2017 International Conference on
Unmanned Aircraft Systems (ICUAS)
June 13-16, 2017, Miami, FL, USA

False Data Injection on EKF-based Navigation Control

Wenxin Chen
Department of Electrical Engineering
University of Hawaii
Honolulu, HI 96822

Zhenhai Duan
Department of Computer Science
Florida State University
Tallahassee, FL 32306

Yingfei Dong
Department of Electrical Engineering
University of Hawaii
Honolulu, HI 96822

***Abstract*—Small consumer unmanned aircraft systems (sUASs) have been involved in many security incidents recently. Current defense methods have serious limitations because they mostly try to physically capture an sUAS or jam its communications and its GPS signals to trigger its fail-safe features. While physical methods are usually unscalable and often slow in response, the default fail-safe features can be easily changed to other options such as continuing its pre-configured mission, regardless of jamming. Therefore, new defense solutions must be developed to address these issues. To our best knowledge, very little research has been published in this area.**

In this paper, we propose a false data injection attack to exploit the limitation of sensoring and processing capabilities on an sUAS and manipulate its on-board sensors, in order to deviate it from its (pre-configured) flight plan. We have analyzed the popular *ArduPilot* flight control system, identified potential vulnerabilities in its sensor data fusion process, and developed a method to manipulate on-board magnetometer readings while avoiding being detected by its bad-data detection schemes. Our initial results show that the proposed method is able to misguide the navigation system and deviate an sUAS from its target destination far away. While we present our main ideas in this paper, we are further investigating related theoretical issues and improving the proposed method with practical experimental tests.

***Keywords*—false data injection, UAS security, inertial navigation, extended Kalman filter (EKF), magnetometer**

I. Introduction

Consumer-level small unmanned aircraft systems (sUASs) usually includes an unmanned air vehicle, a ground control station, a human operator, and Radio Frequency (RF) communication links or guidance controls. Recently, as consumer sUASs become very affordable and popular, they have been involved in many security incidents in various situations, such as invading restricted airspace (e.g., the White House), conducting criminal activities (e.g., smuggling drugs across borders, or delivering contrabands into prisons), or getting too close to manned airplanes [1]–[3]. As the number of sUASs is rapidly increasing, it is urgent to develop effective solutions to deal with such potential threats to important assets.

To prevent sUASs from invading a restricted airspace, e.g., repelling them from approaching an airport, we need to quickly and effectively detect, identify, locate, monitor, and deter them around the restricted area. For detection and monitoring, researchers have proposed various solutions based on radar, acoustic, RF, optical imaging, etc [4]–[6]. In general, radar and RF detection methods are most effective because they are reliable in most conditions, while other methods may not work very well, e.g., at night, in noisy environments, or under bad weather. (We are also working on profiling sUAS communications to identify specific sUASs [7]; it is out of scope of this paper.)

For determent, many *physical methods* have been developed, such as casting a net over an sUAS from another UAS, training eagles to snatch it in the air, or shooting it down. However, these methods often have serious limitations, such as slow and limited responses, potential collateral damages, or ineffective in dealing with a swarm. We still need to develop fast and effective solutions to handle potentially many sUASs at the same time. Remote *cyber solutions* for countering sUASs are very promising in this situation, because we can use common computers and radio devices to automatically explore the vulnerabilities of sUASs and scale up easily when dealing with a swarm. A direct method is to hack into the sUAS control channel and issue fake control commands, because most current sUASs are poorly secured [8], [9]. However, in this paper, we explore a different direction and focus on how to exploit the potential vulnerability in common sUAS navigation systems, i.e., limited capability to handle injected false sensor readings.

One popular idea in current sUAS defense methods is to jam its communications with its ground control station and its GPS receiver, in order to trigger its fail-safe features (immediately landing or return-to-home), such as Silent Archer [10], AUDS [11] and DroneDefender [12]. Because the common default setting is either to land or return-to-home in the case of loss of control communications or GPS signals (e.g., after 10 seconds without GPS signals), these methods effectively disable common sUASs. However, it is relatively easy to change such default settings, e.g., rebuilding an ArduPilot firmware to "continue" a pre-configured mission by only using on-board sensors, regardless of the loss of communications. To our best knowledge, almost no

978-1-5090-4494-8/17/$31.00 ©2017 IEEE

research has been published in this area. To address this issue, we propose a false data injection attack to exploit the limitation of sensoring and data fusing capabilities on a sUAS and manipulate its on-board sensors (such as IMU and magnetometers), in order to deviate it from its (pre-configured) flight plan.

To investigate the practical limitations of current navigation systems, we have analyzed the popular open-source ArduPilot flight control system [13], [14], and identified methods to manipulate on-board sensor readings while avoiding being detected by its bad-data detection schemes. Since FCC requires special approvals to perform physical wireless jamming, we assume that *the jamming is conducted by authorized parties*, and we focus on disrupting the control of the navigation system under jamming. We assume that we can inject false data into sensor readings either on-board or via remote devices. Our preliminary studies in this paper show that: (i) injected false sensor readings can serious affect flight plans, and (ii) we can manipulate the sensor readings on ArduPilot without being detected by its EKF-based bad-data detector. Our initial results show that the proposed method is able to misguide the navigation system and deviate a sUAS from its target destination far away. While we present our main ideas in this paper, we are further investigating related theoretical issues and improving the proposed method with practical experimental tests.

The remainder of this paper is organized as follows. We will discuss background and related work in Sec. II. We will present the proposed attack and preliminary evaluation in Sec. III. We will further present the EKF-based navigation system on ArduPilot and discuss our attack on the system in Sec. IV. We will conclude this paper and discuss our current investigation in Sec. V.

II. Background and Related Work

A. Inertial Navigation Systems (INSs) on sUASs

sUASs usually use strapdown inertial navigation systems with MEMS sensors due to cost and size constraints. An Inertial Measurement Unit (IMU) typically contain three rate-gyroscopes and three accelerometers, measuring 3-dimension angular velocity and linear acceleration, respectively. Based on the measurements, the INS can infer the position, orientation, velocity of an object relative to a known starting point. Excellent INS introductions are presented in many papers such as [15], [16].

sUAS gyroscopes measure the three-orthogonal angular velocity of the body at each time slot and calculate the rotation by integration, to track its global orientation coordinates. In practice, we use a matrix to represent the angular velocity, then the orientation can be calculated based on standard navigation theory [15].

After receiving the orientation information from the gyroscope, the acceleration in the three orthogonal directions of the global coordinates – North, East, Down (NED) – can be also obtained. Then, by applying a double integration with the initial position, an sUAS can infer its position with some error terms. Note that when calculating the acceleration in the Down direction, the gravity factor need to be taken into consideration. The general inertial navigation algorithm is illustrated in Fig.1.

Ideally, an INS is self-contained, i.e., as long as the initial position information and the accurate gyroscope and accelerometer signals in every time slot are given, the INS can determine the exact position at any time. However, due to the propagation of orientation and acceleration errors caused by sensor signal noises, the drift between an actual position and an estimated position may be rapidly exaggerated in a short duration, e.g., over 150 meters in a 60-second mission [15].

B. Reducing Drift in INSs using Sensor Fusion

Due to errors in low-end MEMS gyroscopes and accelerometers, an INS may have large drifts in velocity, position, and attitude. To limit such drifts, sensor fusion is often used by combining the direct measurements from other sensors with IMU measurements to obtain more accurate systems states, e.g., using Absolute Positioning Systems such as GPS [16] or a magnetometer [15]. Such data fusion enables low-cost sensors on sUASs to achieve required performance and robustness such that: faulty sensors 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.

A proposed method to reduce the orientation drifts on an sUAS is to fuse IMU readings with the readings from a magnetometer [14], [15], [17]. The magnetometer measures the strength and the direction of the local magnetic field, allowing the North direction to be found. However, a low-end magnetometer on an sUAS alone is often unable to provide a high accuracy; sensor fusion is necessary to enhance the accuracy. Combining a magnetometer with a common IMU, one study shows that the average error in position can be reduced from over 150 meters to 5 meters for a 60-second mission [15].

While sensor fusion is 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 sensor fusion methods emphasize performance and reliability, and do not consider malicious cyber attacks. Sensor fusion mistakes can result in unexpected mission changes and loss of control [14], [18].

We use ArduPilot as our subject in this project. A detailed discussion on ArduPilot sensor fusion in navigation will be presented in Sec.IV. We summarize its key features here: estimate critical parameters such as vehicle position, velocity, and orientation; wind speed and direction; rate gyroscope and accelerometer offsets; airspeed rate of change; airspeed scale factor error; height above ground; magnetometer offset errors; compass motor interference, etc. Main data fusion techniques in ArduPilot are Nonlinear Least Squares methods for batch processing of sensor

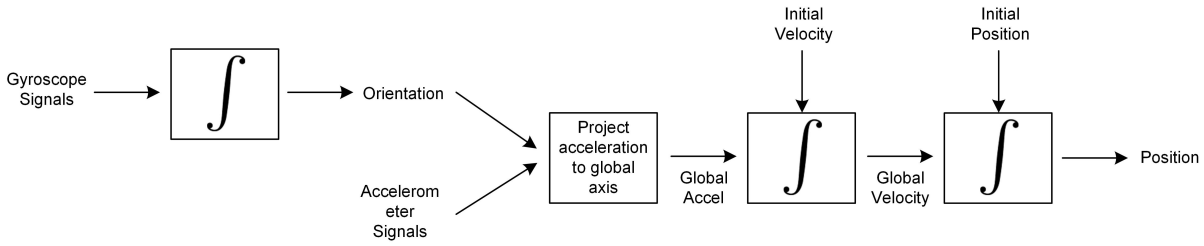


Fig. 1: Typical Inertial Navigation Algorithm [15].

calibration, and Extended Kalman Filters (EKFs) for flight navigation [14].

C. Extended Kalman Filter (EKF)

Kalman Filters (KFs) (also known as linear quadratic estimations) are widely used to estimate the underlying states based on a stream of noisy data since 1960s. A KF assumes zero-mean Gaussian errors in both a system model and corresponding measurements. It recursively estimates the current state at time slot k based on the estimation at time slot $(k-1)$ and the current state measurement at time slot k. While traditional KFs worked mostly for linear systems, EKFs are broadly used for non-linear systems, such as the EKFs implemented in ArduPilot [19].

In a KF, a state space model is usually defined in two steps [16]: *Prediction* and *Update*. In this paper, we consider a system state that includes the position, velocity, and acceleration of a vehicle in each of the three orthogonal directions, as follows:

$$\mathbf{x}(k) = \begin{bmatrix} p(k) \\ v(k) \\ a(k) \end{bmatrix}, \tag{1}$$

where $p(k), v(k)$ and $a(k)$ is the position, velocity, and acceleration at time slot k, respectively. In practice, $\mathbf{x}(k)$ can be calculated with the following equations (for the ease of discussion, assume the acceleration does not change over time):

$$\begin{aligned} p(k) &= p(k-1) + v(k-1)T + 0.5a(k-1)T^2; \\ v(k) &= v(k-1) + a(k-1)T; \\ a(k) &= a(k-1), \end{aligned} \tag{2}$$

which can be represented in matrix form:

$$\begin{bmatrix} p(k) \\ v(k) \\ a(k) \end{bmatrix} = \begin{bmatrix} 1 & T & 0.5T^2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p(k-1) \\ v(k-1) \\ a(k-1) \end{bmatrix}, \tag{3}$$

or

$$\mathbf{x}(k) = \mathbf{A}\mathbf{x}(k-1), \tag{4}$$

where T is the length of one time step. The measurement vector $\mathbf{z}(k)$, has the following equation:

$$\mathbf{z}(k) = \mathbf{C}\mathbf{x}(k). \tag{5}$$

With the above notation, the KF can be defined as two steps:

Prediction:

$$\begin{aligned} \mathbf{x}^-(k) &= \mathbf{A}\mathbf{x}(k-1) \\ \mathbf{P}^-(k) &= \mathbf{A}\mathbf{P}(k-1)\mathbf{A}^T + \mathbf{Q}, \end{aligned} \tag{6}$$

Update:

$$\begin{aligned} \Delta(k) &= \mathbf{z}(k) - \mathbf{C}(\mathbf{x}^-(k)) \\ \mathbf{K} &= \mathbf{P}^-(k)(\mathbf{P}^-(k) + \mathbf{R})^{-1} \\ \mathbf{x}(k) &= \mathbf{x}^-(k) + \mathbf{K}\Delta(k) \\ \mathbf{P}(k) &= (\mathbf{I} - \mathbf{K})\mathbf{P}^-(k), \end{aligned} \tag{7}$$

where $k = 1, 2, \ldots,$ are the time slots, $\mathbf{x}(k)$ is a state vector, $\mathbf{x}^-(k)$ is an estimate state vector, and $\mathbf{A}$ is the state transition matrix.

In the Prediction step, the KF predicts an estimation of the state $\mathbf{x}(k)$. Note that the $\mathbf{x}^-(0)$ in the first iteration is calculated from gyroscope and accelerometer readings. While in the Update step, $\mathbf{x}(k)$ is updated based on its prediction as well as $\mathbf{z}(k)$. The Kalman gain, $\mathbf{K}$, assigns different weights for $\mathbf{x}(k)$ and $\mathbf{z}(k)$, to correct the state estimation. $\mathbf{P}(k)$ is the predict error, and $\mathbf{Q}$ and $\mathbf{R}$ are the process and measurement noise covariance matrices, respectively. In practice, $\mathbf{Q}$ and $\mathbf{R}$ are tuned for desired performance. KF is an iterative process which is executed upon a new $\mathbf{z}(k)$ 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 model can be rewritten as

$$\begin{aligned} \mathbf{x}(k) &= f(\mathbf{x}(k-1)) \\ \mathbf{z}(k) &= h(\mathbf{x}(k)), \end{aligned} \tag{8}$$

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

Prediction:

$$\begin{aligned} \mathbf{x}^{-}(k) &= f(\mathbf{x}(k-1)) \\ \mathbf{P}^{-}(k) &= \mathbf{F}(k-1)\mathbf{P}(k-1)\mathbf{F}^{T}(k-1) + \mathbf{Q}, \end{aligned} \tag{9}$$

Update:

$$\begin{aligned} \Delta(k) &= \mathbf{z}(k) - h(\mathbf{x}^{-}(k)) \\ \mathbf{K} &= \mathbf{P}^{-}(k)\mathbf{H}^{T}(k)(\mathbf{H}(k)\mathbf{P}^{-}(k)\mathbf{H}^{T}(k) + \mathbf{R})^{-1} \\ \mathbf{x}(k) &= \mathbf{x}^{-}(k) + \mathbf{K}\Delta(k) \\ \mathbf{P}(k) &= (\mathbf{I} - \mathbf{K}\mathbf{H}(k))\mathbf{P}^{-}(k), \end{aligned} \tag{10}$$

where $\mathbf{F}$ is the Jacobian matrix of the state-transition function f, which contains the first derivative of the sensor value with respect to each state. Similarly, $\mathbf{H}$ is the Jacobian matrix of the sensor function h.

We will present the details of the EKF on ArduPilot and corresponding attack ideas in Sec. IV. We summarize its main stages as follows: 1) Initialization of States and Covariance Matrix; 2) State Prediction; 3) Covariance Prediction; 4) Measurement Fusion which consists of (i) Calculation of innovations; (ii) Update of states using the innovations and 'Kalman Gains'; (iii) Update of the Covariance Matrix [14].

D. False Data Injection (FDI) on Control Systems

FDI attacks have been examined in various control systems [20]. A conventional method for sensor validation is to check and recalibrate a sensor periodically according to a set of predetermined procedures [18]. However, it is not able to accomplish continuous assessment of a sensor, and becomes cost-ineffective and even infeasible as the number of sensors and the frequency of their measurements increase. Therefore, hardware and analytical redundant approaches have been developed to address this issue. However, on a resource-scarce system such as an sUAS, the use of redundant sensors may not be feasible due to cost and space constraints. As a result, analytical redundancy approaches such as EKF have been broadly used in such systems.

Common control systems are designed with efficiency and reliability as their main goals, without considering cyber attacks. As a result, how to handle the false data (due to malicious attacks) in control loops becomes a critical issue. Several FDI attacks on smart grid state estimation exploited their system models and circumvent their threshold detection schemes that mostly filters out bad data points far away from a threshold [20]. In a navigation system, the system model is different, and the bad data detection usually considers a sequence of measurements and estimates, instead of a single threshold. Therefore, in this paper, we design a new FDI attack on the navigation system model, which defeats a common bad data detection scheme in such systems.

One recent work on countering FDI attacks on UASs introduced a neural networks-based algorithm to detect FDI attacks on IMU sensors [21]. Compared with this paper, it focused on a different issue under different assumptions. Because IMU readings are not used in ArduPilot state updates [14], in this paper, we consider a new type of FDI attack on magnetometer readings.

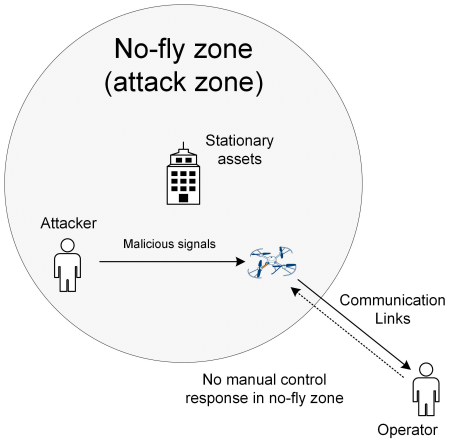


Fig. 2: Attack Model.

III. Proposed FDI Attack

A. Attack Model

As shown in Fig.2, we are protecting a stationary asset (e.g., a building) and set up a *no-fly zone* (also called *attack zone*) around the asset. An sUAS is controlled by an operator via a ground control station (GCS), and its goal is to reach the asset as close as possible, e.g., dropping a contraband. Once we detect that the sUAS is entering the *no-fly zone*, an authorized party will conduct wireless jamming on the sUAS such that it loses its communication link to the GCS and cannot receive valid GPS signals. Consequently, the sUAS has to depend on its on-board sensors to continue its pre-configured flight plan, without the operator's manual control. Assume that we can affect the sensor readings on the sUAS because they are consumer grade devices and not well protected in practice. Particularly in this paper, we assume that we can manipulate the readings of a magnetometer on an sUAS.

Assume that the sUAS uses EKFs to estimate its states, as in ArduPilot [14]. In particular, in the no-fly zone, it has to use its on-board IMU measurements for inertial navigation when under jamming, and use magnetometer measurements as another source to correct the orientation information for its INS. Our goal is to make it deviate from the protected asset and off the scheduled route, without being detected by its bad data detection schemes.

B. FDI Attack based on Magnetometer Readings

1) Attack Idea

In this section, we introduce a new type of FDI attack on an sUAS by manipulating its magnetometer readings. The main idea of our attack is: by modifying the magnetometer measurement readings, the measured orientation can be intentionally skewed, and accordingly the measured state $\mathbf{z}(k)$ in each of the three direction (NED) will also be altered. As long as $\mathbf{z}(k)$ is away from the actual state, it is possible to make the estimated position off the scheduled track, since $\mathbf{z}(k)$ plays an important role to "correct" the estimated state in the update step of EKF. Usually an sUAS has a

mechanism to detect the abnormal data from magnetometers. For example, Ardupilot achieves this by checking the consistency of innovation (the difference between a prediction and a measurement). To pass this check, a simple way is to make the difference between an actual measurement value and a revised value relatively small and also be consistent in the revised sequence.

This type of attack can be widely applied in many situations and have significant practical impacts. For example, in a no-fly zone, security guards can conduct such FDI attacks to force trespassing sUASs to immediately land or fly away to ensure public safety. We show two motivation examples in the following.

2) Two Attack Examples

Assumptions. For the ease of illustration of the attack method, we use the following simplified assumptions to show two attack examples:

- The sUAS enters the attack zone with an initial acceleration a_n in the North direction and an initial height h. The acceleration and the height is expected to maintain constant through the entire flight. It is scheduled to fly straight to the North.
- The attacker starts to inject false magnetometer readings to the sUAS when it enters the no-fly zone.
- Similar to common control systems, the sUAS uses a consistency check detector for magnetometer readings to avoid bad data. If the standard deviation of the reading values is greater than a threshold σ_τ, the current reading will be rejected.
- The process and measurement noises are zero-mean white noises with small standard deviations, i.e., the amplitude of noise is negligible compared to the injected malicious data. There is no intrinsic measurement bias for an IMU and a magnetometer.

1) Example 1. Crash Attack: In this attack, an attacker injects false magnetometer data that suggests the sUAS keeps rotating from the **North** direction to the **Up** direction at a constant angular velocity a_v at each time step. As a result, $\mathbf{z}(k)$ in Eq. 10 will gain a malicious bias at each update step. Since the state $\mathbf{x}(k)$ is updated based on the false $\mathbf{z}(k)$, the gap between the estimated state and the actual state will increase over the time, i.e., the estimated orientation will be closer to the Up direction step by step. To keep the orientation straight to the North, the sUAS will fly downwards and its height is expected to become lower and lower. When its actual orientation points straight down, the attacker can stop injecting the false magnetometer data and leave the sUAS to keep this orientation. The sUAS is expected to hit the ground eventually if no other sensor detects such an issue in time. It is easy to see that the larger ω_e is, the faster the drone will hit the ground.

2) Example 2. Circle Flying Attack: In this attack, an attacker injects a false bias angular velocity ω_d from the **North** direction to the **West** direction. Then, the estimated velocity and the estimated acceleration are expected to rotate around the Down direction axis, and the compromised route will look like a spiral, since it deviates from the North and its velocity is keep increasing. It will become further away from the destination.

Furthermore, this attack can also be extended into other variants. For example, the attacker may want the sUAS to be *trapped* in the attack zone until it runs out of power. To achieve this, it need to set a larger ω_d calculated based on the size of attack zone and the maximum flight time of the sUAS, in order to control the radius of the spiral is sufficient small and fit in the attacking zone. Moreover, the attackers can also choose to stop the data injection when the orientation is totally inversed (i.e., a 180° rotation). In this case, every sUAS entering into the attack zone will turn around and fly back. We call this as *the U-turn attack*.

3) Simulation Results

We evaluate the above two attack examples with simulations to demonstrate their impacts. We use the simplified simulation settings as follows:

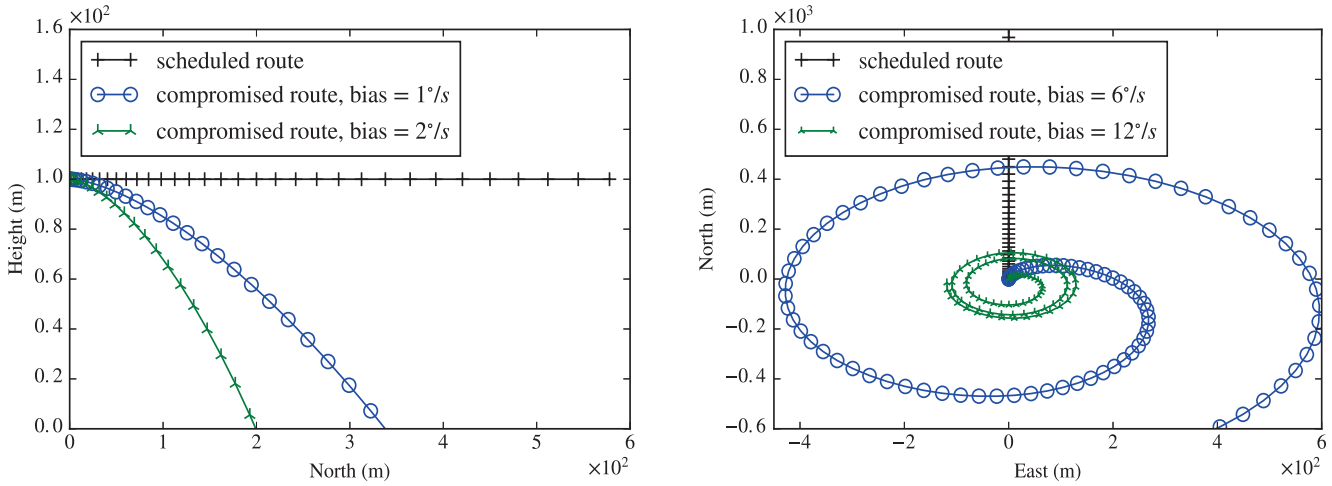
- The sUAS is at position $(0, 0, 100)$ under the NED coordinates when the attack starts, flying straight to the North with an acceleration of $1\ meter/second^2$ and a height of 100 meters, which are constant through the entire flight. Note that apart from the scheduled acceleration, the accelerometer is expected to receive other readings in all directions due to direction changes of the velocity.
- Except for Magnetometer and IMU, all other sensors including GPS do not work properly in the attack zone. This assumption is for ease of the discussion and shows the attack effect under this simplified situation.
- An attack is started at time slot 0, when the sUAS enters the no-fly zone.
- The sUAS does not have any warning mechanism for hitting the ground.
- The accelerometer measurement noises are negligible.
- The intrinsic biases for IMU and magnetometer are negligible.

All the other parameters used in our simulation are shown as Table I. The simulation code is written in Matlab, a variant of the code written for ArduPilot [22].

Fig. 3a compares the scheduled flight route with the routes under different Crash-attack settings. As shown in the figure, we can see that the sUAS under no attack flies straight to the North, and its height is roughly unchanged; in contrast, the sUAS under the two attack settings hit the ground after a short period of time. Moreover, it's not surprised that the drone with a larger injected bias $w_e = 2^\circ/s$ hits the ground faster compared to the one with injected bias $w_e = 1^\circ/s$

TABLE I: Default Simulation Settings

Para.	Val.	Description.
ΔT	$0.01s$	Length of time step.
σ_m	$0.75°/s$	Standard deviation of white noise on magnetometer reading.
v_{wind}	0	Wind velocity.
σ_τ	$20\sigma_m$	Threshold for magnetometer measurement consistency check.
$\mathbf{P}(k)$	diag(1)	Initial predict error matrix.
$\mathbf{Q}$	diag(0.01)	Process noise covariance matrix.
ω_e	$1°/s$ and $2°/s$	Injected angular velocity on magnetometer under crash attack.
ω_d	$6°/s$ and $12°/s$	Injected angular velocity on magnetometer under circle flying attack.



(a) Crash attack

(b) Circle flying attack

Fig. 3: Compromised routes under the Crash attack and the Circle flying attack.

(27 seconds vs 22 seconds in simulations), since the former has a greater velocity in the down direction.

Fig. 3b shows route comparison under the Circle-flying attacks for 100 seconds. It validates our prediction that the compromised sUAS flies along a spiral route within the attack zone. In addition, the radius of the compromised route decreases by more than 50% as the injected bias doubled.

Obviously, the above two simplified examples can not represent more complicated cases. We simply use them shows that the injected false data on sensors can significantly affect a flight path. More other attack strategies can be developed to achieve various effects. In the following, we will present the ArduPilot state estimation algorithms and discuss our ideas to apply the attack on it.

IV. Attacking EKF in ArduPilot

A. EKF in ArduPilot

1) ArduPilot State Estimation Algorithm In the early releases of ArduPilot, simple complementary filtering algorithms were used. As more advanced controller boards become available, more advanced state estimation algorithms are developed for ArduPilot. The current default state estimation algorithm used in the ArduPilot project is EKF [23]. We note that a number of different versions of EKF is included in the ArduPilot source code [24], [25]. Our description focuses on the default EKF in the system.

This EKF is a 24-state estimation algorithm, which includes the position, velocity, and angular orientation of a flight vehicle, in addition to a number of IMU sensor biases and magnetic field state, among others. The overall development of the ArduPilot EKF follows the standard implementation of EKF, in terms of equations to predict and update state covariance matrices, to calculate the Kalman gains, and to update the states. The majority of the EKF equations were first written in Matlab and then converted to the C/C++ programming language to be used by the ArduPilot project, including the state transition equations and the measurement equations. By relying on the Matlab Symbolic Toolbox, various relations and matrices were obtained, for example, the state transition matrix, the measurement (observation) matrix, the predicted state covariance matrix, and the Kalman gains for various state variables. We note that the state covariance matrix update and the state update were directly written in the C/C++ code in the EKF of ArduPilot, instead of relying on the Matlab Symbolic Toolbox to derive the equations. There could be a number of reasons for this, for example, the simplicity of these

equations, the optimization that we can perform due to the nature of matrices involved, and the sequential state update method adopted by the ArduPilot EKF.

One notable feature of the ArduPilot EKF is that the IMU data (the delta angle and velocity measurements of gyroscopes and accelerometers) were treated as control inputs in the state transition equations; they are not used as observations in updating or correcting the state variables. As an example, the NED velocity state transition equation of a flight vehicle is given below [22]:

$$v_{k+1} = v_k + [0; 0; g] * \delta + f(vm, vb, vn) \qquad (11)$$

where v_k and v_{k+1} are the predicted vehicle NED velocity at time k and $k+1$, respectively, g is the earth gravity, δ is the IMU measurement interval (between $k+1$ and k), and $f(vm, vb, vn)$ is a function of vm (IMU angle velocity measurement), vb (IMU angle velocity bias), and vn (IMU angle velocity noise). Please see [22] for the details of the function f. In essence, this state transition equation informs that IMU data is used to predict the flight NED velocity. The measurements of IMU gyroscopes and accelerometers are not used by EKF to fuse with the data of other sensors to correct the states or state covariances. Measurements from other sensors, including GPS, compass (for magnetometer), and barometer, are fused by the EKF to correct the states and state covariances, among others.

At the high level, the ArduPilot EKF algorithm works as the same as a standard EKF as follows. IMU data are read periodically. When IMU data are available, they are used to predict state variables, including the position, velocity, and orientation of the vehicle. In addition, the state covariance matrix is also predicted. Measurements from other sensors such as GPS are fused to correct the states and the state covariance matrix, when they arrive. In particular, when a measurement from another sensor arrives, the innovation (the difference between the predicted state value and the measured value) is computed. In addition, the Kalman gain is also computed, and the state is updated based on the Kalman gain and the innovation using the standard EKF state update equation. At the end of an EKF cycle, the state covariance matrix is also updated based on the Kalman gain, using the standard EKF state covariance update equation.

2) ArduPilot EKF for bad-data detection

Another use of EKF in the ArduPilot project is to detect faults in sensors, or more generally, to perform data consistency check on sensor measurement data, where unhealthy measurement data will be rejected in the fusion process to update the system states. A number of different data consistency checks are performed in the ArduPilot EKF, including the checks on GPS position, velocity measurements, and magnetometer measurements. The data consistency check on different sensors works in a similar manner; in the following, we use the magnetometer measurements as an example to illustrate how data consistency check works in the ArduPilot EKF [24].

For each measurement (observation) variable such as magnetometer measurements, the EKF maintains the standard deviation of the innovation of the variable, that is, the difference between the measured value and the predicted value. (In ArduPilot source code [24], the variance instead of the standard deviation is maintained; for the simplicity of this description, we consider the standard deviation instead of the variance.) When an innovation exceeds a predefined threshold related to the standard deviation, the corresponding measurement is considered to be unhealthy and is rejected in the fusion process of updating the states of the system. In the current ArduPilot EKF, the default threshold is three times of the standard deviation for a magnetometer measurement. Put in another way, if a magnetometer innovation for a measurement exceeds three times of the standard deviation of the innovation variable, the corresponding measurement is considered as being unhealthy and is rejected.

B. Attacking the ArduPilot Navigation Algorithms

Because the EKF implementation in Arduplot works mostly as the same as the standard EKF, our attack validated by the simulation should also work in ArduPilot. However, there may be some details we need to pay attention to when implementing FDI on ArduPilot as follows:

- There are a total of 24 states in EKF to be estimated. To show the attack effectiveness, we can choose to "disable" other unrelated states as much as we can in the implementation. For instance, we can set the wind velocity as 0 in all directions. Setting the biases of these states as 0 will also be helpful.
- Ardupilot EKF estimates the states of non-linear systems. As a result, to linearize the system, we need to approximate the state transition function by a sequence of small line segments. The line segments can be obtained by calculated the partial derivatives of the functions using the Jacobian matrix method.
- For the process and measurement noise covariances $\mathbf{Q}$ and $\mathbf{R}$, we can tune them for desired performance. A good initial choice is set them as diagonal matrices.
- As mentioned before, the EKF implementation in ArduPilot has improved in the prediction step. More specifically, in every iteration, EKF derives prediction from the IMU readings, and update it using other sensor readings. Since the intrinsic bias of IMU is very small compared to the injected bias on a magnetometer, the effect of false data injection attack is expected to be less obvious.

We evaluate our attack scheme based on the Matlab code from the author of the ArduPilot inertial navigation library [22]. The Matlab code is used to generate the real C++ code used in ArduPilot. We mainly monitor the impact of data injection on magnetometer data on the estimation of position and velocity. The data fusion process has three steps based on GPS data, magnetometer data, and airspeed

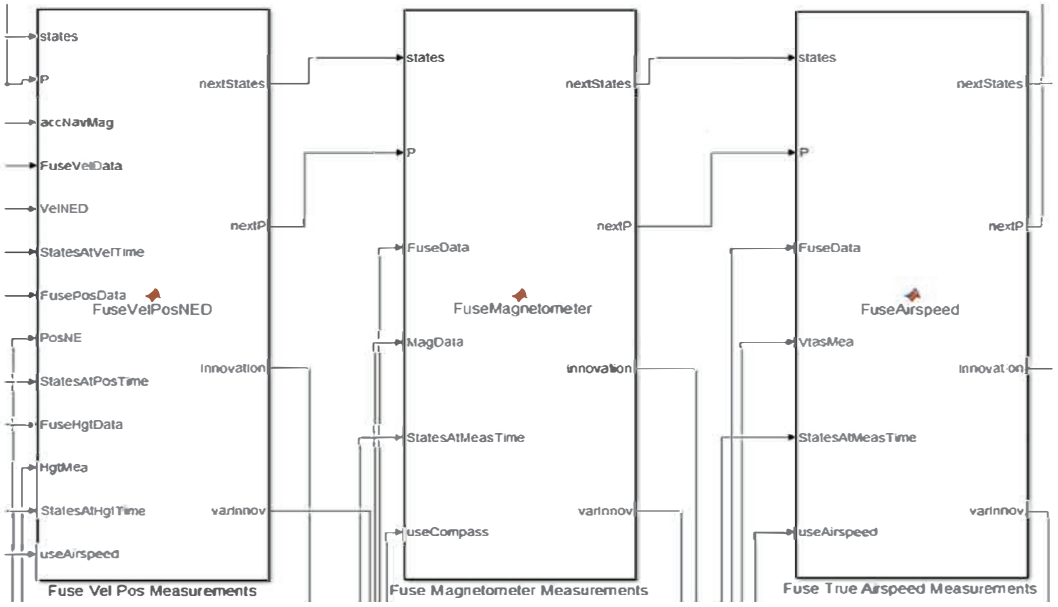


Fig. 4: Three-step fusion process.

data, as fig. 4. To better illustrate the effectiveness of our attack scheme, we disabled the GPS data by indicating all the GPS velocity and position data as unhealthy. Note that we do not indicate the height data as unhealthy, since the height data are not provided by GPS. In addition, we disable the fusion of airspeed measurement. However, we choose to let IMU sensors function correctly. The default thresholds for all consistency checks are set as 5 times of the standard deviation. The whole estimation process starts from `alignTime`, which is calculated based on GPS ground velocity; and will end at approximately $1525s$. In addition, the simulation contains the actual position data as reference. We also perform some other necessary modifications.

1) Impact of the Injection Magnitude on Single Dimension: We know that the estimated orientation of an sUAS can be influenced by the magnetometer data, which are usually 3-dimension vectors. In this part, we evaluate the attack effect by expanding the original magnetometer data on X-dimension to 150% and 200% in each simulation time slot. As Fig. 5a shows, the change in a single dimension of the magnetometer data may result in a distinct position estimation. In particular, we can see that, on the original data, the estimated positions in the north and east direction (the first two rows in each figure) keep increasingly farther from the reference position. We believe that this is because of the error propagation of IMU data. The estimation in the Down direction does not vary too much, since we do not disable the height measurement in data fusion. In contrast, on the injected data, the estimated positions are total different. Moreover, we can find there are some "sharp peaks" in the compromised position estimations (an example is in the 200% case, the estimated North position at time $\approx 585s$, we can see that the estimation falls to 0 sharply from approximately $1.23 \times 10^4 m$), and the number of these peaks increases as the amplification increases. We believe the reason is that a larger amplification may lead to an unhealthier data, which will be detected by the consistency check more frequently.

2) Impact of the Injection on Different Dimensions: We are also very interested in that, if there is a great difference when we inject data in a different dimension of magnetometer data. Fig. 5b shows the results when we expand the original data on X-dimension and Y-dimension to 150% in all time slots. It is easy to see that the compromised estimations are distinct when attacking different dimensions of the original data.

3) Discussion: In conclusion, from the above we can see the fake injection in magnetometer data may lead to a significant drift in position estimations. False data injection on magnetometer data is a very effective way to compromise the position estimation.

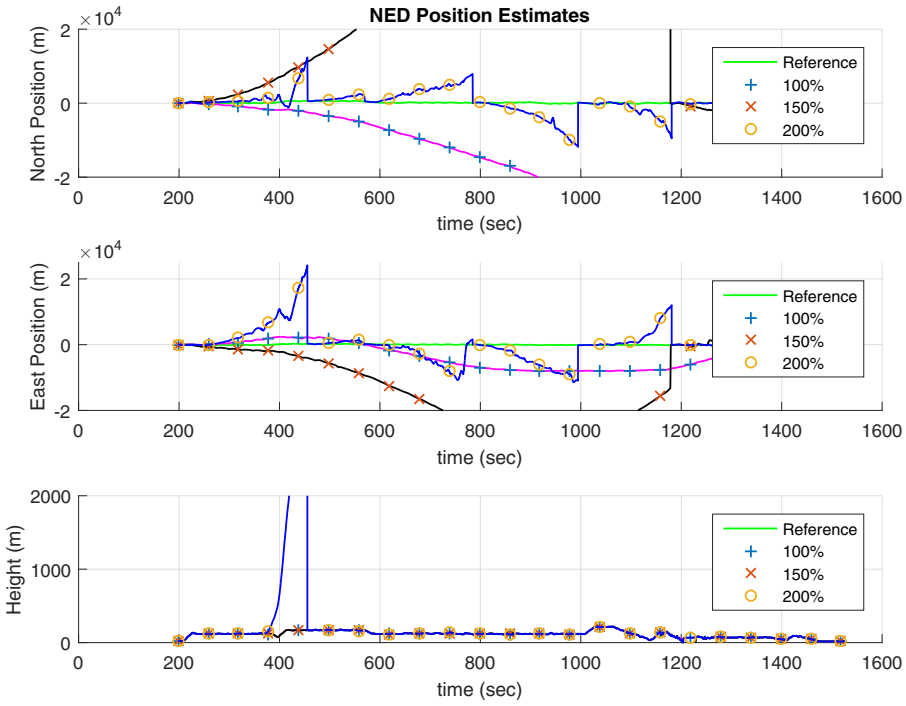
The above simulation is a simple case to demonstrate the effectiveness of our attack scheme. However, in real life, it is unlikely for attackers to manipulate the magnetometer measurement since $t = 0$ in a unlimited time window. Therefore, we are also investigating a scheme that allow attacker to compromise sUASs within a limited time from any certain moment.

V. Discussion and Conclusions

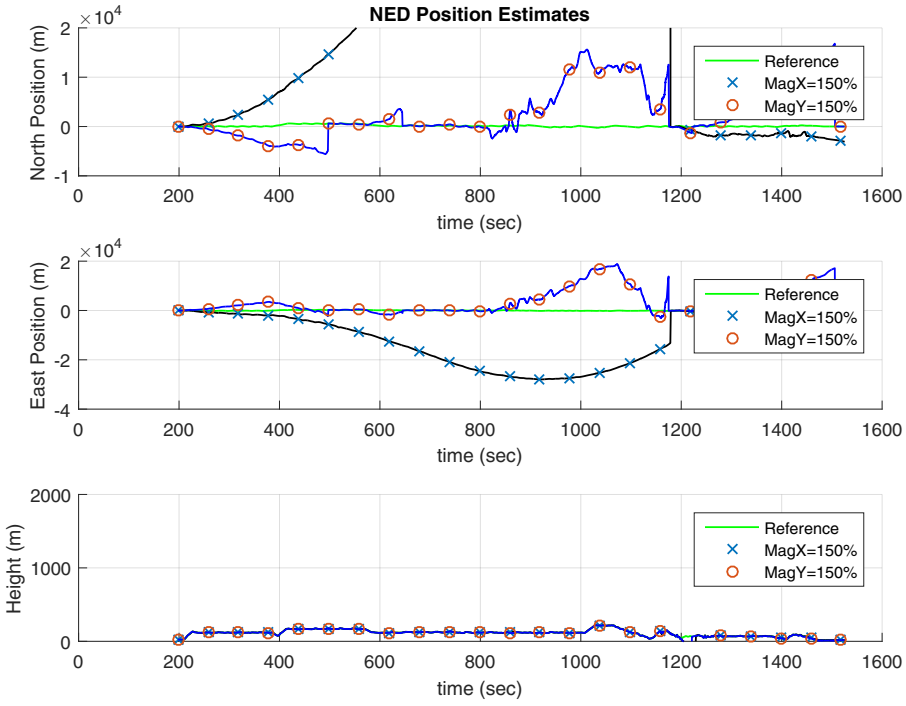
A. Countering FDI Attack on sUASs

While FDI attacks are possible on current sUASs, here are some ideas to identify and maybe counter FDI attacks on future sUASs.

a) Collision detection: If an sUAS is equipped with a collision detection mechanism, it may detect the crash attack.



(a) The impact of the injection magnitude on single dimension.



(b) The impact of the injection on different dimensions.

Fig. 5: The impact of magnetometer data injection on position estimations.

b) Comparison in the means of gyroscope and magnetometer readings: Since the injected bias on magnetometer is significant compared to the intrinsic bias of the gyroscope, we can conclude that the sUAS is under an FDI attack with high probability, when the difference between the means of gyroscope and magnetometer readings is non-negligible.

c) Orientation fluctuation detection: If the sUAS keeps adjusting its orientation to make itself stay on the schedule route, it is likely to be under false data injection attack.

d) Limitation on angular velocity: In a circle flying attack, we find that attacks with small injected bias may need an extreme large attack zone. Therefore, if we set a small velocity limitation for sUAS's rotation, the circle flying attack will not work in a limited-size attack zone.

e) Significant changes in magnetometer readings: In a circle flying attack, if the size of the no-fly zone is not big enough, when the sUAS just flies off the attack zone, the magnetometer reading is likely to experience a significant change. Therefore, when the magnetometer reading fluctuates heavily in a very short time, it was possible that the sUAS was under an FDI attack.

B. Conclusion and Ongoing Work

In this paper, we have proposed an FDI attack to exploit the limitation of sensoring and processing capabilities on an sUAS and manipulate its on-board sensors. We have used the*ArduPilot* flight control system as our subject and identified the vulnerabilities in its sensor data processing, and developed a method to manipulate on-board magnetometer readings while avoiding being detected by its bad-data detection schemes. Our initial results are very promising. Currently, we are investigating various techniques to compromise sUAS routes by subtly manipulating the injected data at each time slot. We are also working to manipulate the injected angular velocity to hold back the growing radius for trapping a transpassing sUAS in the no-fly zone.

ACKNOWLEDGMENT

We like to thank the research supports from NSF DGE-1662487, NSF DGE-1431484, a research grant from Applied Research Laboratory at the University of Hawaii, and a generous donation from Hawaiian Electric Company. We also thank Mr. Conghao Xu of University of Hawaii, who provided insight and expertise that greatly assisted the research.

REFERENCES

[1] 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.

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

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

[4] 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, 2014.

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

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

[7] Hualiang Li, Garret Johnson, Maverick Jennings, and Yingfei Dong. Drone profiling through wireless fingerprinting. In *Proc. of IEEE-CYBER 2017*, Aug. 2017.

[8] Eddy Deligne. Ardrone corruption, 2012.

[9] Fan Zhang, Shize Guo, Xinjie Zhao, Tao Wang, Jian Yang, François-Xavier Standaert, and Dawu Gu. A framework for the analysis and evaluation of algebraic fault attacks on lightweight block ciphers. *IEEE Trans. Information Forensics and Security*, 11(5):1039–1054, 2016.

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

[11] Blighter Surveillance Systems. Auds anti-uav defence system. http://www.blighter.com/products/auds-anti-uav-defence-system.html, 2016.

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

[13] ArduPilot Project. Ardupilot autopilot suite. http://ardupilot.org/ardupilot/.

[14] P. Riseborough. Application of data fusion to aerial robotics. In *in Proc. of Embedded Linux Conference*, Mar. 14, 2015.

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

[16] K. M. Smalling and K. W. Eure. A Short Tutorial on Inertial Navigation System and Global Positioning System Integration. Technical Report NASA/TMâĂŞ2015-218803, NASA, 2015.

[17] M. El-Diasty. An accurate heading solution using mems-based gyroscope and magnetometer integrated system. In *Proc. of ISPRS Technical Commission II Symposium. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume II-2, 2014.*, 2014.

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

[19] ArduPilot. Extended kalman filter navigation overview and tuning. http://ardupilot.org/dev/docs/extended-kalman-filter.html.

[20] 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.

[21] A. Abbaspour, K. Yen, S. Noei, and A. Sargolzaei. Detection of fault data injection attack on uav using adaptive neural network. *Procedia Computer Science 95, pp.193-200*, 2016.

[22] Paul Riseborough. Inertial navigation filter. https://github.com/priseborough/InertialNav.

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

[24] ArduPilot Project. https://github.com/ArduPilot/ardupilot.

[25] ArduPilot Project. Extended kalman filter navigation overview and tuning. http://ardupilot.org/dev/docs/extended-kalman-filter.html.