PRACTICAL GPS SPOOFING ATTACKS ON CONSUMER DRONES

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

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

DEC 2020

By

Jianqiu Cao

Thesis Committee:

Yingfei Dong, Chairperson

Galen Sasaki

Yao Zheng

Keywords: Drone, GPS, SDR (Software-defined Radio), ArduPilot

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

THESIS COMMITTEE

____
Chairperson

____

____

Acknowledgement

I would like to thank my advisor, Dr. Yingfei Dong, for his guidance, patience, and help throughout my research and study at Department of Electrical Engineering, University of Hawaiʻi. I also would like to thank the members of my thesis committee: Dr. Galen Sasaki and Dr. Yao Zheng. Thanks for your advice and support.

In addition, I would like to thank the students who have collaborated with me in my thesis research project. They are: Wenxin Chen, Frendy Lio Can, Tomas J. Tigley and Isaac Lee.

Finally, my deep and sincere gratitude to my family for their continuous and unparalleled love, help and support. I am grateful to my wife and my parents for always being the strong backing.

Abstract

While the security of drones and unmanned automatic systems has become increasingly important, software vulnerabilities have been broadly examined but hardware vulnerabilities on these systems have not been well investigated. In this project, we utilize the recently popular software-defined radio cards to investigate vulnerability on these unmanned systems from both software and hardware aspects, epically focused on the security of civilian GPS receivers on consumer drones.

We have developed a smart GPS spoofing attack framework on consumer drones, based on our understanding of the drone control software and the vulnerability of civilian GPS system. We will first introduce the background and related work on drone control systems and the GPS system; we will further present the design of our attack framework and our experimental results. Although simple GPS spoofing has been conducted in various settings, as far as we know, most of them are brute-force attacks without precise control. In this project, we have developed a practical framework to achieve better control of drone movement, based on the deep understanding of consumer drone specific issues. The proposed attack has been very successful in our lab environments; however, there are many practical challenges in field tests due to the limitation of the devices, field environments, and other reasons out of our control, e.g., wind speed. We have conducted many field tests to understand and address the practical limitations and show the capabilities of the proposed system. We will then present our research results and conclude this thesis with discussion and future work.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

The security of drone and unmanned automatic systems has become increasingly important as we are having more and more such systems surrounding us in our lives. Traditionally, many research projects focused on the vulnerability in the software side; very little research has been conducted in hardware related areas, due to many limitations such as the difficulty to construct hardware attacks. In this project, we utilize the recently popular Software-defined Radio (SDR) cards to investigate vulnerability on these unmanned systems from both software and hardware aspects, epically focused on the security of sensors on these systems.

We have examined sensors on consumer drones including magnetometers, accelerators, and gyroscopes [14][15][16][17], and found that: while brute-force attacks are fairly easy, precise control of these sensors remotely requires high-end devices that are hard to access for us. Therefore, we focused our investigation on civilian Global Position System (GPS) receivers on consumer drones.

GPS is the most widely used positioning and time synchronization system for many unmanned auto-controlled systems. Particularly, a GPS device receives signals from the GPS satellite constellation, and derives fairly precise 3-dimension position, velocity, and high-precision time.

GPS is essential to the navigation of most consumer drones, which use GPS signals to determine their current positions, navigate themselves to their destinations through a series of waypoints, and achieve the popular return-home feature. However, the GPS input of a consumer drone is an obvious vulnerability, as the civilian GPS positioning algorithms are open to the public, and the civilian GPS signals are unencrypted and authenticated. Furthermore, the satellite signals are fairly weak such that it can be easily overpowered by stronger local attack signals. The devices to precisely generate spoofed GPS signals have become fairly easy to access recently. For example, consumer-grade SDR devices (<$1,000) can be used to transmit forged GPS signals with fake position or trajectory signals to civilian GPS receivers.

This thesis research focused on the vulnerability of GPS receivers on consumer drones, in particular, exploiting the weaknesses of GPS navigation and performing smart GPS spoofing attacks on these drones, in order to make malicious drones deviate from their targets. In the meantime, we will also examine the countermeasures of such GPS attacks.

We have conducted preliminary studies of the feasibility and performance of GPS spoofing in our lab. We used software to generate spoofing GPS data streams based on the ephemeris data (obtained from a NASA public web site) and desired positions, and then transmit the data through our SDR platform to a GPS receiver or a drone for testing. Our results show that the drone can be easily fooled and accept the fake GPS positions in its readings. Currently, we are investigating the navigation algorithms on *ArduPilot* (one of the most popular autopilot systems) for developing smart GPS attacks when a drone is in operation, such as hovering or in a given mission. Our goal is to make it deviate from the original destination while not being detected by the control schemes on the drone.

We have conducted both lab and field tests to show the effectiveness of the proposed framework. Our experimental results have shown both advantages and limitations of the proposed approach. The main contributions of this project are:

- Developing a GPS spoofing framework to attack civilian GPS receivers on drones and other systems.
- Conducting many lab and field tests to examine the effectiveness of the proposed system.
- Compiling many practical experiences through field tests, which are critical to future research in this direction.

Chapter 2. Background and Related work

2.1. GPS principles

The Global Positioning System (GPS) provides users with positioning, navigation, and timing (PNT) services, and it consists of three segments: the space segment, the control segment, and the user segment. The U.S. Air Force develops, maintains, and operates the space segment (i.e., 24 satellites) and the control segments (a set of ground monitoring and control stations). The ground stations monitor the status of satellites, adjust their positions and times, and share their data with users for better use of the system. The satellites broadcast one-way signals of the current GPS satellite position and time for receivers to perform position triangulation and time synchronization. The user segments are GPS receivers that help users find their global positions and synchronize times [1].

In the following, we will first introduce the GPS triangulating method, and then discuss the basic concepts related GPS signals, GPS message frames, GPS device starting processing, and one of the most commonly used GPS message formats.

2.1.1. Positioning Principles: Triangulating

A full constellation of 24 GPS satellites in orbit is providing position, velocity, and time services to users. When a receiver can receive signals from at least 4 satellites, it can find out its 3D position on the global and the accurate time based on satellite position data and GPS time using trigonometry principles [2].

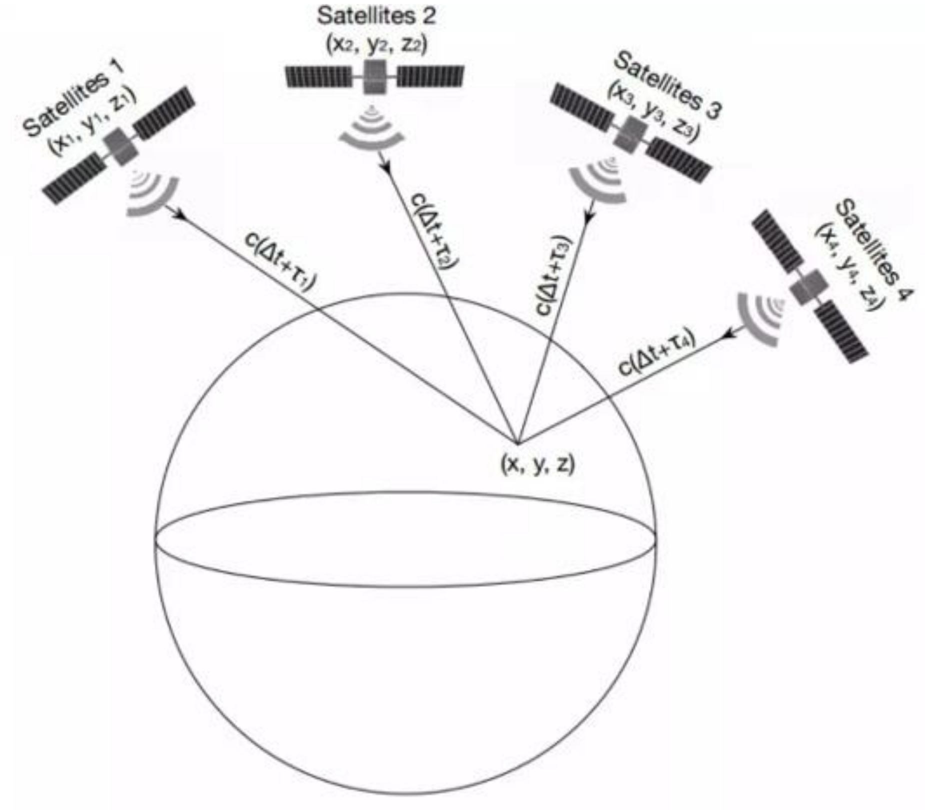


Figure 1. GPS positioning principle.

Figure 1 shows the basic setup using GPS signals to determine a receiver's global position and the GPS time. Suppose we have three satellites (Satellite 1, 2, and 3) in the constellation broadcasting their coordinates (x1, y1, z1), (x2, y2, z2), (x3, y3, z3) with timestamps, and we have also measured the durations (τ1, τ2, τ3) between when a broadcasting signal is sent from a satellite and when it is received by the receiver equipment, we can then "triangulate" the receiver's coordinates (x, y, z) by the following equations:

$$\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} = c \cdot \tau_i,\ i \in \{1,\ 2,\ 3\},$$

where c is the velocity of electromagnet wave, and $c \cdot \tau_i$ are the range from the satellites to the receiver.

However, the problem is more complex in practice. The satellites have precise atomic clocks on board, but the receiver does not, which means the local receiver's clock and the satellite clock are usually not synchronized. So, the measured durations τ1, τ2, and τ3 are not precisely the true transmission durations; rather, the true transmission durations should be $\Delta t_i + \tau_i, i \in \{1, 2, 3\}$, where Δt_i is the time difference between the local clock and the satellite clock. The revised equations are as below:

$$\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} = c(\Delta t_i + \tau_i),\ i \in \{1, 2, 3\}.$$

We call $c(\Delta t_i + \tau_i)$ "*pseudorange*".

As the satellite clocks are well synchronized, we can assume that $\Delta t_1 = \Delta t_2 = \Delta t_3 = \Delta t$. If we receive the data from a 4th satellite, we can then add the 4th equation to the equation set, which becomes:

$$\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} = c(\Delta t + \tau_i),\ i \in \{1, 2, 3, 4\}.$$

So, we can solve the four unknowns (x, y, z and Δt) based on the four equations. Note that not only the receiver's position (x, y, z) can be solved, but also the local clock can be synchronized to the GPS time in this procedure.

2.1.2. GPS Signals

GPS signals are broadcasted on two L-band carrier frequencies, which are shown in Figure *2*.

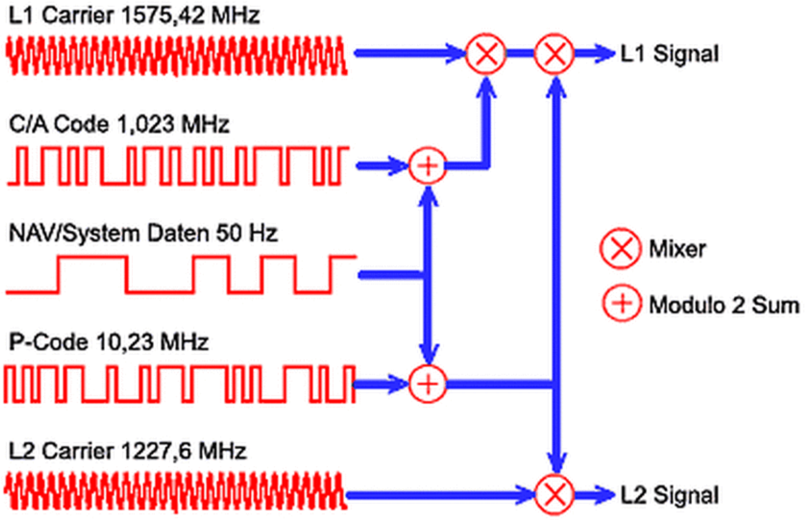


Figure 2. GPS signal coding schemes [4].

The L1 band (1575.42 MHz) carries *Civil Coarse/Acquisition (C/A) Code* (not encrypted), Navigation/System message and Military Precision (P(Y)) code (encrypted), while the L2 band (1227.6 MHz) carries Military code (encrypted). In this research, we exploit only the civilian signals on L1 band.

Based on the C/A code, a receiver calculates the time difference and the pseudo range between the satellite and itself [3]. A C/A code is a type of *Pseudorandom Noise (PRN) code*. Each satellite uses a unique PRN code, which does not correlate with any other satellite's PRN codes. The PRN codes are highly orthogonal to one another. The receiver generates a replica C/A code and shifts it in time until it lines up with the incoming signal from a satellite. The C/A code is used to (1) Identify the ranging code that a satellite uses, and (2) Make a *pseudorange measurement* (ρ), which is defined in the above.

2.1.3. Navigation Message

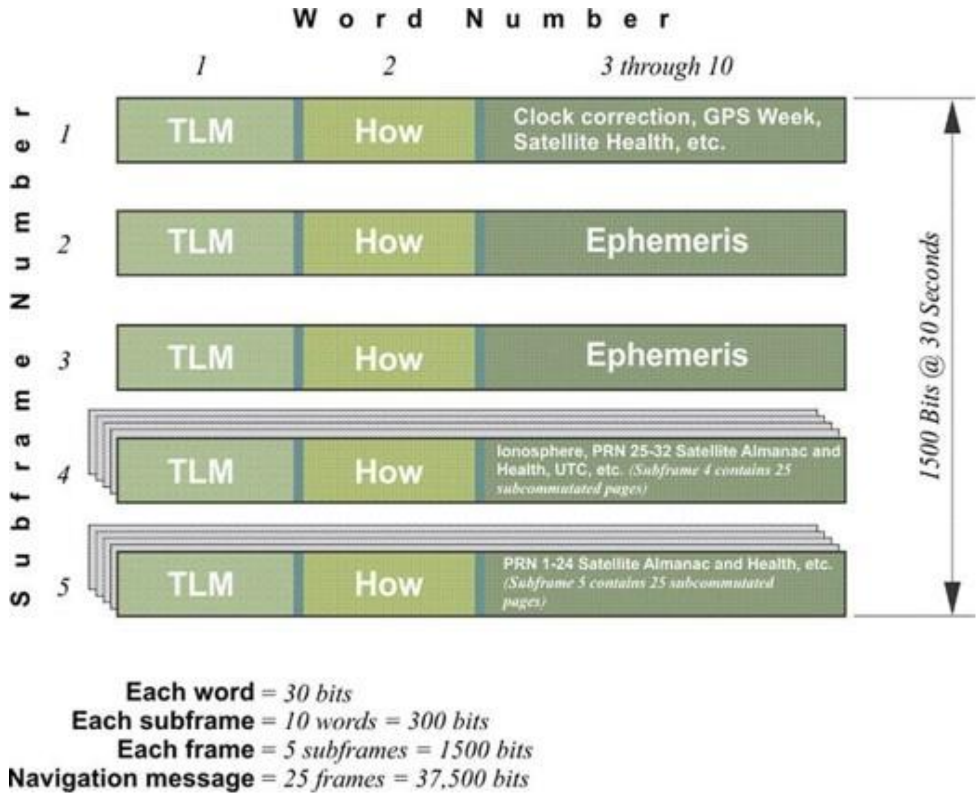

Word Number
1
2
3 through 10
Subframe Number
1
TLM
How
Clock correction, GPS Week, Satellite Health, etc.
2
TLM
How
Ephemeris
3
TLM
How
Ephemeris
4
TLM
How
Ionosphere, PRN 25-32 Satellite Almanac and Health, UTC, etc. (Subframe 4 contains 25 subcommutated pages)
5
TLM
How
PRN 1-24 Satellite Almanac and Health, etc. (Subframe 5 contains 25 subcommutated pages)
1500 Bits @ 30 Seconds
Each word = 30 bits
Each subframe = 10 words = 300 bits
Each frame = 5 subframes = 1500 bits
Navigation message = 25 frames = 37,500 bits

Figure 3. Navigation message [5].

The navigation message conveys information of three types [6][7] (See Figure 3):

- The GPS date and time and the satellite's status. (subframe 1).
- The ephemeris (subframe 2-3): precise orbital information for the transmitting satellite.
    - Obtain satellite coordinates.
    - EACH satellite broadcasts ONLY its own ephemeris data [7], which is valid for only 30 minutes to four hours [33].
- The almanac (subframe 4-5): status and low-resolution orbital information for every satellite.
    - Determine correct nearby satellites upon power-up.
    - Predictions on ionospheric conditions that could change the time of flight for a signal traveling from space to Earth.
    - Each satellite broadcasts almanac data for ALL satellites [7], which is valid with little dilution of precision for up to two weeks [33].

The navigation message conveys GPS date and time, ephemeris, and almanac data, which can be used for satellite searching and determining satellite positions [6][7].

We can obtain the broadcast GPS ephemeris data from the NASA web site to construct fake GPS signals. NASA provides broadcast GPS navigation data in addition to observation GPS data at receivers. The NASA CDDIS creates daily broadcast ephemeris files transmitted by the GPS ground control stations. These files contain the unique GPS satellite ephemeris messages for each day. A similar file is created at the start of the UTC day and updated on an hourly basis from the hourly broadcast navigation files. We can download a single file each day or hour, which contains all broadcast ephemeris messages required for post-processing [29].

The daily GPS broadcast ephemeris file is a merge of the individual site navigation files into one, non-redundant file that can be utilized by users instead of the many individual navigation files [29].

2.1.4. Time to First Fix

Time to first fix (TTFF) is a measure of the time required for a GPS navigation device to acquire satellite signals and navigation data, and calculate a position solution (called a fix) [34][35]. TTFF varies in the following three different scenarios:

Cold start: if a receiver has no previous almanac or ephemeris data, or the almanac data has become invalid due to position changes or loss of accurate time, it will have to perform a cold start, a.k.a., a factory start. In this process, the receiver must search for all the satellites. When it does finally manage to acquire the signal from one, it can use it to obtain an almanac. The period needed to receive the full almanac information is about 12.5 minutes.

Warm start: If a receiver has some leftover almanac, time and position data from its previous observations, it can begin its search with a warm start, a.k.a., *a normal start*. The receiver can estimate the ranges to satellites and restrict its search for satellites to those likely overhead to collect their ephemeris data. The *time to first fix (TTFF)* [35] can be as short as 30 seconds with a warm start.

Hot start: A receiver that has a current almanac, a current ephemeris, time and position can have a hot start. A hot start can take from 0.5 to 20 seconds.

Many receivers can use as many as twelve channels to track satellite signals simultaneously.

2.1.5. NMEA Data Format

NMEA is an acronym for the *National Marine Electronics Association* [36], which provides but is not limited in GPS times and positions. NMEA is a standard data format supported by all GPS manufacturers, much like *ASCII* is the standard for digital computer characters in the computer world.

A NMEA data file consists of multiple NMEA messages. There are many different types of NMEA messages with different capabilities. $*GPGGA* is a frequently used type of NMEA message, which was output from an RTK GPS receiver we used for testing. The following line is an example of NMEA $*GPGGA* message:

```
$GPGGA,181908.00,3404.7041778,N,07044.3966270,W,4,13,1.00,495.144,M,
29.200,M,0.10,0000,*40
```

All NMEA messages start with the $ character, and each data field is separated by a comma. Table 1 describes the meaning of each field in the NMEA GGA message above.

Table 1. Description of NMEA GGA message [37].

Field	Description	Value	Interpretation
0	Message ID	$GPGGA	GPS Time, position, and fix related data
1	UTC of position fix	181908.00	18:19:08
2	Latitude in the DDMM.MMMMM format	3404.7041778	34°04.7041778'
3	Direction of latitude: N: North, S: South	N	North
4	Longitude in the DDDMM.MMMMM format	07044.3966270	070°44.3966270'
5	Direction of longitude: E: East, W: West	W	West
6	GPS Quality indicator: 0: Fix not valid 1: GPS fix 2: Differential GPS fix, OmniSTAR VBS 4: Real-Time Kinematic, fixed integers 5: Real-Time Kinematic, float integers, OmniSTAR XP/HP or Location RTK	4	Real-Time Kinematic, fixed integers
7	Number of satellites in use, range from 00 through to 24+	13	
8	HDOP (horizontal dilution of precision)	1.0	
9	Orthometric height (MSL reference), e.g. altitude of the antenna	495.144	
10	Unit of measure for orthometric height (eg. Meters or Feet)	M	Meter
11	Geoid separation	29.200	
12	Unit of geoid separation (eg. Meters or Feet)	M	Meter
13	Age of differential GPS data record, Type 1 or Type 9 (if any)	1.0	
14	Reference station ID, range 0000-4095 (if any)	0000	
15	The checksum data	*40	

There are also alternative and companion NMEA messages that provide similar or additional information, such as *$GPGLL, $GPRMC, $GPGSA, $GPGSV, $GPVTG, $GPGST* [38][39][40].

$GPGSA: Overall Satellite data which includes GPS DOP and active satellites (i.e., the numbers of the satellites being used in the current solution).

$GPGSV: Detailed Satellite data which includes the number of satellites in view, the PRN numbers, elevations, azimuths, and SNR values. The receiver might be able to find based on its viewing mask and almanac data.

$GPRMC: Position, velocity, and time.

In the thesis research, we utilize NMEA data to record the GPS information from the GPS receiver, and to generate spoofed GPS positions for our attack tests.

2.2. Software and Hardware Platform

In this section, we introduce the software and hardware we used for our GPS attack framework.

2.2.1. GPS-SDR-SIM

The broad availability of SDR cards has made signal-level attacks much easier, cheaper, and practical. We can use the open-source GPS simulator GPS-SDR-SIM [18] to generate GPS baseband signal data streams, which can be converted to RF signals using software-defined radio (SDR) platforms. We can specify the GPS satellite constellation through a GPS broadcast ephemeris file and specified a fixed location or a trajectory using forged coordinates.

We can use the following command line to generate a GPS data stream with a *fixed* location:

```
gps-sdr-sim -e brdc3240.20n -l 21.28,-157.81,100 -o gpssim.bin -t
2020/11/19,07:58:56
```

We can use the following command line to generate a GPS data stream with a *dynamic trajectory* where “holmes.txt” file holds the dynamic trajectory:

```
gps-sdr-sim -e brdc3240.20n -g holmes.txt -o gpssim.bin -t
2020/11/19,07:58:56
```

2.2.2. Software-defined radio (SDR) devices

In an SDR system, we use software to emulate various communication components (such as mixers, filters, amplifiers, modulators, demodulators, and detectors), which are used to be implemented in hardware [12].

There are many hardware solutions for SDR, such as bladeRF, HackRF, and USRP systems. In the experiments of this project, we use bladeRF [10] 2.0 shown in Figure 4 as a GPS signal transmitter.

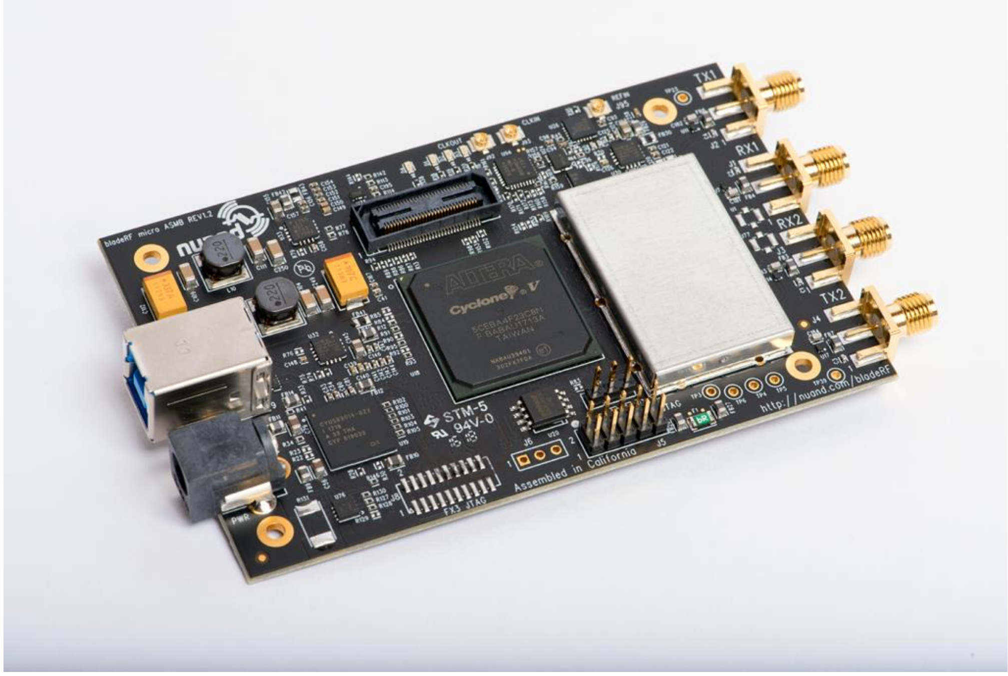


Figure 4. bladeRF 2.0 micro.

2.2.3. Drone and Drone Control Systems

ArduPilot [20] is an open source, unmanned vehicle Autopilot Software Suite, and *ArduPilot Copter* is specially designed for controlling copters. An autopilot controller adopts *PID control* to uses a control feedback mechanism to dynamically adjust control inputs according to the differences between outputs and setpoints [13][21].

The SkyViper Journey [22] is light-weighted drones which use *ArduPilot* as flight control software and features a GPS module for position control. The drone is also equipped with a *MAVLink* interface for the communication with ground control stations. Figure 5 shows the drone and the controller. We use this consumer-grade drone as an attack victim for our testing.

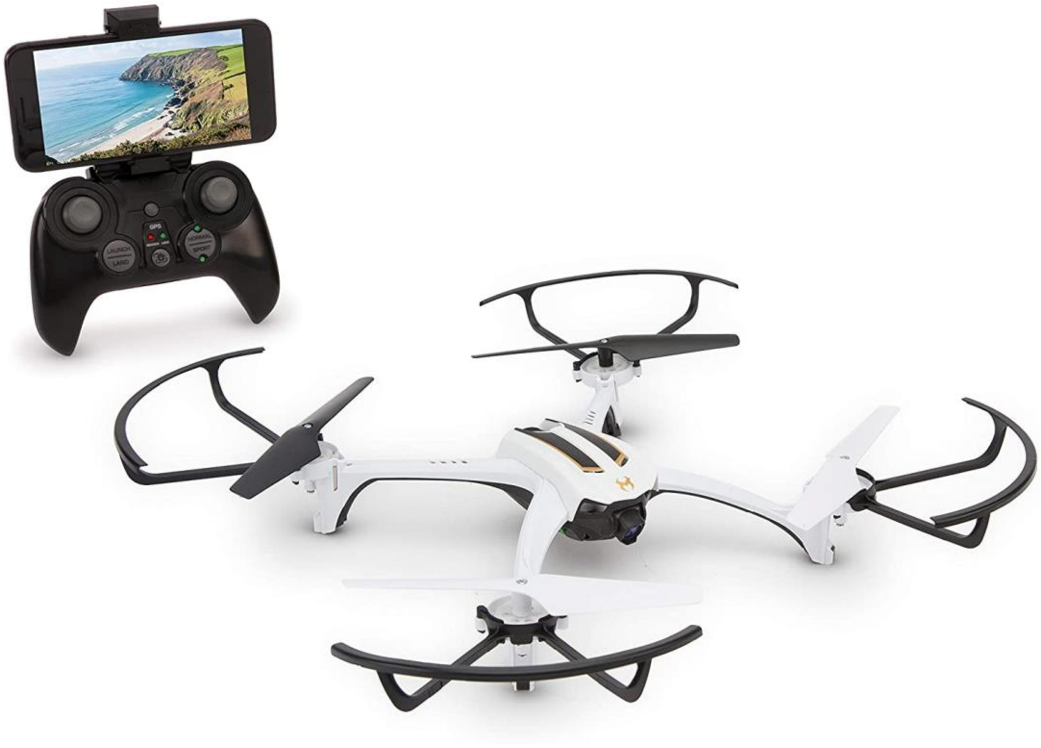


Figure 5. Sky Viper Journey GPS drone.

MAVLink is a lightweight messaging protocol for ground control stations to communicate with drones. It facilitates us to read drone status and send control massages [23]. *MAVLink* provides a default common message set, which can be used for either reading a drone’s internal status or controlling a drone’s flying mission. For example, message GPS_RAW_INT provides the status from a GPS receiver, such as the received global position and ground speed. Message GLOBAL_POSITION_INT includes the information from the fusion of GPS and inertial sensor. We used those two messages in our experiments to read a drone’s position and velocity information. The details are shown in Table 2 and Table 3.

Table 2. GPS_RAW_INT (#24) message [24].

Field Name	Type	Units	Description
time_usec	uint64_t	us	Timestamp (UNIX Epoch time or time since system boot). The receiving end can infer timestamp format (since 1.1.1970 or since system boot) by checking for the magnitude of the number.
fix_type	uint8_t		GPS fix type.
lat	int32_t	degE7	Latitude (WGS84, EGM96 ellipsoid)
lon	int32_t	degE7	Longitude (WGS84, EGM96 ellipsoid)
alt	int32_t	mm	Altitude (MSL). Positive for up. Note that virtually all GPS modules provide the MSL altitude in addition to the WGS84 altitude.
eph	uint16_t		GPS HDOP horizontal dilution of position (unitless). If unknown, set to: UINT16_MAX
epv	uint16_t		GPS VDOP vertical dilution of position (unitless). If unknown, set to: UINT16_MAX
vel	uint16_t	cm/s	GPS ground speed. If unknown, set to: UINT16_MAX
cog	uint16_t	cdeg	Course over ground (NOT heading, but direction of movement) in degrees * 100, 0.0..359.99 degrees. If unknown, set to: UINT16_MAX
satellites_visible	uint8_t		Number of satellites visible. If unknown, set to 255
alt_ellipsoid	int32_t	mm	Altitude (above WGS84, EGM96 ellipsoid). Positive for up.
h_acc	uint32_t	mm	Position uncertainty.
v_acc	uint32_t	mm	Altitude uncertainty.
vel_acc	uint32_t	mm	Speed uncertainty.
hdg_acc	uint32_t	degE5	Heading / track uncertainty
yaw	uint16_t	cdeg	Yaw in earth frame from north. Use 0 if this GPS does not provide yaw. Use 65535 if this GPS is configured to provide yaw and is currently unable to provide it. Use 36000 for north.

Table 3. GLOBAL_POSITION_INT (#33) message [24].

Field Name	Type	Units	Description
time_boot_ms	uint32_t	ms	Timestamp (time since system boot).
lat	int32_t	degE7	Latitude, expressed
lon	int32_t	degE7	Longitude, expressed
alt	int32_t	mm	Altitude (MSL). Note that virtually all GPS modules provide both WGS84 and MSL.
relative_alt	int32_t	mm	Altitude above ground
vx	int16_t	cm/s	Ground X Speed (Latitude, positive north)
vy	int16_t	cm/s	Ground Y Speed (Longitude, positive east)
vz	int16_t	cm/s	Ground Z Speed (Altitude, positive down)
hdg	uint16_t	cdeg	Vehicle heading (yaw angle), 0.0..359.99 degrees. If unknown, set to: UINT16_MAX

Pymavlink is a Python library for handling *MAVLink* protocol streams and log files. This allows for the creation of simple scripts to analyze telemetry logs from autopilots such as *ArduPilot* which use the *MAVLink* protocol [25][26].

Also, *MAVLink* provides various messages for concrete flight mission controls, e.g. MAV_CMD_NAV_WAYPOINT, MAV_CMD_NAV_TAKEOFF, MAV_CMD_NAV_LAND and MISSION_ITEM_INT. However, we used an upper-level API *DroneKit* for flight control as it has more simple and friendly interfaces. *DroneKit-Python* allows developers to create apps that run on an onboard companion computer and communicate with the *ArduPilot* flight controller using a low-latency link. The API communicates with vehicles over *MAVLink*. It provides programmatic access to a connected vehicle's telemetry, state and parameter information, and enables both mission management and direct control over vehicle movement and operations [27].

2.3. Literature Review

Several successful GPS spoofing attacks have been demonstrated in simulation or in the real world.

In [2], the authors used an open source project GPS-SDR-SIM [8] to generate GPS data stream with fake positions and transmit them through HackRF [9] and BladeRF [10] SDR platforms. They spoofed the signals to a cell phone and a smart watch, and shifted their GPS locations from Beijing, China to a static location in Nagoya, Japan.

In the paper [11], the researchers proposed an attack that manipulates road navigation system and implement a portable spoofer (based on HackRF) to divert a vehicle to a different destination in the physical world.

In the paper [12], the authors investigated the requirements for successful GPS spoofing attacks on individuals and groups of victims with civilian or military GPS receivers. The paper shows that the attacker is restricted to only few transmission locations when spoofing a group of receivers while preserving their constellations.

Focused on drone countermeasures, Wenxin Chen et al. proposed False Data Injection (FDI) attacks on the EKF position estimator and ArduPilot drone controller to compromise drone position, altitude, and flight paths [13][14][15][16][17]. In the paper [13], drone positions are manipulated by injecting carefully-built GPS positions to affect drone flight paths; countermeasures to these attacks have also been discussed. They also investigated the attacks on drone altitude control in [15]. The weakness of drone state estimation was explored in [16]. The effect of FDI attacks on drone state estimation and control was first proposed in [17].

Chapter 3. Problem Formulation

As we have identified the GPS input as a low-level vulnerability that has not been properly address in practice, we aim at designing a concrete platform to investigate such an issue. In this platform, we need to figure out how to generate fake GPS signals, and how to deliver the fake signals to a drone, such that we can investigate various attacks and potential countermeasures.

3.1. Hardware Platform

In the thesis research, we generated forged GPS positions and transmitted the signal to the drone, to make it deviate from the original destination while not being detected by the normal control schemes on the drone.

Before attacking the drone, however, we conducted a preliminary attack test on a GPS receiver to validate the feasibility of our GPS attack framework. The hardware setup for the test is shown in Figure 6. A GPS receiver first locked to real GPS satellites and showing its real position. Next we obtained *broadcast ephemeris (BRDC) data* from the NASA CDDIS website [29], generated bitstream of spoofing GPS positions using GPS-SDR-SIM [18], and transmitted the signal through bladeRF [10]. Then the spoofing signal took over the signal from the satellites at a GPS receiver such that the reading of the receiver shows the forged position. The experiment results are given in Section 4.1.

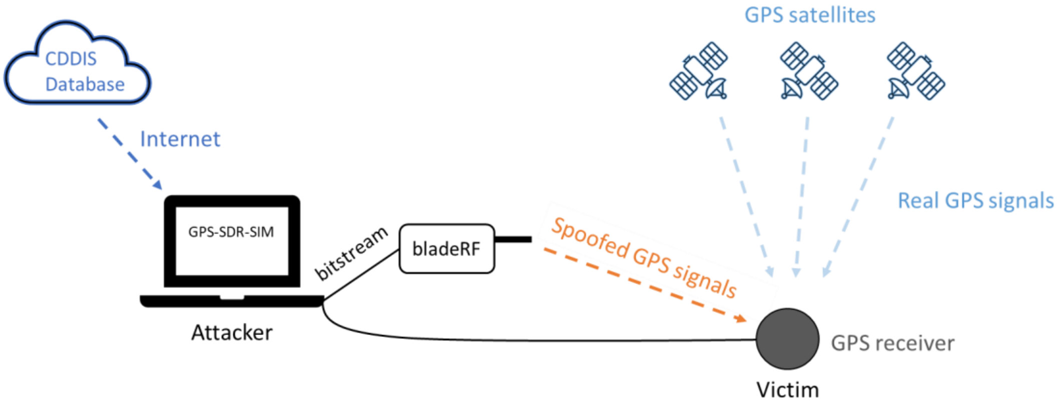


Figure 6. Setup for attacking a GPS receiver

After our attacking method was validated, we conducted GPS spoofing tests to a drone. The setup is shown in Figure 7. We used two computers in the tests: one as an attacker to generate spoofing GPS signal and transmit it to the drone, and another as an observer to read and record the drone’s status. The attacker also acted as a controller, which sent control commands to launch the drone or to let it fly a mission. The two computers were connected to the drone via Wi-Fi network using the *MAVLink* protocol [23] for data communication. The drone originally locked to real GPS satellites and after the spoofing signal took over the GPS fix, we could manipulate the drone’s flying path by sending elaborately generated GPS position trajectories.

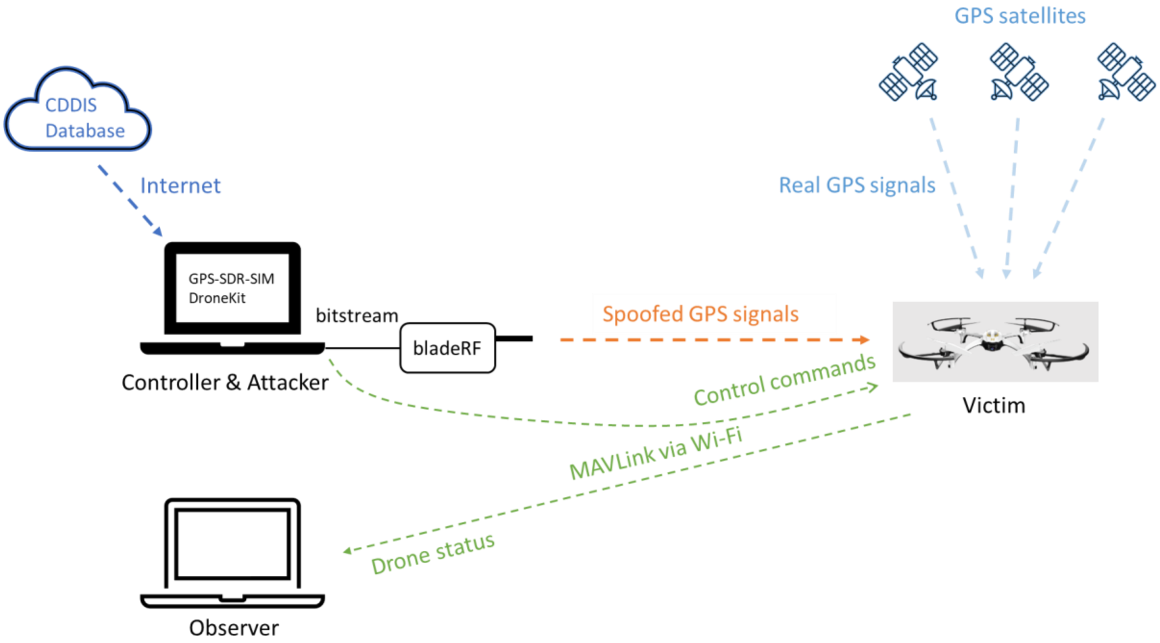


Figure 7. Setup for attacking a drone

3.2. Attack Model

ArduPilot utilizes the data fusion from different types of sensors, including standard IMUs (accelerometers, gyroscopes, and magnetometers in some cases), GPS, and barometers, to estimate the drone’s position and minimize estimation errors. Extended Kalman Filter (EKF), the most commonly used state estimation methods, is implemented to provide more accurate position estimations.

The control loop in a drone is illustrated in Figure 8, which is a real-time automatic control without manual inputs. Specifically, the system first performs state estimation based on sensor measurements. Then according to the current state estimation, the autopilot controller takes control actions to automatically adjust drone movements. Given the autocontrol actions, the actuators take corresponding adjustments.

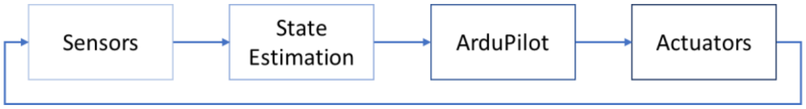


Figure 8. Drone control loop.

To achieve our attack goals, we transmit fake GPS positions to deceive the GPS receiver on the drone, and then the forged sensor readings compromise the position estimation. Furthermore, *ArduPilot* regulates the actuators based on the compromised positions to move the drone to the compromised direction or destination.

We will conduct the GPS spoofing attacks in the following two scenarios, which are shown in Figure 9 and Figure 10.

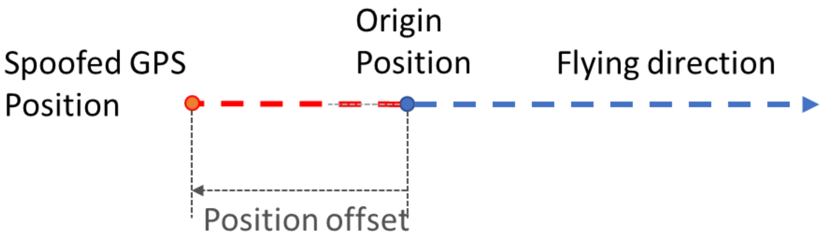


Figure 9. Hovering attack.

In Figure 9, when the drone is hovering at a fixed point, we shift GPS positions gradually a location a few meters away from the origin point. Based on the drone position control algorithms, the drone will find the drift of its position and will move to the opposite direction to maintain its location “unchanged”. If we keep feeding the spoofed GPS position, the drone will keep moving to the opposite direction. We noticed that *ArduPilot* has a has a GPS failsafe mechanism [45] to detect abnormal GPS position changes, so we limit the shift speed of the GPS position to avoid being detected.

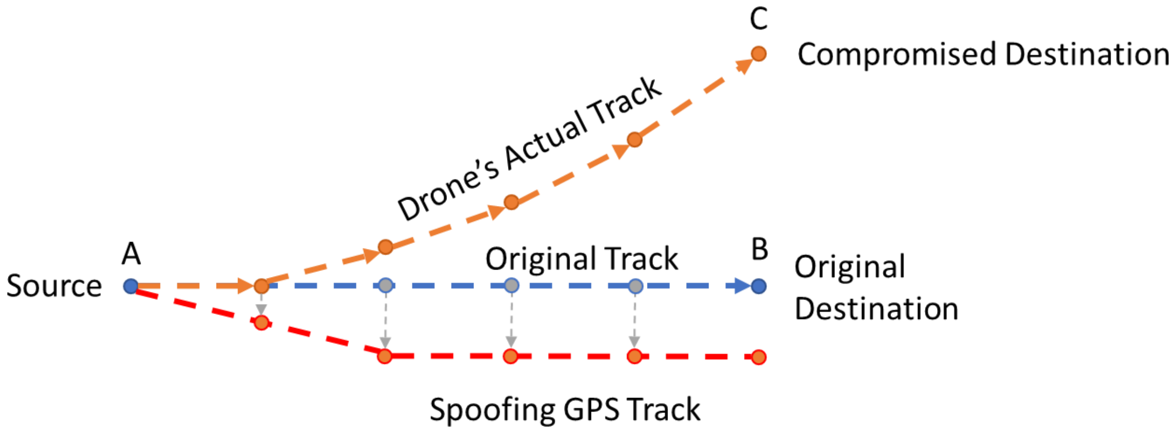


Figure 10. Mission flying attack.

Figure 10 is a simplified illustration of the attack on a drone flying a mission from source point A to destination point B at the same altitude. The original mission track is from source point A to destination point B. We send spoofing GPS positions during the mission with the East direction unchanged but shift the position to the South at a fixed rate. In every control cycle, the drone discovers its deviation from the original track and move to the North for correction. By this attacking approach, we are able to bend the drone's flying track and direct it to a compromised destination C [13].

3.3. Software Setups

In this section, we introduce the software setups for GPS signal generation and transmission, and a Python script for fetching drone status.

3.3.1. GPS Signal Generation and Transmission

Figure 11 illustrates the flow of generating and transmitting the spoofing GPS signal in our research project.

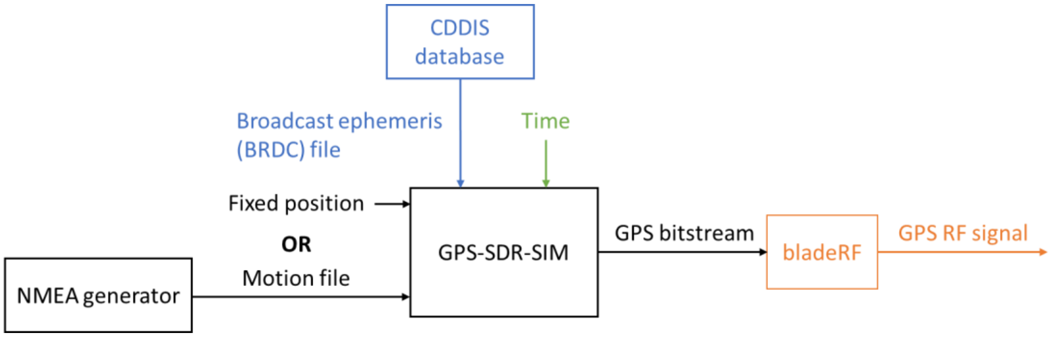


Figure 11. Flowchart of GPS signal generation and transmission.

The process is as follows:

- Download the most recent ephemeris data in a BRDC file (the daily broadcast ephemeris file) from CDDIS using the following command:

```
wget --no-check-certificate
"ftps://gdc.cddis.eosdis.nasa.gov/gnss/data/daily/$(date -u
+%Y)/brdc/brdc$(date -u +%j0.%g)n.gz"
```

, or download the file manually from the archive https://cddis.nasa.gov/archive/gnss/data/daily/

This file is a merge of the individual site navigation files into one, non-redundant file that can be utilized by users instead of the many individual navigation files. The daily file created at a location in Germany each day contains unique navigation messages from sites in Europe.

The combined broadcast ephemeris file is generated on an hourly basis from all hourly navigation files archived at the CDDIS. The hourly navigation file contains all broadcast messages with the TOE of the day that are available when the file is created at the top of the hour. The file is updated each hour with new navigation messages [29].

- Extract the brdc file from the compressed file:

```
uncompress brdc$(date -u +%j0.%g)n.gz
```

- Generate bitstream with a fixed position at the current time using GPS-SDR-SIM, e.g., the following command generates a GPS bitstream for a point at Honolulu (latitude = 21.28N, longitude = 157.81W, height = 100m).

```
./gps-sdr-sim -e brdc$(date -u +%j0.%g)n -l 21.296965,-
157.815687,100 -d 60 -b 16 -o gpssim.bin -t $(date -u +%Y/%m/%d,+%X)
```

Also, we can generate bitstream with a trajectory of moving positions specified in a NMEA file, e.g., holmes.txt, using the following command:

```
./gps-sdr-sim -e brdc$(date -u +%j)0.$(date -u +%g)n -g holmes.txt -
o gpssim.bin -t $(date -u +%Y/%m/%d,+%X)
```

- Transmit the bitstream through bladeRF.

```
bladeRF-cli -s bladerf2.0.script
```

The content of bladerf2.0.script is

```
set frequency tx 1575.42M
set samplerate 2.6M
set bandwidth 2.5M
set gain tx 56
set biastee tx on
tx config file=gpssim.bin format=bin
tx start
tx wait
```

3.3.2. Drone Status Data Fetching

We use the Python library *Pymavlink* to access a drone’s status, such as position and velocity information from the GPS raw input or the fused data from the EKF, through *MAVLink* protocol on a UDP port. The following Python code gets a set of data points: time, GPS fix type, EKF status and GPS position from the drone, and saves it into a CSV file. In addition, we can use *Matplotlib* [30] in Python to visualize data in various types of charts. The data graphs in Section 4.2 are all obtained by using *Pymavlink* and *Matplotlib* libraries.

```
import csv
from pymavlink import mavutil
import keyboard

the_connection = mavutil.mavlink_connection('0.0.0.0:14550')

csv_filename = 'drone_status.csv'
with open(csv_filename, 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile)
    field = ['time', 'fix_type', 'EKF_status','lat', 'lon', 'alt']
    csvwriter.writerow(field)
    print(field)

    while not keyboard.is_pressed('c'):
        the_connection.wait_heartbeat()
        time_usec = the_connection.messages['GPS_RAW_INT'].time_usec # time
since system boot in us
        fix_type = the_connection.messages['GPS_RAW_INT'].fix_type # 1: No
fix, 3: 3D fix
        ekf_status = the_connection.messages['EKF_STATUS_REPORT'].flags
        lat = the_connection.messages['GPS_RAW_INT'].lat # degE7
        lon = the_connection.messages['GPS_RAW_INT'].lon
        alt = the_connection.messages['GPS_RAW_INT'].alt # mm

        time_s = time_usec / 1000000
        lat_deg = lat / 10000000
        lon_deg = lon / 10000000
        alt_deg = alt / 1000

        row = [time_s, fix_type, ekf_status, lat_deg, lon_deg, alt_deg]
        csvwriter.writerow(row);
        print(row)
```

3.4. Evaluation

We need to evaluation our GPS spoofing attack framework in two aspects: the correctness of GPS signal transmission and the effectiveness of the attack strategy.

Correctness of GPS signal transmission: The prerequisite of the successful attack is that the GPS positions received by a GPS receiver are not deviated from the original positions over a threshold. For a GPS receiver, we are able to read the received GPS positions from NMEA $*GPGGA* messages, and calculate the error from the latitude and longitude coordinates to verify the accuracy. For a drone, we can read its GPS positions from GPS_RAW_INT messages via MAVLink, and compare them to the original positions.

Effectiveness of the attack strategy: For the GPS spoofing attack on a drone, to verify that the victim drone is flying to our expected compromised destination, it is difficult to read the drone’s real positions directly from the MAVLink messages at the moment. We decided to observe the drone’s movement by eyes, record the key points, and reconstruct the flying path afterwards, and compare it to the drone’s original track and the trajectory of spoofing GPS signal. For example, we made the original track to the East and we observed the flight path deviated to the Northeast in our tests, when we applied the position shifts to the South.

Chapter 4. Experiment Design and Results

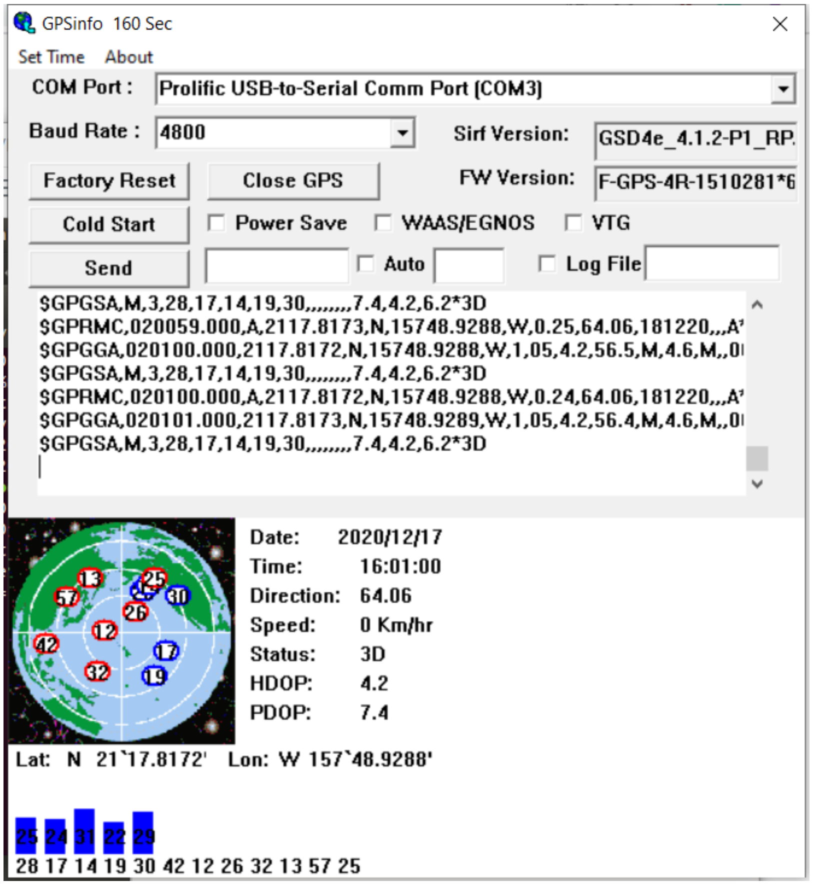
We have transmitted the forged GPS signal to a GPS receiver to verify the feasibility of the GPS spoofing attacks in our lab, and we have further attacked the GPS receivers on a consumer drone in our lab and controlled fields.

4.1. GPS Spoofing attack on a Receiver

In this section, we describe the processes and the results of two GPS spoofing attack experiments on a GPS receiver: fixed position spoofing and moving position spoofing.

4.1.1. Fixed Position Spoofing

We used a BU-353S4 GPS receiver [31] and GPS Info tool [32] (as shown in Figure 12) to display GPS information including latitude, longitude, time, satellite number, signal strength, etc.



COM Port : Prolific USB-to-Serial Comm Port (COM3)

Baud Rate : 4800

Sirf Version: GSD4e_4.1.2-P1_RP.

FW Version: F-GPS-4R-1510281*6

□ Power Save □ WAAS/EGNOS □ VTG

□ Auto □ Log File

```
$GPGSA,M,3,28,17,14,19,30,,,,,,,,7.4,4.2,6.2*3D
$GPRMC,020059.000,A,2117.8173,N,15748.9288,W,0.25,64.06,181220,,,A*
$GPGGA,020100.000,2117.8172,N,15748.9288,W,1,05,4.2,56.5,M,4.6,M,,0
$GPGSA,M,3,28,17,14,19,30,,,,,,,,7.4,4.2,6.2*3D
$GPRMC,020100.000,A,2117.8172,N,15748.9288,W,0.24,64.06,181220,,,A*
$GPGGA,020101.000,2117.8173,N,15748.9289,W,1,05,4.2,56.4,M,4.6,M,,0
$GPGSA,M,3,28,17,14,19,30,,,,,,,,7.4,4.2,6.2*3D
```

Date: 2020/12/17
Time: 16:01:00
Direction: 64.06
Speed: 0 Km/hr
Status: 3D
HDOP: 4.2
PDOP: 7.4

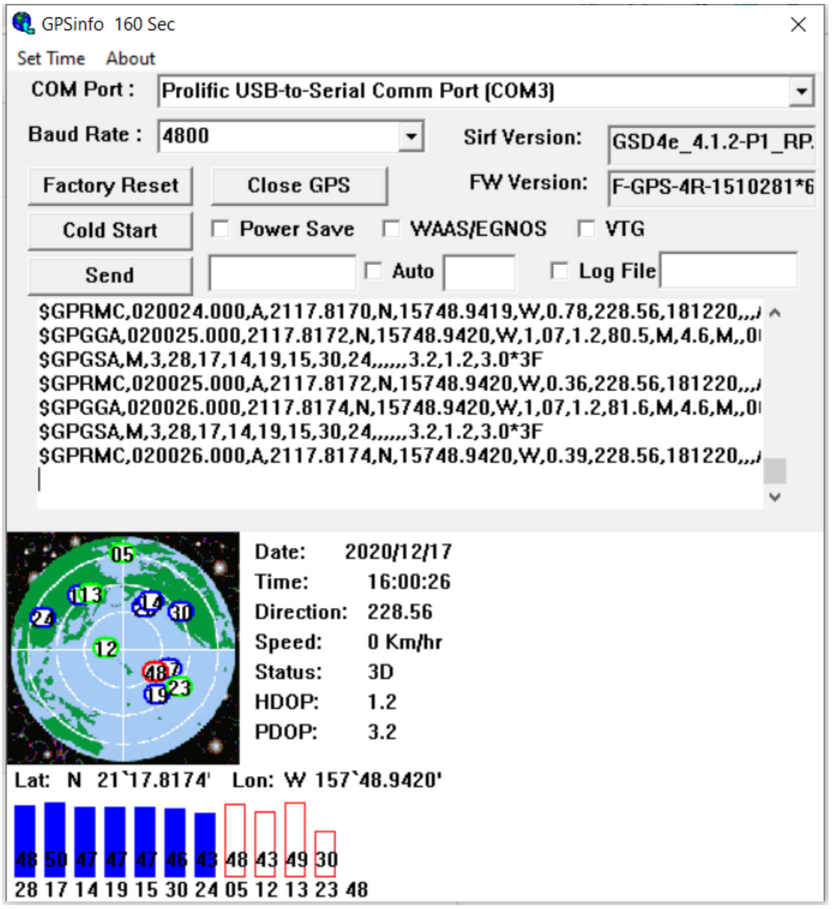
Lat: N 21`17.8172' Lon: W 157`48.9288'

Figure 12. GPS Info, initial status after position fixed by real satellites.

We generated and transmitted the spoofing signal as follows, and the details are described in Section 3.3.1:

- Download the most recent ephemeris data in a brdc file from CIDDS.
- Extract the brdc file from the compressed file.
- Use GPS-SDR-SIM to generate bitstream at the current time with a fixed position, e.g., a point at Honolulu (latitude = 21.28296965N, longitude = 157.815687W, height = 100m).
- Transmit the bitstream through bladeRF.

First the GPS receiver had a 3D fix to real satellites and showed its real position. Then after starting the transmission, as the fake spoofing satellite signals were significantly stronger than the real ones, they dominated the GPS channels, and the receiver could be deceived to the new location within 20 seconds. The results are shown in Figure 13.



COM Port : Prolific USB-to-Serial Comm Port (COM3)

Baud Rate : 4800

Sirf Version: GSD4e_4.1.2-P1_RP

FW Version: F-GPS-4R-1510281*6

```
$GPRMC,020024.000,A,2117.8170,N,15748.9419,W,0.78,228.56,181220,,,A
$GPGGA,020025.000,2117.8172,N,15748.9420,W,1,07,1.2,80.5,M,4.6,M,,0
$GPGSA,M,3,28,17,14,19,15,30,24,,,,,,3.2,1.2,3.0*3F
$GPRMC,020025.000,A,2117.8172,N,15748.9420,W,0.36,228.56,181220,,,A
$GPGGA,020026.000,2117.8174,N,15748.9420,W,1,07,1.2,81.6,M,4.6,M,,0
$GPGSA,M,3,28,17,14,19,15,30,24,,,,,,3.2,1.2,3.0*3F
$GPRMC,020026.000,A,2117.8174,N,15748.9420,W,0.39,228.56,181220,,,A
```

Date: 2020/12/17
Time: 16:00:26
Direction: 228.56
Speed: 0 Km/hr
Status: 3D
HDOP: 1.2
PDOP: 3.2

Lat: N 21`17.8174' Lon: W 157`48.9420'

Figure 13. GPS Info, the status after position fixed by spoofing signal.

The corresponding NMEA messages are:

```
$GPGGA,020024.000,2117.8170,N,15748.9419,W,1,07,1.2,80.7,M,4.6,M,,0000*42
$GPGSA,M,3,28,17,14,19,15,30,24,,,,,,3.2,1.2,3.0*3F
$GPGSV,3,1,12,28,47,030,48,17,40,115,50,14,38,032,47,19,34,144,47*7D
$GPGSV,3,2,12,15,27,321,47,30,24,059,46,24,10,290,43,05,00,000,48*70
$GPGSV,3,3,12,12,75,267,43,13,32,330,50,23,23,125,30,48,50,125,*73
$GPRMC,020024.000,A,2117.8170,N,15748.9419,W,0.78,228.56,181220,,,A*75
$GPGGA,020025.000,2117.8172,N,15748.9420,W,1,07,1.2,80.5,M,4.6,M,,0000*49
$GPGSA,M,3,28,17,14,19,15,30,24,,,,,,3.2,1.2,3.0*3F
$GPRMC,020025.000,A,2117.8172,N,15748.9420,W,0.36,228.56,181220,,,A*76
$GPGGA,020026.000,2117.8174,N,15748.9420,W,1,07,1.2,81.6,M,4.6,M,,0000*4E
$GPGSA,M,3,28,17,14,19,15,30,24,,,,,,3.2,1.2,3.0*3F
$GPRMC,020026.000,A,2117.8174,N,15748.9420,W,0.39,228.56,181220,,,A*7C
```

From the obtained data, we show in Table 4 the list of satellites in different scenes: bitstream data generated by gps-sdr-sim; position fixed with real satellites; position fixed with spoofed satellites.

Table 4. List of Satellite PRNs in each scene.

Scene	5	7	12	13	14	15	17	19	23	24	25	26	28	30	32	42	48	57
Bitstream	●	●	●	●	●	●	●	●		●			●	●				
Real satellites			○	○	●		●	●			○	○	●	●	○	○		○
Spoofed satellites	◎		◎	◎	●	●	●	●	◎	●			●	●			○	

(● = satellites both in view and in use, ◎ = satellites in view with valid SNRs but not in use; ○ = satellites in view without valid SNRs and not in use)

As we generated GPS bitstream using the most recent ephemeris data and the current time, the PRNs of valid satellites in the bitstream and real satellites are nearly the same.

Therefore, as the ephemeris, satellite PRNs and time in the spoofing signal are consistent with the real GPS signal, we can fool the receiver with a fake position near the real position in a short time when the receiver has already got a position fix, similarly to a "hot start".

As to the GPS positions in each scene, Table 5 shows that the 2D distance between the real position and the specified spoofing position is 21.45, and the error between the transmitted and received spoofing positions is 1.35m.

Table 5. Position coordinates.

Scene	Coordinates (Lat, Lon, Height)	2D distance from the spoofed position
Bitstream	21°17.8174'N, 157°48.9412'W, 100	-
Real satellites	21°17.8172'N, 157°48.9288'W, 56.5	21.45m
Spoofed satellites	21°17.8174'N, 157°48.9420'W, 80.5	1.35m

4.1.2. Moving Position Spoofing

We could also generate locations in a moving trajectory and let the receiver virtually move on the trajectory through the following steps:

- Draw a route in Google Maps, which is shown in Figure 14, and save it as a KML file used to display geographic data in an Earth browser (such as Google Earth). It uses a tag-based structure with nested elements and attributes and is based on the XML standard.

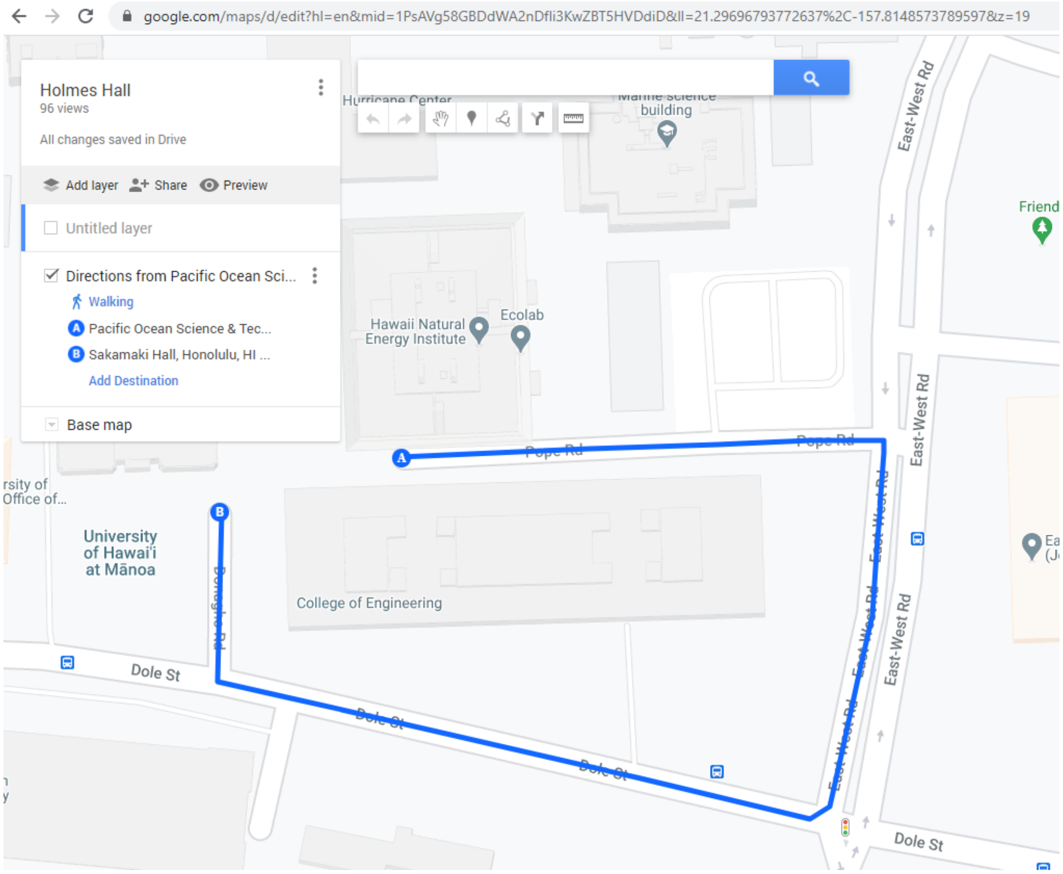
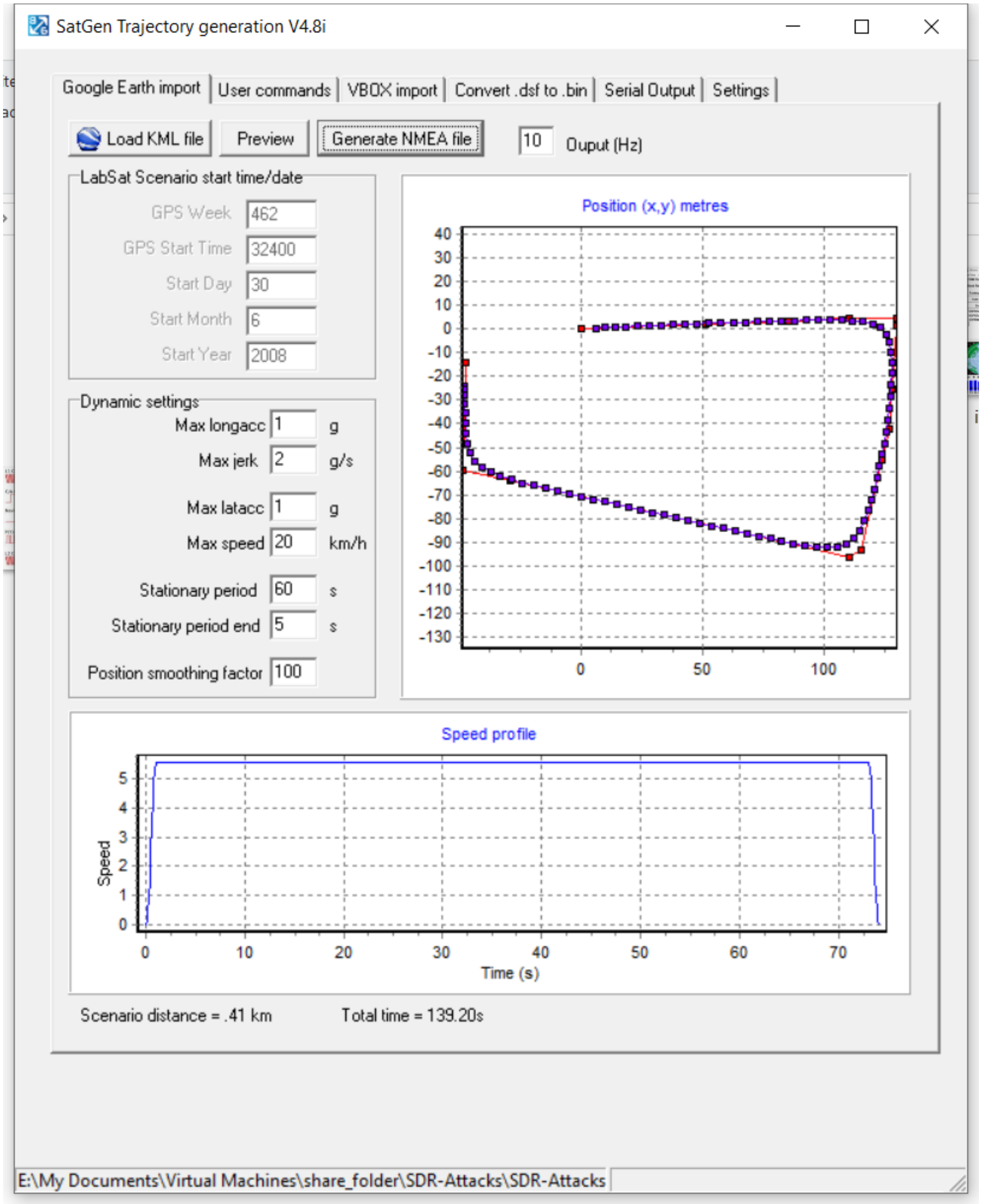

google.com/maps/d/edit?hl=en&mid=1PsAVg58GBDdWA2nDfli3KwZBT5HVDdiD&ll=21.29696793772637%2C-157.8148573789597&z=19
Holmes Hall
96 views
All changes saved in Drive
Add layer
Share
Preview
Untitled layer
Directions from Pacific Ocean Sci...
Walking
Pacific Ocean Science & Tec...
Sakamaki Hall, Honolulu, HI ...
Add Destination
Base map
Hurricane Center
Marine science building
East-West Rd
Friend
Hawaii Natural Energy Institute
Ecolab
Pope Rd
A
B
University of Hawai'i at Mānoa
College of Engineering
Dole St

Figure 14. A route around Holmes Hall in Google Maps

- Convert it into an NMEA file through SatGen Trajectory generation. The software interface is shown in Figure 15, where we set the maximum speed to 20 km/s and the output rate to 10 Hz.



10 Ouput (Hz)

LabSat Scenario start time/date

GPS Week 462
GPS Start Time 32400
Start Day 30
Start Month 6
Start Year 2008

Dynamic settings

Max longacc 1 g
Max jerk 2 g/s
Max latacc 1 g
Max speed 20 km/h
Stationary period 60 s
Stationary period end 5 s
Position smoothing factor 100

Scenario distance = .41 km

Total time = 139.20s

E:\My Documents\Virtual Machines\share_folder\SDR-Attacks\SDR-Attacks

Figure 15. Generate NMEA file in SatGen Trajectory generation.

(Note the Y-axis label is m/s)

- Generate the bitstream using gps-sdr-sim.
- Transmit the bitstream through bladeRF. The script is the same as the one in the fixed-position test.

We plotted the transmitted and received GPS routes, as shown in Figure 16. Also, we calculated the maximum position deviation in X-Y coordinate and height, which are 7.7m and 35.9m, respectively.

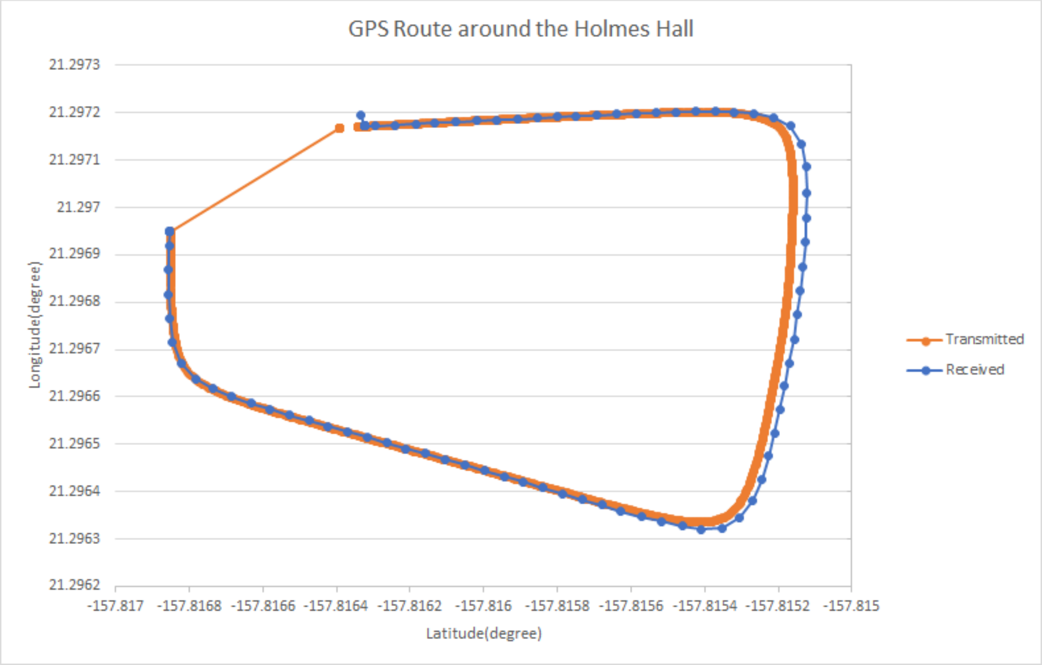


Figure 16. GPS route around the Holmes Hall.

4.2. GPS Spoofing attack on a Drone

We have carried out the spoofing attack when the drone was flying outdoors in the following two scenarios shown in Figure 9 and Figure 10 from Section 3.2:

a. When the drone was hovering at a fixed point, we sent GPS positions which were gradually shifting to a location a few meters away from the origin point in longitude and latitude. (We do not change the altitude.) Based on the drone position control algorithms, the drone should be drifting to the opposite direction to maintain its location “unchanged”. Figure 9 illustrates this scenario.

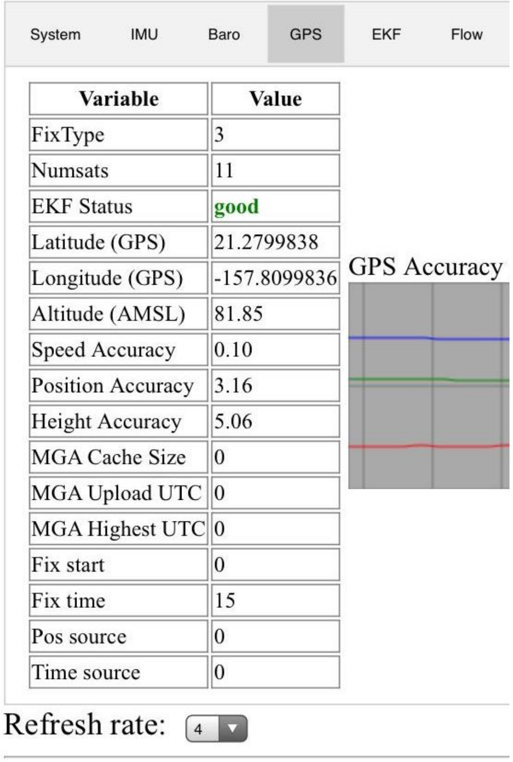
b. When the drone was flying a mission from a source point A to a destination point B, we sent spoofing locations to deviate it from the actual flight path. Figure 10 shows this scenario.

For safety reasons, we always attach a string on the small drone to avoid accidents.

4.2.1. Experiment Method for Signal and Data Transmission

The signal and data transmission for the spoofing attacks was done in a similar procedure to spoof the GPS receiver: we first generated and transmitted data streams, and then observed the GPS information on the drone.

We used a python script to read the GPS status from the *MAVLink* interface on the drone, which is described in Section 3.3.2. The script obtained the GPS status on the drone including latitude, longitude, fix type, position accuracy, velocity accuracy, and other data. In the meantime, SkyViper Journey provides a web interface, shown in Figure 17, for monitoring system status including GPS information in real time.



Variable	Value
FixType	3
Numsats	11
EKF Status	good
Latitude (GPS)	21.2799838
Longitude (GPS)	-157.8099836
Altitude (AMSL)	81.85
Speed Accuracy	0.10
Position Accuracy	3.16
Height Accuracy	5.06
MGA Cache Size	0
MGA Upload UTC	0
MGA Highest UTC	0
Fix start	0
Fix time	15
Pos source	0
Time source	0

Refresh rate: 4

Figure 17. SkyViper drone’s web interface for GPS status.

4.2.2. Hovering Test Result

First, we did a spoofing test when the drone was sitting on the ground to verify the effectiveness of the attack approach. (We conducted this test many times to ensure we obtained the results not affected by external factors such as strong wind.)

In the first 80 seconds, the spoofing GPS position stayed at the origin position to allow the drone to get fixed to the attack signal. Then, we sent the GPS positions shifted 5 meters to the west of the origin position in 5 seconds and stay there for 60 seconds. The shift velocity of 1m/s is low enough to avoid being detected by *ArduPilot*. Figure 18 and Figure 19 (a)-(d) show the results. In Figure 18, the purple points show the original spoofing signal we sent and the blue points are the spoofing signal that the drone received. We can see that there were offsets (mostly in Y-axis) from the original spoofing positions transmitted to the positions that the drone received. However, the offsets were constant and less than 1 meter, which means the spoofing signals that the drone received were within the civilian GPS accuracy (< 10 feet) and acceptable.

Figure 19 (a) shows the fix type and EKF status; Figure 19 (b) shows the position and speed error. At the very beginning of the spoofing GPS signal transmission period and after the transmission was just finished, due to the unstable GPS signal, the fix type was “no fix”, the EKF status was *bad,* and the position and speed errors were fluctuating. During most of the transmission period the signal was good enough and the position error was lower than 5 meters. Also, in Figure 19 (c) and Figure 19 (d) we can see the ground speed was 1m/s for 5 seconds in the middle, and the course changed to around 270 degrees. Such results correspond to the position movement in the spoofing signal.

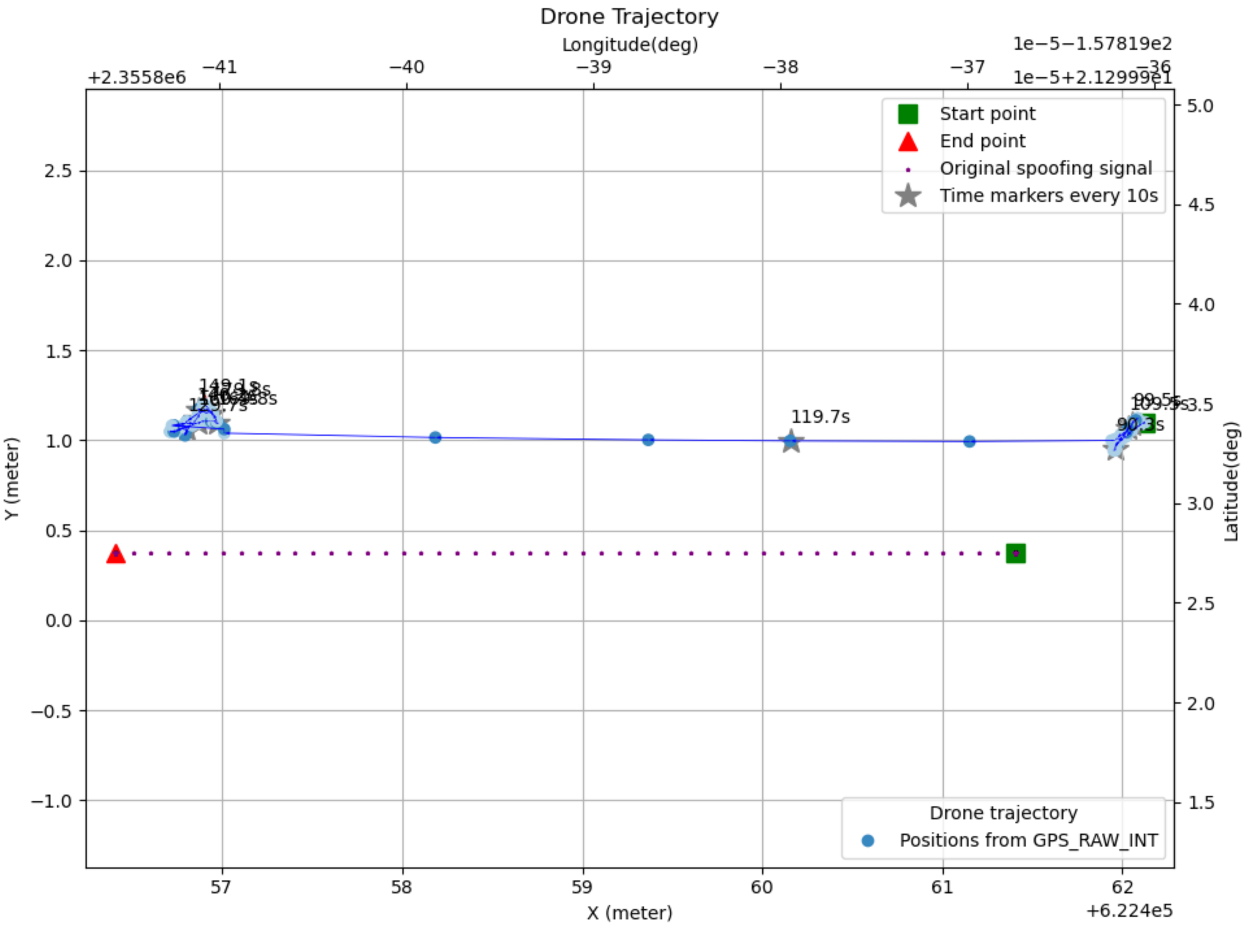
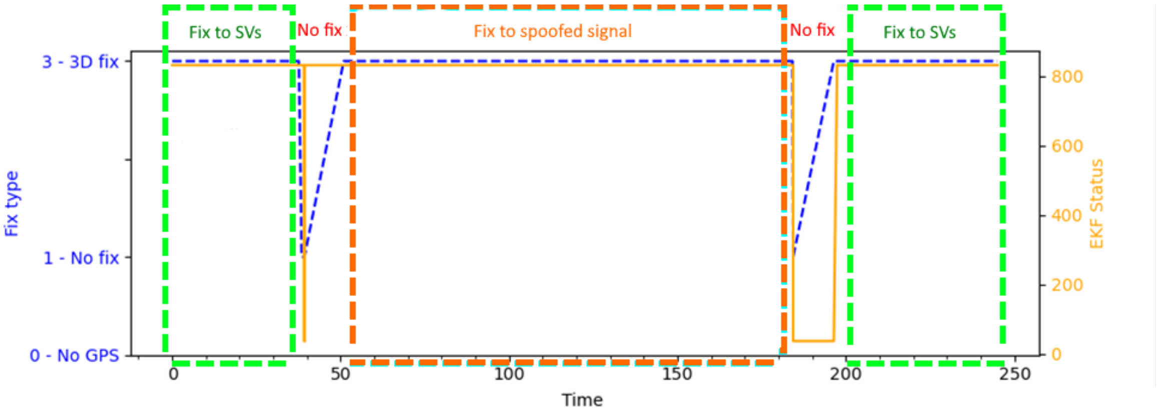
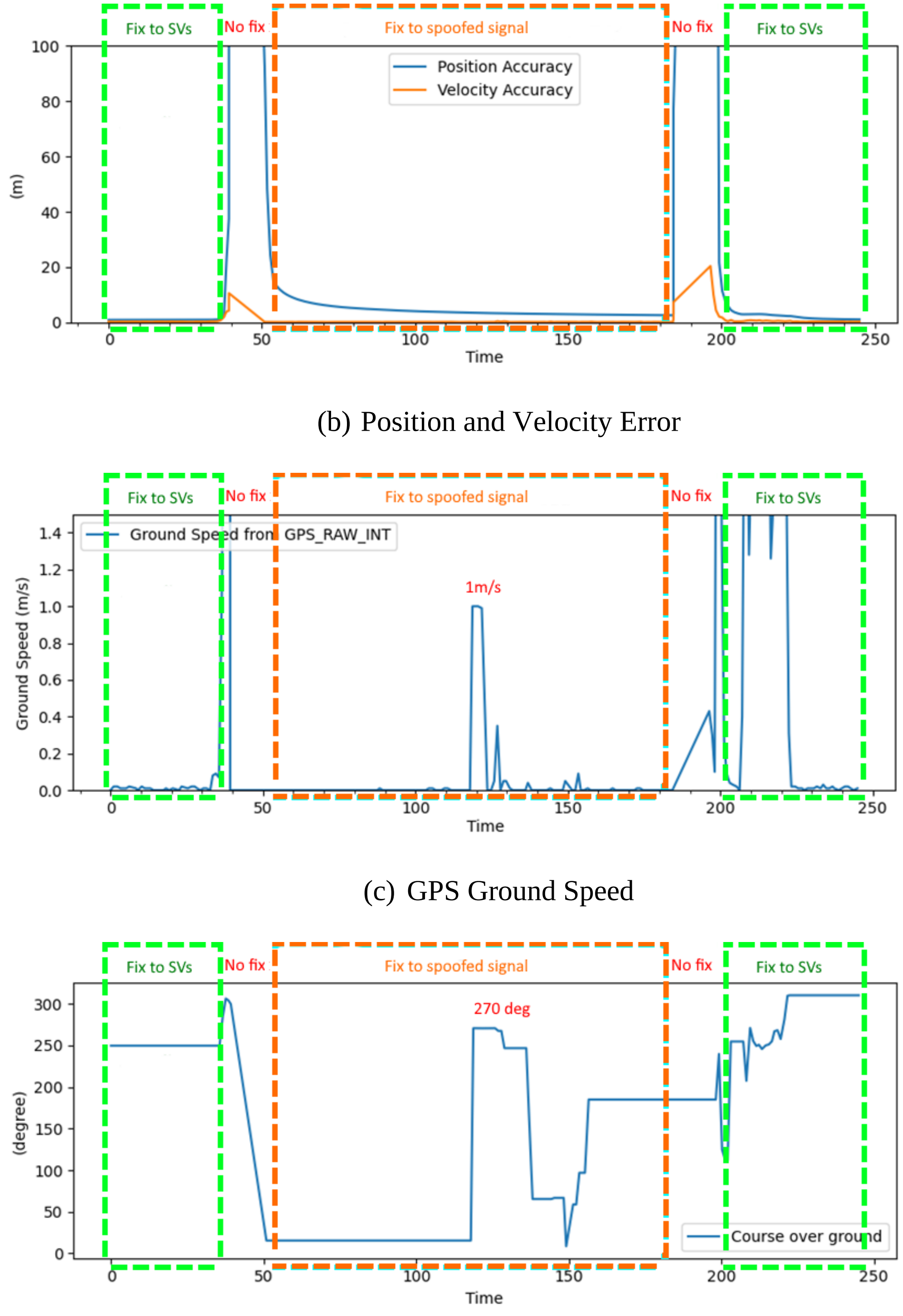


Figure 18. GPS Trajectory when the drone was sitting on the ground.



(a) Fix type and EKF Status



(b) Position and Velocity Error

(c) GPS Ground Speed

(d) GPS Course

Figure 19. Drone status when the drone was sitting on the ground.

Then we conducted a test with a spoofing GPS signal with a 15-meter position shift (shown in Figure 20) and launched the drone 70 second after the start of the GPS transmission when the GPS fix was stable. (Again, we repeated this test many times to deal with uncontrollable external factors.)

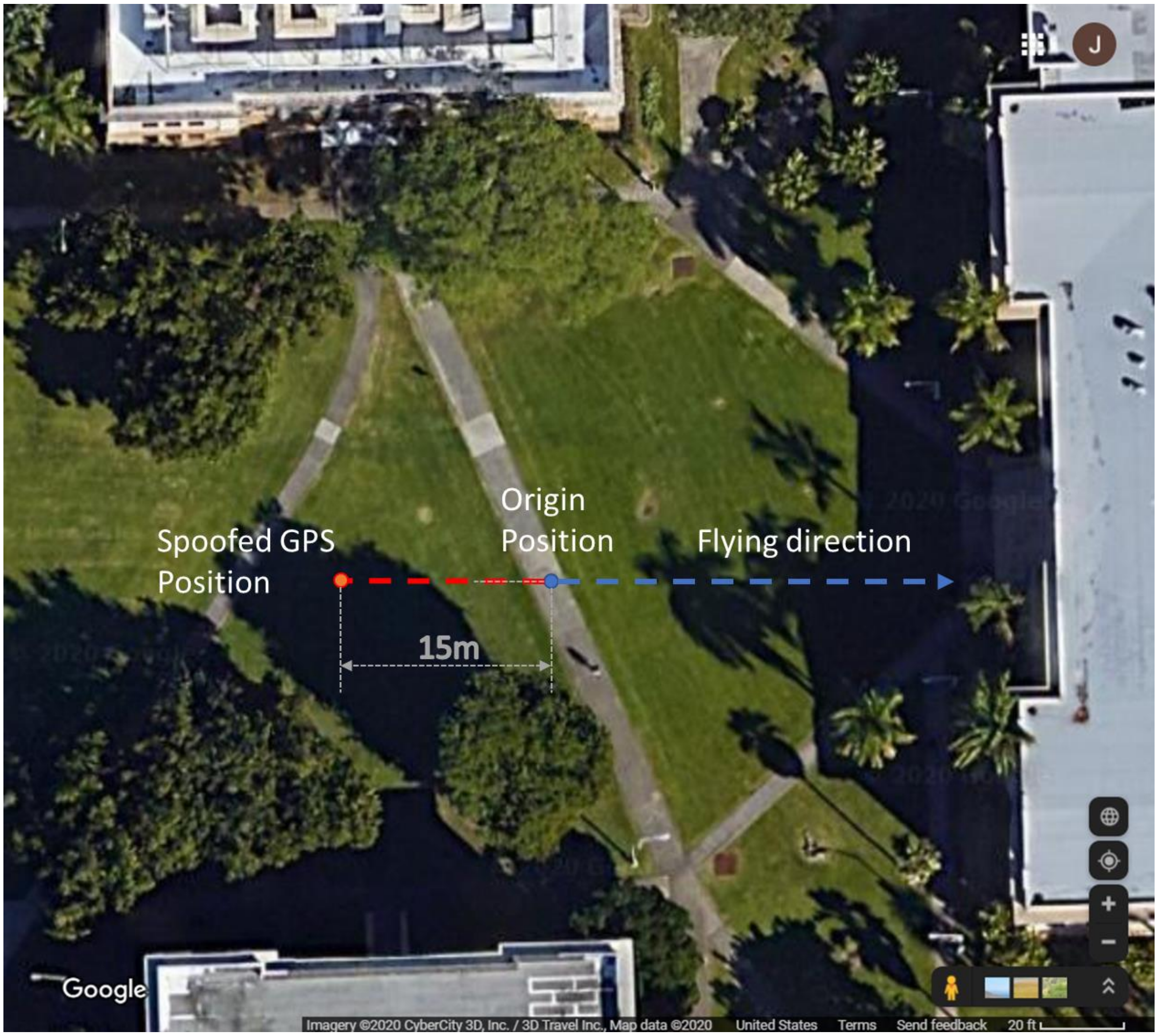


Figure 20. Illustration of the hovering test

The results are shown in Figure 21 and Figure 22, and they look similar to the stationary test above. After the GPS positions started to move to the West, although it is not able to show the drone's actual displacement from the status readings, we did observe that the drone flew to the east fast trying to compensate the offset of the GPS position (shown in Figure 20).

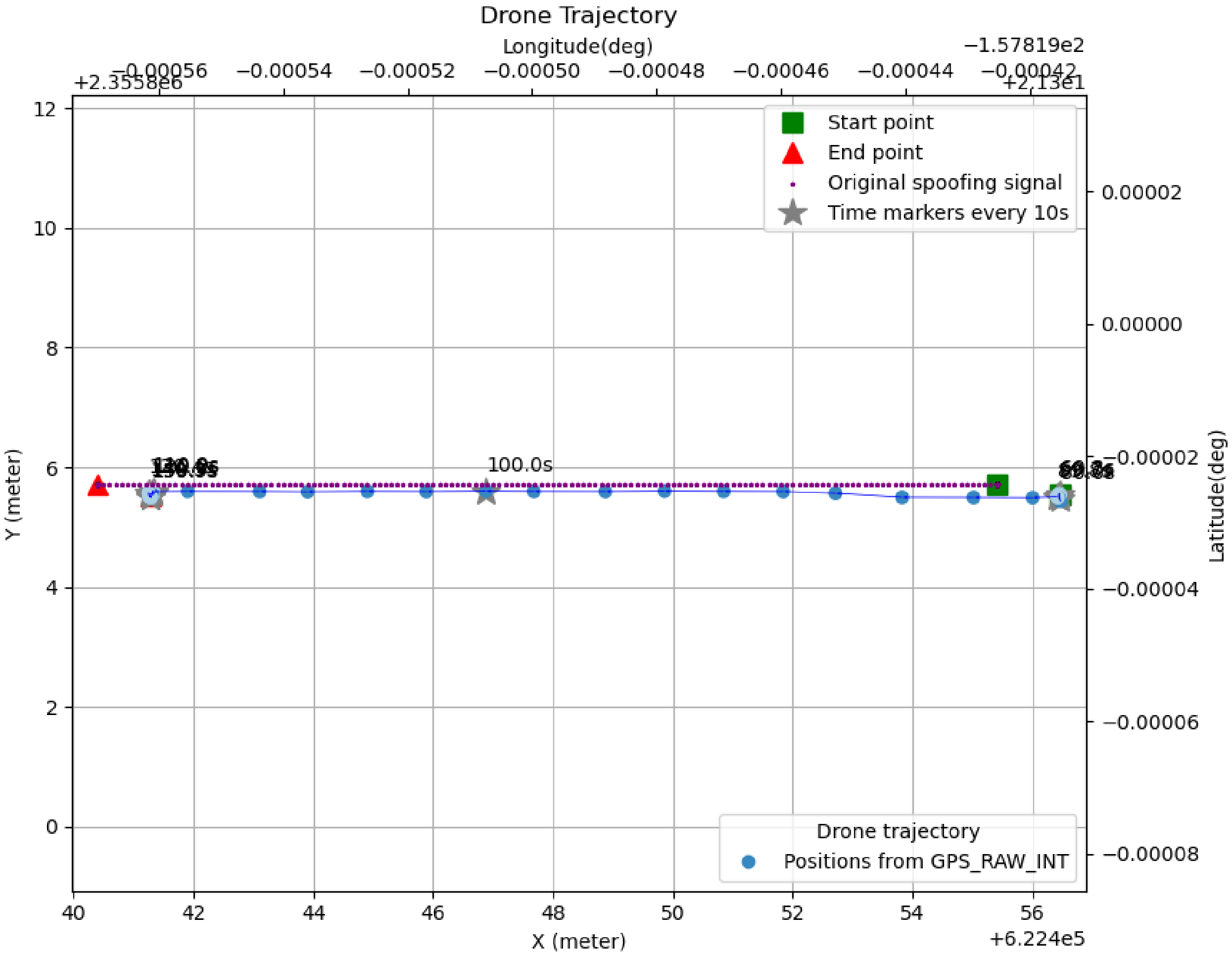
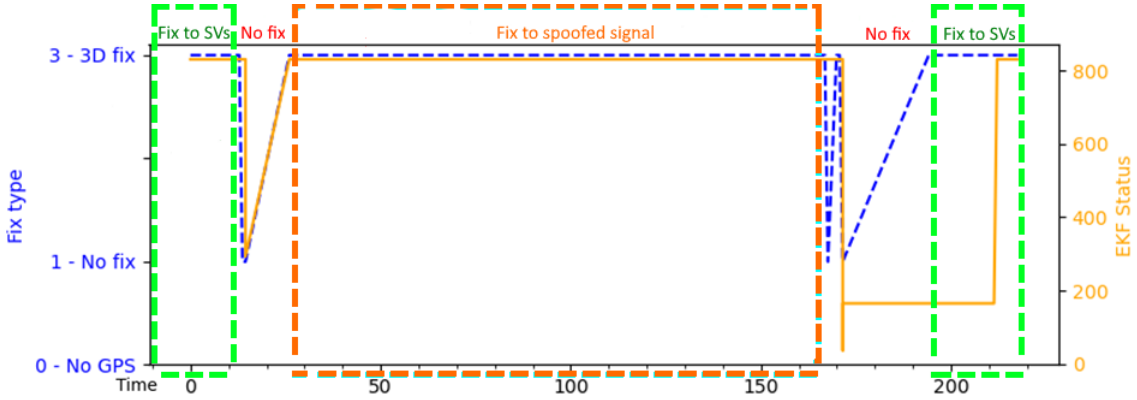
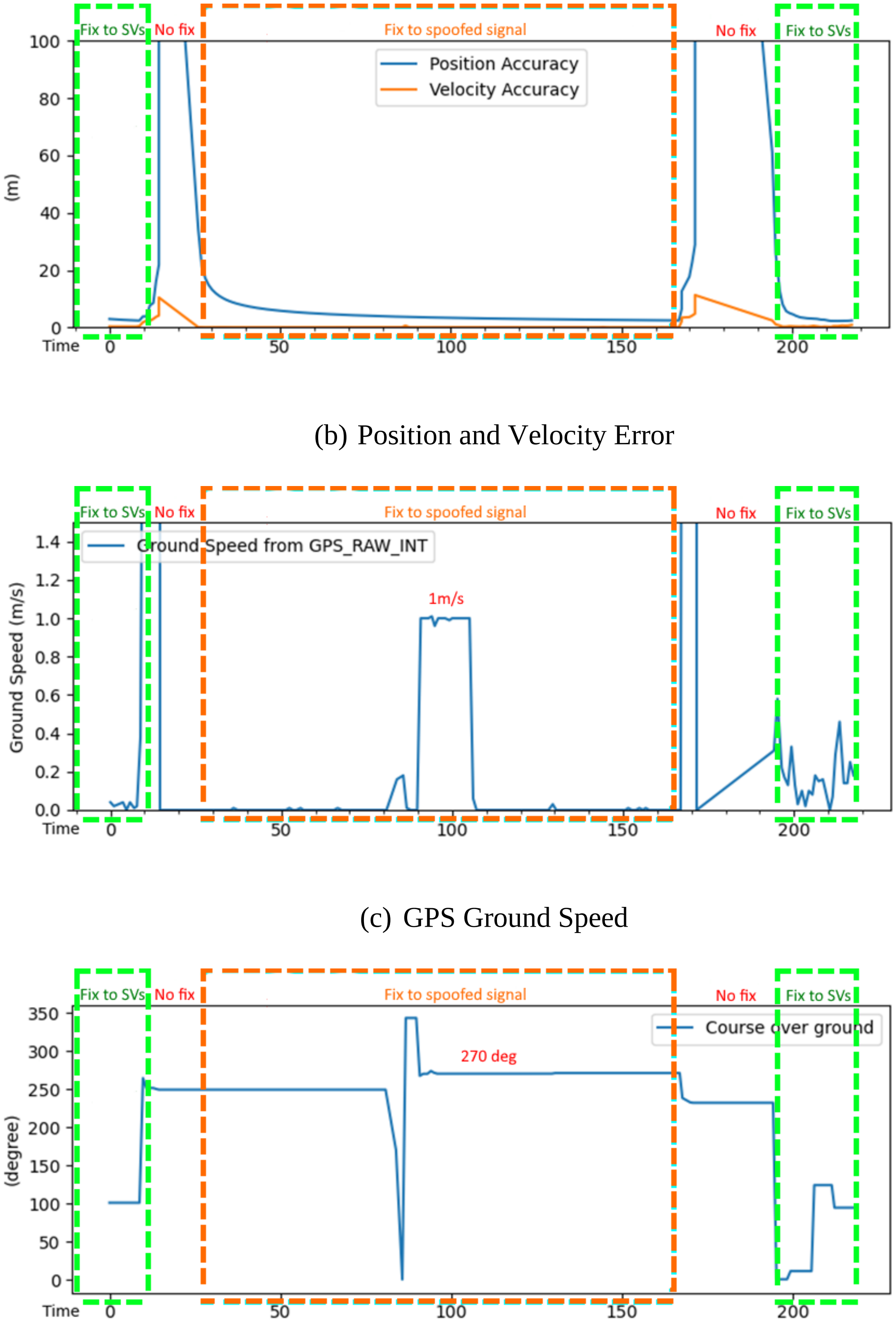


Figure 21. GPS Trajectory when the drone was hovering



(a) Fix type and EKF Status



(b) Position and Velocity Error

(c) GPS Ground Speed

(d) GPS Course

Figure 22. Drone status when the drone was hovering.

4.2.3. Mission Flying Test Result

Before we conducted the GPS attack, we first sent a flying mission to the drone to test its behavior under the guidance of read GPS signal navigation. The mission was flying to the destination 30 meters away to the East at the speed of 1m/s. The drone's GPS trajectory and GPS ground speed are shown in Figure 23 and Figure 24, respectively. From the time of 37s to 80s, the drone was flying to the East. We observed that the true ground speed could not always 1m/s and the average ground speed was approximately 0.7m/s. Currently we cannot location the cause of the inaccurate speed as there were many internal and external factors. Nevertheless, we know that we need to adjust the speed of the GPS spoofing signal to make it as close to the drone's speed as possible.

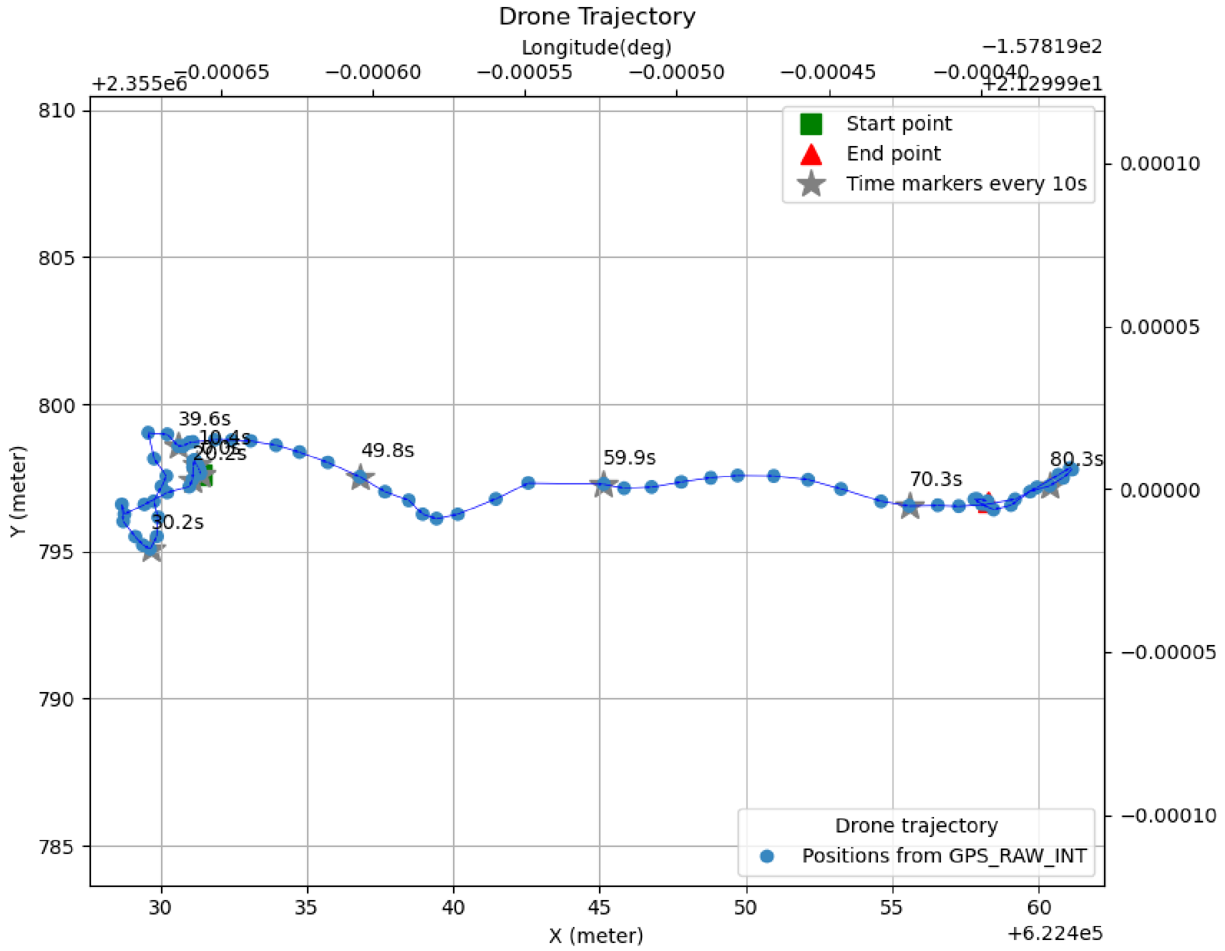


Figure 23. GPS Trajectory when the drone was flying a mission without spoofing GPS signal.

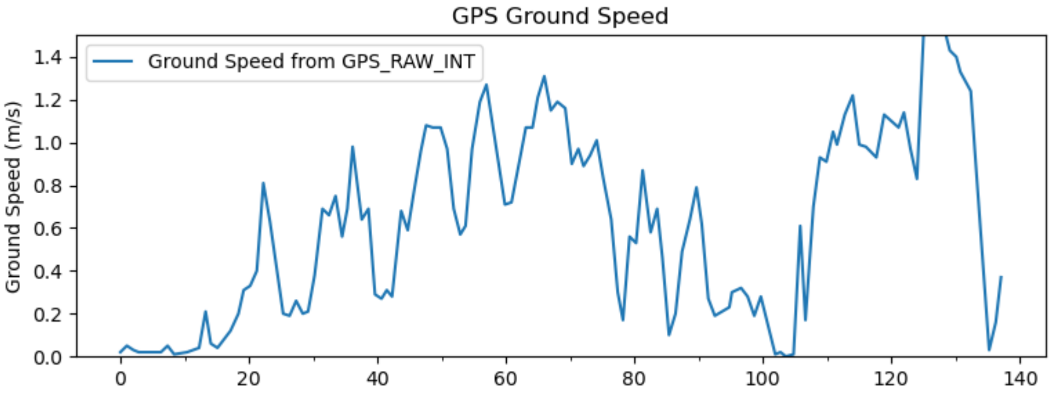


Figure 24. GPS Ground Speed when the drone was flying a mission without spoofing GPS signal.

The mission flying test is similar to the previous hovering test except for the GPS trajectory and the drone's mission. We first started the transmission of spoofing GPS signal with a fixed position, after the drone got a stable 3D fix, then launched the drone and sent a mission through DroneKit. The spoofing GPS positions started to move by following up the drone's movement at the same speed.

Figure 25 illustrates the test plan. The mission directed the drone to a destination 30 meters away eastward at the speed of 1m/s, but the actual ground speed was approximately 0.7m/s. We generated a spoofing GPS trajectory as follows: It begun to move to the Southeast at the Eastward speed component of 0.7m/s and southward speed component of 1m/s. Five seconds later, it bended to the East with the 0.7m/s Eastward speed unchanged. The total Eastward displacement of the GPS positions is 30 meters. As a result, the spoofing positions followed the drone's track in the East direction but shifted 5 meters to the South. Therefore, instead of flying to the East, the drone was expected to fly to the Northeast to compensate the northward GPS position drift.

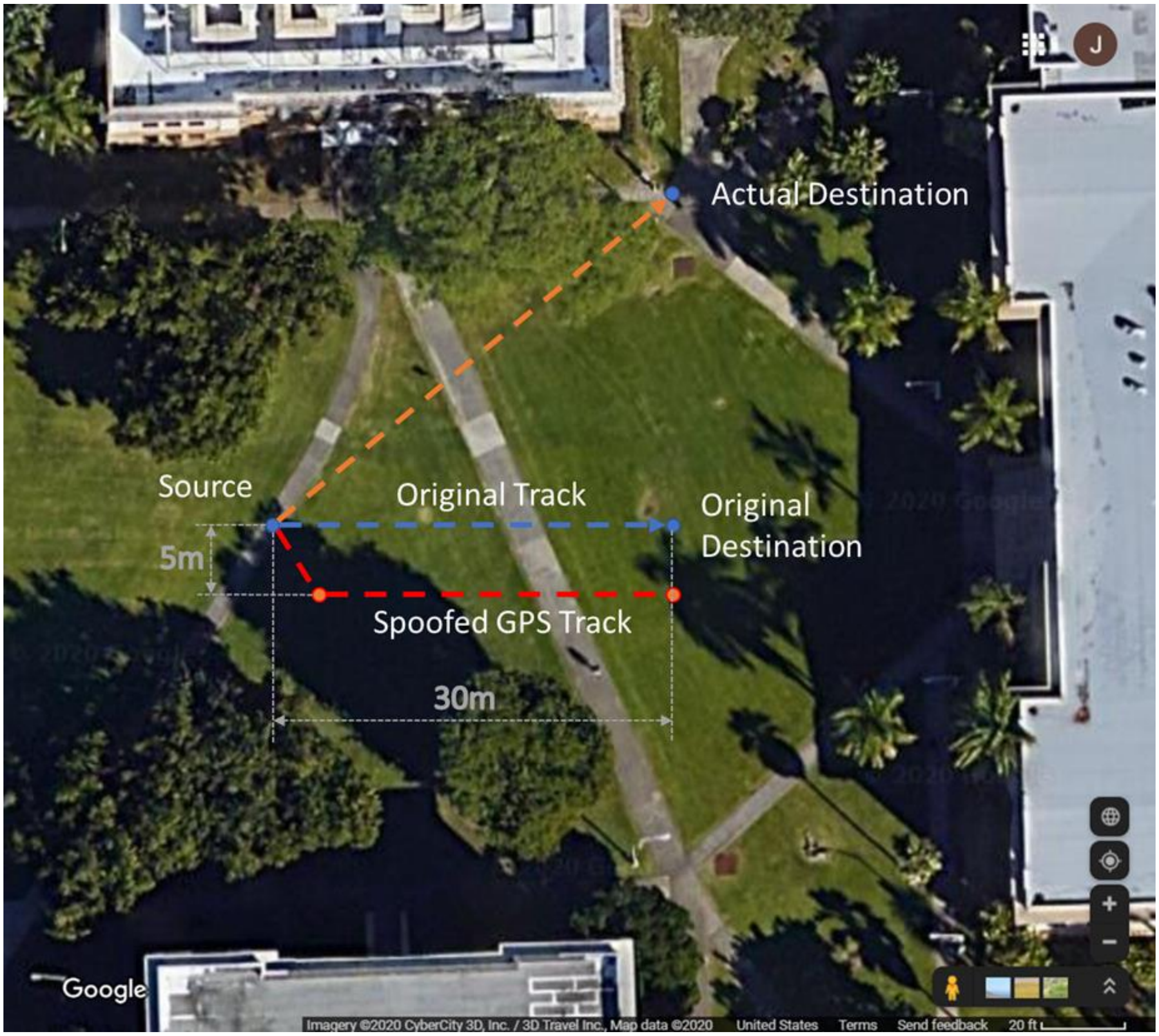

Actual Destination
Source
Original Track
Original
Destination
5m
Spoofed GPS Track
30m
Google
Imagery ©2020 CyberCity 3D, Inc. / 3D Travel Inc., Map data ©2020 United States Terms Send feedback 20 ft

Figure 25. Illustration of the mission flying test on Google Maps

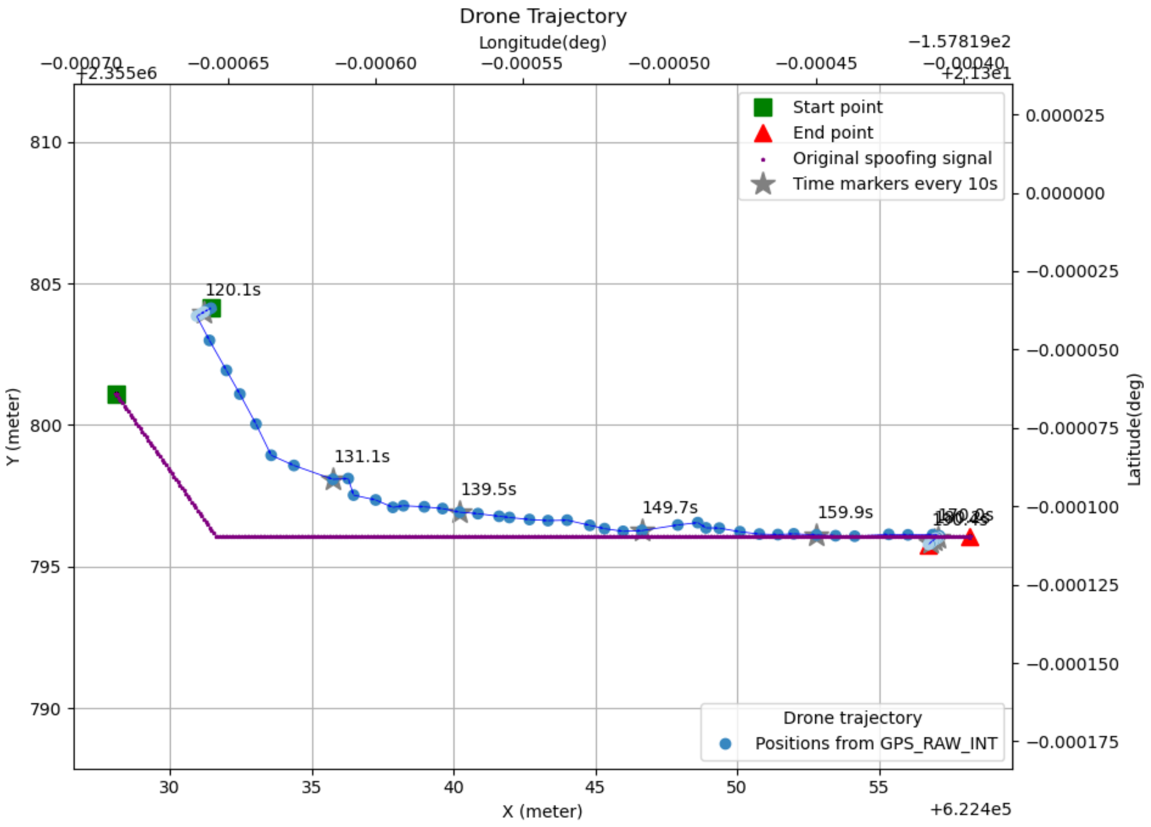
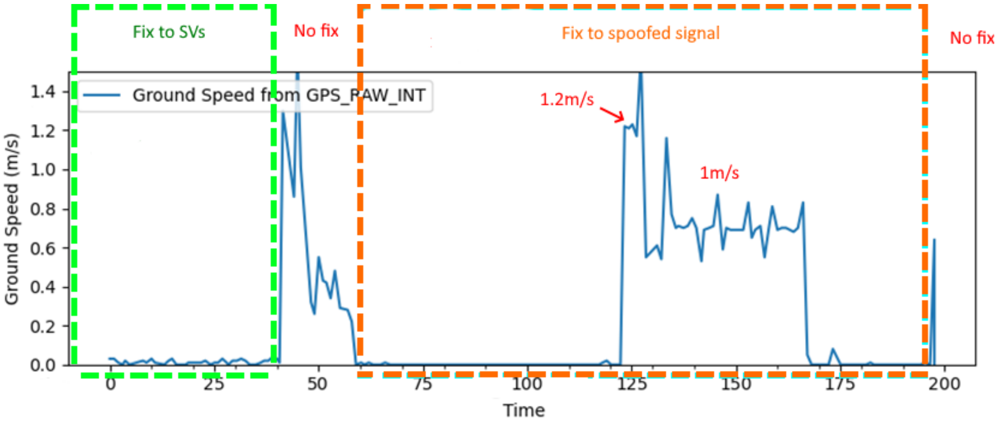
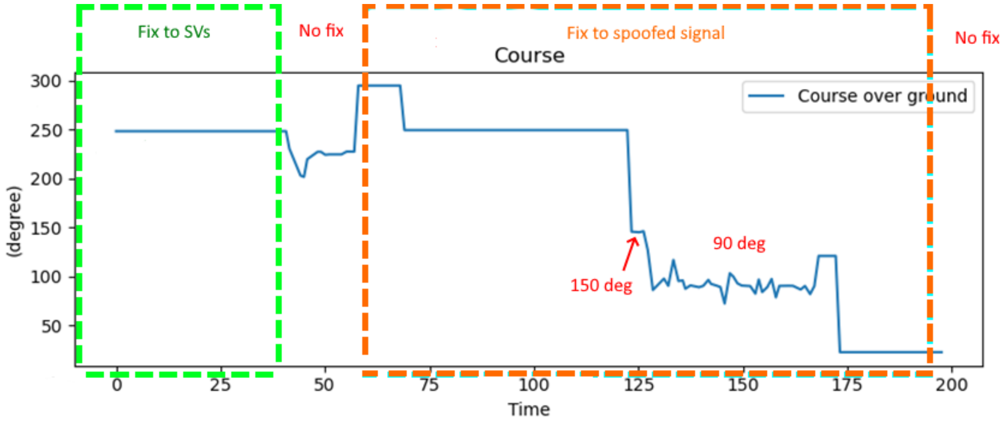


Figure 26. GPS Trajectory when the drone was flying a mission



(a) GPS Ground Speed



(b) GPS Course

Figure 27. Drone status when the drone was flying a mission

Figure 26 and Figure 27 show the test results. In Figure 26, the GPS positions that the drone received (the blue points) is not perfectly consistent with the original GPS positions we sent (the purple line) but still acceptable with the maxim error of approximately 4 meters. From Figure 27 (a) we can see that around the time of 125s, the GPS ground speed, which was approximately 1.2 m/s, corresponds to the segment moving southeast in Figure 26, and the zigzag ground speed at around 0.7m/s from the time of 130s to 165s corresponds to the segment moving east. Also Figure 27 (b) shows that the course of 150° at around 125s implies the Southeast movement, and that the course of approximately 90° corresponds the East movement. The drone's actual track was heading to the Northeast, which is consistent with the orange track in Figure 25.

4.2.4. Discussion on Experiment Results

We have succeeded several times for each GPS spoofing attack in a relatively long range (≥ 30m).

However, the success rate is relatively low. There were frequent problems that the drone could not get stable 3D fix with accurate positions, or it did not fly to our expected destination. We figured out multiple obstacles as follows:

1. *GPS signal generation and transmission*

Our GPS signal generation depends on the ephemeris data in the BRDC file downloaded from the NASA CDDIS database. This data from real satellites is only considered valid for about 30 minutes and is broadcast by each satellite every 30 seconds [7]. The file is updated every 20 – 30 minutes. However, the BRDC file is not always applicable for our test. For example, from 2:00 pm to 3:25 pm HST (12:00 am to 1:20 am UTC) and from 4:00 pm to 4:25 pm HST (2:00 pm to 2:25 pm), there is no updated file available for GPS signal generation at the current time. As another example, from 5:00 pm to 5:15 pm HST (3:00 pm to 3:15 pm UTC), there are only 4 satellites in the generated GPS bitstream while a drone GPS receiver usually requires locking to least 6 satellites. It may lead to low GPS position accuracy and is not sufficient for *ArduPilot* to take off. In short, there are some specific time slots when we are able to conduct successful GPS spoofing attack.

It is reported that some of the bladeRFs have the issue of low on-board clock stability, which may result in failures in GPS applications. A solution of using an external high accuracy clock is given in Section 4.3. Another solution may be using a more precise device, such as high-end USRP devices.

2. Attacking a moving drone

The synchronization between the drone’s movement and the GPS signal its received is critical to the success of the proposed attack. If the GPS positions exceed or lag behind the drone's expected movement significantly, the drone would deviate in an unexpected direction. For example, in this test, the GPS positions were always to the east of the drone’s expected positions, so the drone tried to shift westward for compensation.

The influencing factors include:

1) Timing. The start time of GPS signal transmission and the take-off time of the drone varied a little every time. The difference may be up to a few seconds, which is much larger the interval of GPS updates (0.1 second).
2) Velocity control. In our mission fly test, although the speed was set to 1m/s from a DroneKit command, the speed was not stable, and the average speed was around 0.7m/s. It may be due to the accuracy of the drone motors, the accuracy of the control board, or the wind speed.
3) Wind. We cannot control wind speed and directions. The wind may affect the drone’s velocity and direction control because the testing drone is very small.

4.3. Additional Work: Using an External Clock on bladeRF

Some bladeRF 2.0s are suffered from insufficient clock stability for GPS applications, which results in failed our GPS spoofing experiments. In this section, we introduce a solution to improve clock stability performance with the use of an external high precision clock source.

We purchased a 10MHz TCXO (Temperature Compensated Crystal Oscillator) [41] and applied it to the bladeRF. We connect the TXCO's clock output to the REF_IN connector (J95) on the bladeRF, so that bladeRF can tame the onboard VCTCXO (Voltage Controlled Temperature Compensated Crystal Oscillator) to output a precise 38.4MHz clock with the external 10MHz clock and a PLL. More principle details are discussed in the forum thread [42] and the FAQ .

The block diagram of bladeRF 2.0 from the schematic [44] is shown in Figure 28.

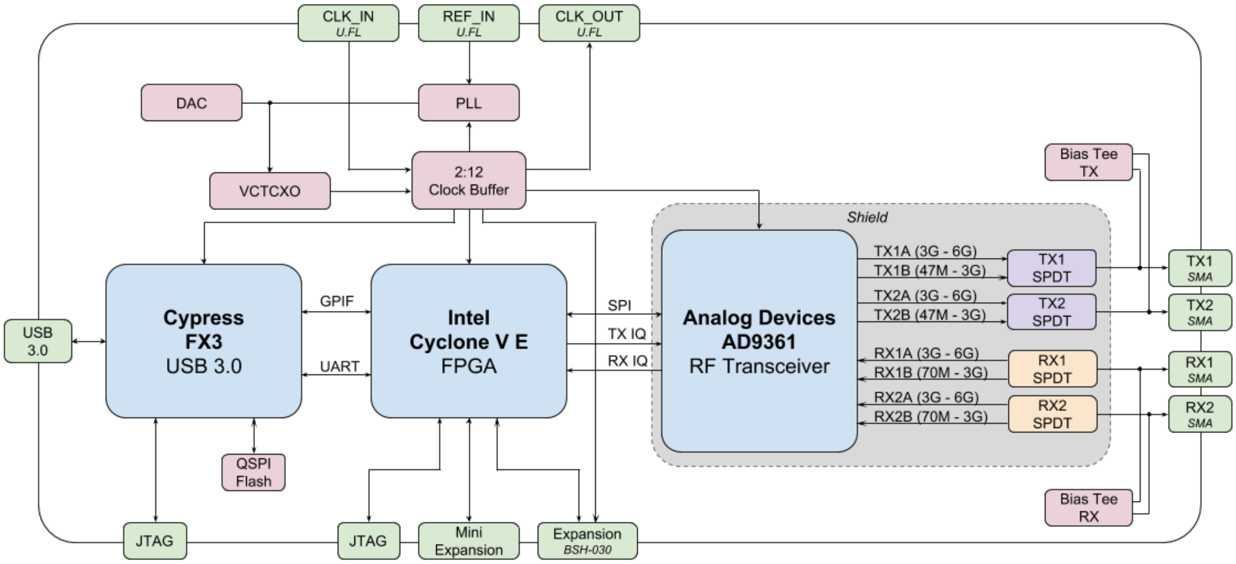


Figure 28. bladeRF 2.0 schematics.

Hardware setup

As the TCXO PCB has only pin headers for clock output and powering, we need to make some wires for connections. I reformed an SMA to UFL coaxial cable by replacing the SMA connector by jumper pin headers, with the shielding layer as the GND and the core as the CLK. I connected the TXCO's clock output to the REF_IN connector (J95) on the bladeRF by this cable.

Also, I soldered a wire to the anode of one of the LEDs on the bladeRF as a 3.3V power and connected it to the TCXO PCB. The hardware setup is shown in Figure 29.

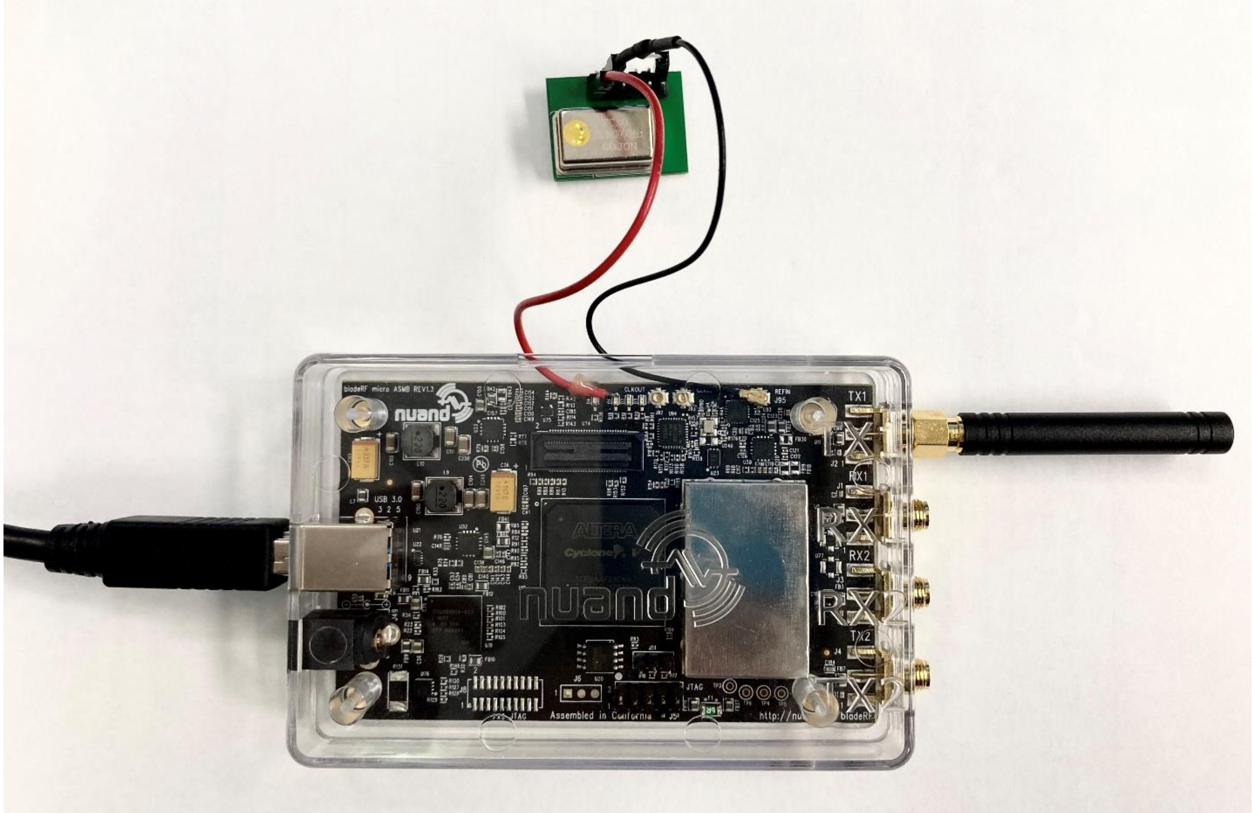


Figure 29. Hardware setup for applying an external TCXO.

Software usage

To enable the external reference input, issue the following bladeRF-cli command3. The output shows the internal clock is locked.

```
bladeRF> set clock_ref enable
Clock reference: REFIN to ADF4002 (locked)
```

The default reference clock frequency is 10MHz. If using other frequencies, set the reference frequency to, e.g., 20 MHz:

```
bladeRF> set refin_freq 20M
REFIN frequency: 20000000 Hz
```

We applied the 10MHz external clock to both the bladeRF xA4 and xA9 we have, and both of them worked for GPS spoofing. The xA4 didn't work for our test before with the onboard clock.

A script for transmitting GPS bitstream with an external clock is

```
set clock_ref enable
set frequency tx 1575.42M
set samplerate 2.6M
set bandwidth 2.5M
set gain tx 56
set biastee tx on
tx config file=gpssim.bin format=bin
tx start
tx wait
```

Chapter 5. Conclusion and Future Work

In this thesis, we described GPS positioning principles and data formats, and introduced the software and hardware platform for GPS spoofing attacks as well as a consumer drone's control and communication protocol. Then we proposed a practical GPS spoofing attack framework and verified it by experiments. Our experimental results show that the drone is vulnerable to such attacks and it is feasible to make the drone deviate from the original destination.

The countermeasures for the proposed GPS spoofing attack may include:

- Using encrypted military-grade GPS code. Attackers cannot generate such encrypted code by open-source algorithms, so it will be very challenging to spoof the GPS positions in such cases.
- GPS failsafe and glitch protection. *ArduPilot* has a mechanism to recognize glitches and warn the controller if abnormal GPS changes are detected [45]. However, since the detection was designed for dealing with random error and not for malicious attacks, if the spoofing GPS signal shifts slowly enough, the drone is still vulnerable. The protection algorithm could be improved to detect continuous small shifts such as the proposed attack.
- Track and verify drone motion using wireless localization technologies [48], then the deviation of drone motion will be detectable.
- Calibration with other navigation methods, such as high-precision inertial navigation, vision navigation or even manual controls, in case of GPS failures.

Based on our current research, we plan to further investigate the following issues.

- Figure out a way to solve drone's real positions from inertial sensor measurement readings from the MAVLink messages. Then, we can easily verify a drone's real flightpath.
- Exploit the capability of GPS simulator and SDR hardware, and conduct experiments with real-time generated spoofing positions and more precise synchronization with the drone. If we are able to calculate the precise attacking positions based on the drone's current position in real time, then fine-grained and flexible attacks would be viable. The GPS simulator software GPS-SDR-SIM and SDR device bladeRF are both open-source. We

can examine the designs of them and modify the configuration to generate and transmit spoofing signals in real time.

- Develop drone tracking techniques to measure the drone's real-time position and velocity, so that we can perform the attacks remotely in practice. Some measurement approaches include using a PTZ-camera [46] and Doppler radar [47]. We will explore drone-focued measurement methods based on those ideas.

References

[1] GPS Overview, https://www.gps.gov/systems/gps/, accessed on 12/17/2020

[2] K. Wang, S. Chen, A. Pan, "Time and Position Spoofing with Open Source Projects".

[3] James Bao-Yen Tsui, Fundamentals of Global Positioning System Receivers: A Software Approach, Chapter Five GPS C/A Code Signal Structure, https://pdfs.semanticscholar.org/d6f5/812c9f91d68862cf3e2a8af6a3f9db14ba2b.pdf?_ga=2.30629631.1351239999.1579687368-1382589764.1579687368, accessed on 12/17/2020

[4] Peter H. Dana, "Global Positioning System Overview", https://foote.geography.uconn.edu/gcraft/notes/gps/gps_f.html, accessed on 12/17/2020

[5] Department of Geography, Penn State, The Navigation Message, https://www.e-education.psu.edu/geog862/node/1734, accessed on 12/17/2020

[6] Jan Van Sickle, GPS for Land Surveyors 4th Edition

[7] Almanac and Ephemeris Data as used by GPS receivers, http://gpsinformation.net/main/almanac.txt, accessed on 12/17/2020

[8] Takuji Ebinuma, GPS-SDR-SIM, https://github.com/osqzss/gps-sdr-sim, accessed on 12/17/2020

[9] Great Scott Gadgets, HackRF, https://greatscottgadgets.com/hackrf/, accessed on 12/17/2020

[10] Nuand, BladeRF wiki, https://github.com/nuand/bladeRF/wiki, accessed on 12/17/2020

[11] Kexiong (Curtis) Zeng et al., "All Your GPS Are Belong To Us: Towards Stealthy Manipulation of Road Navigation Systems".

[12] Nils Ole Tippenhauer et al., "On the Requirements for Successful GPS Spoofing Attacks"

[13] W. Chen, Y. Dong, and Z. Duan. "Manipulating Drone Position Control," *in Proc. of IEEE CNS 2019 - IEEE Conference on Computer Communications and Network Security*.

[14] W. Chen, Z. Duan, and Y. Dong, "Compromising Flight Paths of Autopiloted Drones," *in Proc. of IEEE International Conference on Unmanned Aircraft Systems (ICUAS), 2019.*

[15] W. Chen, Y. Dong, and Z. Duan. "Attacking Altitude Estimation in Drone Navigation," *INFOCOM workshop WiSARN 2018: Wireless Sensor, Robot and UAV Networks.", April 2018.*

[16] W. Chen, Y. Dong, and Z. Duan, “Manipulating Drone Dynamic State Estimation to Compromise Navigation,” *in Proc. of IEEE Conference on Communications and Network Security (CNS), May, 2018*.

[17] W. Chen, Z. Duan, and Y. Dong, “False Data Injection on EKF-Based Navigation Control,” Invited paper. *in Proc. of IEEE International Conference on Unmanned Aircraft Systems (ICUAS), 2017*.

[18] GPS-SDR-SIM, https://github.com/osqzss/gps-sdr-sim, accessed on 12/17/2020

[19] Software Defined Radio: Architectures, Systems and Functions (Markus Dillinger, Kambiz Madani, Nancy Alonistioti) Page xxxiii (Wiley & Sons, 2003, ISBN 0-470-85164-3)

[20] ArduPilot, https://ardupilot.org/ardupilot/, accessed on 12/17/2020

[21] H. Chao, Y. Cao, and Y. Chen, “Autopilots for small unmanned aerial vehicles: a survey,” *International Journal of Control, Automation and Systems, vol. 8, no. 1, pp. 36–44, 2010*.

[22] Sky Viper support center: https://support.skyrocketon.com/hc/en-us/categories/115001355028-SkyViper, accessed on 12/17/2020

[23] MAVLink, https://mavlink.io/en/, accessed on 12/17/2020

[24] MAVLINK Common Message Set, https://mavlink.io/en/messages/common.html#messages, accessed on 12/17/2020

[25] Pymavlink, https://pypi.org/project/pymavlink/, accessed on 12/17/2020

[26] Using Pymavlink Libraries (mavgen), https://mavlink.io/en/mavgen_python/, accessed on 12/17/2020

[27] About DroneKit, https://dronekit-python.readthedocs.io/en/latest/about/overview.html, accessed on 12/17/2020

[28] NASA, The Crustal Dynamics Data Information System (CDDIS), https://cddis.nasa.gov/, accessed on 12/17/2020

[29] NASA, Broadcast ephemeris data, https://cddis.nasa.gov/Data_and_Derived_Products/GNSS/broadcast_ephemeris_data.html, accessed on 12/17/2020

[30] Matplotlib, https://matplotlib.org/, accessed on 12/17/2020

[31] BU-353S4 GPS receiver, https://www.globalsat.com.tw/en/product-199952/Cable-GPS-with-USB-interface-SiRF-Star-IV-BU-353S4.html, accessed on 12/17/2020

[32] BU-353S4 GPS receiver downloads (USB driver and GPS Info tool), https://www.globalsat.com.tw/style/frame/m5/features.asp?content_set=color_2&lang=2&customer_id=909&name_id=10593, accessed on 12/17/2020

[33] Petovello, Mark (November 2008), "Satellite Almanac Life Expectancy", Inside GNSS: 14–19, available: https://teddriver.net/Papers/novdec08-gnss-sol-v1.pdf, accessed on 12/17/2020

[34] Department of Geography, Penn State, The Almanac, Time to First Fix and Satellite Health, https://www.e-education.psu.edu/geog862/node/1739, accessed on 12/17/2020

[35] Wikipedia: Time to first fix, https://en.wikipedia.org/wiki/Time_to_first_fix, accessed on 12/17/2020

[36] National Marine Electronics Association, http://www.nmea.org/, accessed on 12/17/2020

[37] NMEA-0183 message: GGA, https://www.trimble.com/OEM_ReceiverHelp/V4.44/en/NMEA-0183messages_GGA.html, accessed on 12/17/2020

[38] What Exactly Is GPS NMEA Data?, https://www.gpsworld.com/what-exactly-is-gps-nmea-data/, accessed on 12/17/2020

[39] NMEA-0183 messages: Overview, https://www.trimble.com/OEM_ReceiverHelp/V4.44/en/NMEA-0183messages_MessageOverview.html, accessed on 12/17/2020

[40] NMEA RMC Message, http://manuals.spectracom.com/VSP/Content/NC_and_SS/Com/Topics/APPENDIX/NMEA_RMCmess.htm, accessed on 12/17/2020

[41] TCXO Clock, High Precision External TCXO Clock PPM0.1 for HackRF One GPS Application. https://www.amazon.com/Precision-External-PPM0-1-HackRF-Application/dp/B07NXSFFLB/, accessed on 12/17/2020

[42] Nuand forum: BladeRF 2.0 frequency stability. https://nuand.com/forums/viewtopic.php?t=4984, accessed on 12/17/2020

[43] Frequently Asked Questions: How do I use a 10 MHz reference? https://www.nuand.com/frequently-asked-questions/#How_do_I_use_a_10_MHz_reference

[44] bladerRF micro schematics, https://www.nuand.com/bladeRF-micro.pdf, accessed on 12/17/2020

[45] GPS Failsafe and Glitch Protection, https://ardupilot.org/copter/docs/gps-failsafe-glitch-protection.html, accessed on 12/17/2020

[46] J. Park; D. H. Kim; Y. S. Shin; S. Lee, “A comparison of convolutional object detectors for real-time drone tracking using a PTZ camera”, *2017 17th International Conference on Control, Automation and Systems (ICCAS)*

[47] M. Jian; Z. Lu; V. C. Chen, “Drone detection and tracking based on phase-interferometric Doppler radar”, *2018 IEEE Radar Conference (RadarConf18)*

[48] M. Sun, Y. Man, M. Li, R. Gerdes, “SVM: Secure Vehicle Motion Verification with a Single Wireless Receiver”