The sphinx simulator project

Nicolas CARRIER

April, 6, 2016

Presentation
- The problem
- Overview
- Features
- Limitations
- Architecture

Firmwared

The firmwares

Gazebo

Conclusion

The problem

Parrot:

- Builds wireless devices since 1994, bluetooth, WiFi...
- First commercial drone, the AR.Drone in 2010
- Linux based drones, custom distribution
- Located in the center of Paris

Problem:

- No accessible zone to fly drones
- How to develop something like flight plans ?
- How to test big drones ? Flying wings ?

Overview

Sphinx:

- French word for the death head hawk moth
- It's the one which flies, not the one which asks questions
- Means Simulator Project Hopefully Implemented for Next Xmas
- Originally an internal tool for development / automatic testing
- Team created in february 2015, first usable release in fall

Features 1/3

- Based on gazebo
- Mostly open-source (more on that later)
- Allows to test nearly original Parrot drone firmwares
- There'll always be a Next Xmas -> no deadline missed, never

Features 2/3

- Support for Bebop, Bebop 2, Rolling Spider, Disco
- Partial support for our top secret future drones
- Integration of WiFi, Bluetooth
- Uses exprtk, for easier models tweaking
- Support for multiple drones in the same simulation
- Seamless use for any controller above the SDK
  Even for autonomous tests in Jenkins CI servers

Features 3/3

Used for:

- Development of new features (flightplan), debug
- Fine-tune our drone control algorithms
- Autonomous regression tests
- Manual validation tests
- Discussions ongoing for a public release for app developers
- One day for the training of our FPV racing team ?
  Who knows...

Limitations

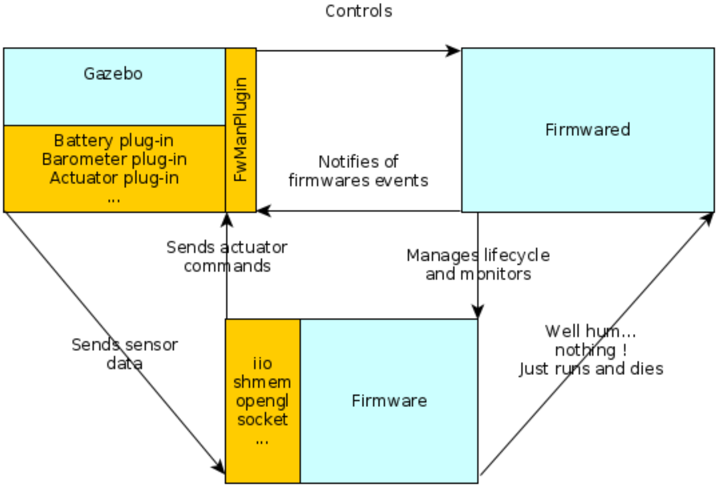
- Runs on Linux only
- Firmware adaptations are not trivial
  but things are getting simpler as we support more and more drones
- Not strictly the original firmware
- Not fully deterministic

Architecture

3 main parts

- Gazebo + some custom plug-ins
- A firmware, adapted and recompiled for x86 / amd64
- Firmwared (https://github.com/ncarrier/firmwared-manifest)

Architecture



Presentation

Firmwared

- Presentation
- Control
- OverlayFS
- Namespaces
- AppArmor

The firmwares

Gazebo

Conclusion

Presentation

System daemon responsible of spawning instances of drones firmwares.

- Firmware's programs run as if they were on a target root privileges...
- Implements containers "by hand" for a basic isolation
  - Chroot on overlayfs
  - Namespaces
  - Apparmor
- Firmwares ext2 filesystem images or a "final" directories (more on that later)
- Multiple instances can be spawned from a single firmware
- Open-source (see Parrot-Developer's github)

Control

- Driven by a named unix socket
- Uses libpomp (https://github.com/Parrot-Developers/libpomp)
  Marshalling API with an a-la-printf protocol
- Two clients:
  - shell: fdc based on pomp-cli, complete
  - C++ gazebo plug-in: fwman

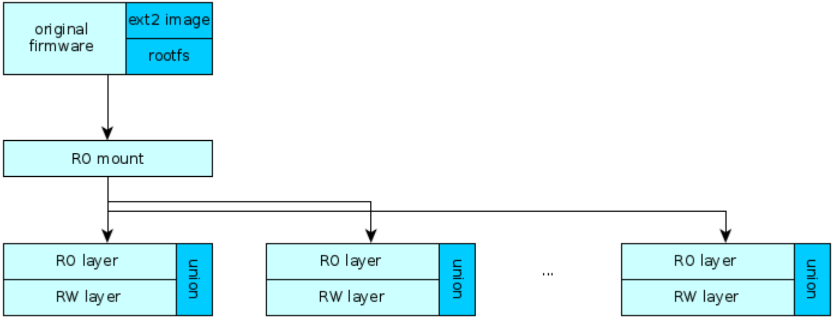
Example

```
$ fdc prepare firmwares /.../final
 ... the firmware garrulous_bellatrix has been created
$ fdc prepare instances garrulous_bellatrix
 ... the instance tremulous_nevena has been created
$ fdc start tremulous_nevena
 ...
```

OverlayFS

- An RW layer on top of an RO one (the rootfs)
- The RO layer is preserved and can be
  - the rootfs produced by the compilation (final dir)
  - an ext2 image, produced from the final by Alchemy
- the RW layer contains the diff on the file system, due to the execution: can be used for postmortem analysis

OverlayFS


original firmware
ext2 image
rootfs
RO mount
RO layer
RW layer
union
RO layer
RW layer
union
...
RO layer
RW layer
union

Namespaces

Unshare some global resources to protect their access from inside an instance

- Network namespace
  - No impact on the host's networking
  - An interface can be stolen for the instance (WiFi AP)
- Pid namespace
  - Renumbering of processes, starting from 1 in the namespace
  - Our init process (fork of Android's) runs as if on target
- Mount namespace
  - No access from the host to the instance's mount points
  - No access from the instance to the host's mount points
  - All mounts are automatically unmounted when the namespace is destroyed (read: when the instance's pid 1 dies)

AppArmor

- Some other global resources are still shared
- AppArmor allows to restrict their access, for example:
  - Capabilities (sys_time, hahem...)
  - Filesystem entities (/dev/mem, /proc/sysrq-trigger, hahem...)
- Uses a shell-like "glob" syntax
  => we managed to use it !

Presentation

Firmwared

The firmwares

Content

Gazebo

Conclusion

Content 1/2

Our build system is Alchemy

- Produces a "full" root fs: the staging (with symbols, headers...)
- Produces a stripped down rootfs for use on target: the final
- From this final, firmware images are produced, in plf, ext2, tar.gz...
- Firmwared can use directly a final directory or an ext2 image

Content 2/2

The firmware's variant used in the simulator has:

- Different hardware access code (IPC with gazebo)
- Different startup / initialization sequence (not the same "hardware")
- All the rest of the code is (or can be...) the same

Pros:

- No kernel level development
- Negligible impact on performances

Cons:

- Not the exact same code and code path as a real firmware
- Maintenance burden (2 variants of one firmware)

Other methods

- Full virtualization
  - High impact on performances
  - Drivers using IPCs with gazebo would be needed
- Qemu ”transparent” emulation: incomplete (netlink)
- Pseudo-hardware in the loop
  - Soft runs on hardware with IPCs via IP
  - No recompilation, same code with runtime adaptations, but lot of work and not the same code paths anyway
  - Latency and throughput problems (e.g. video) gzserver needs to run on target
- Hardware in the loop
  - It’s the holy grail: test the real firmware on a desktop
  - Forces to develop an hardware device per sensor / actuator
  - Not so hard for a gpio, but for a camera sensor ?
  - High cost for first version
  - My bet (and hope) is: we will come to it

Presentation

Firmwared

The firmwares

Gazebo

Overview

Plug-ins

Simulation description

Conclusion

Overview

- Gzserver simulates a world, including models with physical interactions
- Plug-ins system world, model and gui
- Gzclient optional client for real-time opengl visualization
- Open-source, C++
- XML description (sdf) of simulation and models
- We use a modified gazebo 7 with some added features, given back to the project

Plug-ins

Model plug-ins:

- One per sensor / actuator
  - IPC with gazebo: named socket
  - no IP overhead
  - abstract sockets blocked by netns
  - for now all of them use iio / libiio

World plug-ins:

- fwman: client controlling firmwared
- aerodynamic
- wind

Gui plug-in:

- shake: gui plugin for Disco's take off sequence

World files

- Normal sdf files
- References the fwman plugin
- Include no drone models, but reference to the firmwares used

Drone model files

- Sdf files + xinclude for factoring
  e.g. same body, different hulls
- Embedded inside the firmware
  this way, firmware and model are kept in sync
- References the actuators and sensors plug-ins

Simulation description

- Analysis of a world
- Analysis of a model

Presentation

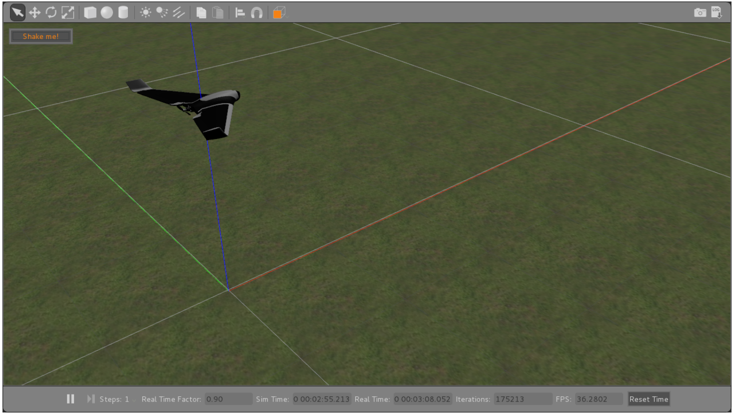
Firmwared

The firmwares

Gazebo

Conclusion

Demonstration



Ongoing and potential work

Ongoing:

- Video support
- Advanced vision features, follow me ...
- RC support

Potential:

- Visual feedback for leds
- HIL ? (finger crossed !)

Thank you for your attention