

$$\bar{\Delta}_i(t) = \begin{cases} \Delta_i(1) & \text{if } t = 1; \\ \frac{\bar{\Delta}_i(t-1)*(t-1)+\Delta_i(k)}{t} & \text{if } t \geq 2. \end{cases} \tag{4.13}$$

In this implementation, the computational overhead increases very little.

In the following, we use the maximum FDI attack as a simple example to test the effectiveness of the proposed detector. According to eq. (4.4), $\Delta(j) = \lambda$ for $j = 1, 2, \ldots$. Then at time step t, the proposed detector checks whether the following statement is true:

$$\left|\frac{\sum_{j=1}^{t}\Delta_i(j)}{t}\right| = \lambda < \frac{\tau_{pd}}{t}. \tag{4.14}$$

In this case, when the time $t \geq \frac{\tau_{pd}}{\lambda}$, the detector will be alerted.

4.2 Proposed Attacks for Altitude Estimation

In this section, we present our ideas for attacking three popular altitude estimation methods -1): EKF-based altitude estimation; 2): altitude estimation based on accurate sen-

sor noise models; and 3): First-order Low-pass altitude estimation in drone navigation systems. We focus on designing general methods to compromise the drone altitude estimation, by exploiting the weakness of common altitude estimation approaches.

4.2.1 Models

4.2.1.1 EKF-based altitude estimation Methods

In ArduPilot, the EKF-based altitude estimation has a similar model as the 2D position estimation. Specifically, in EKF-based altitude estimation,

$$\begin{aligned} f(\mathbf{x}(t-1)) &= \mathbf{x}(t-1), \\ h(\mathbf{x}^{-}(t)) &= \mathbf{x}^{-}(t). \end{aligned} \tag{4.15}$$

For the anomaly detection algorithm, it checks if the following condition is true:

$$inn^2 \leq var^{inn} \cdot \tau, \tag{4.16}$$

Here the innovation $inn = \mathbf{z}(t) - h(\mathbf{x}^{-}(t))$. Similarly, we can consider var^{inn} as a constant in a steady state. Then, eq. (4.16) can be simplified as the following:

$$|\mathbf{z}(t) - h(\mathbf{x}^{-}(t))| \leq \lambda, \tag{4.17}$$

where $\lambda = \sqrt{var^{inn} \cdot \tau}$ is a constant parameter.

Due to the similarities between the models for EKF-based 2D position and altitude estimation , we can use the same attack schemes presented in Sec. 4.1 to quantitatively control EKF-based altitude estimation. For details, please check Sec. 4.1.

4.2.1.2 Altitude Estimation based on Accurate Sensor Noise Models

In [76], the authors provide a novel direction for altitude estimation when the altitude of a vehicle does not change dramatically, e.g., in a cruise control mode, or on a flat plane, or on a gentle slope. One advantage of the proposed approach is that it improves estimation accuracy and provides tighter confidence bounds from GPS and barometer sensors, without the need for calibration. Their approach is based on the knowledge of relatively accurate noise models of GPS and barometer readings. In particular, the noises of altitude data from GPS and barometers have totally different characteristics: GPS readings usually provide more accurate information on the absolute altitude, while barometer measurements are more accurate on relative altitude changes. The GPS altitude measurement g usually follows a normal distribution: $g \sim \mathcal{N}(a, \sigma_g^2)$, where a is the true value of altitude; while the barometric altitude can be modeled as: $b \sim \mathcal{N}(a + \Delta, \sigma_b^2)$, where Δ is the unknown constant bias of barometer measurements. σ_g^2 can be directly obtained from GPS readings, while σ_b^2 is not provided by the barometer. In addition, σ_b^2 is much smaller than σ_g^2. The details of this estimation method is presented in the following.

Assume all GPS and barometer measurements are sampled at fixed intervals. To estimate the parameters of g, they use MLE, based on the last n GPS samples. The sample index is in a reserved order: 1 is the current time interval, such that we have a slide window of size n with the current interval as interval 1. The pooled variance and the sample mean at time step i are:

$$s_g^2[i] = \frac{1}{n} \sum_{j=i-n+1}^{i} \sigma_g^2[j], \tag{4.18}$$

$$\mu_g[i] = \frac{1}{n} \sum_{j=i-n+1}^{i} g[j], \tag{4.19}$$

where $g[j]$ is the jth altitude sample, and $\sigma_g^2[j]$ is the jth standard deviation sample. The

mean follows the normal distribution: $\mu_g[i] \sim \mathcal{N}(a, \frac{s_g^2[i]}{n})$.

Next, they estimate the parameters of b (for barometer) based on the last m barometer samples. The sample mean and the sample variance (biased) are

$$\mu_b[i] = \frac{1}{m} \sum_{j=i-m+1}^{i} b[j], \tag{4.20}$$

$$s_b^2[i] = \frac{1}{m} \sum_{j=i-m+1}^{i} (b[j] - \mu_b[i])^2, \tag{4.21}$$

where $b[j]$ is the jth altitude sample. Here we assume $m \geq n$, since barometer readings are always available, but GPS readings may be not. The mean follows: $\mu_b[i] \sim \mathcal{N}(a+\Delta, \frac{s_b^2[i]}{m})$.

The bias $\Delta[i]$ can be estimated as

$$\hat{\Delta}[i] = \mu_b[i] - \mu_g[i]. \tag{4.22}$$

Because $\hat{\Delta}[i] \sim \mathcal{N}(\Delta, \frac{s_b^2[i]}{m} + \frac{s_g^2[i]}{n})$, we can get the estimation for sample i as:

$$\hat{a}[i] = b[i] - \hat{\Delta}[i], \tag{4.23}$$

and

$$\hat{a}[i] \sim \mathcal{N}(\mu_b[i] - \Delta, s_b^2[i] + \frac{s_b^2[i]}{m} + \frac{s_g^2[i]}{n}). \tag{4.24}$$

For Gaussian distributed variables, the tolerance interval is measured by the number of standard deviations, D. The probability that a value is within a region around the mean with a width of $2D$ standard deviations is $erf(\frac{D}{\sqrt{2}})$, where erf is the error function encountered in integrating normal distributions. Since $s_g^2[i]$ is usually much greater than $s_b^2[i]$, this method shrinks the tolerance intervals, compared to the estimation based solely on GPS

data.

4.2.1.3 First-order Low-pass Filter Model for Altitude Estimation

In [70], the authors utilize a naive first-order low-pass filter to get a coarse altitude estimation from barometer readings, since its noise is at a high frequency. In particular, they use the following equation to estimate the altitude:

$$\mathbf{x}(t) = \mathbf{G}\mathbf{x}(t-1) + (1-\mathbf{G})\mathbf{x}^b(t), \tag{4.25}$$

where $\mathbf{x}(t)$ is the state, $\mathbf{x}^b(t)$ is the barometer reading, and $\mathbf{G}$ is the gain which is set to be a constant as 0.9.

4.2.2 Countermeasures to EKF-based Altitude Estimation

As the attack methods for EKF-based altitude estimation are the same as the ones for EKF-based 2D position estimation presented in Sec. 4.1.1, here we only discuss the corresponding countermeasures.

As we showed before, the implemented anomaly detection algorithm in the ArduPilot flight control system cannot detect the proposed FDI attacks on the EKF-based altitude estimation. In Sec. 4.1.3, we have designed a detector that can effectively detect the proposed FDI attacks with low computational costs. In this subsection, we propose a novel detector exclusively for EKF-based altitude estimation, which utilizes the characteristics of the two sensors - GPS and barometers.

On *ArduPilot*, the flight control system usually estimates the altitude from either barometer or GPS measurements. That is, only one of the two sensors is used for altitude estimation. Then the adversaries only need to compromise the single sensor that accounts for the altitude estimation. From the above analysis, we can see that the altitude estimation

will be far away from the true value, when under attack. That means, the compromised estimation will also be far away from the measurement from the other sensor. Based on this observation, we design this detector.

Without loss of generality, assume the flight control system choose barometer readings for the KF-based altitude estimation. The system should further check whether the following condition is true:

$$|\mathbf{z}_{baro}(t) - \mathbf{z}_{GPS}(t)| \leq \tau \cdot \sqrt{\sigma_g^2 + \sigma_b^2} + |\epsilon|\,, \tag{4.26}$$

where $\mathbf{z}_{baro}(t)$ and $\mathbf{z}_{GPS}(t)$ are barometer and GPS measurements, respectively; $|\epsilon|$ is the upbound of the absolute value of the barometric reading bias; τ is a pre-set threshold. σ_b^2 and σ_g^2 are the variances of barometer and GPS readings, respectively. σ_g^2 can be directly obtained from GPS inputs, and σ_b^2 can be estimated using the method described in Sec. 4.2.1.2. $|\epsilon|$ should be estimated beforehand or set empirically. The detector is designed based on the following fact [76]:

$$\mathbf{z}_{baro}(t) \sim \mathcal{N}(a + \Delta, \sigma_b^2);$$
$$\mathbf{z}_{GPS}(t) \sim \mathcal{N}(a, \sigma_g^2).$$

In addition, $\mathbf{z}_{baro}(t)$ and $\mathbf{z}_{GPS}(t)$ are independent. Therefore,

$$\mathbf{z}_{baro}(t) - \mathbf{z}_{GPS}(t) \sim \mathcal{N}(\Delta, \sigma_g^2 + \sigma_b^2). \tag{4.27}$$

4.2.3 Attacking Accurate Sensor Noise Models based Altitude Estimation

In this method, the authors use MLE to estimate the altitude $\hat{a}[i]$. A credible estimation of $\hat{a}[i]$ depends on the accurate estimations of other parameters. Base on this observation, we proposed the following attacks.

4.2.3.1 Attacks by Modifying Barometer Readings

In this attack, we assume that we are able to manipulate barometric readings. We are presenting the theoretical results under this assumption in the following. Specifically, we first use an example to show that altitude estimation can be influenced by the manipulation of barometer readings. Then we investigate the optimal attack strategy under certain assumptions.

Depending on our ability to manipulate the barometer, many different bad data injections can be performed. For example, a straightforward attack is as follows. We let the malicious barometer reading samples as:

$$b'[i] = b[i] + i \cdot c, \tag{4.28}$$

where $i \cdot c$ is the injected value on the ith sample, and c is a constant. Then

$$\begin{aligned}
\mu'_b[i] =& \frac{1}{m} \sum_{j=i-m+1}^{i} b'[j], \\
=& \frac{1}{m} \sum_{j=i-m+1}^{i} (b[j] + j \cdot c), \\
=& \frac{1}{m} \sum_{j=i-m+1}^{i} b[j] + \frac{1}{m} \sum_{j=i-m+1}^{i} j \cdot c, \\
=& \mu_b[i] + \frac{2i - m + 1}{2} \cdot c,
\end{aligned} \tag{4.29}$$

$\Delta'[i]$ will be estimated as

$$\begin{aligned}\hat{\Delta}'[i] =& \mu'_b[i] - \mu_g[i], \\ =& \mu_b[i] + \frac{2i - m + 1}{2} \cdot c - \mu_g[i], \\ =& \hat{\Delta}[i] + \frac{2i - m + 1}{2} \cdot c.\end{aligned} \tag{4.30}$$

Then we can get the compromised estimation for sample i as:

$$\begin{aligned}\hat{a}'[i] =& b'[i] - \hat{\Delta}'[i], \\ =& b[i] + i \cdot c - \hat{\Delta} - \frac{2i - m + 1}{2} \cdot c, \\ =& \hat{a}[i] + \frac{m - 1}{2} \cdot c,\end{aligned} \tag{4.31}$$

which means a constant bias will be added to the estimated altitude after the proposed attack. The amplitude of the bias is proportional to $m - 1$ and c.

The above example shows that the altitude estimation can be seriously affected by modifying barometer data. In the following, we will explore how to maximize the attack consequences. Assume that the altitude estimation process will end at time step k. At time step $i \in \{1, 2, 3, \ldots, k\}$, the attackers can set the malicious barometer reading $b'[i] = b[i]+c[i]$, where $c[i] \in [-\tau_{baro}, \tau_{baro}]$ is called the attack injection. In addition, we choose the MSE to measure the attack consequences. That is, our goal is to maximize the following term:

$$E(\sum_{i=m}^{k} (\hat{a}'[i] - a)^2). \tag{4.32}$$

Now, we can first rewrite eq. (4.32) as:

$$
\begin{aligned}
E(\sum_{i=m}^{k}(\hat{a}'[i]-a)^2) =& E(\sum_{i=m}^{k}(b[i]+c[i]-\hat{\Delta}'[i]-a)^2),\\
=& E(\sum_{i=m}^{k}(b[i]+c[i]-(\mu_b'[i]-\mu_g[i])-a)^2),\\
=& E(\sum_{i=m}^{k}(b[i]+\mu_g[i]-a+c[i]-\\
& \frac{1}{m}\sum_{j=i-m+1}^{i}(b[j]+c[j]))^2),
\end{aligned}
\tag{4.33}
$$

In eq. (4.33), $b[i]$, $\mu_g[i]$ and a are given, then the only term we can control is $c[i] \in [-\tau_{baro}, \tau_{baro}]$. Then the optimal attack injection sequences are:

$$
\underset{c[1],c[2],\ldots,c[k]\in[-\tau_{baro},\tau_{baro}]}{\operatorname{argmax}} f(x), \tag{4.34}
$$

where $f(x) = E(\sum_{i=m}^{k}(b[i]+\mu_g[i]-a+c[i]-\frac{1}{m}\sum_{j=i-m+1}^{i}(b[j]+c[j]))^2)$. Eq. (4.34) can be further simplified as:

$$
\underset{c[1],c[2],\ldots,c[k]\in[-\tau_{baro},\tau_{baro}]}{\operatorname{argmax}} \sum_{i=m}^{k}\left(c[i]-\frac{1}{m}\sum_{j=i-m+1}^{i}c[i]\right)^2. \tag{4.35}
$$

How to solve eq. (4.35) will be our future work.

4.2.3.2 Attacks by Blocking GPS

Unlike a barometer, GPS signals are weak and may be easily affected. If we disable the GPS readings in most of the sample time, the number of valid GPS samples will be very small. Since the altitude estimation is obtained by

$$
\hat{a}[i] \sim \mathcal{N}(\mu_b[i]-\Delta, s_b^2[i]+\frac{s_b^2[i]}{m}+\frac{s_g^2[i]}{n}). \tag{4.36}
$$

a decrease of n will result in an increase of $s_b^2[i] + \frac{s_b^2[i]}{m} + \frac{s_g^2[i]}{n}$, which will expand the tolerance interval of $\hat{a}[i]$.

4.2.4 First-order Low-pass Filter

This method uses a naive first-order low-pass filter to estimate the altitude, which can be easily attacked: since the gain for the barometer readings is non-negligible, we can simply inject values on the barometer readings, which will influence the accuracy of the estimation significantly. In particular, since

$$\mathbf{x}(t) = \mathbf{G}\mathbf{x}(t-1) + (1-\mathbf{G})\mathbf{x}^b(t). \tag{4.37}$$

Then

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{G}\mathbf{x}(t-1) + (1-\mathbf{G})\mathbf{x}^b(t), \\ &= \mathbf{G}^2\mathbf{x}(t-2) + (1-\mathbf{G})\mathbf{x}^b(t) + \mathbf{G}(1-\mathbf{G})\mathbf{x}^b(t-1), \\ &\dots \\ &\approx (1-\mathbf{G}) \cdot \sum_{l=1}^{t} \mathbf{G}^{t-l}\mathbf{x}^b(l). \end{aligned} \tag{4.38}$$

Now let the malicious barometer readings $\mathbf{x}^{b'}(t)$ be

$$\mathbf{x}^{b'}(t) = \mathbf{x}^b(t) + \mathbf{c}, \tag{4.39}$$

then

$$
\begin{aligned}
\mathbf{x}^{'}(t) &= \mathbf{G}\mathbf{x}^{'}(t-1) + (1-\mathbf{G})\mathbf{x}^{b'}(t), \\
&= \mathbf{G}\mathbf{x}^{'}(t-1) + (1-\mathbf{G})(\mathbf{x}^{b}(t) + \mathbf{c}), \\
&= \mathbf{G}^2\mathbf{x}^{'}(t-2) + (1-\mathbf{G})(\mathbf{x}^{b}(t) + \mathbf{c}) + \mathbf{G}(1-\mathbf{G})(\mathbf{x}^{b}(t-1) + \mathbf{c}), \\
&\dots \\
&\approx (1-\mathbf{G}) \cdot \sum_{l=1}^{t} \mathbf{G}^{t-l}\mathbf{x}_l^b + \mathbf{c}, \\
&= \mathbf{x}(t) + \mathbf{c},
\end{aligned}
\tag{4.40}
$$

which means a constant injection on the barometer reading over time will result in the same injection on the altitude estimation.

4.2.5 Performance Evaluation for Altitude Estimation

In this section, we use simulations of real-world altitude estimation algorithms to evaluate the proposed attacks on EKF-based and sensor-model-based altitude estimation methods.

4.2.5.1 Attacks on EKF-based Altitude Estimation

Simulation Settings The simulations are based on the altitude estimation algorithm of the ArduPilot project [55]. We mainly compare the altitude estimations before and after the proposed attacks.

Our simulations are performed on `TestLog3` data given in [55]. In the experiments, without loss of generality, we perform the maximum FDI attack to make the attack effects more legible. The default threshold τ in the ArduPilot code is pre-set to be 5. In addition, all the sensors remain operational except the airspeed module, which is disabled manually

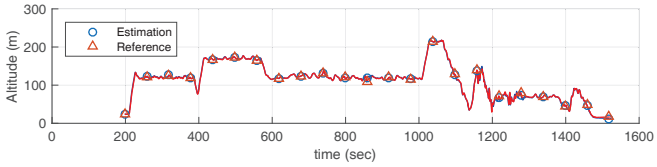


Figure 4.1: Altitude estimation before attack.

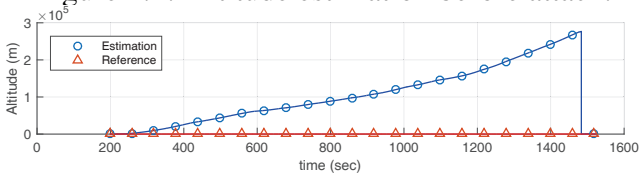


Figure 4.2: Altitude estimation after attack.

to show the attack effects.

In this simulation, we force the flight control system to perform altitude estimation at each time interval. (Under default settings, the estimation may not proceed at certain time steps if some measurements are missing.) In the simulations, we performed the attacks through the entire estimation process.

Simulation Results Fig. 4.1 and Fig. 4.2 depict the altitude estimations before and after the maximum FDI attack, respectively. The x-axis shows the simulation time and the y-axis shows the actual altitude and the altitude estimation (Note that the scales of the two y-axes in Fig. 4.1 and Fig. 4.2 are very different). In the figures, the blue line shows the estimations of altitude while the red line represents the actual values. In Fig. 4.1, without attacks, two lines mostly overlap. However, in Fig. 4.2, with the attacks, the estimation of altitude surges. In addition, the compromised estimations increase almost linearly. Furthermore, we can also validate the effectiveness of our attack from the altitude innovation data. From Fig. 4.3 and Fig. 4.4, we can see that the innovation of the altitude remains constant during the entire estimation process when under attack, which is as expected.

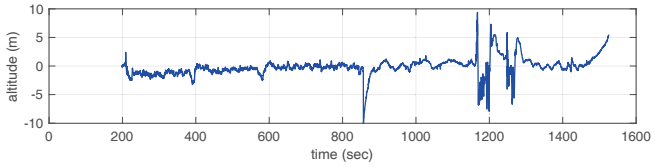


Figure 4.3: Innovation before attack.

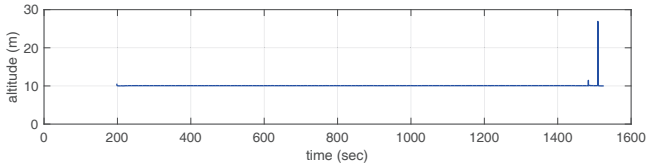


Figure 4.4: Innovation after attack.

4.2.5.2 Attacks on Sensor-model-based Altitude Estimation

Simulation Settings In this simulation, we evaluate the attack under the manipulation of barometer readings. We mainly focus on the results of the attack described in Sec. 4.2.3.1. All the test data are generated in MATLAB. In particular, there are 50 GPS and barometer samples that are available for altitude estimation. We assume $m = n = 5$, i.e., all the parameters are estimated based on the last 5 samples. The drone is set to keep its altitude at 100 m at all time. The GPS altitude measurement g follows a normal distribution: $g \sim \mathcal{N}(100, 10^2)$; and the barometer readings follow: $b \sim \mathcal{N}(110, 0.5^2)$, with a bias of 10 m. In the attack, we let the adversaries modify the barometer samples as:

$$b'[i] = b[i] + i \cdot 5. \tag{4.41}$$

Simulation Results Figure 4.5 illustrates the altitude estimations before and after the attack. We find that there are non-negligible constant differences between the altitude estimations before and after the attack, as expected.

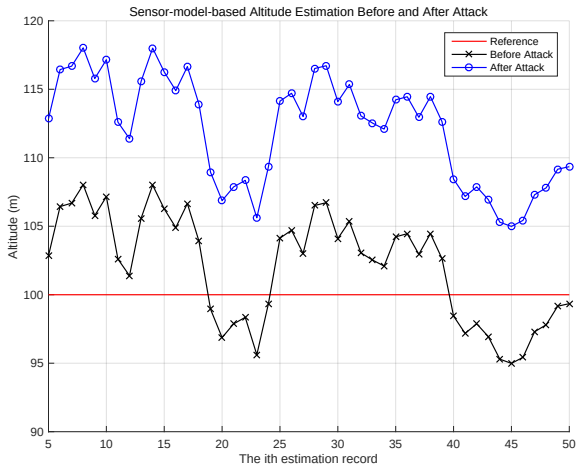


Figure 4.5: The altitude estimations before and after the attack (Sensor-model-based Altitude Estimation).

4.3 Proposed Attacks for Magnetic State Estimation

In this section we propose two FDI attacks - maximum FDI attack and generic FDI attack against EKF-based magnetic state estimation, just as the ones against 2D position estimation. In the attack schemes, we assume that we can access real-time EKF-based estimated state and several related parameters for theoretical analysis. The proposed attacks have great impacts on rotation motions (roll, yaw, pitch), which may result in significant consequences on the drone navigation, stability, and power consumption, among others, and could eventually jeopardize the flight mission of the drone.

4.3.1 EKF-based magnetic estimation Methods

In the ArduPilot EKF model, $\mathbf{x}$ is a 24-state vector. gyro bias offsets (X, Y, Z), gyro scale factors (X, Y, Z), Z accelerator bias, earth magnetic field (North, East, Down), body magnetic field (X, Y, Z), and wind velocity (North, East). Here we focus on the magnetic states, where $\mathbf{x}_{16}$, $\mathbf{x}_{17}$, $\mathbf{x}_{18}$ are earth magnetic field (North, East, Down): $MagN$, $MagE$, $MagD$, and $\mathbf{x}_{19}$, $\mathbf{x}_{20}$, $\mathbf{x}_{21}$ are biases of body magnetic field: $MagX$, $MagY$, $MagZ$. Let

$\mathbf{x}_{i:j}$ represent the entries in vector $\mathbf{x}$ from index i to index j, then $\mathbf{x}_{0:3}$, which defines the rotation quaternion $\mathbf{q} = \mathbf{x}_0 + \mathbf{x}_1\mathbf{i} + \mathbf{x}_2\mathbf{j} + \mathbf{x}_3\mathbf{k}$, represents the rotations of a drone. Here our attacks only focus on $\mathbf{x}_{19}$, $\mathbf{x}_{20}$, $\mathbf{x}_{21}$ - $MagX$, $MagY$, $MagZ$. In the next, we will use the term $\mathbf{x}$ and $\mathbf{x}_{19:21}$ interchangeably.

For $\mathbf{x}_{19}$, $\mathbf{x}_{20}$, $\mathbf{x}_{21}$, $f(\mathbf{x}) = \mathbf{x}$ and $\mathbf{F} = \mathbf{I}$ (identity matrix). Then the Prediction step becomes

Prediction:

$$\begin{aligned} \mathbf{x}^{-}(t) &= \mathbf{x}(t-1), \\ \mathbf{P}^{-}(t) &= \mathbf{P}(t-1) + \mathbf{Q}(t-1). \end{aligned} \tag{4.42}$$

In addition, $h(\mathbf{x}^{-}(t))$ is defined as follows:

$$h(\mathbf{x}_k^{-}) = Tnb * \mathbf{x}_{16:18}^{-} + \mathbf{x}_{19:21}^{-}, \tag{4.43}$$

where the *Tnb* matrix converts the earth fixed magnetic field in the NED coordinates to the XYZ body coordinates after the rotation, and has the following form:

$$\begin{pmatrix} \mathbf{x}_0^2 + \mathbf{x}_1^2 - \mathbf{x}_2^2 - \mathbf{x}_3^2 & 2(\mathbf{x}_1\mathbf{x}_2 + \mathbf{x}_0\mathbf{x}_3) & 2(\mathbf{x}_1\mathbf{x}_3 - \mathbf{x}_0\mathbf{x}_2) \\ 2(\mathbf{x}_1\mathbf{x}_2 - \mathbf{x}_0\mathbf{x}_3) & \mathbf{x}_0^2 - \mathbf{x}_1^2 + \mathbf{x}_2^2 - \mathbf{x}_3^2 & 2(\mathbf{x}_2\mathbf{x}_3 + \mathbf{x}_0\mathbf{x}_1) \\ 2(\mathbf{x}_1\mathbf{x}_3 + \mathbf{x}_0\mathbf{x}_2) & 2(\mathbf{x}_2\mathbf{x}_3 - \mathbf{x}_0\mathbf{x}_1) & \mathbf{x}_0^2 - \mathbf{x}_1^2 - \mathbf{x}_2^2 + \mathbf{x}_3^2 \end{pmatrix} \tag{4.44}$$

The bad data detector for magnetic states checks if the following condition is true:

$$inn_i^2 \leq var_i^{inn} \cdot \tau, \tag{4.45}$$

for $i = 1, 2, 3$ (which represents three dimensions of measurement data of a magnetometer). Here the inn_i is defined as follows:

$$inn_i = \mathbf{z}_i(t) - h(\mathbf{x}_i^{-}(t)), \tag{4.46}$$

Similarly, we can consider var_i^{inn} as a constant in a steady state. Then Eq. (4.45) becomes

$$\left|\mathbf{z}_i(t) - h(\mathbf{x}_i^-(t))\right| \leq \lambda_i, \tag{4.47}$$

where

$$\lambda_i = \sqrt{var_i^{inn} \cdot \tau}, \tag{4.48}$$

for $i = 1, 2, 3$.

4.3.2 Attack Methods

The attack methods for magnetic states are the same as the ones for 2D position estimation and altitude estimation. In particular, for maximum FDI attack, at every time cycle k, the malicious measurement $\mathbf{z'}_k$ should be set as

$$\mathbf{z'}_i(t) = h(\mathbf{x}_i^-(t)) + \lambda_i. \tag{4.49}$$

where λ_i is defined in eq. 4.48. Then in the maximum FDI attack, according to Eqs. (3.10), (4.42) and (4.49), we have

$$\mathbf{x}_{19:21}(t) = \mathbf{x}_{19:21}(t-1) + \begin{pmatrix} \mathbf{K}_{19}(t)\lambda_1 \\ \mathbf{K}_{20}(t)\lambda_2 \\ \mathbf{K}_{21}(t)\lambda_3 \end{pmatrix}, \tag{4.50}$$

which means that the differences between two adjacent estimations of $MagX$, $MagY$ and $MagZ$ is $\begin{pmatrix} \mathbf{K}_{19}(t)\lambda_1 \\ \mathbf{K}_{20}(t)\lambda_2 \\ \mathbf{K}_{21}(t)\lambda_3 \end{pmatrix}$. Since we know that $\mathbf{K}_{19:21}(t)$ and $\begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix}$ are roughly constants, it is expected that the estimation of $\mathbf{x}_{19:21}(t)$ will increase linearly after a short time. In the

following, we just simplify the notation $\mathbf{x}_{19:21}(t)$ as $\mathbf{x}(t)$, and $\begin{pmatrix} \mathbf{K}_{19}(t)\lambda_1 \\ \mathbf{K}_{20}(t)\lambda_2 \\ \mathbf{K}_{21}(t)\lambda_3 \end{pmatrix}$ as $[\mathbf{K}\lambda]$.

Similarly, the linearity of the maximum FDI attack results accordingly inspires the generic FDI attack. To set $\mathbf{x}_i(t_0 + n)$ to be the specific value ξ at time $t_0 + n$, a simple way is to divide $\xi - \mathbf{x}_i(t_0)$ into n equal small values $\frac{\xi - \mathbf{x}_i(t_0)}{n}$ and allocate this value to each attack cycle. In particular, if $\xi \in [\mathbf{x}_i(t_0) - n[\mathbf{K}\lambda]_i, \mathbf{x}_i(t_0) + n[\mathbf{K}\lambda]_i]$, we set the malicious measurement as:

$$\mathbf{z'}_i(t) = h(\mathbf{x}_i^-(t)) + \frac{\xi - \mathbf{x}_i(t_0)}{n\mathbf{K}_i}, \tag{4.51}$$

for $t = t_0+1, t_0+2, \ldots, t_0+n$. We note that, using this method, an attack cannot be achieved if the attack requires ξ to be outside the range $[\mathbf{x}_i(t_0) - n[\mathbf{K}\lambda]_i, \mathbf{x}_i(t_0) + n[\mathbf{K}\lambda]_i]$.

4.3.3 Realistic Attack Scenarios

FDI attacks on magnetic state estimation will greatly affect the navigation system of a drone. A drone under these FDI attacks may experience rapid and random rotations. These rotations may cause significant consequences in terms of attacking real drones. We propose two scenarios as examples to examine the consequences of the proposed attacks on drones:

- **Scenario 1**: A drone is going to take high-quality photos and videos at the restricted area, which requires a high level of stability.
- **Scenario 2**: A drone is scheduled for a relatively long mission, which is limited by its battery life. However, frequent and unnecessary rotations will quickly reduce the battery life.

A good criterion to measure the rotation of drones is the $roll$, $pitch$, and yaw data. For Scenario 1, we measure the stability of drone according to the variances and accelerations

of the $roll$, $pitch$, and yaw data. For Scenario 2, we would like to estimate the power consumption of the drone in terms of the following aspects:

- **Sum of rotation angles**: We will calculate the total rotation angles of the drone in terms of $roll$, $pitch$, and yaw, which is assumed to be proportionally related to the power consumption;
- **Total number of switching times**: A switch event occurs when the values of $roll$, $pitch$, and yaw switch from a negative value to a positive value (or from positive to negative). In general, a higher switching frequency will result in faster power consumption.

We will evaluate the effects of FDI attacks in the two scenarios in Section 4.3.4.

4.3.4 Simulation Studies for Magnetic State Estimation

4.3.4.1 Simulation Settings

In this section we evaluate the proposed FDI attacks through simulations based on the Matlab code used to derive the C++ code of the EKF-based state estimation algorithm of the ArduPilot project [55]. We are mainly interested in the attack results on the estimation of magnetometer data as well as the attack effects in the two scenarios described in Section 4.3.3.

Our simulations are mainly performed on `TestLog3` data given in the Matlab code. In all experiments, without loss of generality, we simulated the maximum FDI attack to evaluate the effectiveness of the attack scheme. In particular, in the first experiment in Section 4.3.4.2, we compare the differences between the original and compromised estimations of $MagX$, $MagY$ and $MagZ$. In the second experiment in Section 4.3.4.3, we evaluate the impact of Maximum FDI attacks on the $roll$, $pitch$, and yaw data of drone.

Then in Section 4.3.4.4, we use the $roll$, $pitch$, and yaw data to measure the stability and power consumption in the two attack scenarios.

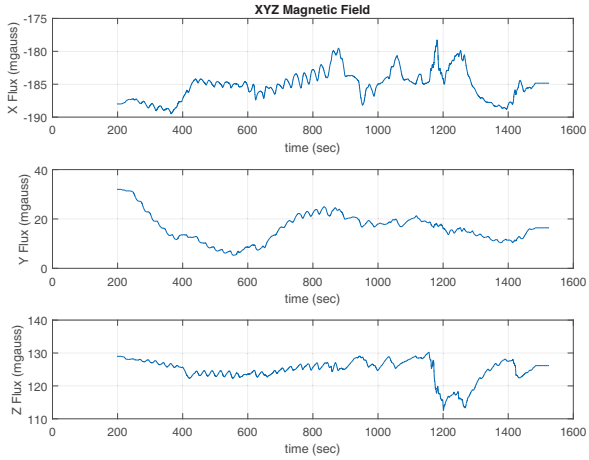
The default threshold τ in the ArduPilot Matlab code is pre-set to be 5^2. However, in simulations for attack scenarios 1 and 2, even though we can get meaningful attack results with $\tau = 5^2$, it cannot be easily seen from the figure. To make the attack effects more legible, we change the threshold τ into 50^2 for attack scenarios 1 and 2, which will result in a much larger magnitude of injection value on the magnetometer data. Note that in our simulations, the airspeed module is disabled. All the other sensors, unless otherwise stated, function correctly. In addition, the wind velocity is set to be 0. For all estimation processes, they start from `alignTime` around 200s, and will end at a pre-set time—approximately 1525s. The attacks are performed through the whole estimation processes.

4.3.4.2 Estimation of Magnetometer Data

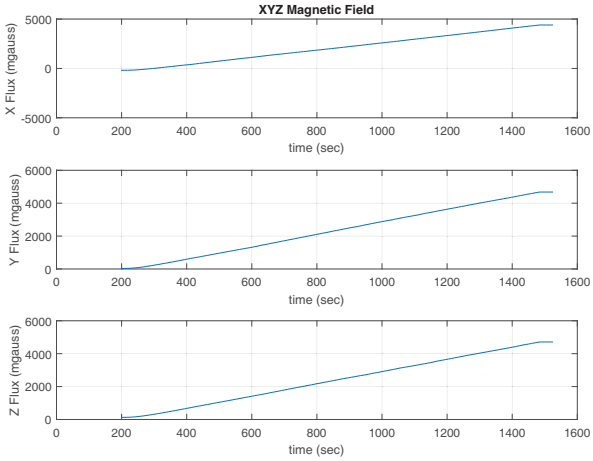
Figs.4.6a and 4.6b show the estimations for $MagX$, $MagY$ and $MagZ$ before and after the maximum FDI attack, respectively. From the figure, we can see that compared to the original estimation, the compromised magnetometer data increased significantly (from several hundred miligauss (mG) to around 5000 mG). In addition, it is easy to see that the compromised estimations of $MagX$, $MagY$ and $MagZ$ increase linearly as expected.

4.3.4.3 Estimations of $Roll$, $Pitch$ and Yaw

Fig.4.7 shows the influence of our FDI attack on the $roll$, $pitch$ and yaw of drones. From Fig.4.7b we can easily see that compromised magnetometer data caused by the attack can result in different estimations of rotation motions.

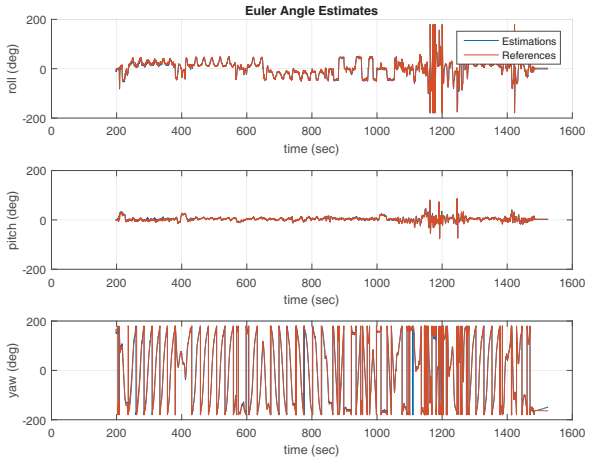


(a) The original estimations for $MagX$, $MagY$, and $MagZ$.

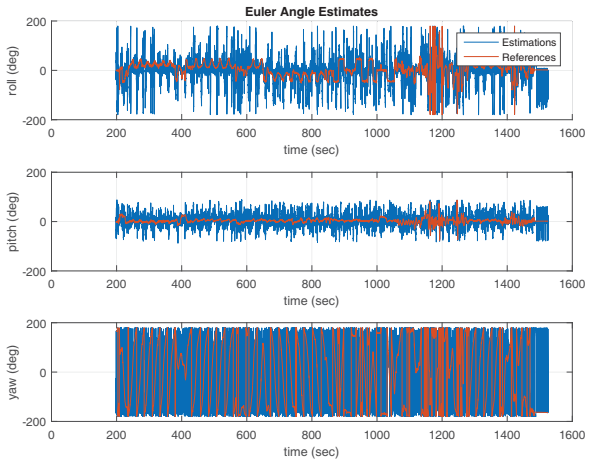


(b) The compromised estimations for $MagX$, $MagY$, and $MagZ$.

Figure 4.6: Attack results in terms of estimations of magnetometer data.



(a) The original estimations for $roll$, $pitch$ and yaw.



(b) The compromised estimations for $roll$, $pitch$ and yaw.

Figure 4.7: Attack results in terms of estimations of $roll$, $pitch$ and yaw.

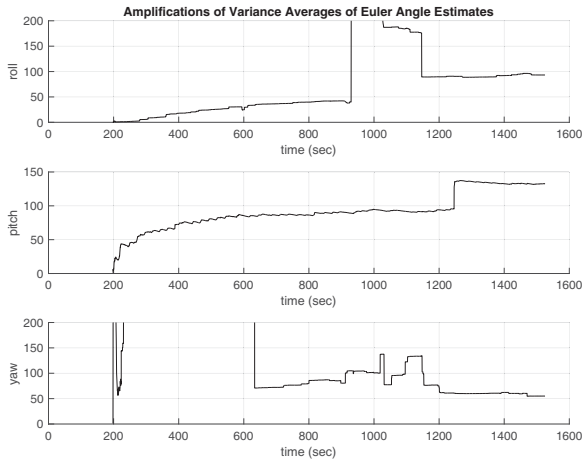


Figure 4.8: The amplification of variances in terms of $roll$, $pitch$, and yaw after attack.

4.3.4.4 Attack Results in Realistic Scenarios

In this subsection we evaluate the effects of our attack scheme in the two realistic scenarios outlined in Section 4.3.3.

For Scenario 1, we evaluate the stability of drones under our FDI attacks using the criterion of variances of $roll$, $pitch$, and yaw data. Fig. 4.8 shows the amplification ratios of variance averages in terms of $roll$, $pitch$, and yaw data after attack. From the figure we can see that the variances in all three axes has increased by approximately 100 times in the end, which shows that the attack can destabilize the drone intensely. Similarly, Fig. 4.9 displays the total rotation angles in terms of $roll$, $pitch$, and yaw. From the figure we can see that the drones get much more rotations in all three axes after an attack, which suggests that our attack schemes will drain power from the battery of drone much faster.

We also provide another criterion to measure the stability and power consumption, which considers the number of switching in terms of $roll$, $pitch$, and yaw (see Sec. 4.3.3). Fig. 4.10 shows that the number of switching of $roll$, $pitch$ and yaw all experience a surge after an attack, which again confirms the effectiveness of our attack schemes on destabilizing a drone and consuming drone battery power.

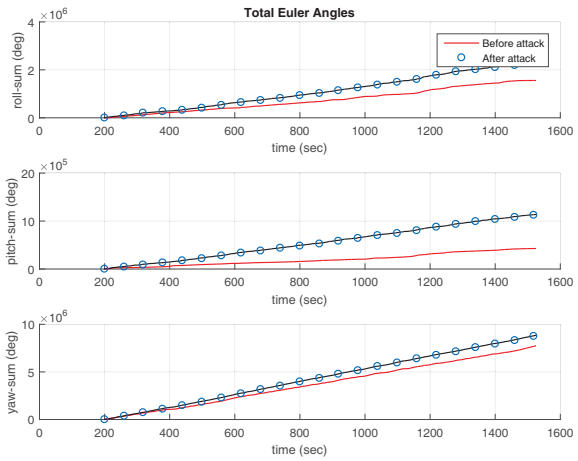


Figure 4.9: The total rotation angles in terms of *roll*, *pitch*, and *yaw* before and after attack.

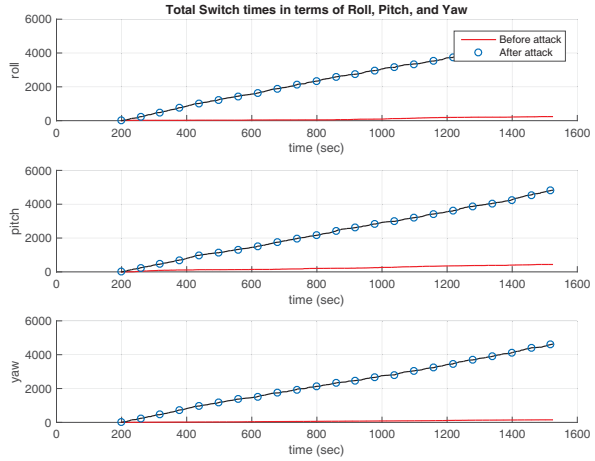


Figure 4.10: The number of switching times of *roll*, *pitch*, and *yaw* before and after attack.

4.4 Conclusion

In this chapter, we have carefully investigated popular state estimation algorithms for 2D position, altitude, and magnetic states, and identified potential attacks to compromise estimation of these states. We have proposed a maximum FDI attack and a generic FDI attack on the EKF-based 2D position, altitude, and magnetic state estimation. For altitude, we have also proposed two attacks on the model-based estimation. We have evaluated the proposed attacks with simulations. The simulation results have shown that the proposed attacks can significantly affect the state estimation and cause serious control issues for drones. Note that the attack schemes proposed in this chapter are only theoretical. In actual flight, to achieve the same goal, the procedure of the attacks could be different.

CHAPTER 5
MANIPULATION OF DRONES' PHYSICAL POSITION

With the help of GPS spoofing techniques and FDI attacks against EKF-based state estimation presented in Chapter 4, in this chapter we propose two third-level attack schemes - *basic Drone Position Manipulation (bDPM)* and *Practical Measurement-based Drone Position Manipulation (mDPM)*, which are able to accurately manipulate a drone's physical position and guide it to a desired location. For both schemes, we first introduce several important propositions, which are the theoretical foundations of the proposed attacks. Then we presents the attack algorithms in detail. We also analyze the feasible ranges of redirected destination under attack. In addition, we validate the DPM on ArduPilot, arguably the most popular opensource flight control system, to show its effectiveness in practical settings.

5.1 Basic DPM (bDPM) Attack

In this section, we will introduce the *basic Drone Position Manipulation (bDPM)* attack. To achieve the attack goal, we need to craft well-designed spoofed GPS inputs to the flight control algorithms based on the redirected destination and other parameters. For bDPM, it directly uses the estimated position states of a drone to craft spoofed GPS positions. Although obtaining EKF states is impractical on a real system, this method helps us better understand the proposed attack on a complicated control system and build a baseline analysis model of the attack. To further develop a practical solution, in the next section, we will introduce the *measurement-based Drone Position Manipulation (mDPM)* attack that uses measured drone positions to craft spoofed GPS positions.

In the following, we will present the bDPM attack and then illustrate its capability by formally analyzing its maximum feasible redirection range for a given original destination.

5.1.1 Theoretical Foundation of bDPM

Let us first present the theoretical foundation of bDPM, consisting of three important propositions.

Notations. We first define related notations for our discussion, as shown in Table 5.1. Because we focus on the drone position in a horizontal 2D plane, we consider the drone's *real velocity* as a 2D vector $V^r = (V^r_N, V^r_E)$, with a sub-component to the North, V^r_N meter/second (m/s), and a sub-component to the East, V^r_E m/s. For example, when a drone moves at 4 m/s to the Northeast, we observe a velocity vector V^r as (2.81, 2.81) m/s. In bDPM, for each GPS cycle, we build the spoofed GPS position inputs by first obtaining the estimated position state and then adding an injection to it with an injection vector of $I = (I_N, I_E)$ m/s ($(0.1 \cdot I_N, 0.1 \cdot I_E)$ m/GPS cycle), which has a sub-component to the North I_N, and a sub-component to the East I_E. Now when applying $I = (0, 10)$ m/s ((0, 1) m/GPS cycle) to a drone flying to the Northeast, i.e., injecting 0 m/s (0 m/GPS cycle) to the GPS position in the North direction and 10 m/s (1 m/GPS cycle) to the GPS position in the East, we observe that the drone's real position drifts away from its position estimation (perceived position) at a stable velocity at (0, -0.44) m/s, i.e., the drone's real position drifts to the West compared to its position estimation at 0.44 m/s as the result of the injections. We call this velocity as the drift velocity V^r_{drift}. With these notations, we introduce the following propositions:

Table 5.1: Notations in the bDPM. Each notation may have a subscript N or E representing its North or East component.

$K(t)$	Kalman gain
C^a	Attack coefficient
V^r	Real drone velocity when under attack
V^r_{drift}	Drone drift velocity
V^{EKF}	Drone velocity estimation
$I(t)$	Position injection rate at cycle t
$P^{EKF}(t)$	Drone position estimation at cycle t
$P^{GPS}(t)$	GPS position input at cycle t

Proposition 1. Drift velocity V^r_{drift} is proportional to injection rate I under a bDPM attack with an attack coefficient C^a defined as follows:

$$C^a = (C^a_N, C^a_E) = (\frac{V^r_{drift,N}}{I_N}, \frac{V^r_{drift,E}}{I_E}) \tag{5.1}$$

for $I_N \neq 0$ and $I_E \neq 0$. If I_N (or I_E) == 0, C^a_N (or C^a_E) = 0.

Proposition 2. For attacks on the same drone in the same environment, C^a remains unchanged for any proper injection size in any direction.

Propositions 1 and 2 are corollaries of the results in Chapter 4. In the bDPM, we choose the injection rate I as a fixed value in each GPS cycle, which results in nearly fixed innovations (the differences between the crafted GPS position input and the drone position estimation) in each EKF position estimation cycle, denoted as Δ (because $I \approx \Delta$). Since the Kalman gain $K(t)$ usually quickly becomes a constant in a steady state, the deviation of position estimation $V^{EKF}_{drift} = K(t) \cdot \Delta$ becomes constant. Because the deviation of position estimation will be corrected in each cycle, i.e., the position estimation will move towards the flight track by $-V^{EKF}_{drift}$ to correct this deviation in each cycle, the real position will also move by $V^r_{drift} = -V^{EKF}_{drift}$ in each cycle, which is a constant as well. Therefore, $C^a = \frac{V^r_{drift}}{I} \approx \frac{-V^{EKF}_{drift}}{\Delta} = -K(t)$, which can be regarded as the same constant for attacks on the same drone in the same environment.

We have further validated these propositions with ArduPilot SITL simulations. Figure 5.1 shows the relationship between the injection rate and the drift velocity observed during bDPM attacks on SITL. With the injection rate (x-axis) increasing from 4 m/s to 40 m/s, we observe that the drift velocity (y-axis) decreases proportionally to the injection rate with coefficient C^a. Furthermore, we have validated that Propositions 1 and 2 hold for injections in any direction in the 2D plane, when we apply a feasible constant injection rate. We measured that $C^a_E \approx -0.0455$ and $C^a_N \approx -0.0491$ in these simulations.

Proposition 3. When we apply a 2D injection rate $I = (I_N, I_E)$, the effect is equivalent to the combined effects of attacking only in the North direction with $I_1 = (I_N, 0)$ and attacking only the East direction with $I_2 = (0, I_E)$.

Proposition 3 (Decomposition Proposition) holds because the drone state estimation algorithms usually decompose the 3D positions and velocities into North, East, and Down sub-components [3]. Based on Propositions 2 and 3, we can simplify the analysis of the attack result under an injection rate I in any direction in the 2D plane, by decomposing the injection rate $I = (I_N, I_E)$ into $I_1 = (I_N, 0)$ and $I_2 = (0, I_E)$. Using measured attack coefficients C_E^a and C_N^a, we can find the drift velocities of the drone in the North and East directions: $V_{drift,N}^r = I_N \cdot C_N^a$ and $V_{drift,E}^r = I_E \cdot C_E^a$; then we can find the attack result with the injection rate I by combining the drift velocities in the two directions.

5.1.2 bDPM Attack

We illustrate how the bDPM attack redirects a drone to a specific redirected destination in a 2D plane, as shown in Figure 5.2. Given the redirected destination $R = (R_N, R_E)$, where we would like to guide the drone to reach, we define a redirection vector $\mathbf{DR} = (DR_N, DR_E)$ as $((R_N, R_E) - (D_N, D_E))$, the vector difference between the redirected destination (R_N, R_E) and the original destination (D_N, D_E). Assume we perform a FDI attack on a drone's position for n cycles to achieve **DR**; we evenly distribute the required injection in each cycle, i.e., in cycle t, we apply an injection $I(t) = (\frac{DR_N}{n \cdot C_N^a}, \frac{DR_E}{n \cdot C_E^a})$ on the current position state $P(t)$ to build its position input $P'(t)$, $0 \leq t < n$. As a result, the navigation algorithm observes that the drone had drifted away from the track, and it will make an adjustment to move it back to the track. After the adjustment, the system considers the drone has returned to the track at $P(t+1)$, but its real position is actually at $P^r(t+1)$. In this example, we only need 5 injection cycles to achieve the required redirection; after 5

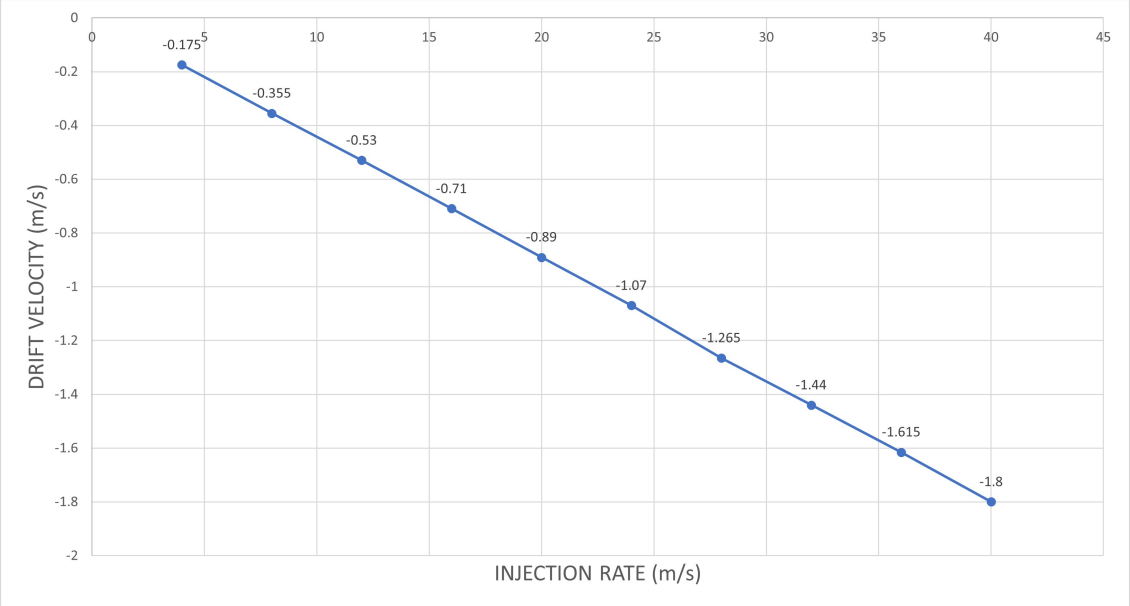


Figure 5.1: Relationship between the drift velocity and the injection rate. As the injection rate increases, we can see the drift velocity increases proportionally, and the attack coefficient C^a stays roughly constant.

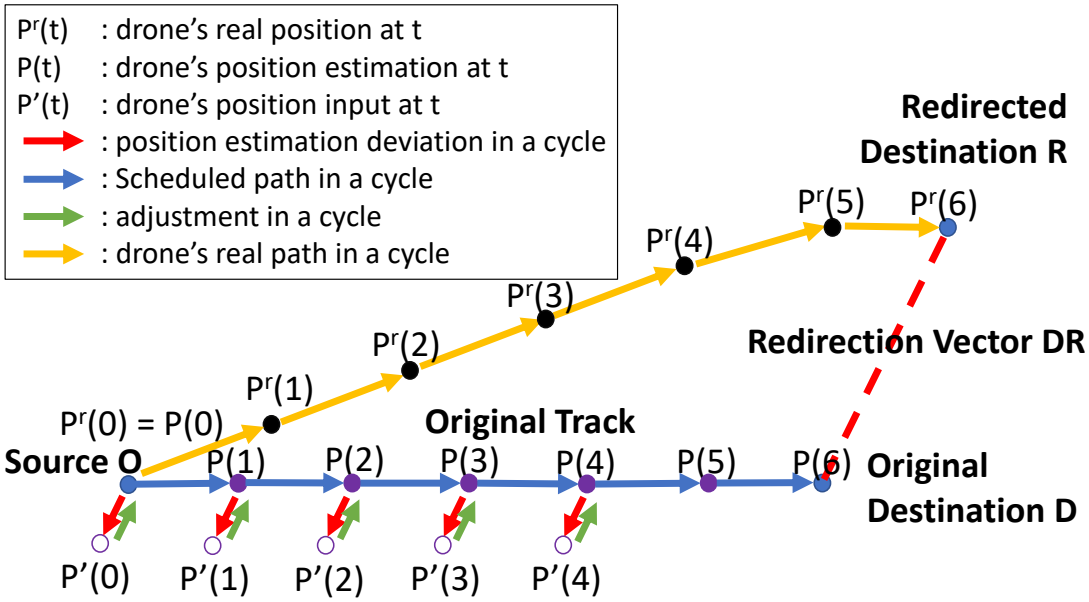


Figure 5.2: Illustration for the bDPM attack.

cycles, we stop injections and the drone will fly towards the redirected destination in a path parallel to the original track.

Similar to the above illustration, we introduce the main steps of bDPM in Algorithm 1. Given the flight track of a drone, a redirected destination, and drone position states, the attack builds a position injection in each cycle in order to lead the drone away from its original track to achieve the redirection, as explained in the above. The maximum injection rate $I^{max} = (I_N^{max}, I_E^{max})$ per cycle can be determined based on the parameters associated with the bad-data detector in theory and it also can be measured in practice. Attack coefficient $C^a = (C_N^a, C_E^a)$ is defined in Proposition 1, and can be measured in advance. A video demonstration of a bDPM attack on ArduPilot SITL is at Youtube [17], and we will evaluate the accuracy and feasible redirection range of bDPM in Section 5.3.

Algorithm 1: bDPM Attack Algorithm.

input: Original track from (O_N, O_E) to (D_N, D_E);
Redirected destination (R_N, R_E);
Drone position state estimation $P^{EKF}(t)$.

Initialization: $\{I(t)\} \longleftarrow \emptyset, t \longleftarrow 0$;
n_0 = the remaining number of cycles on the original track;
DR = $(DR_N, DR_E) \longleftarrow (R_N - D_N, R_E - D_E)$;
$n \longleftarrow max(\lceil \frac{DR_N}{I_N^{max} \cdot C_N^a} \rceil, \lceil \frac{DR_E}{I_E^{max} \cdot C_E^a} \rceil)$; find the total no. of injection cycles;
while $t \leq n_0$ **do**
    if $t < n$ **then**
        $I(t) \longleftarrow (\frac{DR_N}{n \cdot C_N^a}, \frac{DR_E}{n \cdot C_E^a})$; injecting until $t \geq n$;
        $P^{GPS}(t) = I(t) + P^{EKF}(t)$; build fake position inputs;
    else
        $P^{GPS}(t) = P^{EKF}(t)$; fly towards R, no injection;
    send $P^{GPS}(t)$ as position input;
    $t \longleftarrow t + 1$;

5.1.3 Feasible Range of Redirected Destination

Obviously, the above attack cannot redirect a drone to an arbitrary redirected destination without being detected, because the maximum physical position deviation per cycle is

constrained by the bad data detector. Therefore, we need to further figure out if a target redirected destination is feasible for a limited attack time under bDPM. In the following, we will analyze such a feasible range of redirected destinations to show the overall capability of bDPM, which can help us determine if a redirected destination is reachable or not.

Eq. 4.2 is the bad data detector for EKF-based 2D horizontal position estimation. $(var_N^{inn} + var_E^{inn}) \cdot \tau$ can be regarded as a constant λ in a steady state, and inn_N and inn_E are roughly equal to I_N and I_E. Plugging them into Eq. 4.2, for the boundary of the feasible range, we have

$$I_N^2 + I_E^2 = \lambda. \tag{5.2}$$

Then, based on Eq. 5.1, we have

$$(\frac{V_{drift,N}^r}{C_N^a})^2 + (\frac{V_{drift,E}^r}{C_E^a})^2 = \lambda, \tag{5.3}$$

or

$$(V_{drift,N}^r)^2 + (\frac{V_{drift,E}^r}{C_E^a/C_N^a})^2 = \lambda \cdot (C_N^a)^2. \tag{5.4}$$

Eq. 5.4 shows the feasible range of the redirected destination for one injection cycle is an ellipse with its center at the original destination and eccentricity $\sqrt{1 - \frac{(C_E^a)^2}{(C_N^a)^2}}$ (close to 0 in practice), as shown in Figure 5.3. After n injection cycles, the feasible range of the redirected destination will be

$$(\frac{R_x - D_x}{C_N^a})^2 + (\frac{R_y - D_y}{C_E^a})^2 = n^2 \cdot \lambda, \tag{5.5}$$

or

$$(R_x - D_x)^2 + (\frac{R_y - D_y}{C_E^a / C_N^a})^2 = n^2 \cdot \lambda \cdot (C_N^a)^2. \tag{5.6}$$

We will show concrete feasible ranges in Section 5.3.

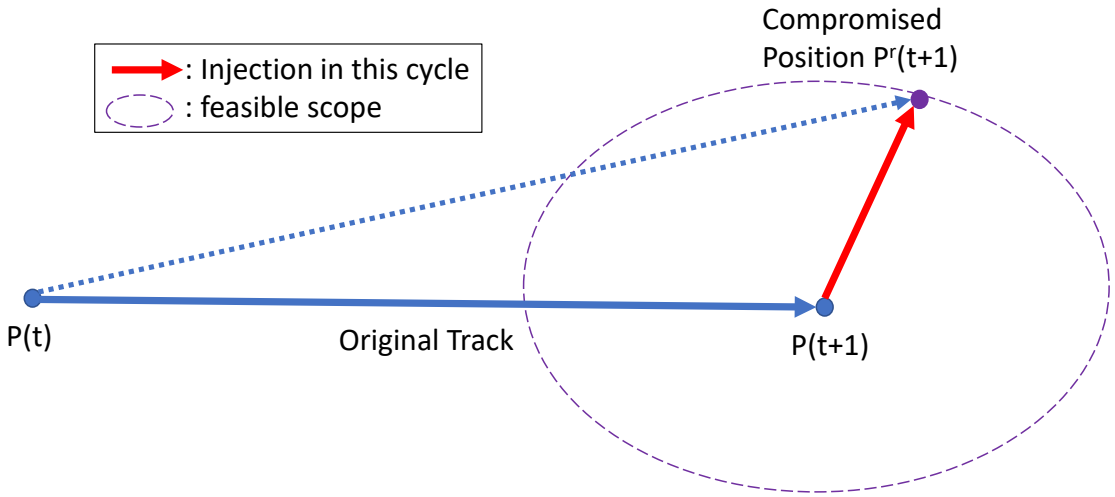


Figure 5.3: Feasible Range of the redirected position in one cycle.

5.2 Practical Measurement-based DPM (mDPM)

Although using the bDPM attack can help us define the previous analysis model, it is impractical because it is very hard to obtain the position estimation from a drone not under our control. Therefore, in this section, we further develop the mDPM attack that builds the GPS position inputs based on measured drone positions. In practice, many methods are available to measure the position and velocity of a drone from a distance [34], e.g., using radar or other localization methods. In a restricted area, we can deploy measurement tools to obtain drone positions and velocities, and use such measurements to build spoofed GPS inputs. In this research, we obtain the measured drone positions and velocities from the SITL simulator for our tests.

5.2.1 Identifying a Practical Injection Method

We first show why the previous constant injection method does not work well in the new practical setting. As shown in Fig. 5.4, the thick blue line in the middle is the drone's original track. When the drone's real position is at $P^r(t)$, we feed the drone with a compromised position input by applying a position injection to the East, such that the autopilot system believes that the drone deviates from the track to the East at $P^{GPS}(t)$. Consequently, the control algorithm will compensate the difference by moving the drone to the West, which makes the drone's real position moves further to the West at $P^r(t+1)$ in the next cycle. However, if we still use the same injection size to build the next position input: $P^{GPS}(t+1)$ is equal to the real position $P^r(t+1)$ plus the constant injection as in the bDPM, $P^{GPS}(t+1)$ will also move to the West, which makes it closer to the drone's position estimation $P^{EKF}(t+1)$ in this cycle. In the next cycle, the smaller difference between $P^{GPS}(t+1)$ and $P^{EKF}(t+1)$ leads to a shorter West drift of the drone's real position; and the following GPS position inputs crafted with the same method will continue to move towards the drone EKF position estimation, as the drone's real position moves to the West. After some n cycles, the difference between the crafted GPS position input $P^{GPS}(t+n)$ and the drone position estimation $P^{EKF}(t+n)$ will be close to 0. In the following, the

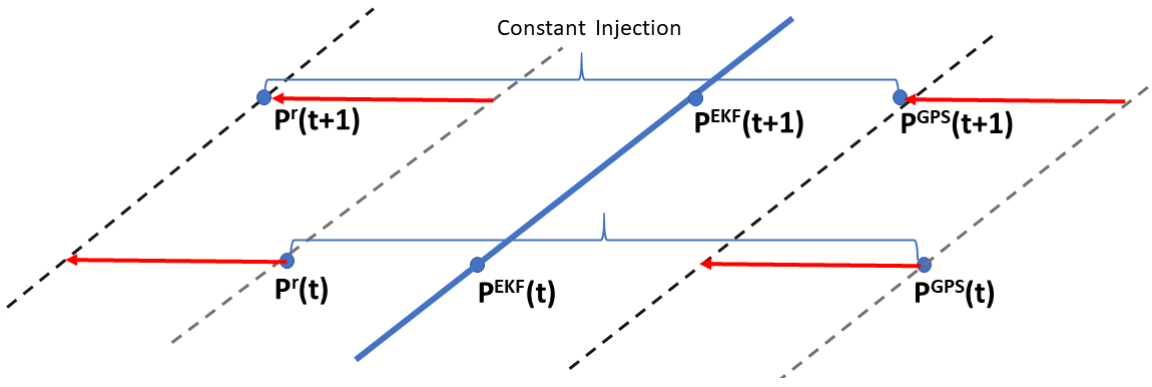


Figure 5.4: Crafting GPS position inputs based on measured drone positions with constant injection sizes.

drone's physical position will hardly drift, and the drone will move on a track *parallel to the original track at a fixed distance (equal to the injection size)*.

We have also validated the above process in the simulation. When the injection rate is constant, the drone's real position will first start to drift, and then quickly stabilize, i.e., the difference between its real position and its drone position estimation keeps unchanged. The drone flies parallel to the original track as we discussed in the above.

Table 5.2: Notations in mDPM.

$P^r(t)$	Drone's real position at cycle t
$P^{GPS}(t)$	Compromised GPS position input at cycle t
$P^{EKF}(t)$	Drone position estimation at cycle t
$\Delta(t)$	Innovation between position input $P^{GPS}(t)$ and drone position estimation $P^{EKF}(t)$

We analyze the details of this process in the following. We denote the difference between the crafted GPS position input and the drone position estimation (i.e., the innovation in drone state estimation) at cycle t as $\Delta(t) = (\Delta_N(t), \Delta_E(t))$; we denote the constant injection to the East on the position inputs as $I = (0, I_E)$. In time cycle 0, $\Delta(0) = (0, I_E)$, since the real position and the drone position estimation are the same at the beginning. In cycle 1, because the drone's real position moves by $(0, -K_E(t)\Delta_E(0))$ due to the innovation, (where $K_E(t)$ is the steady-state Kalman Gain in the East), the position input will also move by $(0, -K_E(t)\Delta_E(0))$, then $\Delta_E(1) = \Delta_E(0) - K_E(t) \cdot \Delta_E(0) = (1 - K_E(t)) \cdot \Delta_E(0)$. Similarly, $\Delta_E(2) = (1 - K_E(t)) \cdot \Delta_E(1), \ldots$; then, we have $\Delta_E(t) = (1 - K_E(t))^t \cdot I_E$. Furthermore, because $0 < K_E(t) < 1$, we then have

$$\lim_{t\to\infty} \Delta_E(t) = \lim_{t\to\infty} (1 - K_E(t))^t \cdot I_E \to 0. \tag{5.7}$$

This analysis shows us that the crafted GPS position input and the drone EKF position estimation will eventually converge. Since the difference between the GPS position input and the drone real position is the constant injection, the difference between the drone's real

position and the drone EKF position estimation will also stabilize.

New Attack Strategy. The above analysis shows that the previous constant injection method will have limited effects in this setting, because the maximum drift distance is limited by the injection size. However, as the injection size must be smaller than a threshold determined by the bad data detector, the drift distance is limited by the threshold. Therefore, we must explore a different attack strategy in mDPM.

After carefully analysis and testing, we propose to increase the injection rate in our attack linearly over time. In particular, assume the injection rate applied to the GPS position input is: $I = I^0 + X^I \cdot t$, i.e.,

$$I = (I_N, I_E) = (I_N^0 + X_N^I \cdot t, I_E^0 + X_E^I \cdot t) \tag{5.8}$$

where the base injection rate I^0 is a constant determined based on the drift velocity that we like to induce on the the drone, X^I is the increase of injection in each cycle, and t indicates the t-th attack cycle. We will introduce how to determine I^0 in Section 5.2.2. Under this attack strategy, we have the following propositions:

Proposition 4. Drift velocity V_{drift}^r is proportional to injection increase rate X^I in a mDPM attack with an attack coefficient $\hat{C}^a$ defined as follows:

$$\hat{C}^a = (\hat{C}_N^a, \hat{C}_E^a) = (\frac{V_{drift,N}^r}{X_N^I}, \frac{V_{drift,E}^r}{X_E^I}) \tag{5.9}$$

for $X_N^I \neq 0$ and $X_E^I \neq 0$; if $X_N^I = 0$, $\hat{C}_N^a = 0$; if $X_E^I = 0$, $\hat{C}_E^a = 0$.

Proposition 5. For attacks on the same drone in the same environment, attack coefficient $\hat{C}^a$ remains unchanged for any valid injection size in any direction.

Proposition 6. When applying an injection rate $I = (I_N, I_E)$, the effect is equivalent to the combined effects of attacking only in the North direction with $I_1 = (I_N, 0)$ and attacking

only the East direction with $I_2 = (0, I_E)$, respectively.

Propositions 4 to 6 are similar to the results in the bDPM scheme due to the drone control algorithms. In the following, we further explore the unique requirement of mDPM and discover interesting results about how the injection size should be increased.

Although there are many potential methods to build the compromised position inputs, most of them will not be effective due to the specific setting of drone control and bad-data detection. For example, simply increasing the injection rate may not work in mDPM. (For simplicity, we omit the subscripts indicating the directions in the following paragraph. The following analysis works for any direction.) In particular, when $I(t+1)-I(t)$ monotonically increases, we have $\lim_{t\to\infty}(I(t+1) - I(t)) \to \infty$; V_r^{drift} will also increase over time. When $I(t)$ increases but $I(t+1)-I(t)$ monotonically decreases, and $\lim_{t\to\infty}(I(t+1)-I(t)) \to 0$, the innovation will converge to 0, which results in the similar results as the constant injection case. Therefore, we have

Proposition 7. To successfully perform the mDPM attack, the injection rate should increase at least linearly, and the increase amount per cycle should have a finite upper bound.

Proof: we denote the innovation, i.e., the difference between the GPS input and the EKF position estimation at cycle t as $\Delta(t)$, and the position injection on the GPS input at cycle t as $I(t)$. Then, at cycle $t+1$, with Kalman gain $K(t)$ we have:

$$\Delta(t+1) = \Delta(t) - \Delta(t) \cdot K(t) + I(t+1) - I(t), \tag{5.10}$$

Eq. 5.10 is based on the following facts. First, we find the change of Δ in cycle $(t+1)$ as $\Delta(t+1) - \Delta(t)$. Since such a change cannot be directly obtained, we can decompose it into two components: (1) the change of the difference between the real position of a drone and its estimated position; (2) the change of the difference between the GPS position input

and the drone's real position. The first component is equal to $\Delta(t) \cdot K(t)$ because the real position moves away from the position estimation by $\Delta(t) \cdot K(t)$, due to the innovation $\Delta(t)$ at the last cycle. The second component is $I(t+1) - I(t)$ due to the change of the injection. Adding them up, we have the difference between $\Delta(t+1)$ and $\Delta(t)$, which leads to Eq. 5.10.

Based on Eq. 5.10, we analyzed the following three types of injection methods:

1. If $I(t)$ increases linearly, i.e., $I(t) = I^0 + X^I \cdot t$, $I(t+1) - I(t) = X^I$. We have $\Delta(t+1) = \Delta(t) - \Delta(t) \cdot K(t) + X^I$.

   If $-\Delta(t) \cdot K(t) + X^I > 0$, then $\Delta(t+1)$ will continue increase until $-\Delta(t) \cdot K(t) + X^I = 0$. Otherwise, if $-\Delta(t) \cdot K(t) + X^I < 0$, then $\Delta(t+1)$ will continue decrease until $-\Delta(t) \cdot K(t) + X^I = 0$. In either case, $\Delta(t+1)$ will be finally equal to $\Delta(t)$. Then, $\Delta(t+1)$ will keep unchanged afterwards. Because $\Delta(t+1)$ keeps unchanged but the injection increases linearly, the real drone position moves away from the GPS position input with a constant V^r_{drift}.

2. If $(I(t+1) - I(t))$ monotonically decreases, and $\lim_{t \to \infty} (I(t+1) - I(t)) \to 0$, we denote $I(t+1) - I(t)$ as $A(t+1)$. Therefore,

$$\Delta(t+1) = \Delta(t) \cdot (1 - K(t)) + A(t+1).$$

   Now given a time T ($T \gg 0$), we define another series

$$B(t) = \begin{cases} \Delta(t) & \text{t<T} \\ \Delta_{T-1} \cdot (1 - \bar{K(t)}) + A_T & \text{t=T} \\ B_{t-1} \cdot (1 - \bar{K(t)}) + A_T & \text{t>T} \end{cases}$$

where the $\bar{K(t)}$ is the upper bound of $K(t)$ for $t \geq T$, $A_T = I(T+1) - I(T)$, then we have

$$B(t) - \frac{A_T}{\bar{K(t)}} = (1 - \bar{K(t)}) \cdot (B_{t-1} - \frac{A_T}{\bar{K(t)}})$$

and

$$B(t) - \frac{A_T}{\bar{K(t)}} = (1 - \bar{K(t)})^{t-T+1} \cdot (B_{T-1} - \frac{A_T}{\bar{K(t)}})$$

for $t > T$.

Since $0 < \bar{K(t)} < 1$, then

$$\lim_{t \to \infty} B(t) - \frac{A_T}{\bar{K(t)}} \to 0,$$

and

$$\lim_{t \to \infty} B(t) \to \frac{A_T}{\bar{K(t)}}.$$

Because

$$\lim_{T \to \infty} A_T \to 0,$$

then

$$\lim_{t \to \infty} B(t) \to 0.$$

As $\Delta(t) \leq B(t)$, and $\Delta(t) > 0$,

$$\lim_{t\to\infty} \Delta(t) \to 0.$$

3. If $(I(t+1) - I(t))$ monotonically increases, and $\lim_{t\to\infty}(I(t+1) - I(t)) \to +\infty$, from Eq. 5.10, we know $\Delta(t+1) > I(t+1) - I(t)$. Therefore $\lim_{t\to\infty} \Delta(t) \to +\infty$. In practice, the system will be alerted by the bad data detector after the innovation is over a threshold, which leads to the attack failure.

Based on the analysis above, we can conclude that to successfully perform the attack, the injection rate should increase at least linearly, and the increase step per cycle should have a finite upper bound.

In summary, we need to carefully determine a proper increasing injection to move the drone away from its original track and not being detected. Although more complicated injection methods could be developed, we must carefully monitor the innovations caused by the injections, which is another complicated problem to be further explored in the future.

5.2.2 mDPM Attack

A mDPM attack has the similar procedure as the bDPM attack, but with the spoofed GPS position inputs constructed based on measured drone positions. The main steps of mDPM is shown in Algorithm 2. Based on the above analysis, we construct the spoofed GPS position inputs with a linearly-increased injection rate for mDPM attacks. Assume that we know the flight track of a drone in advance, we can build shifted GPS positions based on the drone's original destination, a redirected destination, and the maximum injection rate (which is determined by the bad-data detector and specific parameters of a drone and can also be measured). The vector difference between the redirected destination (R_N, R_E) and the original destination (D_N, D_E) is defined as **DR**. We first find the total number of attack cycles n for achieving this redirection vector. We determine $I_0 = (I_N^0, I_E^0)$ as $I_N^0 = \frac{V_{drift,N}^r}{C_N^a}$

Algorithm 2: mDPM Attack Algorithm.

input: Original destination (D_N, D_E);
Redirected destination (R_N, R_E);
Drone position measurements $P^m(t)$.

Initialization: $\{I(t)\} \longleftarrow \emptyset, t \longleftarrow 0$;
n_0 = the remaining number of cycles on the original track;
DR = $(DR_N, DR_E) \longleftarrow (R_N - D_N, R_E - D_E)$;
$n \longleftarrow max(\lceil \frac{DR_N}{I_N^{max} \cdot C_N^a} \rceil, \lceil \frac{DR_E}{I_E^{max} \cdot C_E^a} \rceil)$;
while $t \leq n_0$ **do**
    if $t < n$ **then**
        $I(t) \longleftarrow (\frac{DR_N}{n \cdot \hat{C}_N^a} \cdot t + I_N^0, \frac{DR_E}{n \cdot \hat{C}_E^a} \cdot t + I_E^0)$; injecting until $t \geq n$;
        $P^{GPS}(t) = I(t) + P^m(t)$; build fake position inputs;
    else
        $P^{GPS}(t) = I(n-1) + P^m(t)$; fly towards to R, injection size not increasing any more;
    send $P^{GPS}(t)$ as position input;
    $t \longleftarrow t + 1$;

and $I_E^0 = \frac{V_{drift,E}^r}{C_E^a}$, where $(V_{drift,N}^r, V_{drift,E}^r)$ is the drift velocity that we like to achieve, and the attack coefficient C^a is defined in Proposition 1. Under this setting of I_0, the innovation between the GPS input and EKF position estimation in the initial attack cycle will stabilize at this value and result in the drift velocity $(V_{drift,N}^r, V_{drift,E}^r)$ immediately. By simulation we find that even if we set I_0 to different values (e.g., 0), the innovation will still converge to the previous setting values $I_N^0 = \frac{V_{drift,N}^r}{C_N^a}$ and $I_E^0 = \frac{V_{drift,E}^r}{C_E^a}$ very quickly. In each attack cycle t, $0 \leq t < n$, we apply an injection $I(t) = (\frac{DR_N}{n \cdot \hat{C}_N^a} \cdot t + I_N^0, \frac{DR_E}{n \cdot \hat{C}_E^a} \cdot t + I_E^0)$ to the measured position. Attack coefficient $\hat{C}^a = (\hat{C}_N^a, \hat{C}_E^a)$ is defined in Proposition 4 and can be measured in advance.

5.2.3 Feasible Range of Compromised Destination

Similarly, to figure out if the mDPM attack can guide a drone to a target redirected destination in a limited time, we further analyze the feasible range of redirected destinations for a given attack time to show the overall capability of mDPM. According to Eq. 5.2, we have the feasible range of redirected destination under one injection cycle:

$$(\frac{V^r_{drift,N}}{\hat{C}^a_N})^2 + (\frac{V^r_{drift,E}}{\hat{C}^a_E})^2 = \lambda, \tag{5.11}$$

or

$$(V^r_{drift,N})^2 + (\frac{V^r_{drift,E}}{\hat{C}^a_E/\hat{C}^a_N})^2 = \lambda \cdot (\hat{C}^a_N)^2. \tag{5.12}$$

After n injection cycles, the feasible range of the redirected destination will be

$$(\frac{R_x - D_x}{\hat{C}^a_N})^2 + (\frac{R_y - D_y}{\hat{C}^a_E})^2 = n^2 \cdot \lambda, \tag{5.13}$$

or

$$(R_x - D_x)^2 + (\frac{R_y - D_y}{\hat{C}^a_E/\hat{C}^a_N})^2 = n^2 \cdot \lambda \cdot (\hat{C}^a_N)^2. \tag{5.14}$$

We will present the evaluation of mDPM capability in Section 5.3.

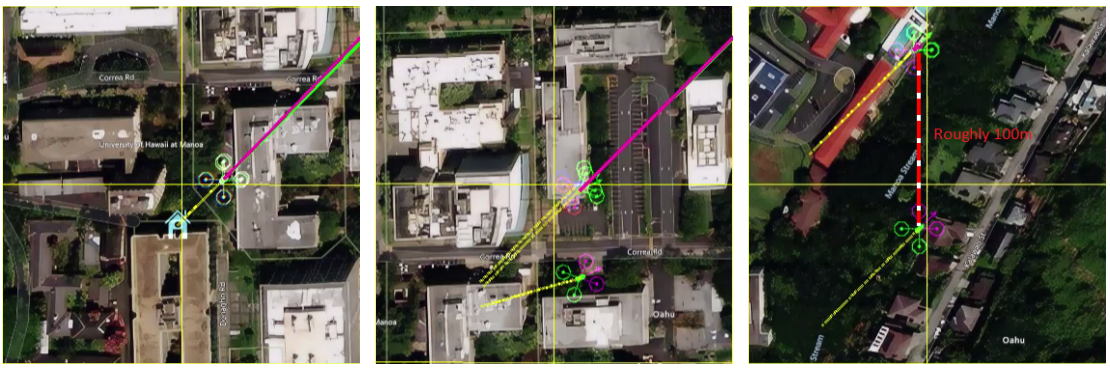
5.3 System Instrumentation and Performance Evaluation

As our goal is to develop a practical solution, different from existing methods, we focused on the broadly-deployed open-source system ArduPilot. In the following, we will first introduce our system instrumentation which not only helps us understand the critical issues in the system but also allows us to evaluate different practical attacks.

5.3.1 System Instrumentation

We have conducted extensive analysis and testing of ArduPilot Copter code to understand the state estimation and navigation control algorithms. For system control, the SITL module runs the same code as a real firmware to simulate a flight with a large set of common

parameters. Although ArduPilot has a powerful logging scheme that enables debugging for many states [2], for our research, we need to look into specific states (such as examining the adjustment parameters in each fast loop, which are not supported by the logging facility), in order to better learn system dynamics and capture real-time states. So we enhanced the system log facility of ArduPilot [4] by instrumenting the source code of its control algorithms (including the key control loops, EKF state estimation algorithms, and navigation control algorithms), such that we were able to save more related system states into its Dataflash logs for our analysis, and observe every state variable associated with control algorithms in each cycle. As a result, we were able to capture all key variables (such as all sensor readings, state estimation variables, control parameters, and system states). Meanwhile, we also utilize the default MAVLink interface to obtain common states such as the simulated position of a drone (i.e., its "physical position" in the simulated world) and its position estimation in each GPS cycle for us to build spoofed GPS inputs in real-time.



(a) Before attack. (b) Under attack. (c) Reached Destination.

Figure 5.5: bDPM Demo in ArduPilot SITL.

As a SITL drone runs in the same way as a real drone, it can take different types of GPS input formats, e.g., popular UBLOX and NMEA formats [5]. In its default setting, it simply uses the simulated (physical) position of a drone as its GPS position input. So, we can perform a covert GPS spoofing by switching its GPS input to take MAVLink *GPS_INPUT*

messages [43], in the same way as it receives GPS messages from a GPS-capable device. We then built a mission control program with the DroneKit Developer tools [26] to obtain drone real-time states, craft GPS_INPUT messages per GPS cycle, and send them to the drone via MAVProxy [7], same as a common ground control station (QGroundControl [53]) communicates with a real drone. We further installed a callback listener to receive position state updates in our control program.

We uploaded a video of a covert bDPM attack on the above platform at Youtube [17]. With the source location as (0, 0), the drone's destination is set to the waypoint of (500 meters, 500 meters) in the Northeast with a velocity of 4 meter/second. Waiting for 20 seconds into the mission after the system entered a steady state, we started to covertly spoof GPS inputs with an injection of 4.07 meter/per GPS cycle to the North. (It takes less than 20 seconds for the drone to enter a steady state.) As shown in Figure 5.5.(b), we then saw that the drone state (shown as the top drone icon) is still on its original track (in pink) to the Northeast; but its real position (shown as the bottom drone icon) is shifted down to the South, below its original track. The drone continued with its mission, without noticing its real position is gradually away from its original track. When the drone position state is close to the original destination, the real drone position is about 100 meters South to the original destination, as shown in Figure 5.5.(c).

With significant efforts in the past years, we were able to build this in-depth instrumentation platform for examining the control algorithms and the proposed attacks in great details, which also facilitated the evaluation presented in the following.

5.3.2 Evaluation of basic DPM (bDPM)

5.3.2.1 Simulation Settings

In each attack simulation, to show the pure attack effect, we did not launch the attack until the system entered a steady state, i.e., we waited for 20 seconds after it reaches the takeoff altitude and begins to fly to a preset waypoint. We used the common settings of consumer drones as key parameters in the evaluation, e.g., a GPS update cycle is set to 0.1 second; the horizontal position accuracy of GPS input is 0.1 meter; the velocity accuracy of GPS input is 0.1 meter/second; a default drone starting velocity is 4 meter/second. We have repeated the simulations many times to observe and measure the coefficients for our basic model presented in Section 5.1: for example, we used linear regression to find the attack coefficients $C_a^N = -0.0491$, and $C_a^E = -0.0455$.

5.3.2.2 Accuracy of bDPM

As our goal is to divert a drone to a redirected destination, we first evaluated the accuracy of bDPM, i.e., the difference between the expected redirected destination and the actual destination. For easy illustration, we first set the injection direction to the East and the total attack duration to 50 seconds. Consider the home position as the original point (0, 0), the original destination was set to (500 meters, 500 meters) in the Northeast in a local frame. To evaluate the accuracy under different attack sizes, we varied the size of **DR** (the vector difference between the redirected destination (R_x, R_y) and the original destination (D_x, D_y)) from 20 to 100 meters. Table 5.3 shows the attack error rates under different injection rates. The 1st row shows the size of intended redirection vector from 20 to 100 meters. The 2nd row is the corresponding injection size derived based on the size of redirection vector using Algorithm I. The 3rd row is the size of the actual redirection vector obtained from the simulation. In the 4th row, we can see that: for different redirection sizes,

the bDPM attack achieved very small errors (under 1.5%), i.e., it can accurately redirect a drone to the intended destination.

Table 5.3: bDPM Attack error under different Injection rates.

expected **DR** size (m)	20	40	60	80	100
Injection (m/GPS-cycle)	0.88	1.76	2.64	3.52	4.40
Actual **DR** size (m)	20.22	39.81	60.01	81.14	100.36
Error Rate	**1.10%**	**-0.475%**	**0.017%**	**1.425%**	**0.36%**

Next, keeping the same source and destination as the above, we evaluated the bDPM's accuracy in eight directions: North, Northeast, East, Southeast, South, Southwest, West, Northwest. The total attack duration is set to 50 seconds as the above, and the redirection vector is set to 100 meters. In Table 5.4 and Table 5.5, the 1st row shows the injection direction; the 2nd and the 3rd row show the injection sub-components in the North and the East directions; the 4th and the 5th row show the errors in the North and the East directions; the 6th and 7th row show the error rates in the North and the East directions. We can see the attack errors for these cases are still very small (under 0.9% in a sub-component), which shows the bDPM attack can accurately redirect the drone to different directions.

Table 5.4: bDPM Attack error under different attack directions (1).

Injection direction	E	W	N	S
Injection: North (m/GPS-cycle)	0	0	5	-5
Injection: East (m/GPS-cycle)	5	-5	0	0
Error: North (m)	/	/	-0.23	-0.29
Error: East (m)	0.06	-0.13	/	/
Error Rate: North	/	/	**-0.19%**	**-0.24%**
Error Rate: East	**0.053%**	**-0.11%**	/	/

5.3.2.3 Injection limitation and Feasible range

Although we have shown that the bDPM can redirect a drone to any direction with high accuracy, the maximum size of the redirection is limited by the maximum injection

Table 5.5: bDPM Attack error under different attack directions (2).

Injection direction	NE	NW	SE	SW
Injection: North (m/GPS-cycle)	5	5	-5	-5
Injection: East (m/GPS-cycle)	5	-5	5	-5
Error: North (m)	0.16	0.02	-1.1	-0.59
Error: East (m)	0.62	0.14	0.23	-0.05
Error Rate: North	**0.13%**	**0.016%**	**-0.90%**	**-0.48%**
Error Rate: East	**0.55%**	**0.12%**	**0.20%**	**-0.044%**

allowed in each cycle that is limited by the bad data detector of the drone. To find the maximum injection rate allowed in a direction, we gradually increased the injection rate until the system detects the large error term and raises GPS-fail alarms. We repeated these simulations to confirm the maximum injection rate for bDPM in each direction, which will then give us the largest redirection size **DR** for a given attack duration in the direction. Then we can combine the maximum redirection size in all directions to outline the feasible range of the bDPM attack, i.e., we can redirect the drone to any point within this range under this attack duration. In Table 5.6 and Table 5.7, the attack duration was 50 seconds, and we tested in 8 directions to outline the feasible redirection range for attacking 50 seconds. The 1st row shows the redirection directions; the 2nd row shows the maximum injection rate in a direction; the 3rd row shows the maximum size of redirection. The maximum injection rate for each direction varies from 7.09 to 7.65 meter/GPS cycle; the maximum redirection size in each direction varies from 169.28 meters to 177.03 meters for the attack duration of 50 seconds.

Furthermore, to show the feasible ranges of redirected destinations under different attack duration, we varied the attack duration from 20 to 100 seconds, and determined the corresponding maximum redirection sizes in 8 directions. We then outlined the feasible ranges under different attack durations in Figure 5.6. In this 2D plane, the center location (0, 0) is the original destination; the smallest circle-like range is the feasible range under an attack duration of 20 seconds; the largest circle-like range is the feasible range under an

Table 5.6: bDPM Maximum Redirection Size (1).

Injection Direction	E	W	N	S
Max Injection (m/GPS-cycle)	7.56	7.6.	7.09	7.14
Max **DR** size (m)	171.58	173.32	171.32	173.00

Table 5.7: bDPM Maximum Redirection Size (2).

Injection Direction	NE	NW	SE	SW
Max Injection (m/GPS-cycle)	7.24	7.48	7.50	7.37
Max **DR** size (m)	169.28	176.77	177.03	173.28

attack duration of 100 seconds. It is easy to see that the attack feasible range expands as the attack duration increases.

5.3.3 Evaluation of Measurement-based DPM (mDPM)

5.3.3.1 Settings

The basic simulation settings for mDPM is the same as bDPM. We have conducted extensive simulations to measure the attack coefficients for our mDPM model presented in Section 5.2: we used linear regression to identify the measurement-based attack coefficients $\hat{C}_N^a = -1.00$, and $\hat{C}_E^a = -0.933$; we further determine the parameters of this model as $I_N^0 = \frac{\hat{C}_N^a}{C_N^a} \cdot X_N^I = 20.37 \cdot X_N^I$, $I_E^0 = \frac{\hat{C}_E^a}{C_E^a} \cdot X_E^I = 20.51 \cdot X_E^I$.

5.3.3.2 Accuracy of mDPM

To evaluate the accuracy of the mDPM attack, as shown in the 1st row of Table 5.8, we varied the injection increment velocity X^I from 0.5 to 2.5 meter/$second^2$ to examine the difference between the expected redirected destination and the actual destination. The 2nd row shows the expected redirection size; the 3rd row shows the actual redirection size; and the 4th row shows the attack error rates under different X^I. Here we set the injection direction to the East and the total attack duration to 100 seconds. Consider the home

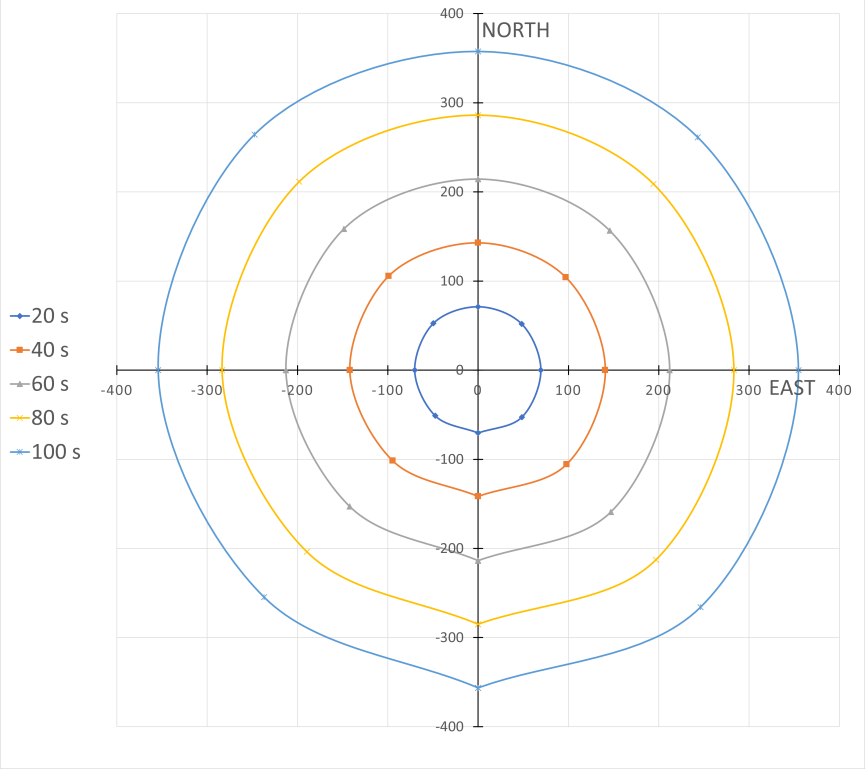


Figure 5.6: Feasible ranges of redirected destinations under different attack durations in bDPM.

Table 5.8: mDPM Attack error rate under different X^I.

X^I (m/s^2)	0.5	1	1.5	2	2.5
Exp. **DR** size (m)	46.65	93.3	139.95	186.6	233.25
Act. **DR** size (m)	46.76	93.34	139.84	186.54	232.96
Error Rate	**0.24%**	**0.043%**	**-0.079%**	**-0.032%**	**-0.12%**

position as location (0, 0), the original destination was set to (500 meters, 500 meters) in the Northeast in a local frame. We can see that: for these X^I's, the mDPM attack shows very small errors (under 0.45%), i.e., it can precisely divert a drone to our redirected destinations.

Furthermore, we tested the attack accuracy in 8 directions as shown in Table 5.9 and Table 5.10. The 1st row represents the redirection direction. The 2nd and 3rd row show the sub-component of X^I's in the North and East directions. The 4th and 5th row show

the actual redirection sizes in the North and East. The 6th and 7th row show the errors between the expected **DR** and the actual **DR**. The 8th and 9th row show the error rates in the North and East. For all cases, the mDPM showed a very small error rates (under 0.45% in a sub-component).

Table 5.9: mDPM Attack error rate under different attack directions (1).

Injection direction	E	W	N	S
X_N^I (m/s^2)	0	0	1	-1
X_E^I (m/s^2)	1	-1	0	0
Act. **DR** size-North (m)	/	/	100.13	99.72
Act. **DR** size -East (m)	93.34	92.88	/	/
Error-North (m)	/	/	0.13	-0.28
Error - East (m)	0.04	-0.42	/	/
Error Rate - North	/	/	**0.13%**	**-0.28%**
Error Rate - East	**0.043%**	**-0.45%**	/	/

Table 5.10: mDPM Attack error rate under different attack directions (2).

Injection direction	NE	NW	SE	SW
X_N^I (m/s^2)	1	1	-1	-1
X_E^I (m/s^2)	1	-1	1	-1
Act. **DR** size-North (m)	100.10	100.18	99.65	99.75
Act. **DR** size -East (m)	93.29	92.91	93.40	92.91
Error-North (m)	0.10	0.18	-0.35	-0.25
Error-East (m)	-0.01	-0.39	0.10	-0.39
Error Rate - North	**0.10%**	**0.18%**	**-0.35%**	**-0.25%**
Error Rate - East	**-0.011%**	**-0.42%**	**-0.11%**	**-0.42%**

Table 5.11: mDPM Max Redirection Sizes for different directions (1).

Injection Direction	E	W	N	S
Max X^I (m/s^2)	3.6	3.4	3.4	3.2
Max **DR** size (m)	167.77	158.09	170.10	159.77

Table 5.12: mDPM Max Redirection Sizes for different directions (2).

Injection Direction	NE	NW	SE	SW
Max X^I (m/s^2)	3.25	3.39	3.39	3.25
Max **DR** size (m)	157.36	163.89	163.88	156.80

5.3.3.3 Injection limitation and Feasible Range

Because the maximum redirection size is determined by the maximum X^I, we gradually increased X^I to find its maximum value allowed until the bad data detector is triggered. We repeated these simulations to confirm the maximum X^I for mDPM in each direction, which will give us the maximum redirection size in the direction for a given attack duration. We then combined these maximum redirection sizes to outline the feasible range of redirected destinations under the mDPM attack, i.e., we can redirect the drone to any point within this range. The attack duration was set to 50 seconds, and we tested in 8 directions to outline the range. As shown in Table 5.11 and Table 5.12, the maximum X^I for each direction varies from 3.2 to 3.6 $meter/second^2$; the maximum redirection size in each direction varies from 156.80 to 170.10 meters.

Furthermore, we varied the attack duration from 20 to 100 seconds, and determined the corresponding maximum redirection sizes in 8 directions. Based on the maximum redirection sizes under different attack durations, we outlined the feasible ranges of redirected destinations in Figure 5.7. In this 2D plane, the center location (0, 0) is the original destination; the smallest circle-like range is the feasible range under an attack duration of 20 seconds; the largest circle-like range is the feasible range under an attack duration of 100 seconds. It is easy to see that the attack feasible range expands as the attack duration increases.

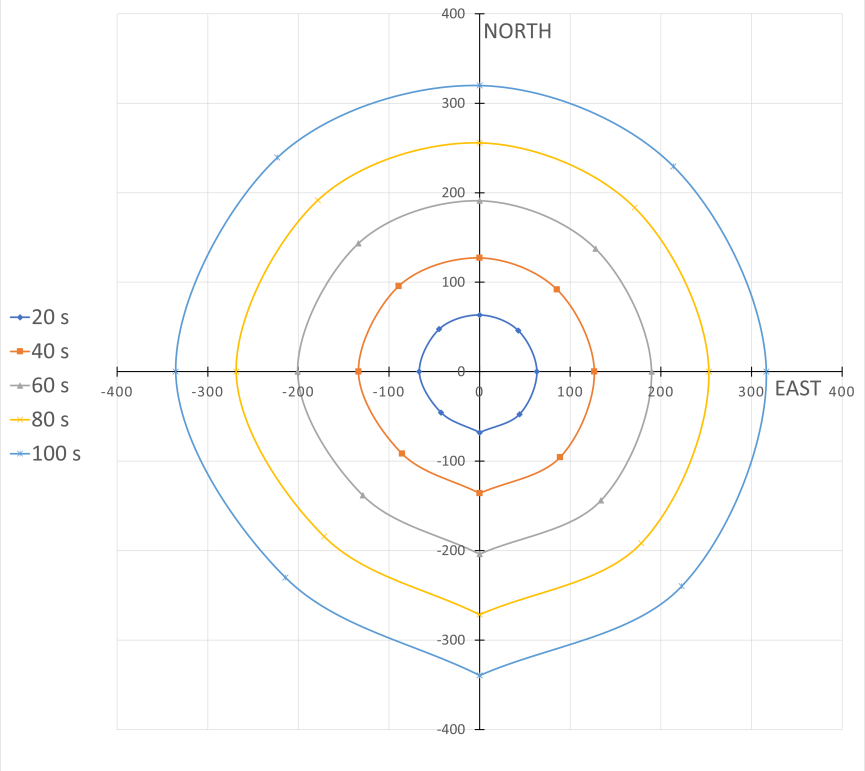


Figure 5.7: Feasible ranges of redirected destinations under different attack durations in mDPM.

5.4 Conclusion

In this chapter, we have developed the third-level DPM attack that can help us accurately manipulate a drone's physical position. We have evaluated the attack on the SITL module of the ArduPilot, which shows that DPM can redirect a drone to a desired location with very small errors. We have also analyzed the maximum feasible range of redirected destination for a given original destination under DPM. Because the weaknesses exploited here are common in many autonomous systems, this work can be applied to these systems.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

In this research, we have first examined the entire stack of guidance sensing, state estimation, and navigation in the control loop of consumer drones; we have then identified the vulnerabilities at each level of the stack: sensor measurements, state estimation, and navigation control, and developed the second-level FDI attacks and third-level DPM attack. Our analysis and evaluation have shown that the proposed attacks can accurately manipulate a drone's state estimation and eventually guide it to a redirected location for safe handling. We believe this is the first work that is able to redirect a consumer drone accurately to a desired destination.

Although FDI and DPM are developed on common consumer drones, the idea of exploring the vulnerabilities of sensing, state estimation, and navigation control in a holistic method is applicable to many other existing and emerging autonomous system. Identifying and exploring such a sequence of vulnerabilities could be a generic attack to many control systems. To stop such an attack, we need to address the vulnerability at each step. First, there have been various proposals to detect GPS spoofing [21, 38, 73]. However, to implement these enhancements on consumer drones may still have a long way to go [34]. Second, we can improve the EKF's anomaly detection algorithm to detect attacks such as FDI and DPM. We have briefly discussed the general idea of how to improve the detector in Chapter 4.1.3 and 4.2.2, and we will further investigate countermeasures based on physical properties of drones. Third, we will look into secure navigation algorithms to detect position/velocity manipulations. Many other important and interesting research problems still need to be explored on consumer drones, such as how to compromise physical altitude control, or how to defeat various optical flow solutions.

BIBLIOGRAPHY

[1] ArduPilot. ArduPilot Autopilot Suite. http://ardupilot.org/ardupilot/.

[2] ArduPilot. Diagnosing problems using Logs. https://ardupilot.org/copter/docs/common-diagnosing-problems-using-logs.html, 2020.

[3] ArduPilot. Extended Kalman Filter Navigation Overview and Tuning. http://ardupilot.org/dev/docs/extended-kalman-filter.html, 2020.

[4] ArduPilot. Adding a New Log Message, Dec. 18, 2019. https://ardupilot.org/dev/docs/code-overview-adding-a-new-log-message.html?highlight=log.

[5] ArduPilot. ArduPilot AP GPS, Dec. 18, 2019. https://github.com/ArduPilot/ardupilot/blob/master/libraries/AP_GPS/AP_GPS.h.

[6] ArduPilot. SkyRocketToys Source Code, Dec. 18, 2019. https://github.com/SkyRocketToys/ardupilot.

[7] ArduPilot. A UAV ground station software package for MAVLink based systems, Dec. 18, 2019. https://ardupilot.org/mavproxy/.

[8] Battelle. Drone defender. https://www.battelle.org/government-offerings/national-security/tactical-systems-vehicles/tactical-equipment/counter-UAS-technologies, 2016.

[9] Minas Benyamin and Geoffrey Goldman. Acoustic Detection and Tracking of a Class I UAS with a Small Tetrahedral Microphone Array. Technical Report ARL-TR-7086, ARL, September 2014.

[10] Suzhi Bi and Ying Jun Zhang. Defending mechanisms against false-data injection attacks in the power system state estimation. In *IEEE GLOBECOM Workshops (GC Wkshps)*, pages 1162–1167, 2011.

[11] Rakesh B Bobba, Katherine M Rogers, Qiyan Wang, Himanshu Khurana, Klara Nahrstedt, and Thomas J Overbye. Detecting false data injection attacks on dc state estimation. In *Preprints of the First Workshop on Secure Control Systems, CPSWEEK*, volume 2010, 2010.

[12] Jianqiu Cao. Practical GPS spoofing attacks on consumer drones. Master's thesis, University of Hawaii, https://scholarspace.manoa.hawaii.edu/bitstream/10125/73336/Cao_hawii_0085O_10909.pdf, December 2020.

[13] Simon Castro, Robert Dean, Grant Roth, George T Flowers, and Brian Grantham. Influence of acoustic noise on the dynamic performance of MEMS gyroscopes. In *ASME 2007 International Mechanical Engineering Congress and Exposition*, pages 1825–1831. American Society of Mechanical Engineers, 2007.

[14] W. Chen, Y. Dong, and Z. Duan. Manipulating Drone Dynamic State Estimation to Compromise Navigation. In *2018 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, May 2018.

[15] W. Chen, Y. Dong, and Z. Duan. Manipulating Drone Position Control. In *in Proc. of IEEE Conference on Communications and Network Security(CNS)*, pages 1–9, June 2019.

[16] Wenxin Chen, Yingfei Dong, and Zhenhai Duan. Compromising Flight Paths of Autopiloted Drones. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1316–1325. IEEE, 2019.

[17] Wenxin Chen, Yingfei Dong, and Zhenhai Duan. A Video Demo of Drone Position Manipulation Attack, Feb 16, 2021. https://youtu.be/kE0T4sFJZ7o.

[18] Wenxin Chen, Zhenhai Duan, and Yingfei Dong. False Data Injection on EKF-based Navigation Control. In *International Conference on Unmanned Aircraft Systems (ICUAS) 2017*, pages 1608–1617, 2017.

[19] AM Leite Da Silva, MB Do Coutto Filho, and JF De Queiroz. State forecasting in electric power systems. In *IEE Proceedings C (Generation, Transmission and Distribution)*, volume 130, pages 237–244, 1983.

[20] P. Dash, M. Karimibiuki, and K. Pattabiraman. Out of control: stealthy attacks against robotic vehicles protected by control-based techniques. In *ACSAC '19: Proceedings of the 35th Annual Computer Security Applications Conference*, Dec. 2019.

[21] D. S. De Lorenzo, J. Gautier, J. Rife, P. Enge, and D. Akos. Adaptive array processing for GPS interference rejection. In *In Proceedings of the ION GNSS Meeting, Long Beach, CA. Institute of Navigation.*, 2005.

[22] Robert N Dean, George T Flowers, A Scotte Hodel, Grant Roth, Simon Castro, Ran Zhou, Alfonso Moreira, Anwar Ahmed, Rifki Rifki, Brian E Grantham, et al. On the degradation of MEMS gyroscope performance in the presence of high power acoustic noise. In *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, pages 1435–1440. IEEE, 2007.

[23] Robert Neal Dean, Simon Thomas Castro, George T Flowers, Grant Roth, Anwar Ahmed, Alan Scottedward Hodel, Brian Eugene Grantham, David Allen Bittle, and James P Brunsch. A characterization of the performance of a MEMS gyroscope in acoustically harsh environments. *IEEE Transactions on Industrial Electronics*, 58(7):2591–2596, 2011.

[24] DeDrone. Secure your airspace now. http://www.dedrone.com/en/dronetracker/drone-protection-software, 2016.

[25] Eddy Deligne. Ardrone corruption. *Journal of Computer Virology, vol.8, pp.15-27*, 2012.

[26] DroneKit. DroneKit Python Development Tools, Dec. 18, 2019. https://github.com/dronekit/dronekit-python.

[27] Damien Eynard, Pascal Vasseur, Cedric Demonceaux, and Vincent Fremont. Real time UAV altitude, attitude and motion estimation from hybrid stereovision. *Autonomous Robots*, 33(1-2):157–172, 2012.

[28] Kenneth Gade. The seven ways to find heading. *The Journal of Navigation, vol.69, pp.955-970*, 2016.

[29] Przemyslaw Gasior, Stanislaw Gardecki, Jaroslaw Goslinski, and Wojciech Giernacki. Estimation of altitude and vertical velocity for multirotor aerial vehicle using Kalman filter. In *Recent Advances in Automation, Robotics and Measuring Techniques*, pages 377–385. Springer, 2014.

[30] Esmaeil Ghahremani and Innocent Kamwa. Dynamic state estimation in power system by applying the extended Kalman filter with unknown inputs to phasor measurements. *IEEE Transactions on Power Systems*, 26(4):2556–2566, 2011.

[31] Paul D. Groves. *Principles of GNSS, Inertial, and Multisensor integrated Navigation Systems*. Artech House, 2008.

[32] Chingiz Hajiyev and Sitki Yenal Vural. LQR controller with Kalman estimator applied to UAV longitudinal dynamics. *Positioning*, 4(1):36, 2013.

[33] Li Jiang. *SENSOR FAULT DETECTION AND ISOLATION USING SYSTEM DYNAMICS IDENTIFICATION TECHNIQUES*. PhD thesis, University of Michigan, 2011.

[34] Andrew J Kerns, Daniel P Shepard, Jahshan A Bhatti, and Todd E Humphreys. Unmanned aircraft capture and control via GPS spoofing. *Journal of Field Robotics*, 31(4):617–636, 2014.

[35] A. Kim, B. Wampler, J. Goppert, and I. Hwang. Cyber attack vulnerabilities analysis for unmanned aerial vehicles. *Infotech at Aerospace*, 2012.

[36] Tung T Kim and H Vincent Poor. Strategic protection against data injection attacks on power grids. *IEEE Transactions on Smart Grid*, 2(2):326–333, 2011.

[37] Drone Labs. Drone detector. http://www.dronedetector.com/how-drone-detection-works/, 2016.

[38] B. M. Ledvina, W. J. Bencze, and I. Galusha, B.and Miller. An in-line anti-spoofing module for legacy civil GPS receivers. In *In Proc. of the ION ITM, San Diego, CA. Institute of Navigation*, 2010.

[39] Xusheng Lei and Jingjing Li. An adaptive altitude information fusion method for autonomous landing processes of small unmanned aerial rotorcraft. *Sensors*, 12(10):13212–13224, 2012.

[40] Yao Liu, Peng Ning, and Michael K. Reiter. False Data Injection Attacks Against State Estimation in Electric Power Grids. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 21–32, New York, NY, USA, 2009. ACM.

[41] JK Mandal, AK Sinha, and L Roy. Incorporating nonlinearities of measurement function in power system dynamic state estimation. *IEE Proceedings-Generation, Transmission and Distribution*, 142(3):289–296, 1995.

[42] MAVLink. MAVLink Developer Guide, Dec. 18, 2019. https://mavlink.io/en/.

[43] MAVLink. MAVLink GPS_INPUT Command, Dec. 18, 2019. https://mavlink.io/en/messages/common.html#GPS_INPUT.

[44] A.H. Michel and D. Gettinger. Analysis of new drone incident reports, May 8, 2017. http://dronecenter.bard.edu/analysis-3-25-faa-incidents/.

[45] Mike Monnik. Hacking the Parrot AR.Drone 2.0. https://dronesec.com/blogs/articles/hacking-the-parrot-ar-drone-2-0, 2019.

[46] K Nishiya, J Hasegawa, and T Koike. Dynamic state estimation including anomaly detection and identification for power systems. In *IEE Proceedings C (Generation, Transmission and Distribution)*, volume 129, pages 192–198, 1982.

[47] Ruixin Niu and Lauren Huie. System state estimation in the presence of false information injection. In *Statistical Signal Processing Workshop (SSP), 2012 IEEE*, pages 385–388. IEEE, 2012.

[48] Juhwan Noh, Yujin Kwon, Yunmok Son, Hocheol Shin, Dohyun Kim, Jaeyeong Choi, and Yongdae Kim. Tractor Beam: Safe-hijacking of Consumer Drones with Adaptive GPS Spoofing. *ACM Transactions on Privacy and Security (TOPS)*, 22(2):12, April 2019.

[49] OpenPilot. DIY Drones: The Leading Community for Personal UAVs. https://diydrones.com/page/openpilot-1, 2021.

[50] Paparazzi. Paparazzi: The Free Autopilot. http://wiki.paparazziuav.org/wiki/Main_Page, 2003.

[51] Fabio Pasqualetti, Ruggero Carli, and Francesco Bullo. A distributed method for state estimation and false data detection in power networks. In *IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 469–474, 2011.

[52] Scott Peterson and Payam Faramarzi. Exclusive: Iran hijacked US drone, says Iranian engineer. https://www.csmonitor.com/World/Middle-East/2011/1215/Exclusive-Iran-hijacked-US-drone-says-Iranian-engineer, 2011.

[53] QGroundControl. Intuitive and Powerful Ground Control Station for the MAVLink protocol, Dec. 18, 2019. http://qgroundcontrol.com/.

[54] P. Riseborough. Application of Data Fusion to Aerial Robotics. In *in Proc. of Embedded Linux Conference*, Mar. 14, 2015.

[55] Paul Riseborough. Inertial Navigation Filter. https://github.com/priseborough/InertialNav.

[56] Fred Samland, Jana Fruth, Mariso Hildebrandt, Tobias Hoppe, and Jana Dittmann. AR. drone: Security threat analysis and exemplary attack to track persons. *Proceedings of the SPIE*, 8301, 2012.

[57] Henrik Sandberg, André Teixeira, and Karl H Johansson. On security indices for state estimators in power networks. In *First Workshop on Secure Control Systems (SCS), Stockholm*, 2010.

[58] M. Schmidt and M. Shear. A Drone, Too Small for Radar to Detect, Rattles the White House. *New York Times, https://www.nytimes.com/2015/01/27/us/white-house-drone.html*, Jan. 26, 2015.

[59] Andrew M. Shull. Analysis of cyberattacks on unmmaned aerial systems. Master's thesis, Purdue University, 2013.

[60] K. M. Smalling and K. W. Eure. A Short Tutorial on Inertial Navigation System and Global Positioning System Integration. Technical Report NASA/TM–2015-218803, NASA, September 2015.

[61] Yunmok Son, Hocheol Shin, Dongkwan Kim, Youngseok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, and Yongdae Kim. Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 881–896, Washington, D.C., August 2015. USENIX Association.

[62] INC SRC. Silent Archer counter-UAS system. http://www.srcinc.com/what-we-do/ew/silent-archer-counter-uas.html, 2016.

[63] P. B. Sujit, Srikanth Saripalli, and Joao Borges Sousa. Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicles. *IEEE Control Systems*, 34(1):42–59, February 2014. Copyright: Copyright 2014 Elsevier B.V., All rights reserved.

[64] Blighter Surveillance Systems. AUDS Anti-UAV Defence System. http://www.blighter.com/products/auds-anti-uav-defence-system.html, 2016.

[65] Nils Ole Tippenhauer, Christina Popper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful GPS spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 75–86, 2011.

[66] Timothy Trippel, Ofir Weisse, Wenyuan Xu, Peter Honeyman, and Kevin Fu. WALNUT: Waging doubt on the integrity of MEMS accelerometers with acoustic injec-

tion attacks. In *2017 IEEE European symposium on security and privacy (EuroS&P)*, pages 3–18. IEEE, 2017.

[67] Gustavo Valverde and Vladimir Terzija. Unscented Kalman filter for power system dynamic state estimation. *IET generation, transmission & distribution*, 5(1):29–37, 2011.

[68] Rudolph Van Der Merwe, Eric Wan, and Simon Julier. Sigma-point Kalman filters for nonlinear estimation and sensor-fusion: Applications to integrated navigation. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 5120, 2004.

[69] J. Vanian. Drone registrations are still soaring. *Fortune,* http://fortune.com/2017/01/06/drones-registrations-soaring-faa/, Jan. 06, 2017.

[70] JL Verboom, Sjoerd Tijmons, C De Wagter, B Remes, Robert Babuska, and Guido CHE de Croon. Attitude and altitude estimation and control on board a flapping wing micro air vehicle. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5846–5851. IEEE, 2015.

[71] Eric A Wan and Rudolph Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium (AS-SPCC)*, pages 153–158, 2000.

[72] G. Welch and G. Bishop. An Introduction to the Kalman Filter. *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 1995.

[73] K. D. Wesson, B. L. Evans, and T. Humphreys. A combined symmetric difference and power monitoring GNSS anti-spoofing technique. In *In Proceedings of the IEEE Global Conference on Signal and Information Processing, Austin, TX.*, 2013.

[74] Oliver J. Woodman. An introduction to inertial navigation. Technical Report UCAM-CL-TR-696, University of Cambridge, Computer Laboratory, August 2007.

[75] Le Xie, Yilin Mo, and Bruno Sinopoli. False data injection attacks in electricity markets. In *First IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 226–231, 2010.

[76] Vadim Zaliva and Franz Franchetti. Barometric and GPS altitude sensor fusion. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 7525–7529. IEEE, 2014.