Telekyb3

A free and open architecture for

Aerial Robotics

Gianluca CORSINI, Ingénieur de Recherche CNRS - IRISA

Rennes – 24/05/2024

Outline

1. Introduction
2. Hardware
    a. Aerial platforms
    b. Electronics
3. Software
    a. The 3 pillars: git.openrobots.org, robotpkg, genom3
    b. Main components
    c. Examples of architectures
4. Examples of applications
5. Journée Drones 2024
6. Conclusions

1. Introduction

What is Telekyb3?

- a.k.a. **TK3** is an *“Open-source **collection** of **software** (and **hardware**) for Unmanned **Aerial Vehicles**”*

When and **where** Telekyb3 is born?

- Around **2015** at **LAAS** (almost 10y ago!), with few users

Who is using it?

Institution		LAAS (Toulouse)
Software		X
Hardware		X
Users	**Active**	≥ 5
	Over time	≈50

1. Introduction

What is Telekyb3?

- a.k.a. **TK3** is an *“Open-source **collection** of **software** (and **hardware**) for Unmanned **Aerial Vehicles**”*

When and **where** Telekyb3 is born?

- Around **2015** at **LAAS** (almost 10y ago!), with few users

Who is using it?

Institution		LAAS (Toulouse)	IRISA (Rennes)	University of Twente (NL)	Saxion (NL)	University of Catania (IT)
Software		X	X	X	X	X
Hardware		X	X	X	X	
Users	**Active**	≥ 5	≥ 10	≥ 5	1	1
	Over time	≈50	≈20	≈10		

1. Introduction

What is Telekyb3?

- a.k.a. **TK3** is an *"Open-source **collection** of **software** (and **hardware**) for Unmanned **Aerial Vehicles**"*

When and **wh**

- Around **201**

Who is using i

Small, but growing community!
≥ 20 users today!
≈ 80 over the years

Institution		LAAS (Toulouse)	IRISA (Rennes)	University of Twente (NL)	Saxion (NL)	University of Catania (IT)
Software		X	X	X	X	X
Hardware		X	X	X	X	
Users	**Active**	≥ 5	≥ 10	≥ 5	1	1
	Over time	≈50	≈20	≈10		

1. Introduction

What is Telekyb3?

- a.k.a. **TK3** is an *"Open-source **collection** of **software** (and **hardware**) for Unmanned **Aerial Vehicles**"*

When and **where** Telekyb3 is born?

- Around **2015** at **LAAS** (almost 10y ago!), with few users

Who is maintaining it?

Institution	LAAS (Toulouse)	IRISA (Rennes)	University of Twente (NL)	Saxion (NL)	University of Catania (IT)
Software	X	X			
Hardware	X	X	X		
Maintainers	3	2	≈1	–	–

1. Introduction

Why Telekyb3?

1. **Modularity, Reusability and Interchangeability**
    - Several components: each one implementing one (or more) functionality(ies)
    - **Interface-based** design: components use interfaces to communicate

1. Introduction

Why Telekyb3?

1. **Modularity, Reusability and Interchangeability**
   - Several components: each one implementing one (or more) functionality(ies)
   - **Interface-based** design: components use interfaces to communicate
2. **Formal description language**
   - Allows software validation and verification, and well-defined behavior

1. Introduction

Why Telekyb3?

1. **Modularity, Reusability and Interchangeability**
   - Several components: each one implementing one (or more) functionality(ies)
   - **Interface-based** design: components use interfaces to communicate
2. **Formal description language**
   - Allows software validation and verification, and well-defined behavior
3. **Middleware abstraction**
   - Component description is unaware of middleware implementation (*templates*)

1. Introduction

Why Telekyb3?

1. **Modularity, Reusability and Interchangeability**
   - Several components: each one implementing one (or more) functionality(ies)
   - **Interface-based** design: components use interfaces to communicate
2. **Formal description language**
   - Allows software validation and verification, and well-defined behavior
3. **Middleware abstraction**
   - Component description is unaware of middleware implementation (*templates*)
4. **Low-level access**
   - Full control on any low-level component (either hardware or software)

1. Introduction

Why Telekyb3?

1. **Modularity, Reusability and Interchangeability**
    - Several components: each one implementing one (or more) functionality(ies)
    - **Interface-based** design: components use interfaces to communicate
2. **Formal description language**
    - Allows software validation and verification, and well-defined behavior
3. **Middleware abstraction**
    - Component description is unaware of middleware implementation (*templates*)
4. **Low-level access**
    - Full control on any low-level component (either hardware or software)
5. **Variety** of **aerial robots**
    - **Single** and **multi-robot** systems with **different rotor configurations**

1. Introduction

Why Telekyb3?

1. **Modularity, Reusability and Interchangeability**
    - Several components: each one implementing one (or more) functionality(ies)
    - **Interface-based** design: components use interfaces to communicate
2. **Formal description language**
    - Allows software validation and verification, and well-defined behavior
3. **Middleware abstraction**
    - Component description is unaware of middleware implementation (*templates*)
4. **Low-level access**
    - Full control on any low-level component (either hardware or software)
5. **Variety** of **aerial robots**
    - **Single** and **multi-robot** systems with **different rotor configurations**
6. **Variety** of **applications**

2. Hardware

Aerial Robots: under-actuated quad-rotor (a.k.a. *QR*)

Quad-rotor platform at LAAS (top) and at IRISA (bottom).

- Features:
  - 4 collinear motor-propeller pairs
  - ≈ 1kg take-off mass
- Mechanical components:
  - mainly from Mikrokopter store (still purchasable)
  - 3D printing
- Applications:
  - Indoor and outdoor navigation
  - Vision-based control
  - Human-robot interaction (handover)
- Institutions: LAAS, IRISA

2. Hardware

Aerial Robots: fully-actuated hexa-rotor (a.k.a. *FiberTHex*)

Hexa-rotor platform at LAAS (top) and at IRISA (bottom).

- Features:
  - 6 fixedly-titled motor-propeller pairs
  - ≈ 2-3kg take-off mass
- Mechanical parts:
  - Several suppliers (e.g. RS, Mikrokopter, Robotshop)
  - 3D printing
- Applications:
  - Indoor and outdoor navigation
  - Vision-based control
  - Physical interaction with the environment (or humans)
- Institutions: LAAS, IRISA, UT, SAXION

2. Hardware

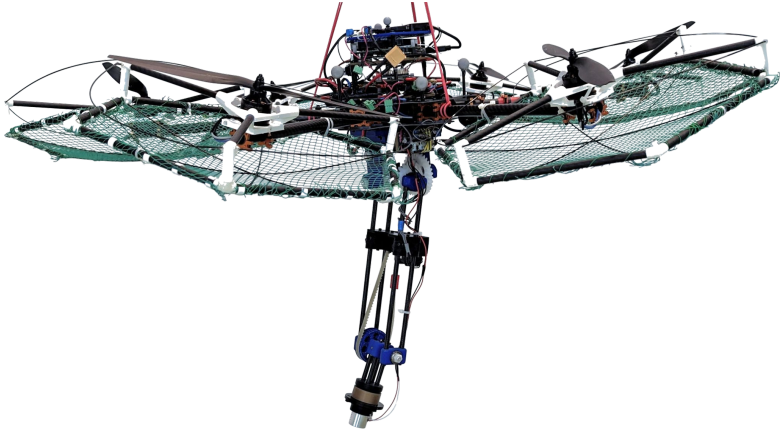
Robotic arms: 3-DoF servo-powered

Mechanical design of the 3-DoF arm realised at LAAS.

- Features:
  - 2-DoF (shoulder) + 1-DoF (elbow)
  - 1:1 weight-2-lift-force ratio → ≈ 1kg lifting mass
  - Dynamixel motors
- Mechanical parts:
  - Several suppliers
  - 3D printing
- Applications:
  - Physical interaction with the environment (or humans)
- Institutions: LAAS, UT

2. Hardware

Aerial Manipulators: FiberTHex + 3-DoF arm (a.k.a. *FiberTHam*)

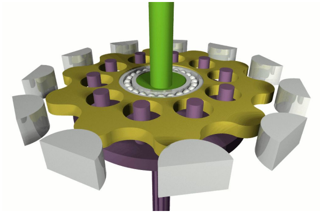


The FiberTHam built at LAAS.

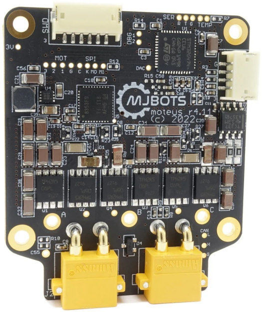
- Features:
  - 9-DoF
  - Propeller guards
- Applications:
  - Physical interaction with the environment (or humans)
- Institutions: LAAS, UT

2. Hardware

(In development) **Other robotic arms**: 6-DoF brushless-powered (a.k.a. Micrurus)



A cycloidal gear.
Courtesy of Wikipedia



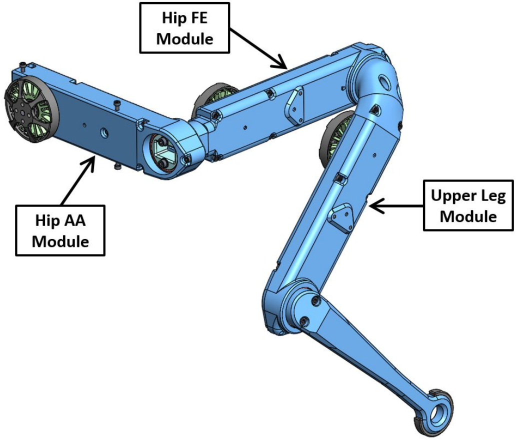
MJBot Moteus ESCs.

- Features:
  - 6x 1-DoF cycloidal gear
  - MJbots Moteus ESCs + Brushless motors (T-motor)
- Mechanical parts:
  - Several suppliers
  - 3D printing
- Applications:
  - Physical interaction with the environment (or humans)
- Institutions: LAAS

2. Hardware

(In development) **Other robotic arms**: 3-DoF arm based on Solo-12's leg (IRISA)

- Open Dynamic Robot Initiative



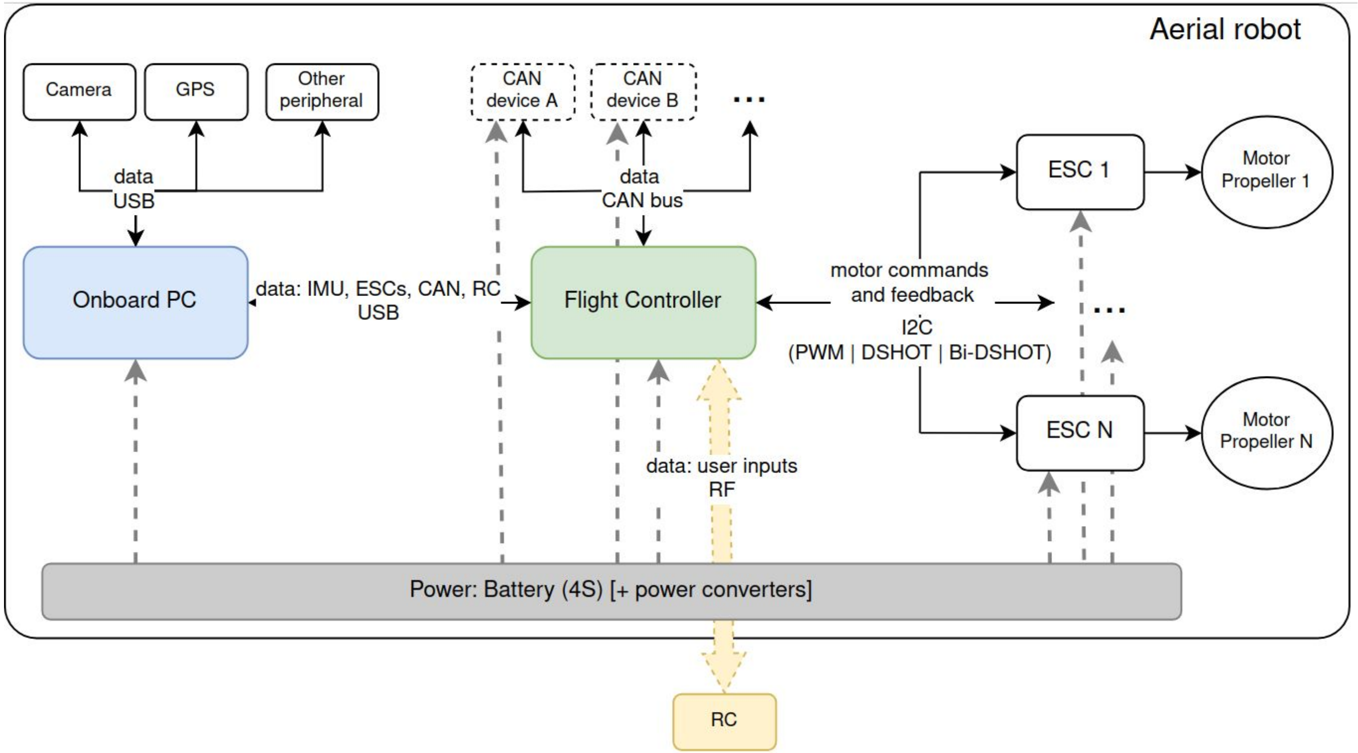
Mechanical design of the Solo-12's 3-DoF leg.
Courtesy of Open Dynamic Robot Initiative.

Torque-controllable aerial manipulator

J. Marti-Saumell et al. "Borinot: an open thrust-torque-controlled robot for research on agile aerial-contact motion." ArXiv abs/2307.14686 (2023).

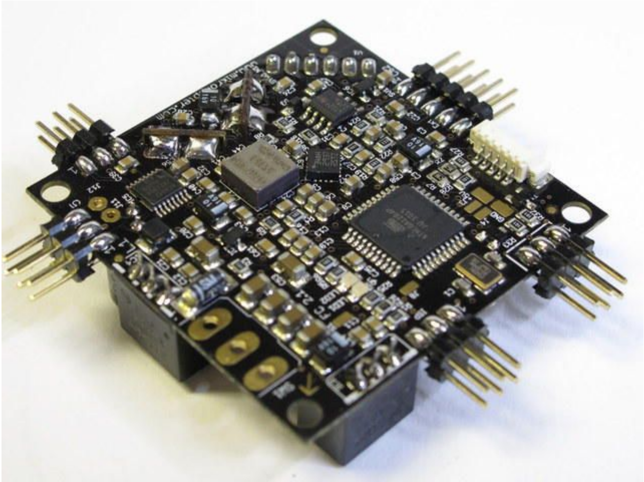
2. Hardware

Electronics: Overview



2. Hardware

Electronics: Mikrokopter Flight Controller (board v2.1 or v2.5)

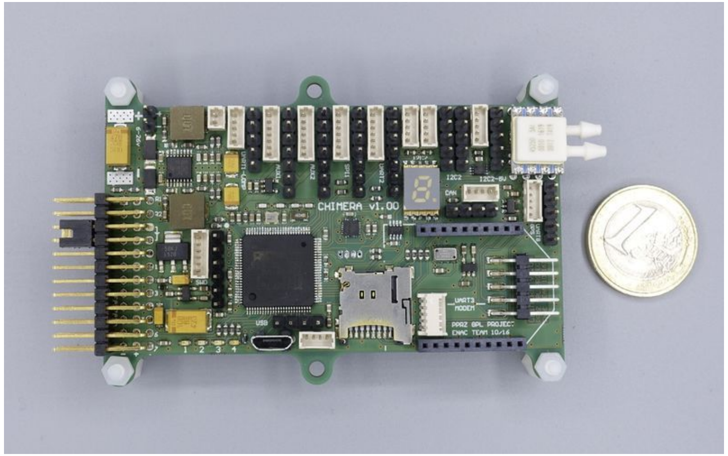


Mikrokopter Flight Controller board v2.1.
https://gallery3.mikrokopter.de/tech/FC21-best_ckt1

- Manufacturer: Mikrokopter
- Functionalities:
    - AVR **8-bit** µC
    - **6-DoF IMU**: accelerometer, gyroscope
    - **1 kHz telemetry** (motor ~ 100Hz)
    - **Serial-2-usb** connection to onboard **PC**
    - Custom firmware: tk3-mikrokopter/mkfl
    - TK3 communication protocol
        - 2µs resolution for RPM command
    - **I2C bus** for **ESCs**
- Status: discontinued
- Local stocks: LAAS, IRISA

2. Hardware

Electronics: Paparazzi Chimera Flight Controller

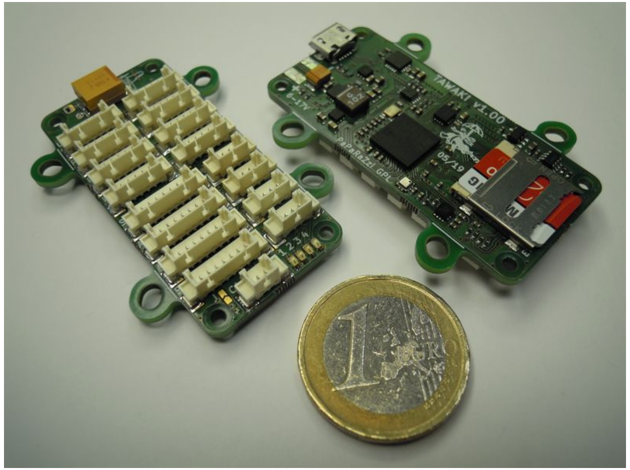


Paparazzi Chimera Flight Controller board v1.00.
https://wiki.paparazziuav.org/wiki/Chimera/v1.00

- Developer: ENAC
- Functionalities:
  - STM32F7 **32bit** ARM µC
  - **9-DoF IMU**: accelerometer, gyroscope, **magnetometer**
  - **1 kHz telemetry** (motor ~ 100Hz)
  - **Serial-2-usb** connection to onboard **PC**
  - Custom firmware: tk3-paparazzi
  - TK3 communication protocol
    - 2µs resolution for RPM command
  - **Can** devices, **I2C**, **SPI**, **UART**s, **radio controller**
- Status: IMU chip is discontinued!
- Local stocks: LAAS, IRISA, UT

2. Hardware

(Future) **Electronics**: Paparazzi Tawaki Flight Controller



Paparazzi Tawaki Flight Controller board v1.10.
https://wiki.paparazziuav.org/wiki/Tawaki/v1.10

- Developer: ENAC
- Functionalities:
  - **Same as Paparazzi Chimera Flight Controller**
  - **Smaller** form factor
  - Support for **can-fd** devices
- Status: requires **minimal firmware adaptation**
- Local stocks: –

2. Hardware

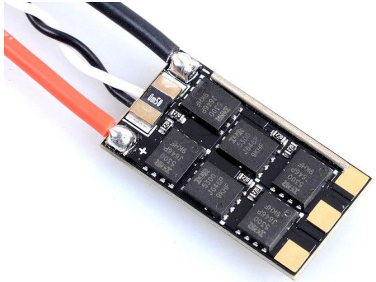
Electronics: Mikrokopter ESC

Mikrokopter BL-Ctrl v2.0 ESC.
https://gallery3.mikrokopter.de/tech/FC21-best_ckt1

- Manufacturer: Mikrokopter
- Functionalities:
  - (Multi-slave) **I2C** connection to **FC**
  - TK3 communication protocol
  - **Closed-loop speed control**
    - Up to 1kHz (on the ESC)
- Status: discontinued
- Local stock: (good amount) in LAAS, IRISA, UT

2. Hardware

Electronics: Commercial (Hobbyist) ESCs

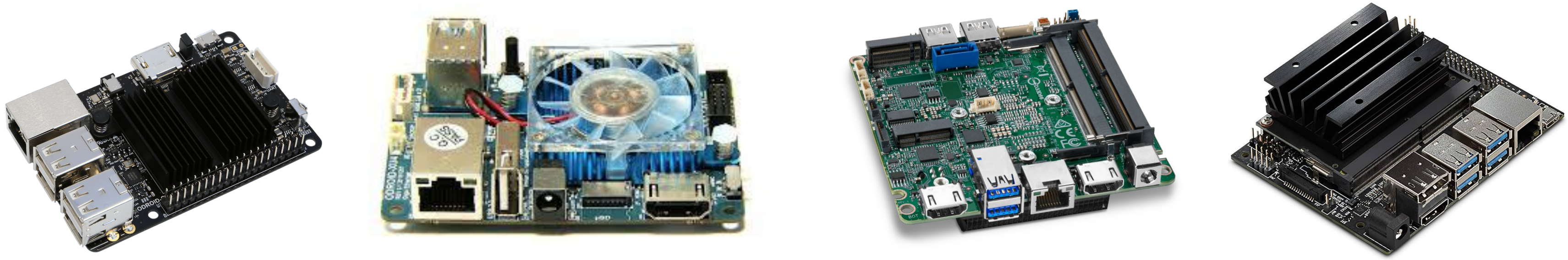


Aikon AK32 ESC.
https://www.drone-fpv-racer.com/aikon-ak32-35a-6s-esc-1969.html

- Manufacturer: any
- Requirements:
  - **PWM** or **DSHOT** communication protocol
    - up to 900kHz (i.e. up to DSHOT900)
    - **open-loop** speed control
  - **Bidirectional-DSHOT** (shortly Bi-DSHOT)
    - **BLHeli-32** firmware
    - **closed-loop** propeller speed control (from FC)
- Status: in testing at LAAS and UT
- Local stock: (good amount) in LAAS, IRISA, UT

2. Hardware

Electronics: Onboard PCs



From left to right: Odroid C2, Odroid XU4, Intel NUC, Jetson Nano.

Requirements:

- Run a **linux-like operating system** (e.g. Ubuntu)
- **1 USB port** →Flight controller
- Optionally (and conveniently) with WiFi →for remote interaction from another machine

2. Hardware

Electronics: Sensors and peripherals

Flight Controller:

- Onboard
  - IMU [+ magnetometer]
- CAN
  - FT sensors: IIT FT45 (discontinued)

Onboard PC:

- USB
  - GPS RTK
  - Cameras: realsense D435 and T265
  - FT sensors: Botasys (MiniOne and Medusa under testing)
  - Dynamixel motors
  - Arduino boards

IIT FT45

Drotek Sirius RTK GNSS Rover

Botasys MiniOne

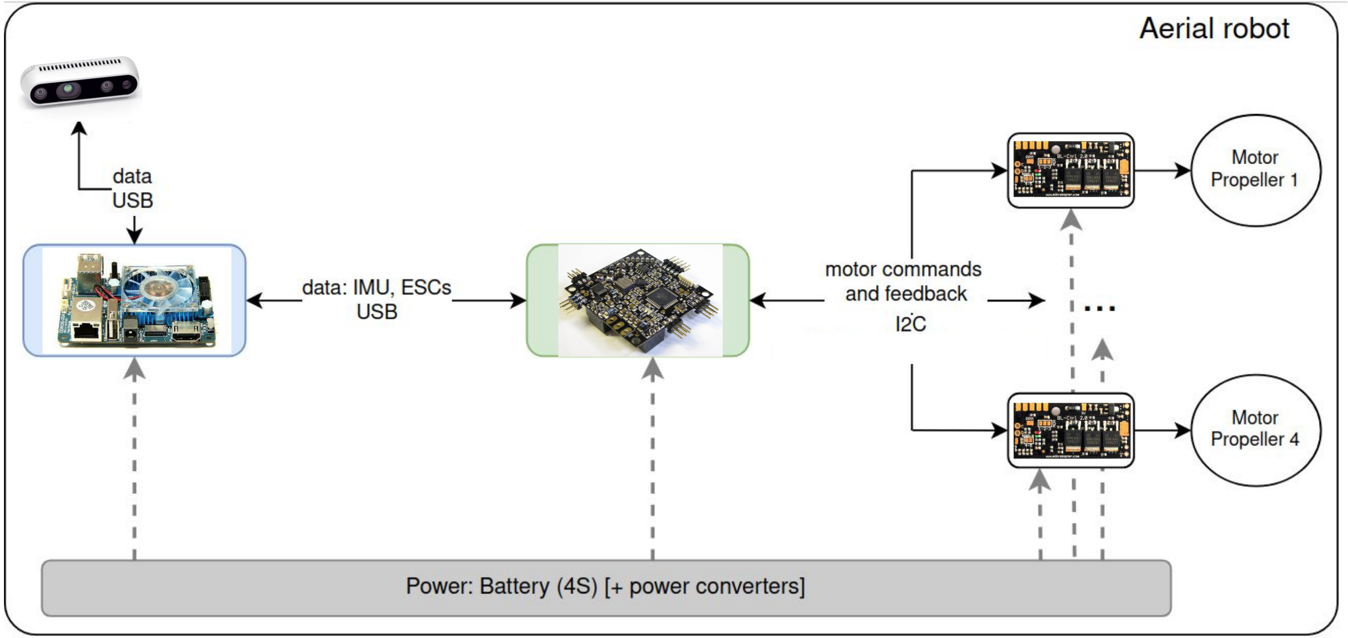
Dynamixel MX-28

Intel Realsense D435 (left) and T265 (left)

Taranis
Radio Controller

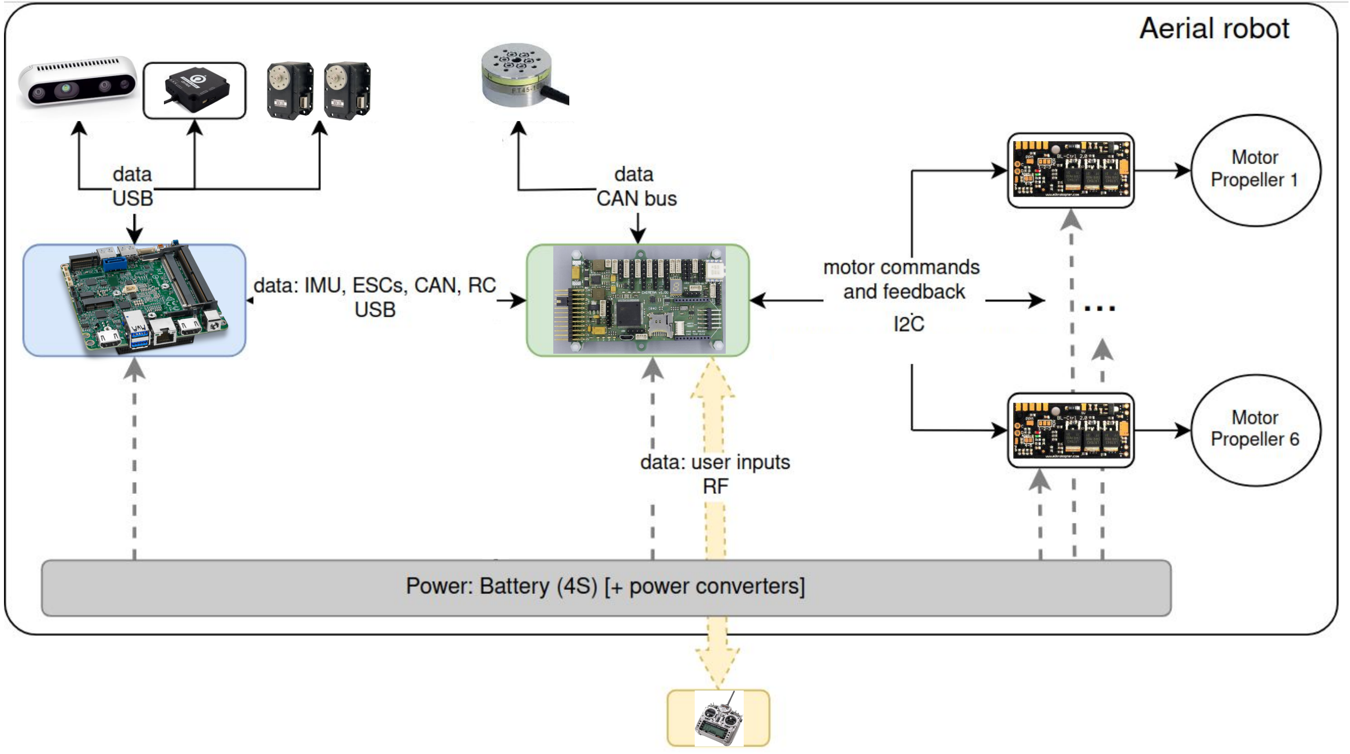
2. Hardware

Example of hardware architecture of a quad-rotor



2. Hardware

Example of hardware architecture of an hexa-rotor



3. Software

The **3 “pillars”** of the Telekyb3 software architecture:

3. Software

The **3 “pillars”** of the Telekyb3 software architecture:

- Part of the git.openrobots.org project
  - *“Robotics Open Source Software developed at CNRS/LAAS”*

3. Software

The **3 “pillars”** of the Telekyb3 software architecture:

- Part of the git.openrobots.org project
  - *“Robotics Open Source Software developed at CNRS/LAAS”*
- Delivered by means of robotpkg
  - Compilation from source on host CPU
  - Provides a PPA and binary packages for debian-based systems (.deb)
  - Stable versions of the software and regular releases (typically >1 per year)

3. Software

The **3 “pillars”** of the Telekyb3 software architecture:

- Part of the git.openrobots.org project
  - *“Robotics Open Source Software developed at CNRS/LAAS”*
- Delivered by means of robotpkg
  - Compilation from source on host CPU
  - Provides a PPA and binary packages for debian-based systems (.deb)
  - Stable versions of the software and regular releases (typically >1 per year)
- Heavily based on Genom3
  - “[...] a tool to design real-time software architectures”

3. Software

Genom3

- Tool designed to write **independent** and **reusable components**
  - **Component** = “[...] a server that provides a number of services and communicates through data ports with other components in the system”
  - **component description files** (data types, services, ports)

3. Software

Genom3

- Tool designed to write **independent** and **reusable components**
  - **Component** = “[...] a server that provides a number of services and communicates through data ports with other components in the system”
  - **component description files** (data types, services, ports)
- **Middleware abstraction**
  - Templates to select target middleware (e.g., pocoLibs, ROS, Yarp, …)

3. Software

Genom3

- Tool designed to write **independent** and **reusable components**
  - **Component** = “[...] a server that provides a number of services and communicates through data ports with other components in the system”
  - **component description files** (data types, services, ports)
- **Middleware abstraction**
  - Templates to select target middleware (e.g., pocoLibs, ROS, Yarp, …)
- Source code automatically generated
  - Target middleware
  - Main component routines

3. Software

Genom3

- Tool designed to write **independent** and **reusable components**
  - **Component** = “[...] a server that provides a number of services and communicates through data ports with other components in the system”
  - **component description files** (data types, services, ports)
- **Middleware abstraction**
  - Templates to select target middleware (e.g., pocoLibs, ROS, Yarp, …)
- Source code automatically generated
  - Target middleware
  - Main component routines
- Allow interfacing with **external clients** (user applications)
  - **Genomix**: abstraction interface
  - Scripting in different programming languages: TCL, Python, MATLAB/Simulink

3. Software

Genom3

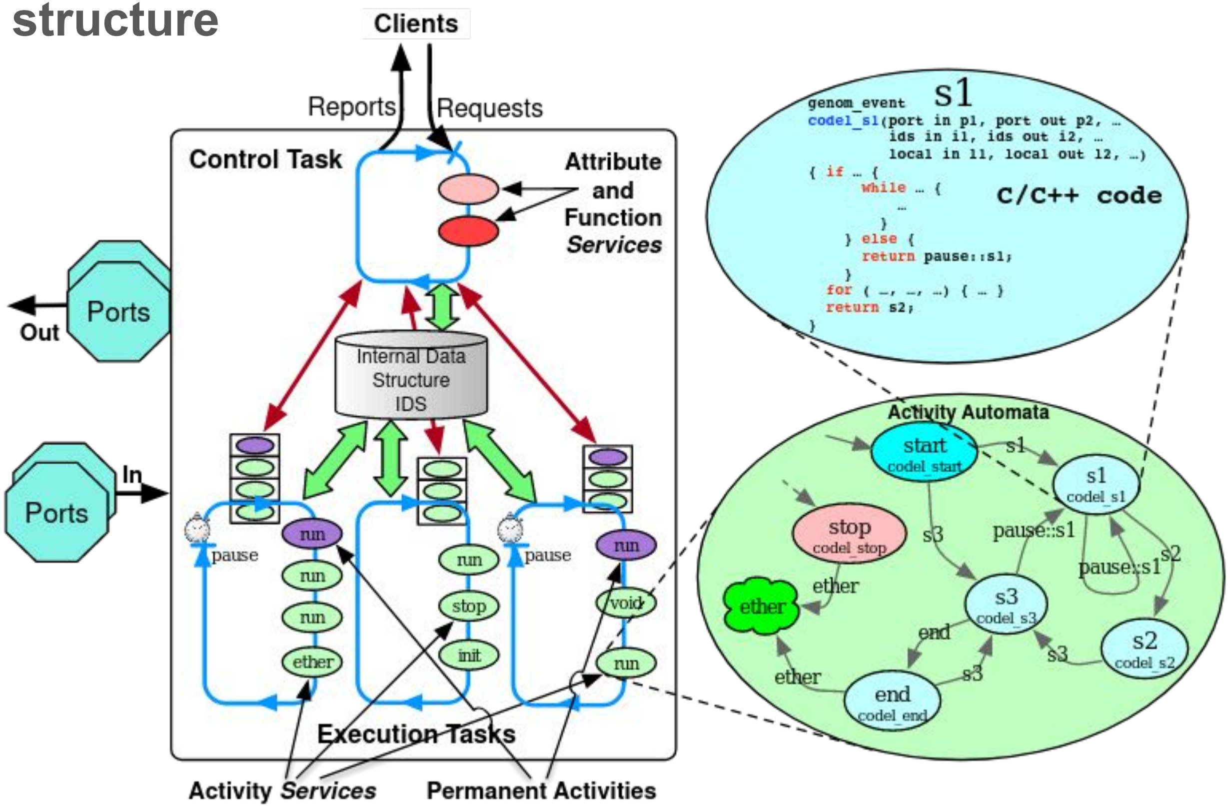
The user needs to:

1. Write the **component description file** (*.gen*)
2. Run the **skeleton generation engine** (i.e., ‘$ *genom3 skeleton component.gen’*)
3. Write implementation of services as elementary bits of code (a.k.a. *codels*)
4. **Build** the component for the desired middleware
   - The configuration (*configure*) and compilation scripts (*Makefiles*) are automatically generated!
   - Based on the *autotools* toolchain
5. **Run** the component and the desired middleware
6. Use ***genomix*** within scripts for interfacing with the components, e.g.:
   - Read output ports
   - Call services (set/get parameters, control component execution, …)

3. Software

A Genom3 **component structure**

1. IDS
2. Ports
3. Tasks
4. Services
5. Codels



3. Software

Example of component description file (*.gen*): [demo-genom3]

```
#include "demoStruct.idl"

/* ---- component declaration ---- */

component demo {
  version       "1.3";
  email         "openrobots@laas.fr";
  lang          "c";
  require       "genom3 >= 2.99.26";

  /* ---- Data structures and IDS ---- */

  ids {
    demo::state state;          /* Current state */
    demo::speed speedRef;       /* Speed reference */
    double      posRef;
  };
```

demoStruct.idl:

```
#ifndef IDL_DEMO_STRUCT
#define IDL_DEMO_STRUCT

module demo {

  const unsigned long task_period = 400;
  const double millisecond = 0.001;

  struct state {
    double position;  /* current position (m) */
    double speed;     /* current speed (m/s) */
  };

  enum speed {
    SLOW,
    FAST
  };
};

#endif /* IDL_DEMO_STRUCT */
```

3. Software

Example of component description file (*.gen*): [demo-genom3]

```
/* ---- Data structures and IDS ---- */

ids {
  demo::state state;          /* Current state */
  demo::speed speedRef;       /* Speed reference */
  double      posRef;
};
```

3. Software

Example of component description file (*.gen*): demo-genom3

```
/* ---- Posters declarations ---- */

port out demo::state Mobile;


exception TOO_FAR_AWAY {double overshoot;};

exception INVALID_SPEED;
```

3. Software

Example of component description file (*.gen*): [demo-genom3]

```
/* ---- Execution task declaration ---- */

task motion {
  period      demo::task_period ms;
  priority    100;
  stack       4000;
  codel <start>       InitDemoSDI(out ::ids, port out Mobile) yield ether;
};
```

codel files:

```
/* --- Task motion ---------------------------------------------------- */


/** Codel InitDemoSDI of task motion.
 *
 * Triggered by demo_start.
 * Yields to demo_ether.
 */
genom_event
InitDemoSDI(demo_ids *ids, const demo_Mobile *Mobile,
            const genom_context self)
{
```

3. Software

Example of component description file (*.gen*): [demo-genom3]

```
/* ---- Services declarations ---- */

attribute SetSpeed(in speedRef = demo::SLOW   :"Mobile speed")
{
  doc         "To change speed";
  validate    controlSpeed (local in speedRef);
  throw       INVALID_SPEED;
};

attribute GetSpeed(out speedRef =     :"Mobile speed")
{
  doc         "To get current speed value";
};

function Stop()
{
  doc         "Stops motion and interrupts all motion requests";
  interrupts  MoveDistance, GotoPosition;
};
```

```
ids {
  demo::state state;
  demo::speed speedRef;
  double      posRef;
};
```

codel files:

```
/* --- Attribute SetSpeed ---------------------------------------------- */

/** Validation codel controlSpeed of attribute SetSpeed.
 *
 * Returns genom_ok.
 * Throws demo_INVALID_SPEED.
 */
genom_event
controlSpeed(demo_speed speedRef, const genom_context self)
{
```

3. Software

Example of component description file (*.gen*): demo-genom3

```
activity GotoPosition (in double posRef = 0   :"Goto position in m")
{
  doc         "Move to the given position";

  validate    controlPosition (local in posRef);

  codel <start> gpStartEngine() yield exec, ether;
  codel <exec> gpGotoPosition(local in posRef, inout ::ids,
                               port out Mobile)
    yield pause::exec, end;
  codel <end, stop> gpStopEngine() yield ether;

  interrupts  MoveDistance, GotoPosition;
  task        motion;
  throw       TOO_FAR_AWAY;
};
```

codel files:

```
/* --- Activity GotoPosition ------------------------------------

/** Validation codel controlPosition of activity GotoPosition.
 *
 * Returns genom_ok.
 * Throws demo_TOO_FAR_AWAY.
 */
genom_event
controlPosition(double posRef, const genom_context self)
{
```

```
/* --- Activity GotoPosition ------------------------------

/** Codel gpStartEngine of activity GotoPosition.
 *
 * Triggered by demo_start.
 * Yields to demo_exec, demo_ether.
 * Throws demo_TOO_FAR_AWAY.
 */
genom_event
gpStartEngine(const genom_context self)
{
```

3. Software

Example of component description file (*.gen*): demo-genom3

```
activity GotoPosition (in double posRef = 0   :"Goto position in m")
{
  doc          "Move to the given position";

  validate     controlPosition (local in posRef);

  codel <start> gpStartEngine() yield exec, ether;
  codel <exec> gpGotoPosition(local in posRef, inout ::ids,
                              port out Mobile)
    yield pause::exec, end;
  codel <end, stop> gpStopEngine() yield ether;

  interrupts   MoveDistance, GotoPosition;
  task         motion;
  throw        TOO_FAR_AWAY;
};
```

codel files:

```
/** Codel gpGotoPosition of activity GotoPosition.
 *
 * Triggered by demo_exec.
 * Yields to demo_pause_exec, demo_end.
 * Throws demo_TOO_FAR_AWAY.
 */
genom_event
gpGotoPosition(double posRef, demo_ids *ids, const demo_Mobile *Mobile,
               const genom_context self)
{
```

```
/** Codel gpStopEngine of activity GotoPosition.
 *
 * Triggered by demo_end, demo_stop.
 * Yields to demo_ether.
 * Throws demo_TOO_FAR_AWAY.
 */
genom_event
gpStopEngine(const genom_context self)
{
```

3. Software

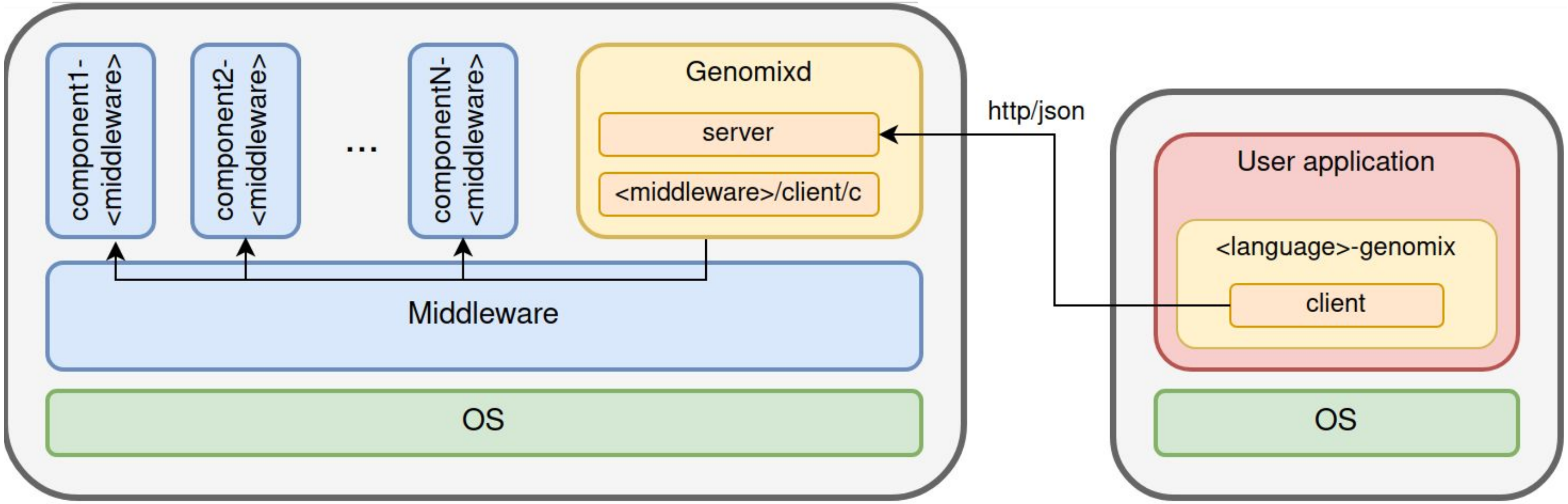
Example of component description file (*.gen*): demo-genom3

```
activity MoveDistance(in double distRef = 0   :"Distance in m")
{
  doc          "Move of the given distance";
  validate     controlDistance(in distRef, in state.position);

  codel <start> mdStartEngine(in distRef, in state.position, out posRef)
    yield exec, ether;
  codel <exec> mdGotoPosition(in speedRef, in posRef, inout state,
                             port out Mobile)
    yield pause::exec, end;
  codel <end, stop> mdStopEngine() yield ether;

  interrupts   MoveDistance, GotoPosition;
  task         motion;
  throw        TOO_FAR_AWAY;
};
```

3. Software

Genomix: daemon **server** (*genomixd*) + **client** (*<language>-genomix*)

where *<language>* = *[tcl* | *python* | *matlab]* , e.g. *python-genomix*



where <middleware> = [ROS | pocoLibs | YARP …]

3. Software

Main TK3 Genom3 components

- Control:
  - *nhfc-genom3*: cascade PID for under-actuated aerial vehicles
  - *uavpos/uavatt-genom3*: positional and attitude controllers for fully-actuated aerial vehicles
  - *phynt-genom3*: admittance filter + wrench observer
- Estimation:
  - *pom-genom3*: Uscented Kalman Filtering
- Motion:
  - *maneuver-genom3*: kinematic trajectory generator
- Robot interfaces:
  - *rotorcraft-genom3*: interface with low-level hardware (flight-controller)
- Sensors:
  - *optitrack/qualisys/vicon-genom3*: interface with Motion Capture Systems
  - *realsense-genom3*: interface with Intel Realsense cameras
  - *gps-genom3*: interface with GPS modules
  - *dynamixel-genom3*: interface with Dynamixel motors

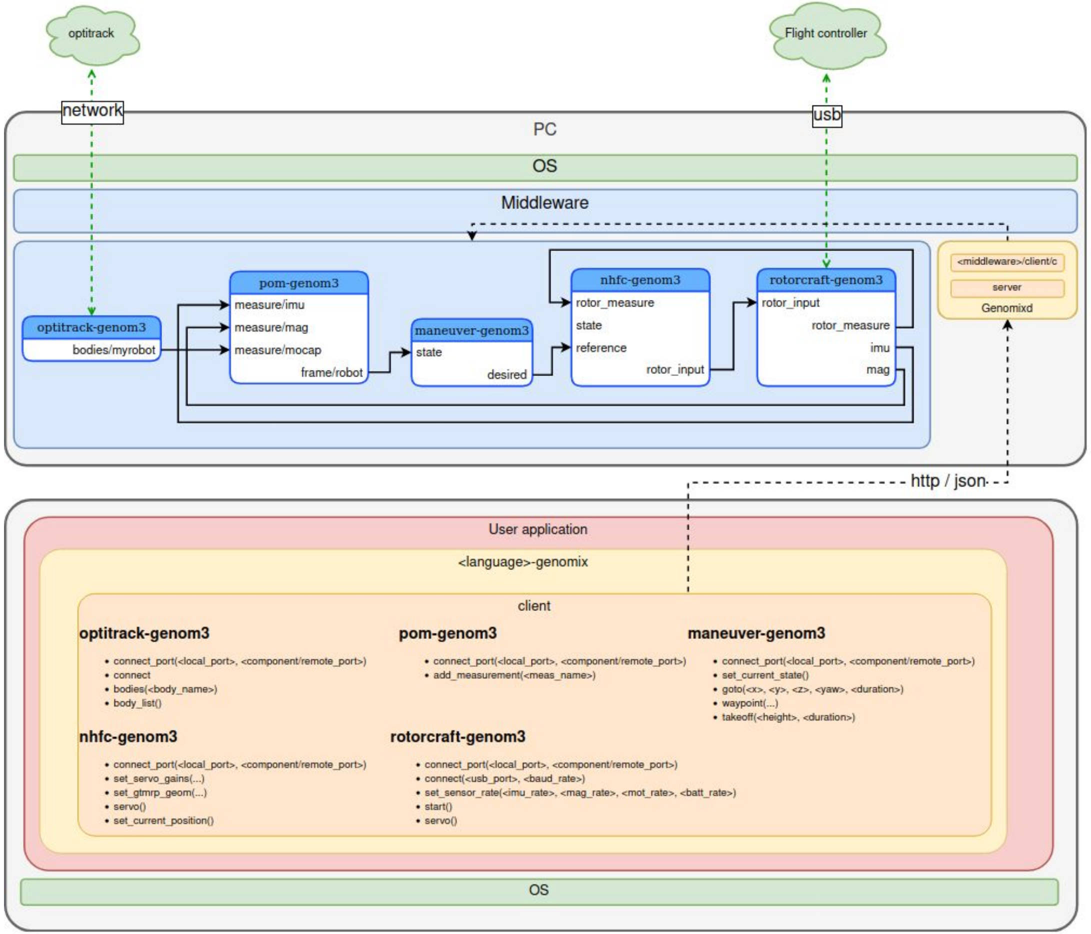
3. Software

Example of software

architecture for a

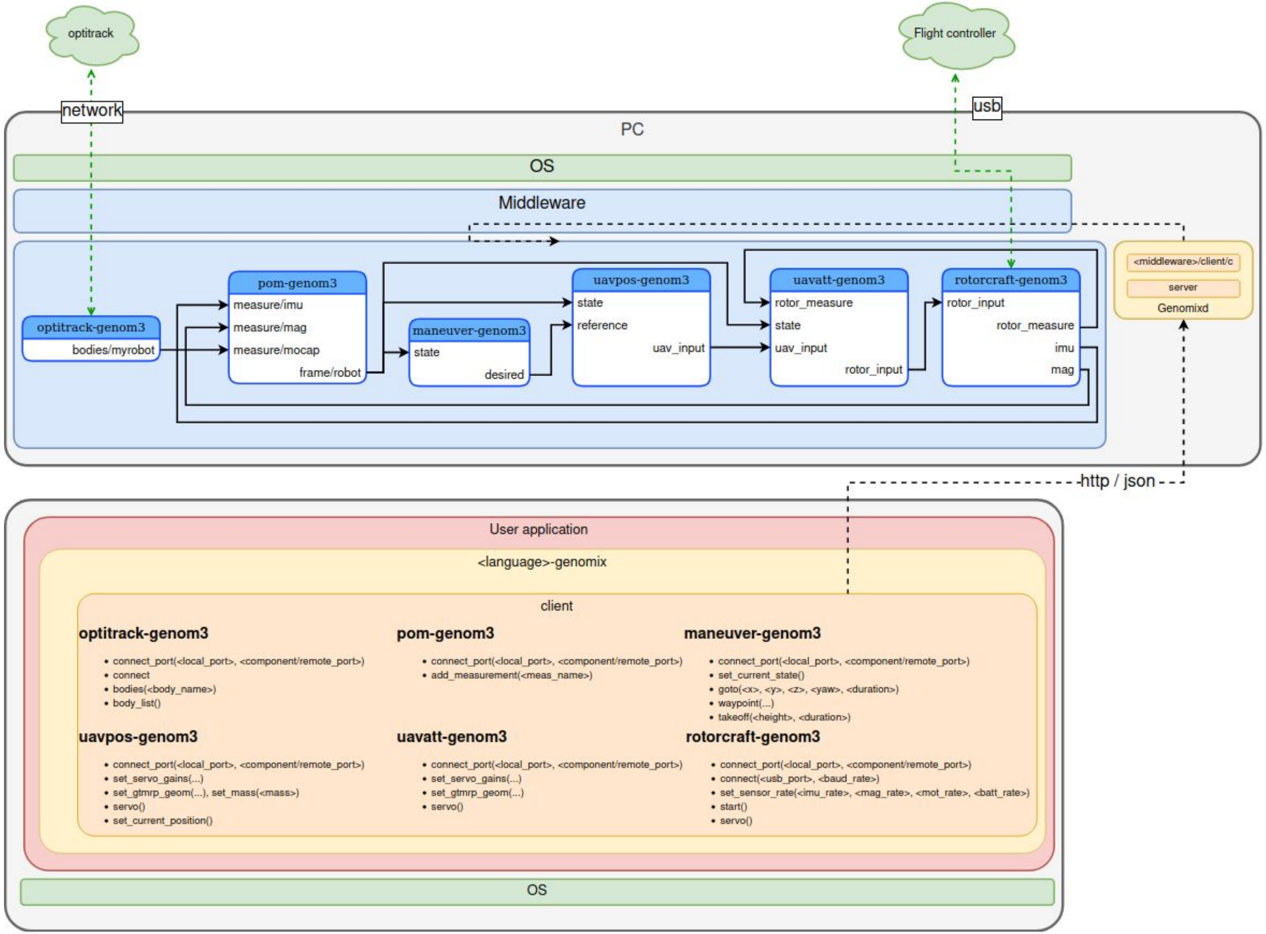
quad-rotor

(in real experiments!)



3. Software

Example of software architecture for an **hexa-rotor** (in real experiments!)



3. Software

What about **simulation**?

- Main **simulator**: Gazebo
- Several **plugins**
  - *mrsim-gazebo*: simulates a generic multi-rotor aerial vehicle
    - also TK3 **low-level hardware**, i.e., FC, ESCs, motor dynamics
  - *optitrack-gazebo*: simulates an Optitrack motion capture system
    - natnet stream (optitrack protocol)
  - *dxsim-gazebo*: simulates a chain of Dynamixel motors
    - RAM, EPPROM, communication protocol
- Other **Genom3 components** for simulation
  - *gazebocam-genom3*: streams a camera sensor of Gazebo
  - *gazeboft-genom3*: steams wrench from a force-torque sensor of Gazebo
- **Seamless simulations-2-experiments** transition
  - Usage of **interfaces** allows interchanging real and simulated hardware (or other components)

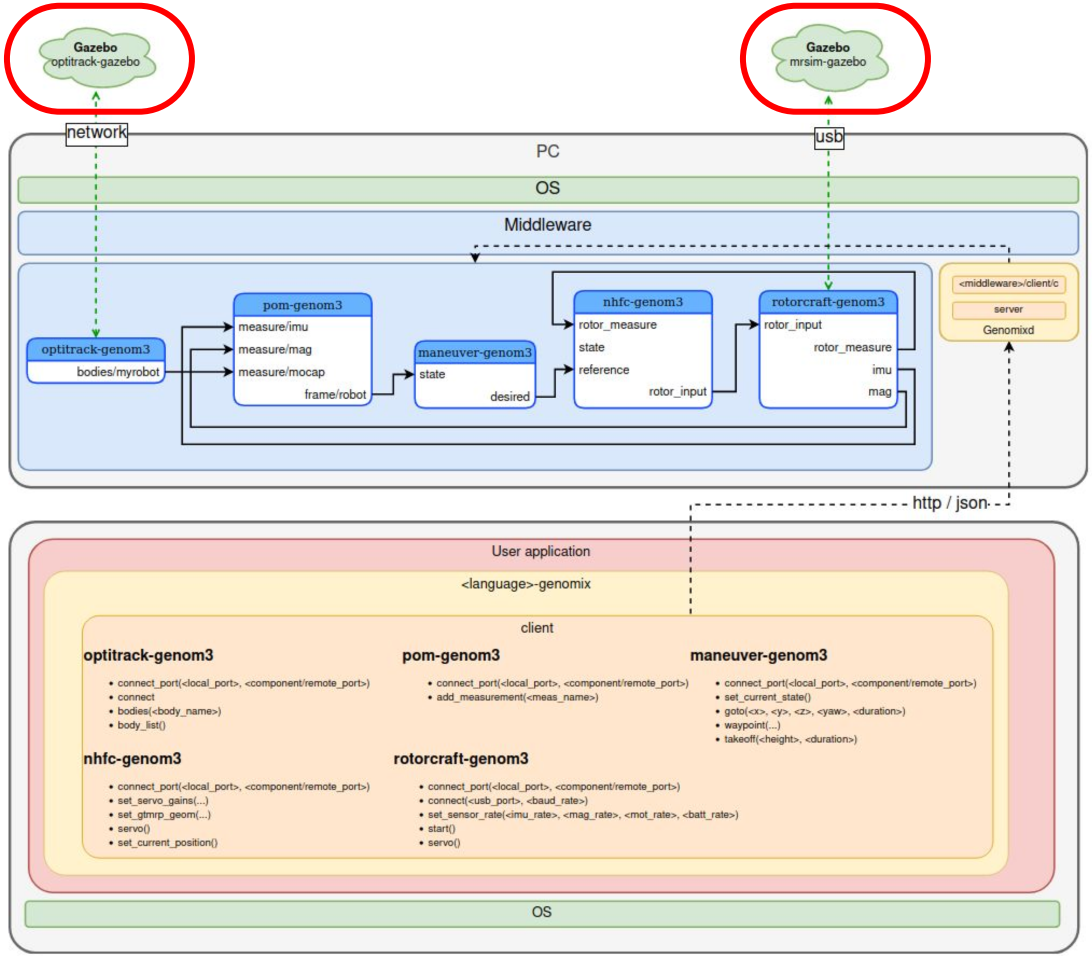
3. Software

Example of software

architecture for a

quad-rotor

in **simulation**



NB: PC = localhost!

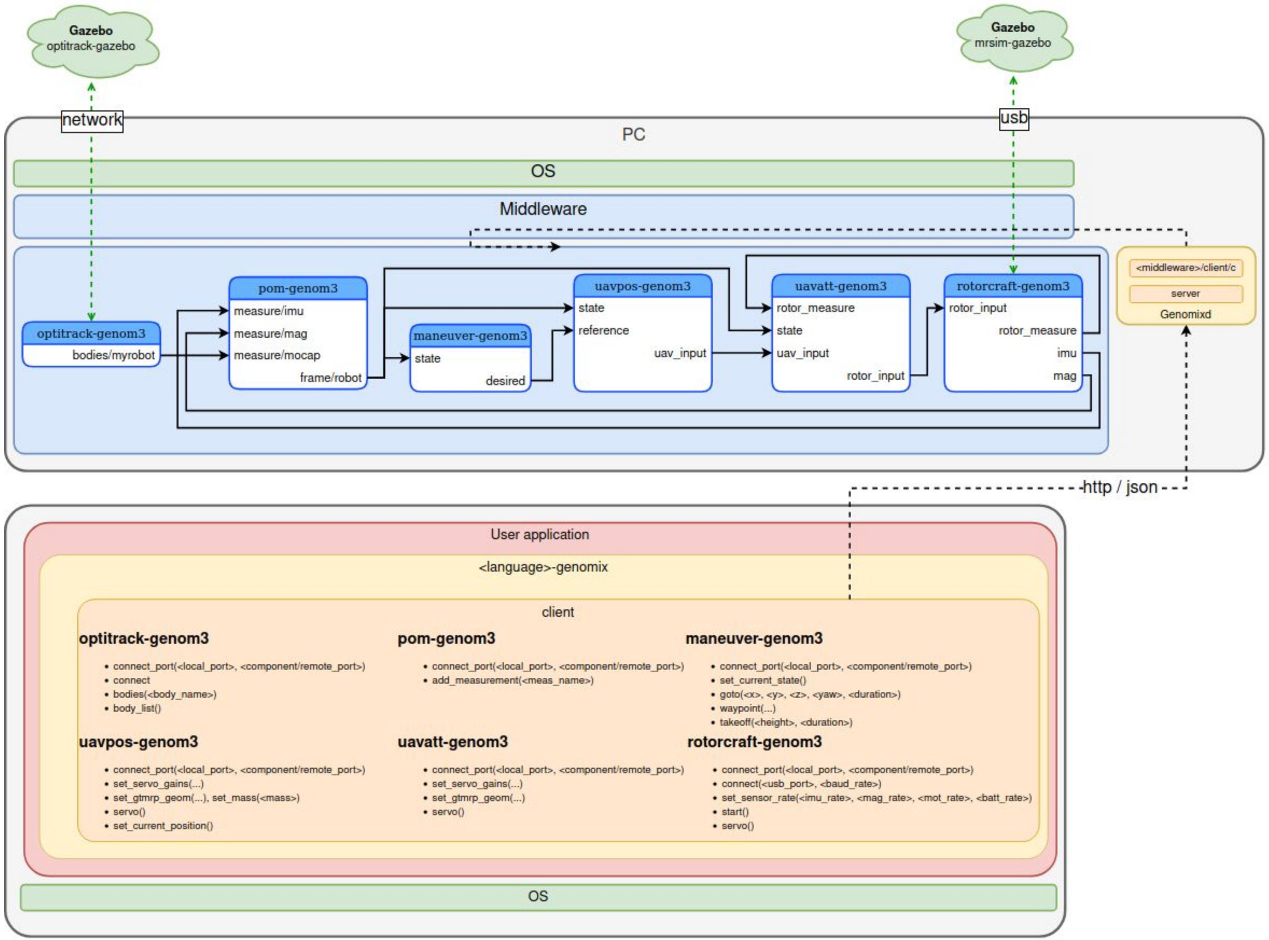
3. Software

Example of software

architecture for an

hexa-rotor

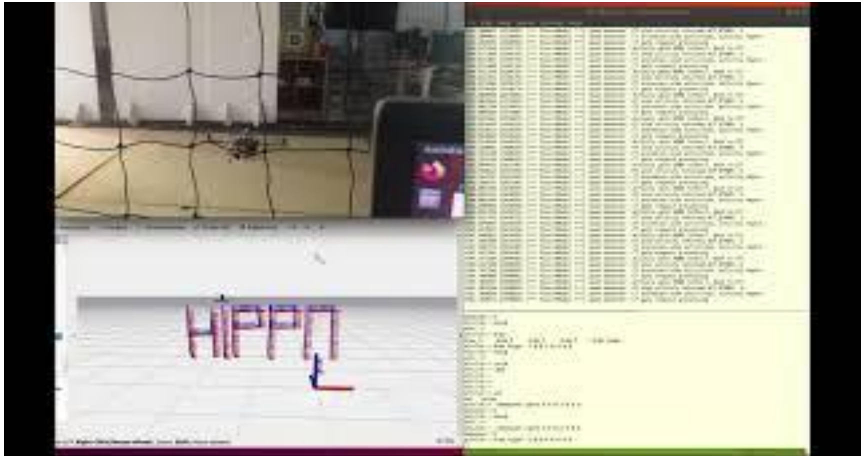
in **simulation**



NB: PC = localhost!

4. Examples of applications

Indoor (left) and outdoor (right) navigation



Courtesy of Felix Ingrand.

Courtesy of Felix Ingrand.

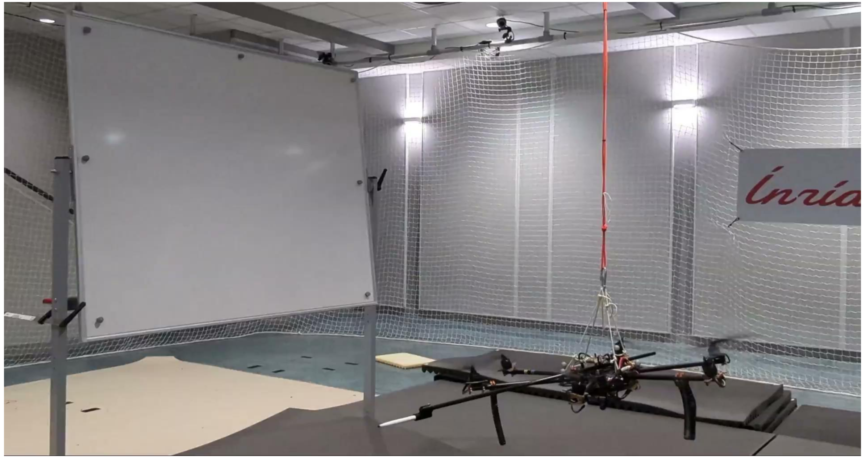
Software validation and verification through
[Genom3 template for the FIACRE language](#).

4. Examples of applications

Physical interaction with the environment

- Pick-and place
- Aerial drawing

Experiments at LAAS: Autonomous pick-and-place application.
G. Corsini et al.. A General Control Architecture for Visual Servoing and Physical Interaction Tasks for Fully-actuated Aerial Vehicles. AIRPHARO 2021.



Experiments at IRISA: aerial drawing with a fully-actuated multi-rotor aerial vehicle.

4. Examples of applications

Physical Human-Aerial robot Interaction

- Human-2-robot handover

Experiments at University of Twente (the Netherlands): human-to-aerial-robot handover.
A. Afifi et al.. Physical Human-Aerial Robot Interaction and Collaboration: Exploratory
Results and Lessons Learned. ICUAS 2023

4. Examples of applications

Agile navigation with a multi-robot system

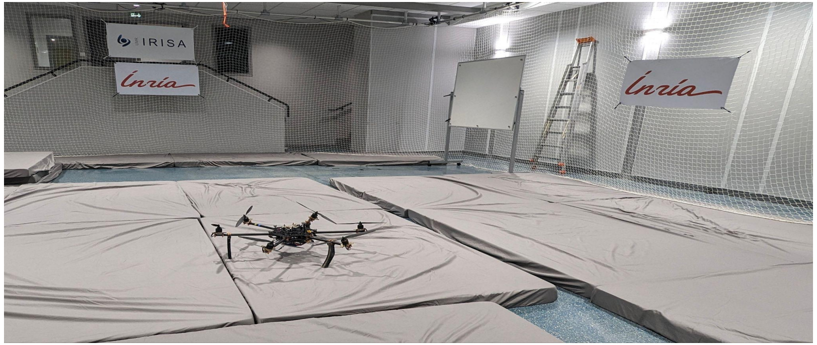
- Flycrane = 3x QR + payload platform + cables

Experiments at IRISA: agile trajectory tracking with the Flycrane system.

5. Journée Drones 2024

Practical session:

- Simulations in Gazebo
    - Under-actuated Quad-rotor flight
    - Under-actuated Hexa-rotor flight
    - Fully-actuated Hexa-rotor flight

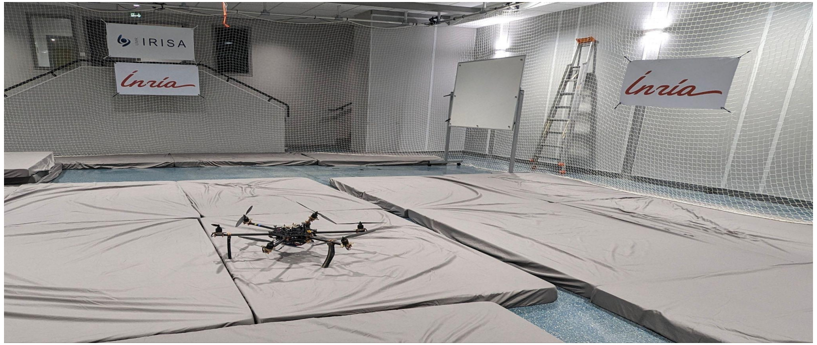


5. Journée Drones 2024

Practical session:

- Simulations in Gazebo
  - Under-actuated Quad-rotor flight
  - Under-actuated Hexa-rotor flight
  - Fully-actuated Hexa-rotor flight

- (Possibly) Indoor Experiment
  - Fully-actuated Hexa-rotor flight



6. Conclusions

Reasons to **consider** TK3:

- **Growing** community
- **Modular** architecture
- Full-access to **low-level** hardware
- **Open-source**
- **Single** and **multi-robot** systems
- Adaptation to **any application**

6. Conclusions

Reasons to **consider** TK3:

- **Growing** community
- **Modular** architecture
- Full-access to **low-level** hardware
- **Open-source**
- **Single** and **multi-robot** systems
- Adaptation to **any application**

Reasons to **NOT consider** TK3:

- Requires **basic understanding** of underlying **architecture** and **tools**
- **Not** really **user-friendly**
  - Requires a bit of motivation and dedication
  - **PhD-friendly**: students willing to add their own functionalities to the basic framework

6. Conclusions

Future directions:

- **Hardware availability**
- **Open-hardware** → release platform designs
- **New ESC** alternative →closed-loop speed control, high-frequency telemetry
- Control of **aerial manipulators** →whole-body and optimization-based control
- Testing the components related to **vision** →realsense cameras

Eager to join the TK3 community?

- Element chat → https://matrix.to/#/#art:laas.fr
- Official project → https://git.openrobots.org/projects/telekyb3
  - Documentation and tutorials (ongoing)
    https://git.openrobots.org/projects/telekyb3/pages/index
  - BSD-like license
- **ART Meetings**
  - Monthly meetings between institutions to discuss status, progress, and future development
- Feel free to make questions and open issues → git.openrobots.org

Thanks for your attention!

Any question?