Compromising Flight Paths of Autopiloted Drones

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

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

***Abstract*—While more and more consumer drones are abused in recent attacks, there is still very little systematical research on countering malicious consumer drones. In this paper, we focus on this issue and develop effective attacks to common autopilot control algorithms to compromise the flight paths of autopiloted drones, e.g., leading them away from its preset paths. We consider attacking an autopiloted drone in three phases: attacking its onboard sensors, attacking its state estimation, and attacking its autopilot algorithms. Several first-phase attacks have been developed (e.g., [1]–[4]); second-phase attacks (including our previous work [5], [6]) have also been investigated. In this paper, we focus on the third-phase attacks. We examine three common autopilot algorithms, and design several attacks by exploiting their weaknesses to mislead a drone from its preset path to a manipulated path. We present the formal analysis of the scope of such manipulated paths. We further discuss how to apply the proposed attacks to disrupt preset drone missions, such as missing a target in searching an area or misleading a drone to intercept another drone, etc. Many potential attacks can be built on top of the proposed attacks. We are currently investigating different models to apply such attacks on common drone missions and also building prototype systems on ArduPilot for real world tests. We will further investigate countermeasures to address the potential damages.**

***Index Terms*—counter drone, autopilot, navigation**

I. Introduction

While they have enabled many new applications [7], consumer drones have been abused in many recent attacks [8], [9]. One recent case was at the UK's second largest airport. Malicious drones disturbed the normal operations during the busy 2018 Christmas season for three days. Clearly, we have to build effective solutions to stop such abuses.

Existing counter-drone solutions usually have two steps: we first identify an unauthorized drone entering a restricted airspace and then apply counter-drone solutions to disrupt or capture it. A typical setting is shown in Fig. 1. We set up a perimeter and apply drone detection schemes, using Radio Frequency (RF) communications, radars, acoustic monitoring, or image processing [10]–[12] to discover incoming drones. We focus on the second step in this paper: how to systematically counter the invading drones. In particular, in the no-fly zone, we assume that remote manual control is disabled and the malicious drone is on autopilot; we can apply the first-phase sensor attacks and the second-phase state estimation attacks to inject fake states into the autopilot navigation control. By examining the existing common autopilot algorithms, we propose several third-phase attacks to mislead invading drones.

Most existing counter-drone systems have been proposed by industry with straightforward solutions, such as jamming drone control channels or GPS receivers to trigger a drone switching to a default failsafe mode (e.g., landing when lost GPS signals over 10s), or capturing a drone with a net, etc. Such direct physical attacks work well when dealing with unsophisticated drone operators; but they also show serious limitations, e.g., they usually do not consider collateral damages. If a drone carries a bomb, we should not make it land in a protected critical space, e.g., an office building. The best solution in such situation is to lead the drone fly away from the target as far as possible. We have conducted a broad literature survey, and have not seen systematic research to address such issues. Therefore, it is urgent to investigate more intelligent counter-drone solutions.

Ideally, we want to precisely control the flight path of an unauthorized drone, e.g., making it miss its preset waypoints in its mission plan. In this paper, we consider attacking an autopiloted drone in three phases. The **first-phase attacks** focus on compromising the sensor readings of an autopiloted drone. Several first-phase attacks have shown that such attacks are completely feasible (e.g., [1]–[4]). Based on such first-phase attacks, we have proposed **the second-phase attacks** [5], [6], e.g., exploiting the weaknesses in common drone state estimation algorithms. In this paper, we focus on **the third-phase attacks** by utilizing the compromised state estimation to fool common autopilot algorithms to make a drone deviate from its flight paths. We do not assume that we can remotely tamper actuators.

Our Goal. Our goal is to compromise drone autopilot control algorithms to manipulate flight paths. We have conducted

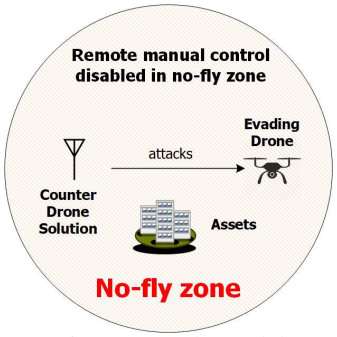


Fig. 1: Attack Model.

extensive investigation on popular open-source flight control systems (such as *ArduPilot* and *Paparazzi*), and discovered multiple weaknesses in common autopilot control algorithms. Because sensors often have occasional errors, drone control systems usually use state estimation algorithms to address these errors. Kalman Filter (KF) and its variants are the most popular estimation algorithms in drone control systems. A drone autopilot system is dependent on these state estimations to adjust flight parameters. We have proposed several second-phase attacks to manipulate state estimation algorithms in our previous papers. In this paper, we focus on exploiting the weakness of autopilot algorithms to manipulate a drone in real time in order to make it follow (or not follow) certain flight paths, e.g., away from a target or missing certain points in a search sweep. To our best knowledge, we have not seen similar work in this direction.

The remainder of this paper is organized as follows. We will introduce related background of autopilot control in Sec. 2. In Sec. 3, we will then present several attacks on autopilot control algorithms, and analyze the scope of feasible paths under such attacks. We will further discuss how to apply the attacks to disrupt drone missions, and show basic simulation evaluations in Sec. 4. We conclude this paper and discuss our current and future research in Sec. 5.

II. Drone Autopilot Background

A. Drone Control Model

We illustrate a common automatic control loop of a drone in Fig. 2. The control system periodically reads physical measurements from sensor readings, and then estimate system states for further control decisions via sensor fusion schemes; based on the estimated system states, the autopilot component makes control adjustments of actuators to achieve control movements.

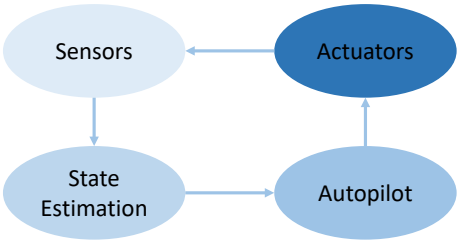


Fig. 2: Common Drone Control Loop.

To deal with an evading drone, we need attack schemes to affect its flight path based on our goals, e.g., making it miss its preset waypoints. As we mentioned in the above, we consider such attacks as the *third-phase attacks* that exploit the weakness of drone autopilot control algorithms. These third-phase attacks are based on the *first-phase attacks* on sensors and the *second-phase attacks* on state estimation algorithms. The first-phase attacks focus on compromising the sensor readings of a drone. By understanding how state estimation algorithms use these sensor data, we can figure out how to manipulate these readings such that we can fool the state estimation algorithms of a drone as we need in the second-phase attacks. Similarly, exploiting the weakness of drone flight control algorithms, we know what states will mislead the drone movement control in the third-phase attacks. In this paper, we assume that sensor attacks help us in state estimation and autopilot attacks; we do not focus on sensor attacks. Attacking sensor readings has been an active research area in recent years, and existing sensor attacks achieves good results in manipulating IMU sensors [3], [4], GPS [1], [2], etc. Therefore, in this paper, we assume that first-phase attacks help us compromise sensors, and second-phase attacks (proposed by other researchers and us [6]) help us mislead the state estimation algorithms. We will focus on the third phase attacks.

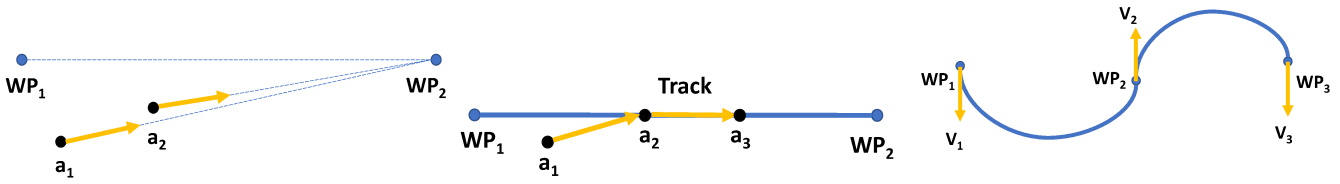
B. Autopilot Control Algorithms in ArduPilot

In this section, we introduce the basic mechanism for autopilot control, which adjust drone actions to follow a preset flight path with specific waypoints. Most popular autopilot control algorithms use GPS-based waypoints navigation. Therefore, the path control of a drone usually includes: adjustment of roll and pitch for desired attitude; adjustment of heading and altitude for trajectory or waypoints tracking; and waypoint navigation. The autopilot system usually consists of two basic controllers: An altitude controller makes sure the drone at the correct altitude, and a velocity/heading controller navigates the drone to fly through the desired waypoints. We focus on both controllers in this work to change drone flight paths. In this paper, we investigate three autopilot navigation algorithms used in the popular open source flight control system *ArduPilot*. ArduPilot uses the Proportional Integral Derivative (PID) method for the controller, which adjust the actuators in each time step to achieve the preset target points.

The *Basic AutoPilot Algorithm* is the fundamental auto-navigation control algorithm. As shown in Fig. 3a, the controller calculates the target bearing in every time step based on the current position of the drone and the target position. The controller will then navigate the drone towards the target with this bearing and a desired velocity.

The *Linear Track-based Navigation Algorithm* leads a drone to a target location following the linear track from a source to a destination, as shown in Fig. 3b. To make sure that the drone flies along the track, the controller will calculate the track error (the distance between the drone's current position and the track) in every time step, and limit the error within a small range. As a result, a drone will not deviate from the track too far. Usually, the rate to correct the track error (i.e., moving the drone back to the track) is stable as a constant. To lead the drone to reach the destination, the controller will calculate the drone's speed along the track and stabilize it at a desired one.

The *Spline Track-based Navigation Algorithm* makes a drone fly smoothly passing through waypoints during a flight. To achieve this, one simple solution is to make a drone slow down when it approaches a waypoint. However, this may



(a) Basic autopilot algorithm. (b) Linear track-based autopilot algorithm. (c) Spline track-based autopilot algorithm.

Fig. 3: Three types of autopilot algorithm.

cost the drone more time and power. In order to tackle this problem, we can make the flight track be a spline. ArduPilot uses Cubic Hermite splines for flight tracks. As shown in Fig. 3c, a Cubic Hermite spline is defined by source WP_1, destination WP_2, origin velocity V_1, and destination velocity V_2. In particular, any position $P(t)$ in this spline can be defined as:

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} WP_1 \\ WP_2 \\ V_1 \\ V_2 \end{bmatrix}, \tag{1}$$

where $t \in [0, 1]$. The corresponding velocity is:

$$V(t) = \begin{bmatrix} 2t^3 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix}^T \cdot \begin{bmatrix} WP_1 \\ WP_2 \\ V_1 \\ V_2 \end{bmatrix}. \tag{2}$$

When we piece splines together, we should ensure that the positions and velocity rates are matched at each connection point. Similarly, in this method, the controller constrains the track error to make the drone move along the track, with a desired speed.

III. Proposed Attacks

In this section, we focus on designing general methods to compromise common autopilot navigation algorithms. In particular, we propose three attacks to the three common autopilot algorithms, to compromise the flight paths of autopiloted drones. We further formally analyze, when under the proposed attacks, the feasible position scopes of an autopiloted drone with a preset track from one waypoint to another. We then discuss how to apply such analysis to multiple tracks and other complicated cases (discussed in the next section). We also examine the key factors that affect the proposed attacks.

We assume that first-phase attacks (cyber attacks or physical attacks) can help us compromise drone sensor readings, such that we can feed "fake" data to drone control algorithms to mislead drone navigation control. Recently, several attacks have been developed to compromise the readings of MEMS sensors, by exploiting their physical vulnerabilities. In [3], the authors showed that an adversary can incapacitate the MEMS gyroscopes of a drone, by crafting acoustic signals with the same resonant frequencies of gyroscopes to degrade their

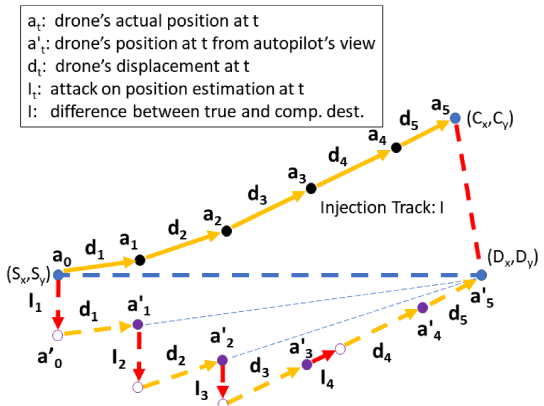


Fig. 4: Illustration of attacking the basic autopilot algorithm.

performance. In [4], the authors further developed acoustic injection attacks on MEMS accelerometers. Moreover, manipulating GPS readings [1], [2] has been conducted in field tests. We are also investigating potential attacks on MEMS barometer sensors used on drones, following other MEMS research [13]. (We are fully aware that this assumption is fairly strong. These attacks need significant efforts, which are not addressed here.) Furthermore, we also assume that we can perform second-phase attacks to manipulate state estimation such as positions. Other research and our previous work [6], [14] have shown such fake-data injection attacks on state estimation. In this paper, we focus on the third-phase attacks to exploit the weaknesses of common autopilot algorithms. We have briefly discussed the ideas of two third-phase attacks in [14] without details due to space limit; that paper focused on attacking state estimation. In this paper, we present the detailed attack algorithms and formal analysis of position scopes; we also develop the attack on spline track-based autopilot with analysis; we further apply our position scope analysis to build more general attacks (such as multi-track attacks).

A. Attacking the Basic Autopilot Algorithm

The main idea to attack the basic autopilot algorithm is: by attacking the position estimation, we continuously adjust the heading of a drone little by little, until the drone exactly points to a compromised destination. As long as the bearing has been determined, we only need to adjust the remaining flight distance, such that it can reach the compromised destination precisely. Although the basic autopilot algorithm is simple, it is actually a little harder to compromise compared

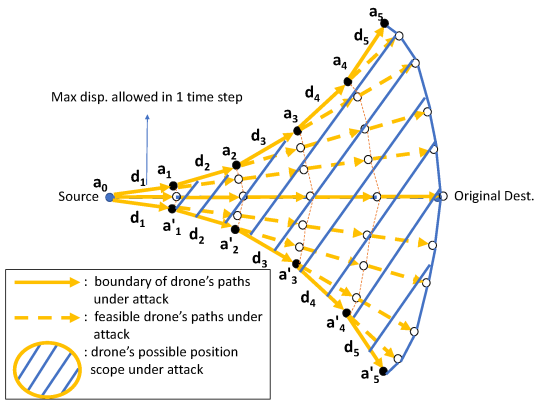


Fig. 5: Feasible position scope for a drone under the attack of basic autopilot algorithm.

to the linear-track autopilot algorithm, because it is difficult to predict and manipulate the bearing information in each time step as the drone moves in each step and calculates a new bearing in every step. We show the algorithm in *Algorithm 1*.

We use an example in Fig. 4 to show the proposed algorithm in detail. For ease of illustration, here we ignore the altitude and only consider the flight path in two dimensions: a source (S_x,S_y), an original destination (D_x,D_y) and a compromised destination (C_x,C_y), where we want to mislead a drone to fly to. Since $C_y > D_y$ in this case, we apply a downward position injection such that the direction of the displacement points to the above. Here the optimal strategy is to make the position injection vector $\mathbf{I}_1$ vertically downward, with the maximum magnitude that it can achieve using the second-phase state estimation attack. This strategy will maximize the bearing change. Then, the autopilot finds the drone at a "new" position $\mathbf{a}'_0$, calculates the current bearing, and moves towards the "destination" with the displacement allowed in one time step (based on its velocity and the length of a time step). Now the drone believes it is at $\mathbf{a}'_1$, but actually at $\mathbf{a}_1$. In the following, the attackers repeat the above process until the drone is exactly towards the compromised destination. Note that the magnitude of the last injection should be tuned such that we set the perfect bearing for the drone. Next, we need an additional injection $\mathbf{I}_4$ with the same or opposite moving direction of the drone, to adjust the remaining flight distance with the given time. More specifically, $\mathbf{I}_4 = \mathbf{I} - \mathbf{I}_1 - \mathbf{I}_2 - \mathbf{I}_3$, where $\mathbf{I}$ is the vector difference between the original destination and the compromised destination. (Note that it is vector arithmetic.) At the end, the drone reaches the compromised destination $\mathbf{a}_5$ at the given time, but it believes that it arrives at the original destination $\mathbf{a}'_5$. We generalize the procedure in Algorithm 1 (without loss of generality, assume $C_x < D_x$, $C_y > D_y$, $S_x < D_x$, and $S_y = D_y$):

Feasible Position Scope under the Attack. In the following, we formally analyze the scope of all feasible positions when a drone is under the proposed attack. As Fig. 5 shows, the top and bottom yellow solid lines indicate the actual flight path when we apply the maximum vertical position injections in each time step. In these boundary cases, the drone experiences the largest bearing change. Intuitively, these two paths limit the feasible position scope. Furthermore, we can also find that every feasible position inside the aforementioned scope is achievable under the attack, because we can adjust the injection at each time step to find a flight path covering any point in the scope.

Assume the maximum position injection in each time

Algorithm 1: Attacking the basic autopilot algorithm

input : maximum position injection in each time step I_{max}, and drone's displacement in one time step d
output: injection vector $\{\mathbf{I_i}\}$ that leads the drone to (C_x, C_y)

Initialization: for time step i, $\mathbf{I_i} \longleftarrow \emptyset$, the bearing towards the original destination from the autopilot's view: $\sigma_i \longleftarrow \emptyset$, the actual bearing towards the compromised destination: $\delta_i \longleftarrow \emptyset$, the actual position: $\mathbf{a_i} \longleftarrow \emptyset$, the position from the autopilot's view: $\mathbf{a'_i} \longleftarrow \emptyset$; $\mathbf{a_0} \longleftarrow (S_x, S_y)$; $\mathbf{a'_0} \longleftarrow (S_x, S_y)$; $\sigma_0 \longleftarrow 0$; $\delta_0 \longleftarrow \arctan \frac{D_y - S_y}{D_x - S_x}$; $\mathbf{I} \longleftarrow (D_x - C_x, D_y - C_y)$; $\mathbf{I_0} \longleftarrow (0,0)$; $i \longleftarrow 1$; $\Delta \longleftarrow \sqrt{(D_x - C_x)^2 + (D_y - C_y)^2}$;
while $|\sigma_{i-1}| < |\delta_{i-1}|$ **do**
  if $i = 1$ **then**
    $\sigma_i \longleftarrow \arctan \frac{I}{\Delta}$;
  else
    $\sigma_i \longleftarrow \arctan \frac{i \cdot I - d \cdot (\sum_{n=0}^{i-1} \sin \sigma_n)}{\Delta - d \cdot (\sum_{n=0}^{i-1} \cos \sigma_n)}$;
  $\mathbf{I_i} \longleftarrow (0, -I_{max})$; keep injecting max until $|\sigma_{i-1}| \geq |\delta_{i-1}|$;
  $a_i \longleftarrow (a_{(i-1)_x} + d \cdot \cos \sigma_i, a_{(i-1)_y} + d \cdot \sin \sigma_i)$;
  $a'_i \longleftarrow a_i + \sum_{n=1}^{i} \mathbf{I_i}$;
  $\delta_i \longleftarrow \arctan \frac{C_y - a_{i_y}}{C_x - a_{i_x}}$;
  $i \longleftarrow i + 1$;
if $|\sigma_{i-1}| > |\delta_{i-1}|$ **then**
  $i \longleftarrow i - 1$; if we over-estimate the bearing change in the previous step, we need to make sure $|\sigma_i| == |\delta_i|$;
  $\delta_i \longleftarrow \arctan \frac{C_y - a_{(i-1)_y}}{C_x - a_{(i-1)_x}}$;
  $\sigma_i \longleftarrow \delta_i$;
  $\mathbf{I_i} \longleftarrow (0, (D_x - a_{(i-1)_x}) \cdot \tan \sigma_i - a_{(i-1)_y})$;
  $a_i \longleftarrow (a_{(i-1)_x} + d \cdot \cos \sigma_i, a_{(i-1)_y} + d \cdot \sin \sigma_i)$;
  $a'_i \longleftarrow a_i + \sum_{n=1}^{i} \mathbf{I_i}$;
  $i \longleftarrow i + 1$;
Now $|\sigma_{i-1}| == |\delta_{i-1}|$, we do not need vertical injection any more. We just need to figure out the remaining injections towards the compromised destination in the following;
while $\sqrt{(D_x - a'_{(i-1)_x})^2 + (D_y - a'_{(i-1)_y})^2} > I_{max}$ **do**
  $\delta_i \longleftarrow \delta_{i-1}$; $\sigma_i \longleftarrow \sigma_{i-1}$;
  if $(\mathbf{I} - \sum_{n=0}^{i-1} \mathbf{I}_n) > I_{max}$ **then**
    $\mathbf{I_i} \longleftarrow (I_{max} \cdot \cos \sigma_i, I_{max} \cdot \sin \sigma_i)$;
  else
    $\mathbf{I_i} \longleftarrow \mathbf{I} - \sum_{n=0}^{i-1} \mathbf{I}_n$;
  $a'_i \longleftarrow (a'_{(i-1)_x} + d \cdot \cos \sigma_i, a'_{(i-1)_y} + d \cdot \sin \sigma_i) + \mathbf{I_i}$;
  $a_i \longleftarrow (a_{(i-1)_x} + d \cdot \cos \sigma_i, a_{(i-1)_y} + d \cdot \sin \sigma_i)$;
  $i \longleftarrow i + 1$;
if $(\mathbf{I} - \sum_{n=0}^{i-1} \mathbf{I}_n) > I_{max}$ **then**
  return "Attack Failure!";
else
  $\mathbf{I_i} = \mathbf{I} - \sum_{n=0}^{i-1} \mathbf{I}_n$;
return injection vectors $\{\mathbf{I_i}\}$;

step is represented as I_{max}. To reach the boundaries, the injections should be either upward or downward. The drone's displacement in one step is d based on its velocity and the length of the time step. With these parameters, we can get the exact position scope at time step t.

As Fig. 5 shows, at time step 1, we apply an upward or downward position injection for reaching the top or bottom boundaries, respectively. In the following, we denote the constant $\sqrt{(D_x - S_x)^2 + (D_y - S_y)^2}$ as Δ. The relative bearing of the drone towards to the original destination at time step 1 is σ_1, where $\sigma_1 = \pm\arctan\frac{I_{max}}{\Delta}$. Then, we have $a_1 = (S_x + d \cdot \cos\sigma_1, S_y + d \cdot \sin\sigma_1), a'_1 = (S_x + d \cdot \cos\sigma_1, S_y - d \cdot \sin\sigma_1)$.

In time step 2, starting at a_1 or a'_1, we continue to let the target drone make a maximum bearing change. Based on similar analysis, we can get the new bearing is $\sigma_2 = \pm\arctan\frac{2I_{max} - d\cdot\sin\sigma_1}{\Delta - d\cdot\cos\sigma_1}$. We then have $a_2 = (a_{1_x} + d \cdot \cos\sigma_2, a_{1_y} + d\cdot\sin\sigma_2), a'_2 = (a'_{1_x} + d\cdot\cos\sigma_2, a'_{1_y} - d\cdot\sin\sigma_2)$, where $a_{1_x} = a'_{1_x} = S_x + d\cdot\cos\sigma_1$, $a_{1_y} = S_y + d\cdot\sin\sigma_1$, and $a'_{1_y} = S_y - d\cdot\sin\sigma_1$. Repeating the above process, at time step $t > 2$, we have

$$\sigma_t = \pm\arctan\frac{t\cdot I_{max} - d\cdot(\sum_{n=1}^{t-1}\sin\sigma_n)}{\Delta - d\cdot(\sum_{n=1}^{t-1}\cos\sigma_n)}. \tag{3}$$

and

$$\begin{aligned} a_t &= (a_{t-1_x} + d\cdot\cos\sigma_t, a_{t-1_y} + d\cdot\sin\sigma_t) \\ &= (S_x + d\cdot(\sum_{n=1}^{t}\cos\sigma_n), S_y + d\cdot(\sum_{n=1}^{t}\sin\sigma_n)); \\ a'_t &= (a'_{t-1_x} + d\cdot\cos\sigma_t, a'_{t-1_y} - d\cdot\sin\sigma_t) \\ &= (S_x + d\cdot(\sum_{n=1}^{t}\cos\sigma_n), S_y - d\cdot(\sum_{n=1}^{t}\sin\sigma_n)), \end{aligned} \tag{4}$$

Now we get all the positions for a_1 to a_t and a'_1 to a'_t, then the top and bottom boundaries are made of all the line segments $\overline{a_{n-1}, a_n}$ and $\overline{a'_{n-1}, a'_n}$ for $n = 1, 2, \ldots, t$ (where $a'_0 = a_0$). For the boundary "arc" $\overline{a_t a'_t}$, we can get its precise shape if needed. But we can easily approximate it with multiple short lines, by finding multiple position points on the "arc". For example, we can give $2t + 1$ position points on this "arc", including: $a_t, (a_{t-1_x} + d\cdot\cos\sigma_{t-1}, a_{t-1_y} + d\cdot\sin\sigma_{t-1}), (a_{t-2_x} + 2d\cdot\cos\sigma_{t-2}, a_{t-2_y} + 2d\cdot\sin\sigma_{t-2}), \ldots (a_{t-n_x} + n\cdot d\cdot\cos\sigma_{t-n}, a_{t-n_y} + n\cdot d\cdot\sin\sigma_{t-n}), \ldots (D_x, D_y), \ldots (a'_{t-n_x} + n\cdot d\cdot\cos\sigma_{t-n}, a'_{t-n_y} - n\cdot d\cdot\sin\sigma_{t-n}), \ldots (a'_{t-2_x} + 2d\cdot\cos\sigma_{t-2}, a'_{t-2_y} - 2d\cdot\sin\sigma_{t-2}), (a'_{t-1_x} + d\cdot\cos\sigma_{t-1}, a'_{t-1_y} - d\cdot\sin\sigma_{t-1}), a'_t$. These are the drone's positions at time step t in the cases when the continuous upward or downward position injections stop at time step n for $n = 0, 1, 2, \cdots, t$ (i.e., there are injections in the first n time steps, but no injection in the last $t - n$ time steps).

For theoretical interests, we also figure out the eventual scope when t is sufficient large. When $I_{max} > d$, the relative bearing towards to the original destination σ_t will be close to $\pm\pi/2$. We have $a_t = (a_{t-1_x} + d\cdot\cos\sigma_t, a_{t-1_y} + d\cdot\sin\sigma_t) \approx (a_{t-1_x}, a_{t-1_y} + d) \approx (D_x, a_{t-1_y} + d); a'_t = (a'_{t-1_x} + d\cdot\cos\sigma_t, a'_{t-1_y} - d\cdot\sin\sigma_t) \approx (a_{t-1_x}, a_{t-1_y} - d) \approx (D_x, a_{t-1_y} - d)$. This means the drone's position will go straight upward or downward near the vertical line at D_x on the x-axis . If $I_{max} < d$ but they are close, then the drone will also fly upward or downward when t is very large. However, it will stop in finite steps, since the drone will eventually reach the "destination" from the autopilot's view. If $I_{max} \ll d$, then the drone will stop after a few time steps before the relative bearing σ_t approaches $\pm\pi/2$.

Applying the above analysis result to lead a drone along a single desired track. Given a desired attack flight track, we can directly use the above results to figure out if it is feasible or how to build position injections to achieve it. First, the desired track must be in the feasible position scope based on the flight parameters. If a part of the desired track is outside the position scope, it cannot be achieved. Second, although the whole desired track is within the scope, it is not necessarily feasible because we have to make sure the bearing change at each time step on the desired track is achievable, due to the limit of bearing change at a time step. In general, as long as we find a feasible path, we just need to backtrace the corresponding position injections such that we make the drone flying on the desired path.

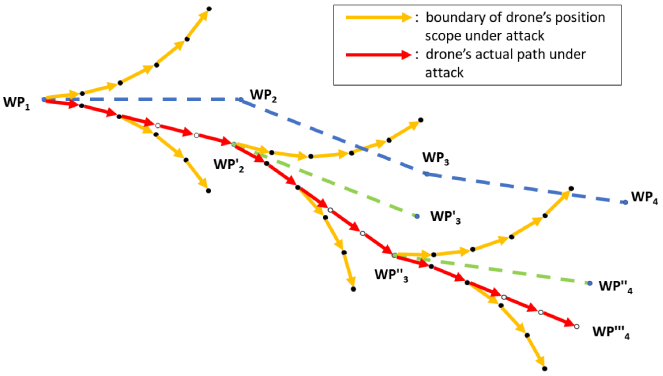


Fig. 6: Compromising drone's flight path via multiple waypoints.

Compromising drone's flight path with multiple waypoints. In the above we have discussed how to attack drone's actual positions on one track between two waypoints. A follow-up question is how to manipulate a drone's path with multiple waypoints. We illustrate the basic idea with Fig. 6. Assume a flight path (starting at WP_1 and ending at WP_4) contains three tracks: $\overline{WP_1 - WP_2}$, $\overline{WP_2 - WP_3}$, and $\overline{WP_3 - WP_4}$. Our attack goal is to let the drone reach WP'''_4, instead of WP_4. We can first mislead the autopilot navigation to make it reach WP'_2. At this point, we apply the attack again to make it reach WP'_3. We apply the same single-track attack to guide it to the compromised destination WP'''_4. Note that all the actual paths should be within the

position scope of each track. In particular, the position scope of the second or third track is based on the virtual track $\overline{WP_2' - WP_3'}$ and $\overline{WP_3'' - WP_4''}$, respectively.

B. Attacking the Linear Track-based Autopilot Algorithm

If the desired flight track is linear, the position error caused by the proposed second-phase attacks is easy to figure out: since any "drift" (due to the injection on the position estimation) will be corrected by the autopilot algorithm, the sum of the total injections on position estimation should be equal to the inverse of the injection track (i.e., the difference between the original and compromised destinations). We present the attack algorithm to mislead the drone to a specific compromised destination in the following.

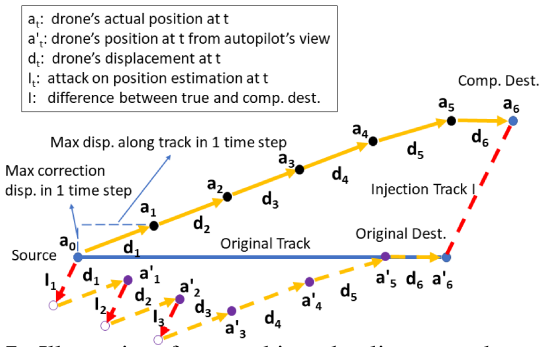


Fig. 7: Illustration for attacking the linear track autopilot algorithm.

As shown in Fig. 7, we first apply a position estimation injection $\mathbf{I}_1$ in time step 1. Without loss of generality, we let the magnitude of the injections $\mathbf{I}_t$ (except the last injection) be the maximum value that it can be achieve with the second-phase attack, and all the injection vectors be parallel to the injection track. After the injection, the autopilot algorithm believes the drone has drifted away from the track and will try to make it return to the track again. It will let the drone make an adjustment $\mathbf{d}_1$, which is the sum of the allowed displacement along the track and the allowed displacement correction towards the track in one time step. As a result, the drone reaches $\mathbf{a}_1$ but believes itself at $\mathbf{a}'_1$. In the following, the attacker applies other two injections $\mathbf{I}_2$ and $\mathbf{I}_3$ on the position estimation such that $\mathbf{I} = \mathbf{I}_1 + \mathbf{I}_2 + \mathbf{I}_3$ (note that it is vector arithmetic). The controller will try to correct the drone's position with displacement $\mathbf{d}_2$ and $\mathbf{d}_3$. By now, we have finished all the injections on position estimation for this attack. Therefore, we will then let the autopilot controller continuously correct the drone's flight path and move along the "track" to the "destination". Finally, the drone arrives at the compromised destination $\mathbf{a}_6$, but believes that it reaches the original destination $\mathbf{a}'_6$. Due to the space limit, we move the detailed attack algorithm to a technical report [15].

Feasible position Scope under this Attack. Similar to attacking the basic autopilot algorithm, as Fig. 8 shows, the top and bottom yellow solid lines indicate the actual flight path when it is applied the proposed position injection with maximum possible magnitudes in each time step. Intuitively,

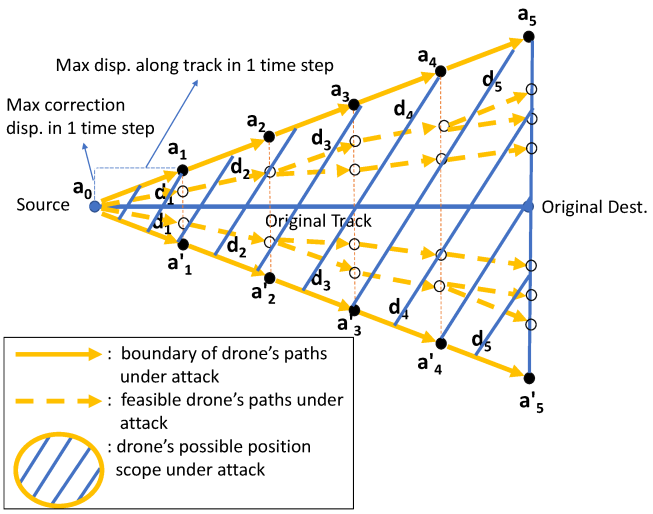


Fig. 8: Feasible position scope of the target drone when the adversaries attack the linear track-based autopilot algorithm.

the two yellow solid paths limit the feasible position scope. In addition, every feasible position inside the aforementioned scope is achievable by the drone under the possible attack for similar reasons.

The mathematical form of the position scope here is relatively straightforward. With the same notation as the above attack (in addition, we denote the correction displacement and the displacement along the track in one time step as d_c and d_t, respectively), we have:

(a) If the drone does not return on the track after the autopilot's correction in one time step, as Fig. 8 shows, at time step 1, after we apply an position injection, we have $a_1 = (S_x + d_t, S_y + d_c); a'_1 = (S_x + d_t, S_y - d_c)$. In addition, the relative bearing of the drone will not change through the whole attack. Thus, we have $a_t = (S_x + t \cdot d_t, S_y + t \cdot d_c); a'_t = (S_x + t \cdot d_t, S_y - t \cdot d_c)$. Therefore the scope is a triangle with three vertices: source, a_t, and a'_t. In this case, the scope will become very large when t is large.

(b) If drone's position return the track after the autopilot's correction in one time step, we have $a_t = (S_x + t \cdot d_t, S_y + t \cdot I_y); a'_t = (S_x + t \cdot d_t, S_y - t \cdot I_y)$, where I_y is the size of the y-component of the position estimation injection. Here the relative bearing of the drone will not change through the whole attack as well, except at the last time step when the drone reaches the "destination" from the autopilot's view. Specifically, at the last time step, the drone's displacement along the track may be shorter to reach the "destination". Therefore the scope here is made of a triangle with three vertices: source, a_{t-1}, and a'_{t-1}, and a trapezoid with four vertices: a_{t-1}, a'_{t-1}, a_t, and a'_t, where t is the final time step.

C. Attacking the Spline Track-based Autopilot Algorithm

In ArduPilot, although the navigation controller supports the spline track-based autopilot algorithm, they do not support a correction approach when the drone is off the track. In this section, we will first design a corresponding correction

algorithm, then develop the attack schemes. We will also analyze the feasible position scope under attack.

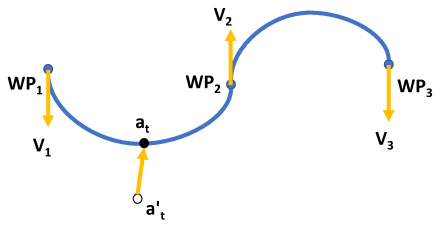


Fig. 9: Position correction algorithm for the spline track-based autopilot algorithm.

Proposed Position Correction Algorithm. As we mentioned before, in each time point $t \in [0, 1]$, the expected position of the target drone can be determined using eq. (1). We propose a simple position correction algorithm as shown in Fig. 9: as long as the autopilot controller finds that the track error (the difference between the expected and the actual positions) is beyond the threshold τ_{spline} at any time, it asks the drone to suspend the current flight plan and fly directly back to the expected position. After reaching the expected position, it restarts the flight plan. If the track error is within the threshold, then the controller will guide the drone fly parallelly to the track. This correction algorithm may not be the most time-efficient one, but it can assure the drone flies closely along the track in most time.

Attack Schemes. Attacking the spline track-based autopilot algorithm with the proposed position correction is similar to attacking the linear track-based autopilot algorithm. In particular, the attackers will put a series of position estimation injections, whose sum is equal to the inverse of the injection track. We will discuss the attack schemes in two different cases in the following. For simplicity, we assume the maximum correction displacement in one time step $d_c \gg \tau_{spline}$.

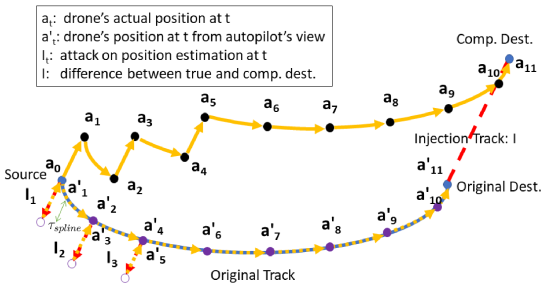


Fig. 10: Attacking the spline track-based autopilot algorithm when $I > \tau_{spline}$.

Case 1: We will first consider the case when the maximum position estimation injection $I > \tau_{spline}$. We will use Fig. 10 to explain the attack scheme. To make the attack process more legible in the figure, we will perform the attack in every other time step if needed (i.e., in the time step of $0, 2, 4, \cdots$). Similar to attacking the linear track-based autopilot algorithm, the attackers will place three position estimation injections $\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3}$ that are parallel to the injection track $\mathbf{I}$ at time step $0, 2, 4$, Such that $\mathbf{I} = \mathbf{I_1}+\mathbf{I_2}+\mathbf{I_3}$. Then the autopilot controller will "correct" the drift caused by these injections, pushing the target drone towards the compromised destination. In this case, the target drone will precisely reach the compromised destination under the attack.

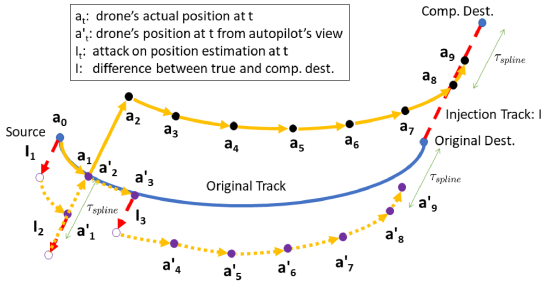


Fig. 11: Attacking the spline track-based autopilot algorithm when $I \leq \tau_{spline}$.

Case 2: When $I \leq \tau_{spline}$, we will apply the same attack scheme as the above. However, as shown in Fig. 11, the target drone may not reach the compromised destination with no error; instead, it reaches the circular region with the desired destination as the center and τ_{spline} as the diameter. For example, in Fig. 11, we can find that the controller will correct the track error after $\mathbf{I_2}$ is injected, since at this time the track error $\mathbf{I_1} + \mathbf{I_2}$ is greater than the threshold τ_{spline}. However the last injection $\mathbf{I_3}$ will not be corrected, since the track error is $\mathbf{I_3} \leq I \leq \tau_{spline}$. In the end, the distance between the actual position and the compromised destination will be $\mathbf{I_3} \leq \tau_{spline}$; in addition, the distance between the position from the autopilot's view and the original destination will also be $\mathbf{I_3} \leq \tau_{spline}$. However, if we apply the injection at time step 0, and every time when the injection is corrected and the target drone has flown back to the source, we continue to apply more injections, as long as the sum of these injections is greater than the threshold τ_{spline}, the drone can still reach the compromised destination precisely.

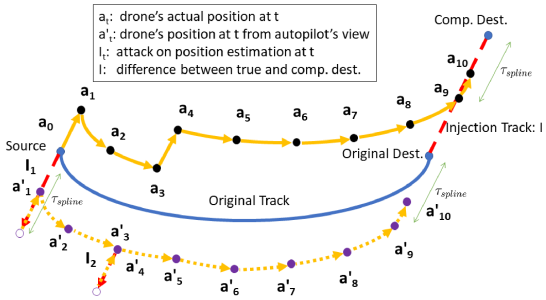


Fig. 12: If $d_c < \tau_{spline}$, it is possible that the drone arrives with a bounded error.

Discussion: what if $d_c < \tau_{spline}$? We can still perform the above attack. However, similar to the above $I \leq \tau_{spline}$ case, it is possible that the drone arrives at the compromised

destination with a bounded error. For example, as Fig. 12 shows, at time step 0/3, after the injection $\mathbf{I}_1/\mathbf{I}_2$ is performed, since the drift at this time is greater than the threshold τ_{spline}, a corrected displacement is applied. Because the corrected displacement $d_c < \tau_{spline}$, the drift is not fully corrected, and the target drone does not reach the original track after correction from autopilot's view. In the following time step, the remaining drift is not corrected either, because the drift is not greater than the threshold any more. Eventually, the target drone arrives with a bounded error. Due to the space limit, we move the detailed attack algorithm to a technical report [15].

Feasible position scope under this attack. In the following, we will analyze the feasible position scope under the proposed attack. For simplicity, we assume we apply the injection at time step 0, and every time when the injection is corrected and the target drone has flown back to the source track, we continue to apply more injections. That is, all the injections are performed at the beginning, and the drone's spline-form path will not start until all the "drifts" caused by the injections are corrected. Therefore, under both Case 1 and 2, the target drone will reach the compromised destination with no error; and both attack results will be the same if under the same attack. Again, we will use a figure to illustrate it.

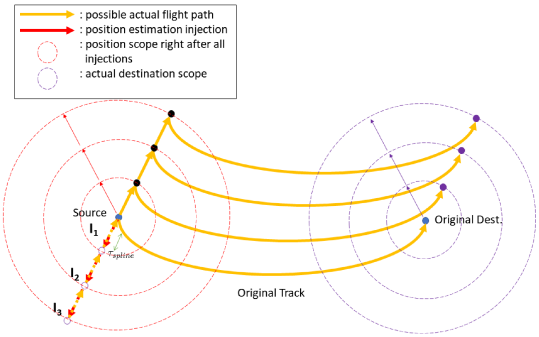


Fig. 13: feasible position scope of the target drone when the adversaries attack the spline track-based autopilot algorithm.

As Fig. 13 shows, we know that the injection in each time step can be in any direction, then the corresponding compensatory displacement for correction can also be in arbitrary direction, since the drone always flies in the opposite direction of the injection for correction. Then after t injections at the beginning, the feasible position scope of the drone will be a circular region with the source as the center and $t \cdot I$ as the diameter. Then the drone will fly along a path that has the same shape and size of the original track, and is parallel to the original track. Therefore, the actual ending position of the drone is also within the circular area with the original destination as the center and $t \cdot I$ as the diameter. As a result, the position scope under this attack is a position set as follows:

{S | S *is a spline that has the same shape and size of the original track, has the origin in the circular region with the source as the center and* $t \cdot I$ *as the diameter, and is parallel to the original track.*}.

D. Factors May Affect the Attacks

The above attacks assume we can perform precise position injection and no obstacles in the no-fly zone. These factors may vary in different situations. We discuss the impact of these factors on the proposed attacks in the following.

1) Position Injection with Noise: In practical navigation systems, the position estimation injections usually contain noises. If the position estimation injection is not precise, the accuracy of the proposed attacks may be impacted. However, we can control the magnitude of attack errors within an acceptable range. In particular, when performing the attack, if we find there is a position injection error in the last time step, we should correct this error in this injection. That is, in this time step, the position injection should be equal to sum of the original injection and the inverse of the last injection error (in vector arithmetic). By repeating the process, we can ensure that the difference between the final position of the drone and the desired compromised destination is equal to the position injection noise in the last time step.

2) Obstacles in the No-fly Zone: In real life, a drone may meet obstacles during a flight. If there are obstables on a flight path, the path needs to be changed. The challenge is how we should adjust the attack strategies such that the target drone can reach the compromised destination in the shortest possible path. We use the basic autopilot algorithm to illustrate our idea to solve this problem in two cases.

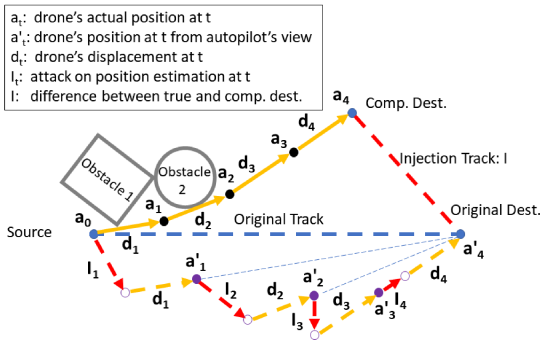


Fig. 14: The obstacles block the actual flight path.

Obstacles blocking the actual flight path. We design an optimal attack algorithm in the presence of obstacles. As Fig. 14, assume two obstacles block the proposed actual path for the target drone. To avoid colliding with the obstacles and maximize the bearing change, a natural selection is to make the first flight path segment tangent to Obstacle 1. Similarly, the second flight path segment should be tangent to Obstacle 2. After that, we can continue the attack as the original one as in Sec. III-A. If in one time step the tangent line is not a feasible flight path segment under the proposed attack, then the attack cannot be performed.

Obstacles blocking the flight path from autopilot's view. In this case, under the proposed attack, from autopilot's view, the target drone will hit obstacles, though it may not actually

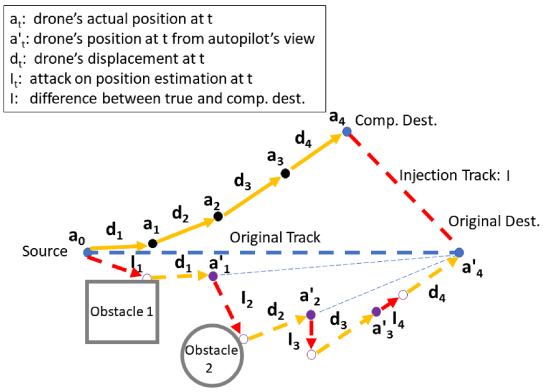


Fig. 15: The obstacles block the flight path from autopilot's view.

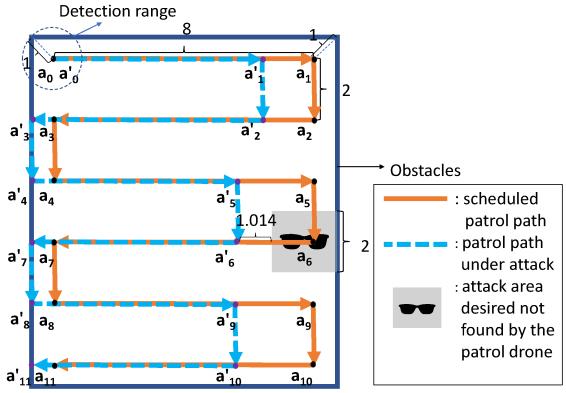


Fig. 16: Make the target drone move along the desired path.

happen. Using a similar strategy, we can let the flight path segment be tangent to the obstacles when the original ones overlap them.

In summary, when facing obstacles, the attackers should change the flight path for the target drone to avoid collision. One efficient way to achieve this is to make the flight path tangent to the obstacles. This also works for attacks on the linear track-based autopilot algorithm or the spline track-based autopilot algorithm. To change the flight path, for the linear track-based autopilot algorithm, we can adjust the direction by shortening the size of injection or changing its direction; and for the spline track-based autopilot algorithm, we can choose to distribute the position estimation injections in different time steps (e.g., every other time step), rather than putting all injections continuously at the beginning.

IV. Applying the Proposed Attacks on Drone Missions

A. Disrupting Patrol Missions

Now we consider a scenario of attacking a drone's patrol mission such that it misses a target within a specific area. As Fig. 16 shows, a drone is performing a common patrol task within a closed rectangular region. Assume the detection range of the drone is 1, then the preset patrol path zigzags with the distance $\frac{\sqrt{2}}{2}$ from each edge of the area. Assume we have an asset within a 2×2 square area. In this case, our goal is to change the actual flight path of the patrol drone so that it cannot detect our asset.

Because there are many ways to achieve our goal, we use a simple attack strategy to illustrate how to apply our proposed attacks to shift the patrol path to the left. Specifically, we apply a right position injection for each horizontal track, which leads the track to move left. In the first injection, we apply an right injection of 1.6, causing a right shift of the waypoint a_1 to a'_1. Then, the drone will make an right turn to fly back along the second horizontal track to the left side of the patrol area. Here we need to perform a left injection of $1.6 - \frac{\sqrt{2}}{2}$ to make sure the drone not flying out the patrol area. After the second downward flight, the drone will continue to go towards right, where we again apply a right injection of 1.6, making the waypoint a'_5 move left. Then the drone will fly downward again, reaching a'_6. Now the track between a'_5 and a'_6 are the closest part to the attack area, where the distance between them is $1.6 + \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2} - 2 \approx 1.014 > 1$ (the detection range of the patrol drone). Therefore the drone will not detect our asset in this area. In the following, the attackers will repeat the previous process, under which the drone will fly along a'_6 to a'_{11}.

B. Intercepting Another nearby Drone

In this scenario, we want to make two nearby drones to collide. Assume we know the flight paths and current positions of a victim drone V and a compromised drone C. There are different ways to make them collided. One simple way is: we do not attack V but make C intercept V by flying into V's path at a certain time. Assume V cannot avoid fast moving drones. Because V flies on its known flight path, we first determine the feasible collision positions on V's path. Because we know the position of V at time t, we can find those positions by checking if they are in the feasible position scopes of C and if we can make C arrive at the position at or before time t. We can easily exclude the position that C will arrive later than V. Furthermore, we select one collision position based on different requirement, e.g., with a shortest delay. Then, we perform the proposed attack on C such that (1) it arrives at a collision position at the same time of V, and they will collide; (2) it arrives a little earlier, and then we have to continue position injections to make it lingering for V's arrival.

We can also change the flight paths of both drones to make them collide. We then have to find the overlap positions of the feasible position scopes of both drones. There are also some realistic limitations that can make the attack much more challenging, including: limited battery life, injections with noise, obstacles in the attack zone, no knowledge of the exact flight paths of two drones. We will further explore these issues.

C. Simulation Evaluation

We propose a simple simulation to demonstrate the feasibility of intercepting another nearby drone.

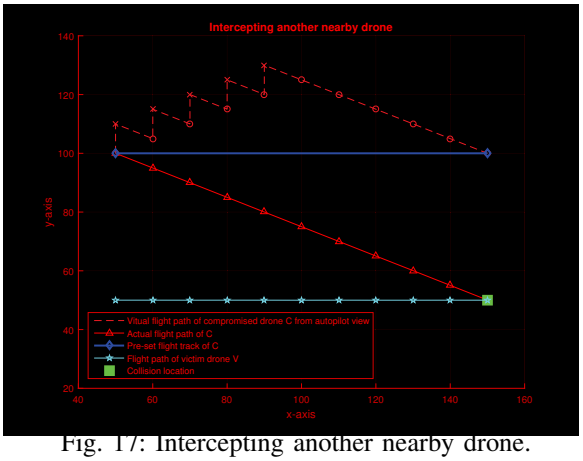


Fig. 17: Intercepting another nearby drone.

Simulation Settings. The attack simulator is written in MATLAB. In this simulation, we assume that the two drones: compromised drone C and victim drone V follow pre-set flight tracks under the linear track-based autopilot algorithm. In addition, we are able to manipulated the position estimation of the C precisely. The injection on the position estimation is set to be at most 5 m/timestep. Furthermore, the noise of each state estimation is assumed to follow $\mathcal{N}(0, 0.01^2)$. In the attack, we assume that we have the knowledge of the pre-set flight track of C and V. In particular, the source and the original destination for C is set to be (50 m, 100 m) and (150 m, 100 m), while the source and the original destination for V is (50 m, 50 m) and (150 m, 50 m) in a 2D plane. At time 0, they are both at their original sources. For the linear track-based autopilot algorithms on both drones, we assume that the maximum speed along the track and towards the track for track error correction is 10 m/timestep and 2.5 m/timestep, respectively. Then our attack goal is: by attack the autopilot controller of C, we would like to make C and V collide,

Simulation Results. Fig. 17 shows the simulation results of intercepting another nearby drone under the linear track-based autopilot algorithm. The red line with circles shows the flight path of C from the system's view while the black line with triangles represents its actual path. In addition, the cyan line with stars is the flight path of V. Since the track of V is under the one of C, we will attack C such that C fly along the bottom edge of its position scope, where C will intercept V at the earliest time. At each time step, the attackers put a maximum positive injection on the y-axis. Then the autopilot controller will direct the drone to fly towards the "destination", which makes it move towards bottom right. Under this attack, at time step 10, C and V will collide at (150 m, 50 m).

V. Conclusions

In this paper, we have focused on compromising the flight paths of autopiloted drones and presented several attacks to three common autopiloted navigation algorithms. Assume the first-phase sensor attacks and the second-phase state estimation attacks can help us build precise position injections, we have exploited the weaknesses of existing autopilot navigation algorithms to mislead a drone from its preset path to a manipulated path. We have presented the formal analysis of the scope of such manipulated paths. We have further discussed how to apply the proposed attacks to disrupt preset drone missions, such as missing a target in searching an area or misleading a drone to intercept another drone, etc. Many potential attacks can be built on top of the proposed attacks. We are currently investigating different models to apply such attacks on common drone missions and also building prototype systems on ArduPilot for real world tests. We will further investigate countermeasures to address the potential damages.

Acknowledgement: We like to thank the support of NSF 1662487 for this research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful gps spoofing attacks," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 75–86.

[2] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, no. 4, pp. 617–636, 2014.

[3] Y. Son, H. Shin, D. Kim, Y.-S. Park, J. Noh, K. Choi, J. Choi, Y. Kim *et al.*, "Rocking drones with intentional sound noise on gyroscopic sensors."

[4] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, "Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks."

[5] W. Chen, Z. Duan, and Y. Dong, "False data injection on ekf-based navigation control," in *ICUAS 2017*, pp. 1608–1617.

[6] 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)*, May 2018, pp. 1–9.

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

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

[9] 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.

[10] M. Benyamin and G. Goldman, "Acoustic Detection and Tracking of a Class I UAS with a Small Tetrahedral Microphone Array," ARL, Tech. Rep. ARL-TR-7086, 2014.

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

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

[13] D. E. Bolanakis, "Mems barometers in a wireless sensor network for position location applications," in *Applications of Commercial Sensors (VCACS), 2016 IEEE Virtual Conference on*. IEEE, 2016, pp. 1–8.

[14] W. Chen, Y. Dong, and Z. Duan, "Manipulating drone position control," in *IEEE CNS 2019*, June 2019.

[15] ——, "Compromising the flight paths of autopiloted drones," https://drive.google.com/open?id=1pH69I2FlxGkQezwZEdCNYczrtZn_G8hE, 2019.