Project “The Interceptor”

Avoiding counter-drone systems with nanodrones

David Melendez Cano

R&D Embedded Software Engineer

Introduction

With massive drone industry growth, comes antidrone systems industry. Due to multiple and even classic vulnerabilities of communications now used by drones, antidrone industry can take down those drones using well documented attacks.

State of the art

Drone-antidrone competition is already put into scene, and we can look at the state-of-the-art from two points of view:

- **Anti-drone systems:**
  - Detection:
    - Image/Thermal cameras, and radar
    - Flight plan
    - Motor noise signatures
    - Image recognition
    - Radio control and telemetry signature
  - Actions:
    - **Radio protocol specific attacks (Deauth WiFi, etc)**
    - **Vendor sub-protocol specific attacks (Command injection/manipulation)**
    - **Radio jamming**
    - GPS spoofing
    - Camera blinding
    - Capture with other aircraft

- **Drone countermeasures**
  - Physical
    - Surrounding protection nets
    - Swarm attacks
  - Logical
    - Spread spectrum radio techniques
    - Inertial navigation with drift correction using image recognition (No GPS)
    - **Use 3G/4G networks to control and telemetry**

The main point of this paper resides behind the fact that state-of-the-art antidrone mitigations claim they are effective, just because the frequency bands they use for control and telemetry are illegal to disturb: (3G/4G) This a reasonable point considering that many antidrone systems are managed, indeed, with 3G/4G links, but it has obvious limitations.

New approach

Using a communication system that claims to be secure ***and immune to jamming, because jamming it would be illegal,*** **is simply not enough** when we are talking about, for example, government/army equipment. Not to mention that you cannot fly if no 3G/4G coverage is available.

Also, physical detection is key to antidrone systems, based mainly on the aircraft size and radar signature.

This new approach offers three main points:

Drone size: Small drones are more difficult to detect. If we can downsize the drone as much as possible, drone detection gets harder. Also drone gets cheaper, if we use standard hardware and cheap parts.

Hidden WiFi protocol for control and telemetry: Most antidrone systems are based on WiFi vendor communication signatures. The key here is to use nearbly existing WiFi networks, by reinjecting their beacon frames, modifying the payload on the fly. The beacon payload can be used both to transmit pilot commands, and to receive drone telemetry. This prevents the creation of an entire and unexpected WiFi network near ~~of~~ the antidrone systems. Making more difficult radio signature detection based methods.

Also, the drone has to be able to act as a hacking platform in the same way as the state-of-the-art hacking drones, with all Wifi hacking capabilities, using only one WiFi adapter for all tasks.

Fallback RF control: If those measures are not enough, and WiFi based communication is indeed jammed, a fall-back system is proposed using a Raspberry Pi with pitx, with arbitrary frequency radio transmission capabilities, and an on-board SDR as receiver. This hardware can also be used as a standard radio sniffer when drone is not being attacked on WiFi band.

Project Interceptor

The Interceptor is a proof-of-concept nano-drone built from scratch. It is an evolution of the drone ATROPOS, a previous project of the author. It's frame is made with Chinese chopsticks, the total cost is $70, and it implements the proposed new approach to communications and hacking capabilities.



Ilustration 1: "Atropos" drone. A PoC of Hardware Hacking

Why chopsticks?. Wood is a good “vibration absorber”, and decreases the total amount of metal parts.

The Interceptor is built with:

Control board: Vocore2.

In order to minimize drone size, the smallest, readily available, Linux based IoT board is chosen. No separate microcontroller is needed to generate time sensitive motor signals (PWM), using the on-board x4 PWM channels of the Vocore2. A custom OpenWrt image is compiled, with a *devicetree* mod since the board comes with only two PWM channels enabled in the default vendor devicetree description.

This keeps cost, weight and hardware complexity ridiculously low, compared to market solutions. Even, some market control boards are almost as big, as this drone itself.

This board manages:

- **Stabilization control program in C, with motor PWM control**
- **WiFi beacon frame based communications**
- **SDR based fallback receiving control protocol**
- **Standard hacking tools (bully, aircrack, pixiewps, fakeap, etc)**
- **SDR standard tools (rtl_sdr, rtl_433, etc)**
- **Power management: Installed MOSFETs can turn on/off all peripherals**

Inertial sensor: GY-953 / BNO055

An inertial measurement sensor attached to Vocore2 serial interface is used to get the aircraft attitude. Refresh rate is 100Hz, in continuous mode, providing Euler angles/Quaternions. Since the Fixed clock calculations are generated inside this sensor, Vocore2 is freed from generating its own time fixed interrupt signals. This approach is chosen against others, to keep CPU load low, while main control is inside Vocore2.

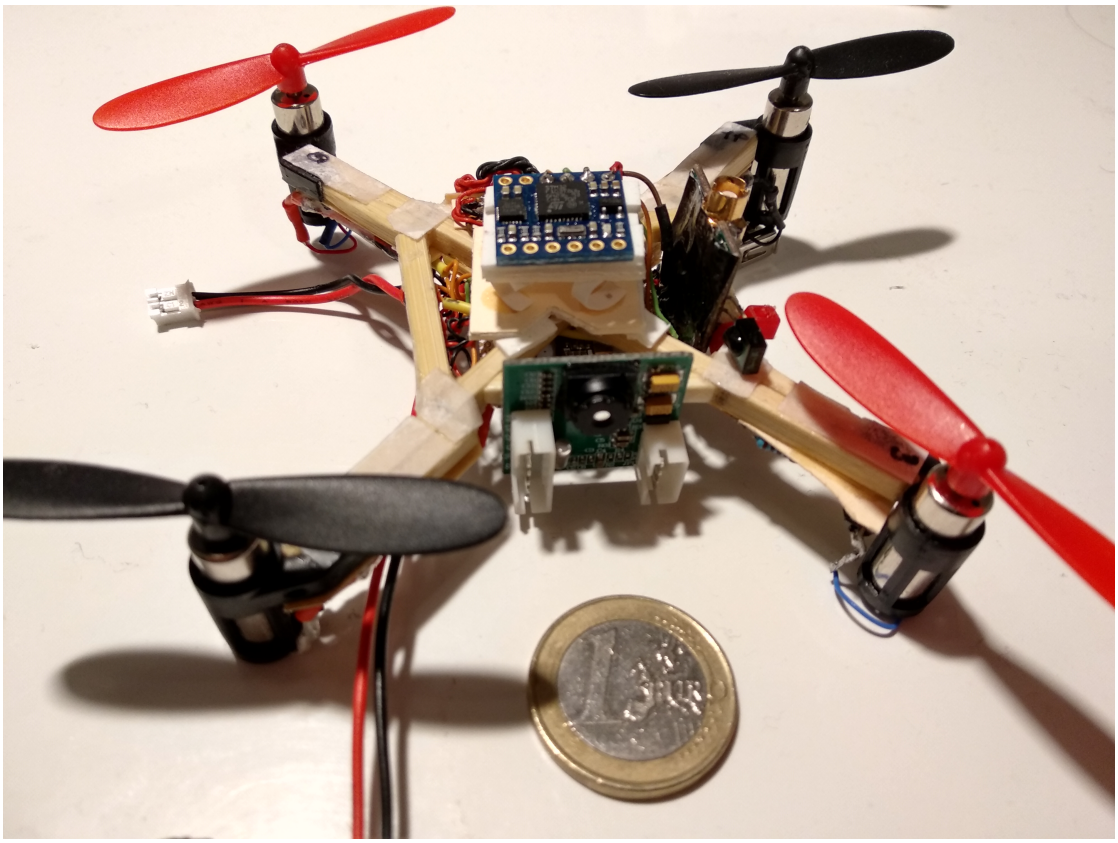


Illustration 2: Top-front view of drone Interceptor. Notice IMU sensor on top, with vibration absorber made with shrink tubes.(Battery and antennas dismounted)

Other hardware parts:

- RTL2832U SDR USB dongle
- 1S LiPo Battery
- x2 channels I2C ADC IC converter. Included in Vocore2 devicetree file, and Kernel driver-access enabled.
- DC-DC converter from 3.7V Lipo battery to 5V for electronics
- Nano-drone-type brushed motors
- MOSFET and passive components to drive brushed motors soldered inline
- UART camera

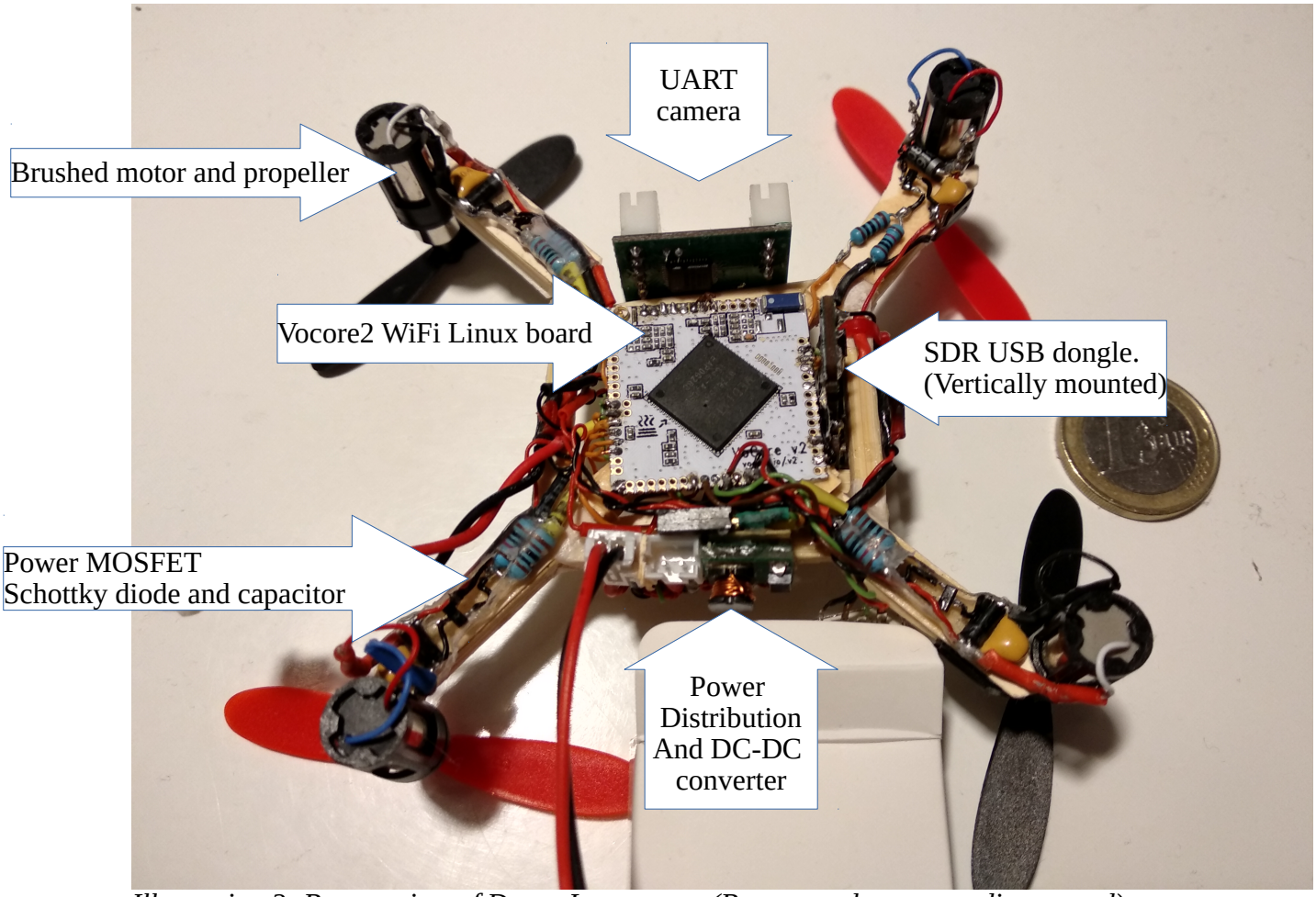


Illustration 3: Bottom view of Drone Interceptor. (Battery and antennas dismounted)

Interceptor Architecture

Interceptor architecture can be divided into two main parts. Pilot side, and drone side.

Pilot side is based on Raspberry-Pi Zero, with a USB pilot trainer (detected as a joystick by Linux *joydev.ko* module), a WiFi module, and rpitx/PiFmRds using GPIO for fallback control.

WiFi based communication

Primary communication method is based on monitor mode, on both drone and pilot sides. A key is exchanged using the previous flight key **or** by wired serial interface between pilot and drone before each flight.

Even in the laziest user mode (using previous key), you only have to turn on the drone and pilot, in a different location to ensure a fresh and safe key is exchanged, taking into account the hypothetical situation that your previous key was somehow compromised by nearby attacker.

Both sides perform a beacon frame scanning on all available channels, in order to capture an arbitrary number of beacon frame headers. Those headers are then reinjected on air, with their payload modified as follows:

Packet payload is forged with:

1. Pseudo-randomized Initialization Vector for AES-CBC encryption
2. **Data type: a pilot command, a telemetry stream or an image stream from camera**
3. **Data length**
4. **Data stream: containing command information/telemetry/camera data**
5. **Current 802.11 channel, and targeted channel (See next chapter "Channel hopping")**
6. **Sequence number: Discards older packets, to avoid command reinjection attacks**
7. SHA256 of all previous data

AES-CBC encryption covers points from 2 to 6, both included

Channel hopping

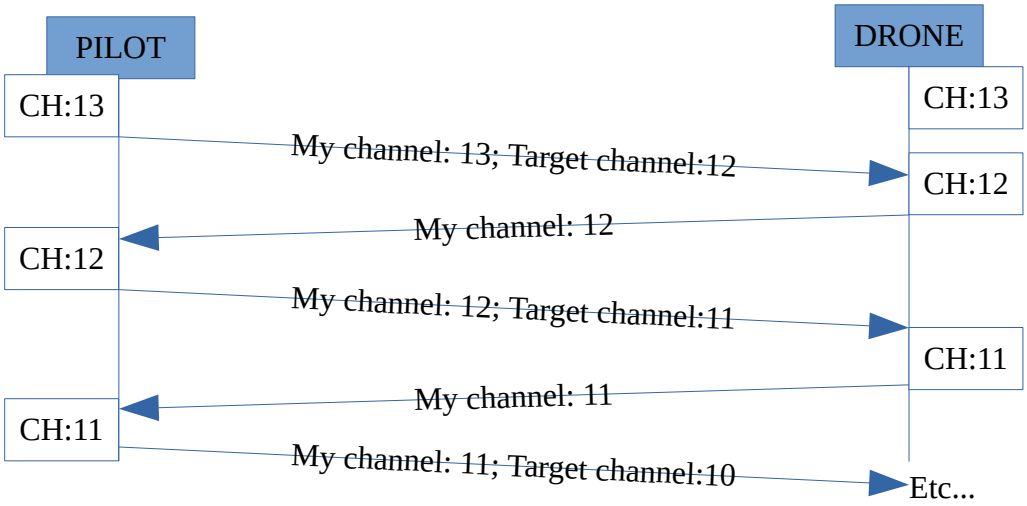
In order to match capabilities of other hacking drones on the scene, the drone has to be able to "visit" any available WiFi channel, even when flying, without losing control, using the same WiFi interface already used for both telemetry/control and WiFi audit. A specific protocol has been developed to change both pilot and drone WiFi channel synchronously.

For example, if drone-pilot system is on channel 13 at the beginning of the flight, and pilot wants to audit a network located on channel 1, the transmitted "targeted channel" field, changes to channel 12. Once drone has changed to 12, "current channel field" coming from drone to pilot, takes value 12. Once pilot receives this packet, pilot changes to 12, and sends "target channel" to 11 to drone. All these steps are repeated until "ultimate target" channel 1 is reached by drone and pilot.

The reason why this works resides on channel design. 802.11 channels are overlapped, and pilot still can receive packets from drone without losing control, even if one is 1 channel away from the other. This situation keeps the protocol simple, and fast enough to work flawlessly.

Negotiation from 1 to 13 and vice-versa, takes ~2-3 seconds keeping smooth control.

Diagram 1: Channel hopping negotiation between pilot and drone, taking advantage of 802.11 channel overlapping and libnl API library



NOTE: every channel change implies automatic stopping of any running WiFi audit tools. Those tools can be respawned by user, once “ultimate target” channel is reached by drone and pilot.

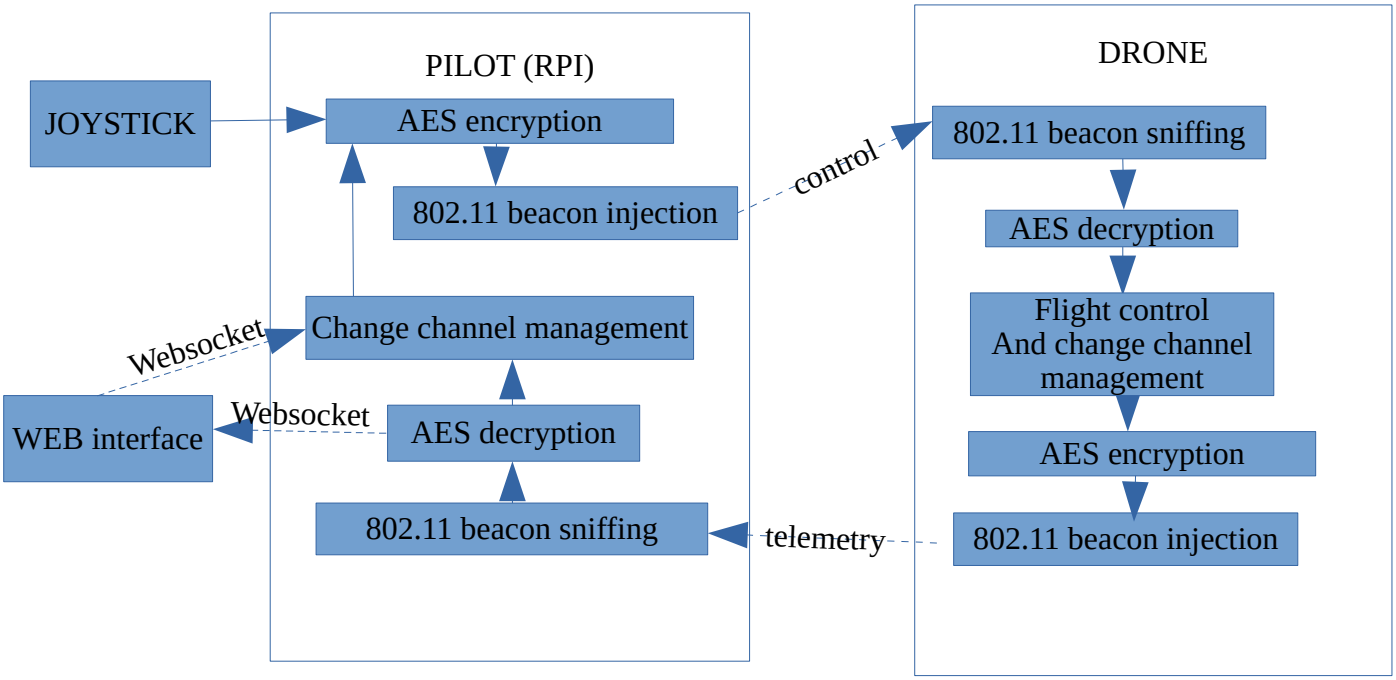


Diagram 2: WiFi protocol architecture

NOTE: a client/AP (infrastructure) mode is also available at the same time as monitor mode. The only restriction is that channel is fixed on this mode, at the same channel of STA/AP mode.

This enables network-level attacking capabilities once the target WiFi network is compromised, and the drone is effectively connected to that network as a regular client.
Examples: reverse SSH tunnel, DNS tunnel, network pentesting, etc.

Web interface

Web interface is complemetary to the joystick in the role of user command & control. A web server is embedded in the josytick application, using mongoose c library, and it uses websockets to send additional commands to drone (channel and mode changes) and to receive telemetry, previously captured from 802.11 beacon frames.

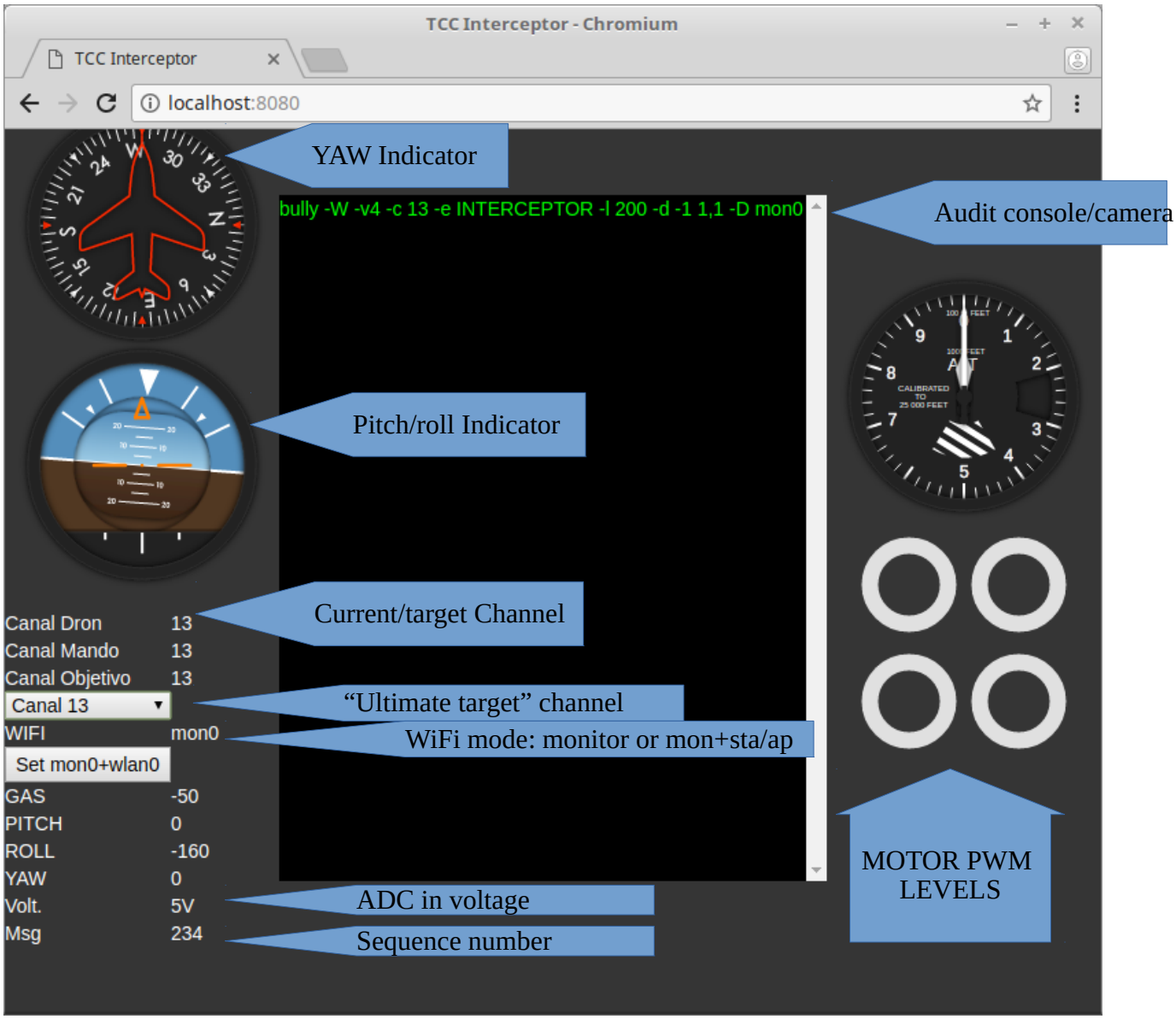


Illustration 4: Interceptor web interface

Fallback control with rpitx/PiFmRds and SDR

In case the Beacon Frame based communication is detected and successfully jammed by any anti-drone system, a fallback system is spawned once a receiving command timeout event is triggered on pilot, or requested by drone, covering the situation where only pilot→drone communication is jammed/attacked.

This feature needs a previous action,, before flight, when the key is exchanged. The drone enables its SDR dongle to perform a Fast Fourier Transform on arbitrary frequency ranges, previously configured by user. **Once FFT is performed, the drone chooses a suitable frequency by using the band containing the least power as reported by the FFT** (Further techniques can be applied here for frequency choosing, for example, next to an FM radio station or any other current transmission) .

This best-suitable frequency is reported to pilot by drone, and stored. In case of a successful attack to WiFi link, pilot Raspberry-Pi rpitx will start to transmit control data over this frequency.

NOTE: Starting rpitx takes some seconds, so, pilot system will be running it even when no attack is performed against drone, BUT GPIO WILL NOT BE CONNECTED TO ANTENNA, until requited. GPIO is switched to GND with a resistor as default position, preventing output to be transmitted until required (avoiding possible premature transmission discovery by antidrone systems).

Packet size is dramatically reduced, limiting all pilot movements to 4 bits per remote control stick. In this emergency scenario, only sticks data are transmitted, (no other non-critical channels) with a payload of two bytes of usable data. Fine trim values are stored previously by drone, assuming all pilot controls are set to default position on engines startup, and applied to received values.

Packet format is simple:

- 1 start byte:0x00 (not allowed in usable data).
- 1 byte Gas-Yaw: Mask 0x0F: Gas, Mask 0xF0 Yaw
- 1 byte Pitch-Roll: Mask 0x0F: Pitch, Mask 0xF0 Roll
- 1 byte checksum

No encryption is implemented in this version for the moment. since this communication system is only used in an emergency situation, and the data is modulated with a very low baudrate to maximize reliability. Further implementation with more resolution and encryption could be available in a few months.

Modulation and demodulation

Once fallback system is triggered, pilot side starts sending 4-byte packets to a Frequency Shift Keying modulator. In this case minimodem project is used. The baudrate is set to an arbitrary number, between 100-300 bauds, previously chosen randomly by drone, and notified to pilot.

Generated audio is then piped and converted to a readable format for rpitx using a DSP library (cdsr) sampling at 48 ksps. Rpitx receives the stream and transmits it using the previouslychosen frequency as FM modulation.

On the drone side, SDR is set to the same frequency, FM demodulated, and piped to an on-board version of minimodem, to demodulate the tones. Once demodulated, packet data is passed by named pipe to stabilization and control process, as pilot commands.

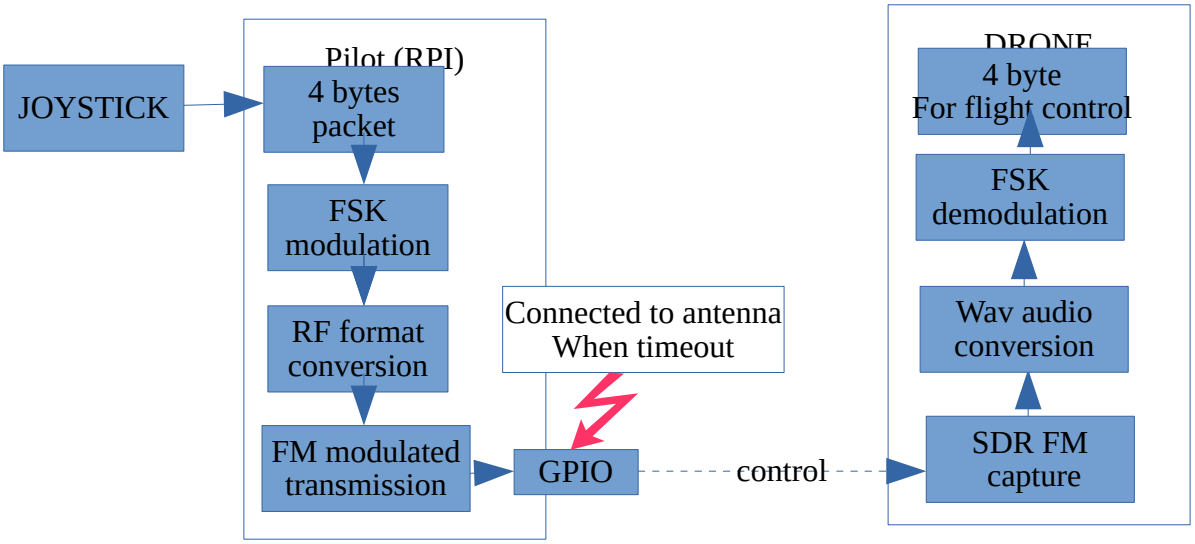


Diagram 3: SDR fall-back communication

Further upgrades

Further upgrades can be implemented, for example, transmitting a “standard” FM audio transmission, simulating being a radio station, (for example:https://www.youtube.com/watch?v=oHg5SJYRHA0) and transmitting the commands over PS, RT and TA (Traffic Announcement flag), as hidden side channel communication.

Raspberry-Pi GPIO harmonics: really a *problem*? Huh…

Transmitting RF over digital GPIO causes harmonics. For example, transmitting at 28 MHz will generate other signals at 84 MHz, 140Mhz, 196Mhz etc…

This is a **legal issue,** because it can interfere with reserved frequencies. **The point here is: if somebody flies a drone to enter in a no-drone area/installation, maybe transmitting in forbidden frequencies is the least of their problems**.

In this PoC, instead of using an external analog filter to remove the harmonics, we take advantage of them so that the drone is able to receive, not only in the main frequency, but it could use all the harmonics thanks yo its SDR receiver capabilities, selecting the best frequency automatically, without extra or fancy hardware.

If main frequency is also jammed, (and WiFi signal too, at the same time), the aircraft could have a last chance to receive commands by Round-Robin change of frequency, over the main frequency and its harmonics, inside the SDR frequency range.

NOTE: The point of this PoC is to demonstrate that for an attacker, legal issues are not a problem, making most common Anti-Drone systems vulnerable against a technique like this, because they are looking into expected frequencies, at expected protocols and signatures. A wide number of parameters (baudrate, modulation, encryption, frequency) can be arbitrarily changed (within limits) automatically, to make signature identification a pain.

Conclusions

- Ridiculously small size, weight and cost. This bring us to the next point:
- Hardware hacking and from scratch build, from flight control to hacking. **Not a “regular drone” with extra stuff put inside.**
- Side/hidden channels communication as central philosophy. No vendor or 3G/4G comms.
- Built from scratch, two-way communication, 802.11 beacon frame based protocol. Unexpected system by antidrone systems.
- Channel hopping over only one WiFi adapter on each side. (Lower cost, lower weight/power consumption on drone side)
- Fallback RF control communications, also unexpected by antidrone systems.