DRONE & ROS 2 HUMBLE

COMPRENDRE LE FLIGHT CONTROLLER ET SON INTÉGRATION DANS ROS 2

Sommaire

Introduction aux drones

- Rôle et fonction.
- Exemples de firmware de flight controllers populaires (Pixhawk, Ardupilot, Betaflight).
- Communication avec le drone (protocoles MAVLink, RTPS, etc.).

Notions avancées

- Boucles de contrôle (PID, LQR).
- Gestion des capteurs et estimation d'état (IMU, GPS, LiDAR).
- Planification de trajectoires et évitement d'obstacles.

Communication entre le flight controller et ROS2

- Utilisation de Fast DDS pour RTPS.
- Topologie de communication : topics pour la télémétrie, commandes de vol, états.

Présentation de ROS2 dans le contexte des drones

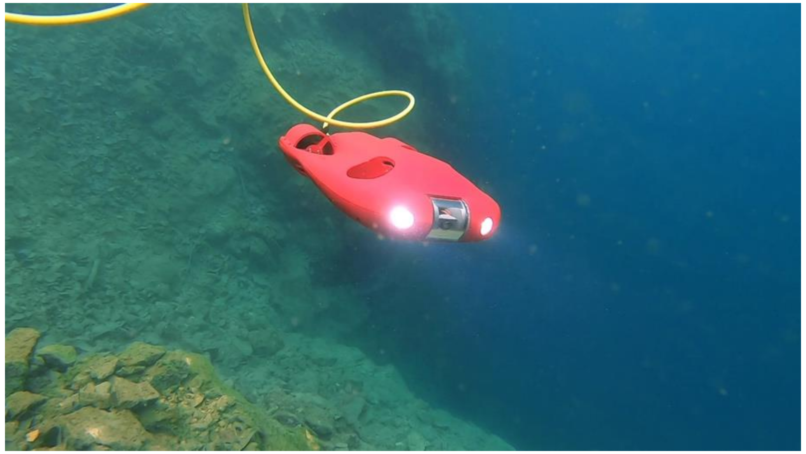
- Concepts de base de ROS2 (nodes, topics, services).
- Pourquoi ROS2 pour les drones (temps réel, modularité).
- Intégration avec un simulateur (Gazebo, Ignition) et un flight controller.

Introduction aux drones

Un **drone** est un véhicule **sans pilote**, **télécommandé** ou **autonome**, utilisé pour diverses applications.

Caractéristiques principales :

- Capacité à voler sans intervention humaine directe.
- Équipé de capteurs, d’actionneurs et d’un système de contrôle embarqué.

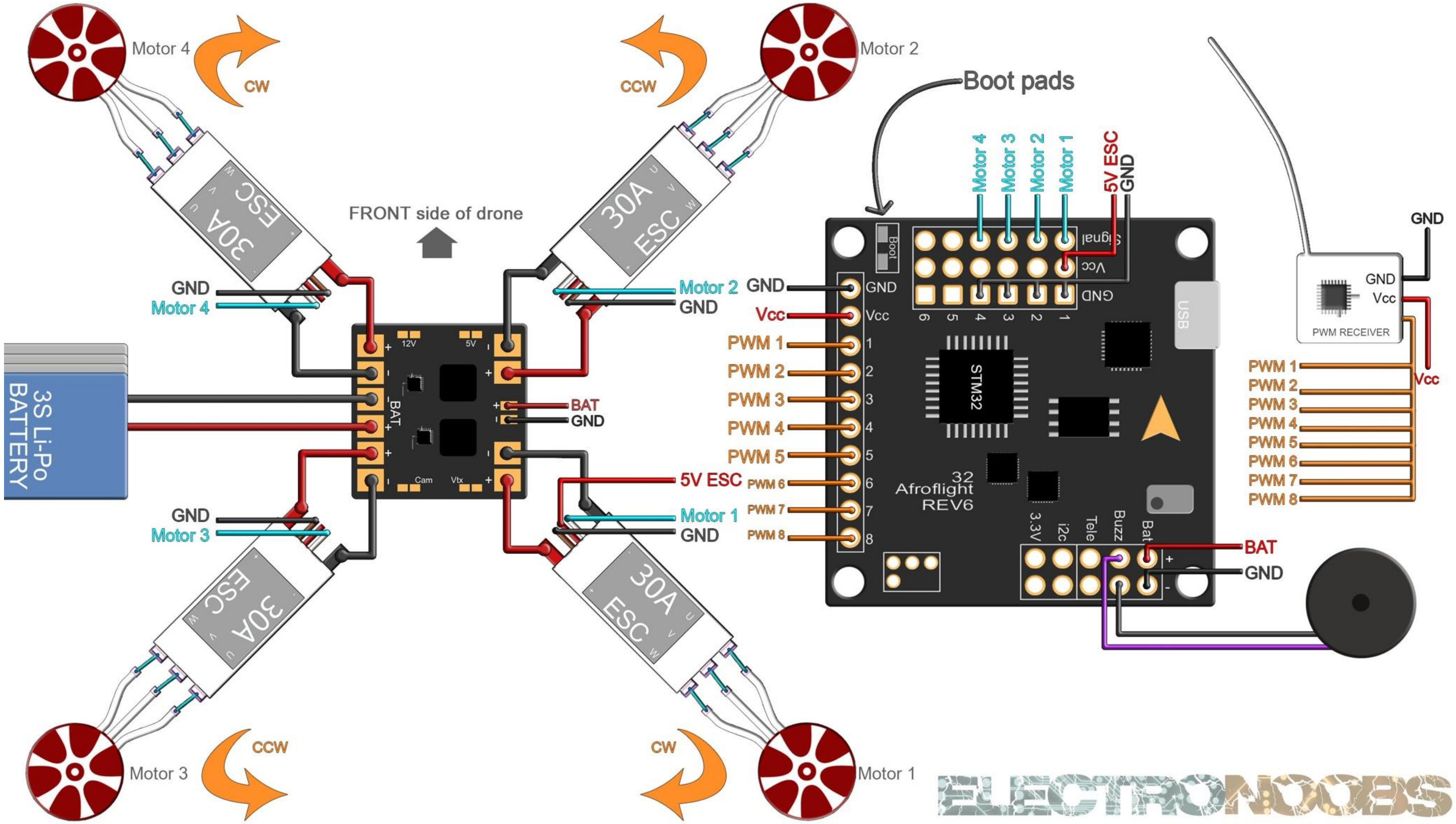


Introduction aux drones

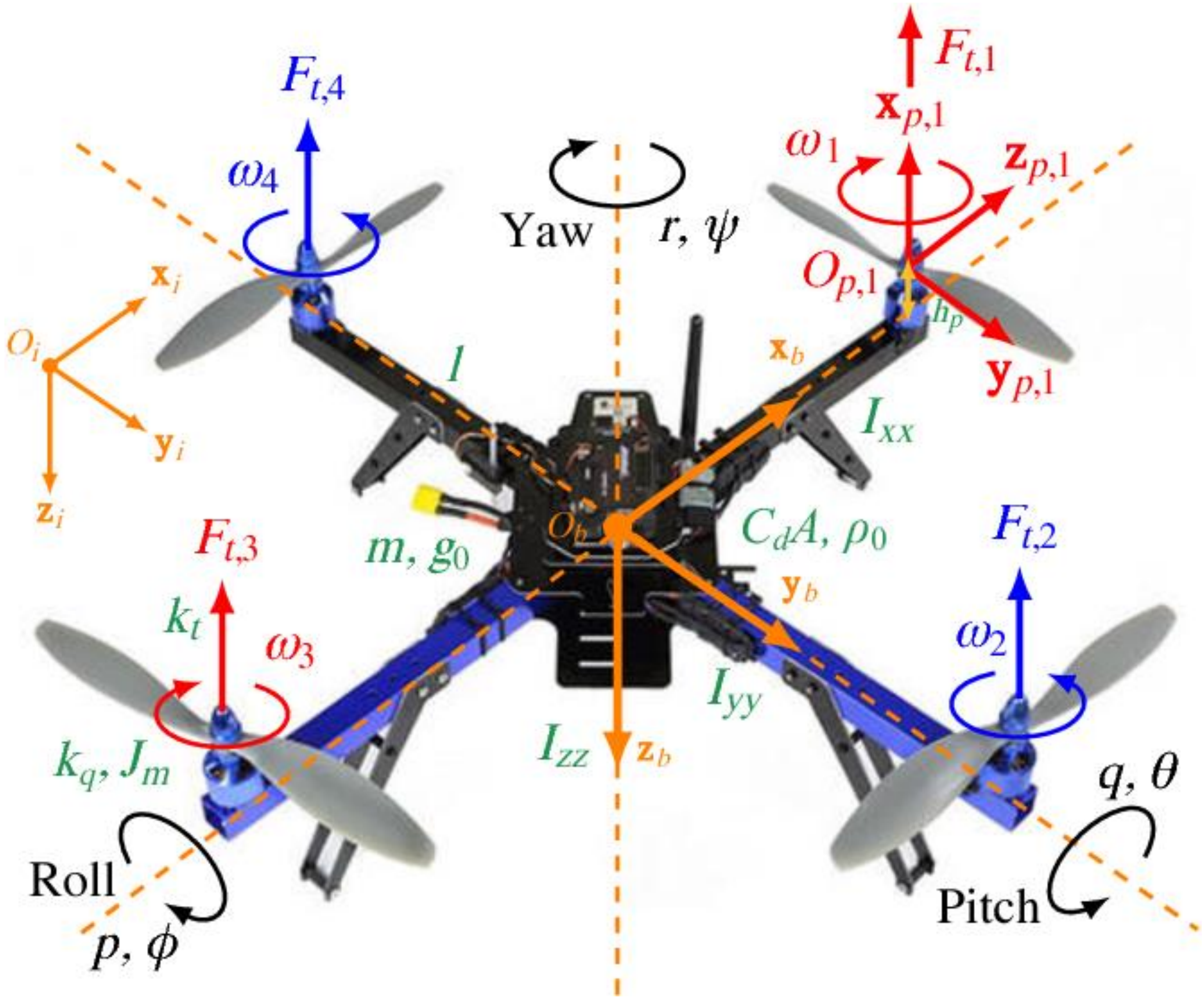
Types de drones aériens :

1. **Multirotors** (quadricoptères, hexacoptères) :
    - Stabilité, maniabilité.
    - Idéal pour les prises de vue et les environnements complexes.
2. **Drones à voilure fixe** :
    - Grande autonomie.
    - Adapté pour la surveillance ou la cartographie.
3. **Drones hybrides** :
    - Combinaison voilure fixe + multirotor.

Introduction aux drones



Introduction aux drones



Introduction aux drones

Le **flight controller** est le cerveau du drone. Il reçoit des commandes de vol, les interprète, et agit sur les moteurs pour contrôler le vol.

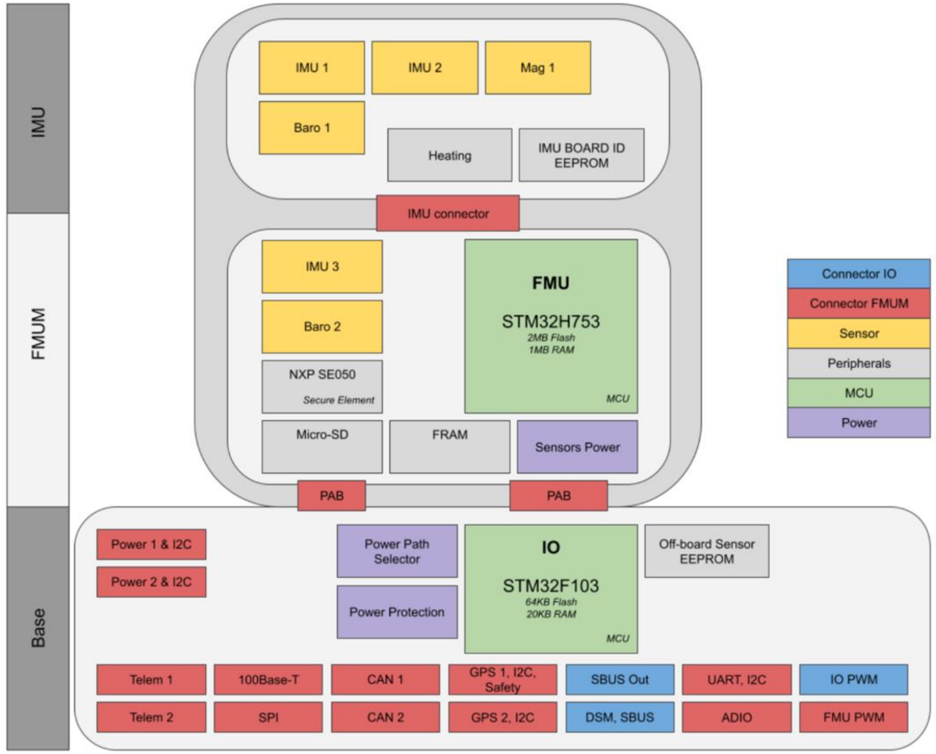
Il utilise :

- **Capteurs internes** (IMU, baromètre, GPS) pour évaluer l'état du drone.
- **Commandes externes** envoyées par un pilote ou un système autonome.
- **Protocoles de communication** comme MAVLink pour interagir avec ROS2.

Introduction aux drones

FMUv6X Summary

Overview



NOTE: FMUv6X has the same architecture as v5X, but is based on STM32H7.

Introduction aux drones

CLEANFLIGHT

LibrePilot

ARDUPILOT

PX4 autopilot

BETAFLIGHT

Introduction aux drones

- Créé en 2011 par ETH Zurich, axé sur les systèmes de **drones professionnels et de recherche**.

  Multirotors, avions, rovers, sous-marins, ballons (plusieurs plateformes prises en charge).

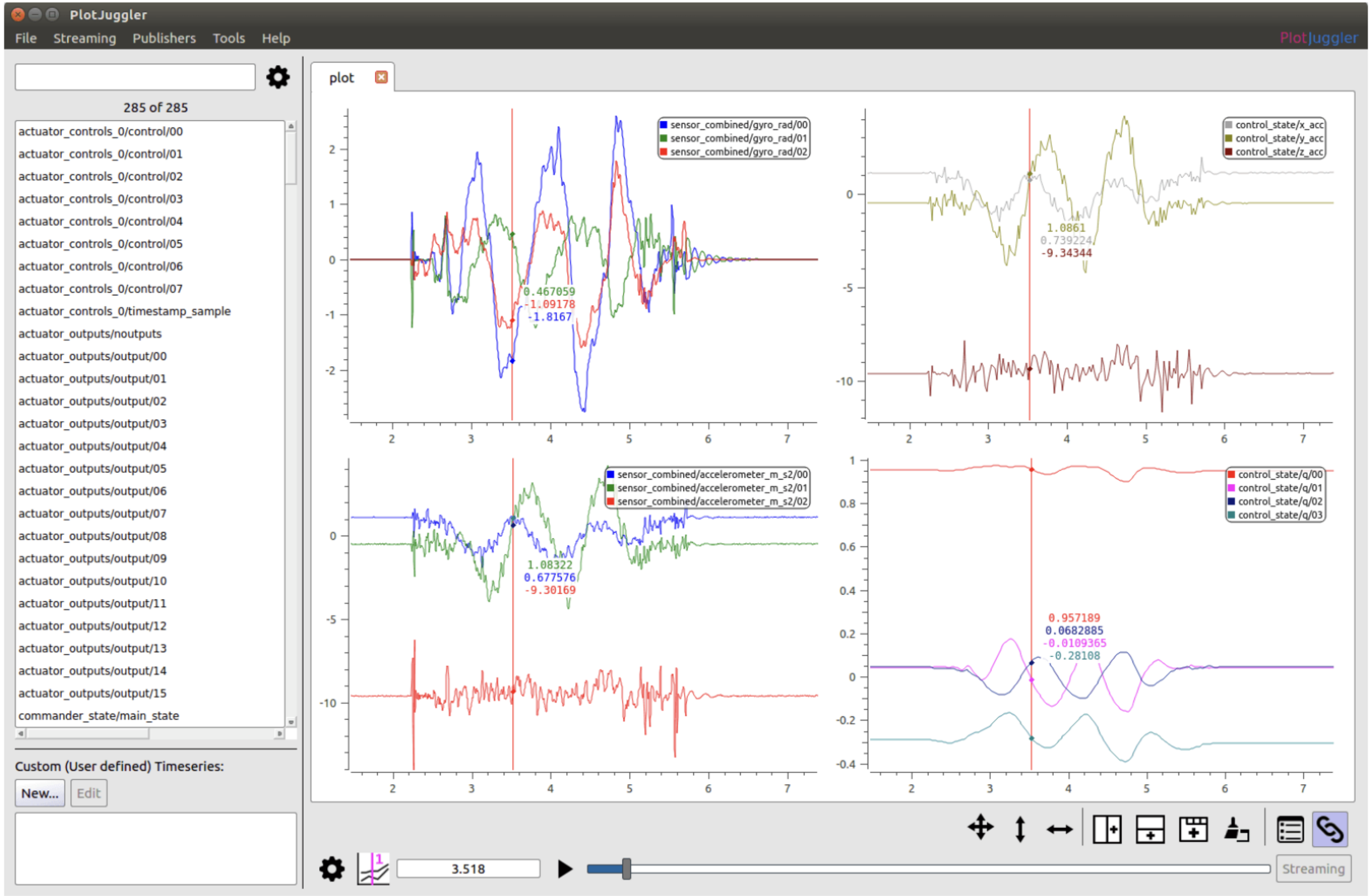
  Haute modularité, mais davantage orienté vers des applications professionnelles.

  Intégration native avec **ROS2, MAVSDK**, et d’autres outils pour la recherche et les cas complexes.

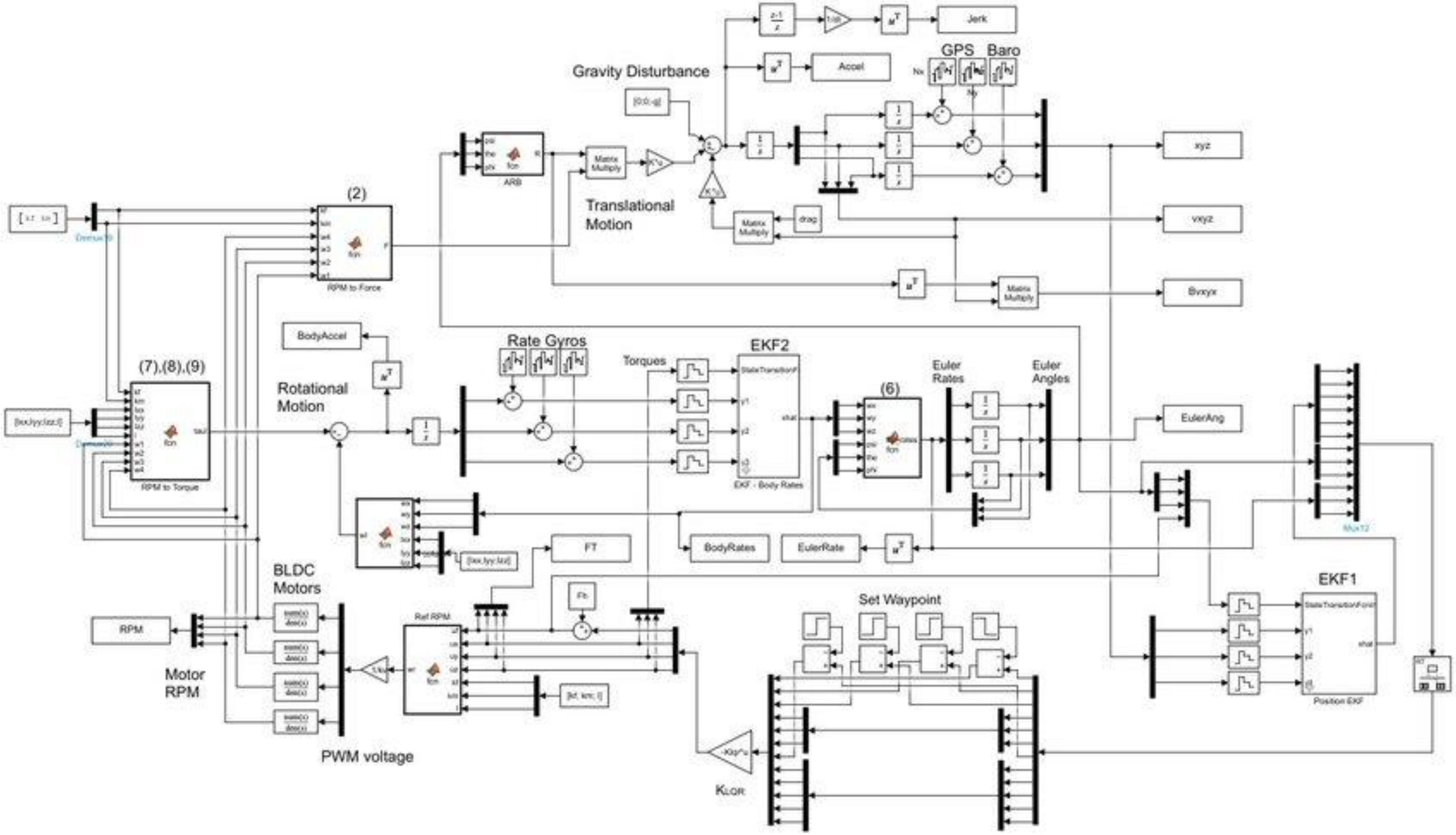
  Open-source, licence BSD.

Notions avancées

- Raw data to odom ?



Notions avancées



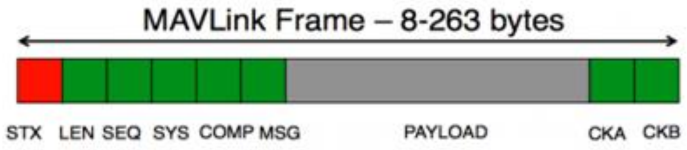
Communication entre le flight controller et ROS2

- Les différentes stacks de communication entre les systèmes des drones

Communication entre le flight controller et ROS2

- Communication avec le drone

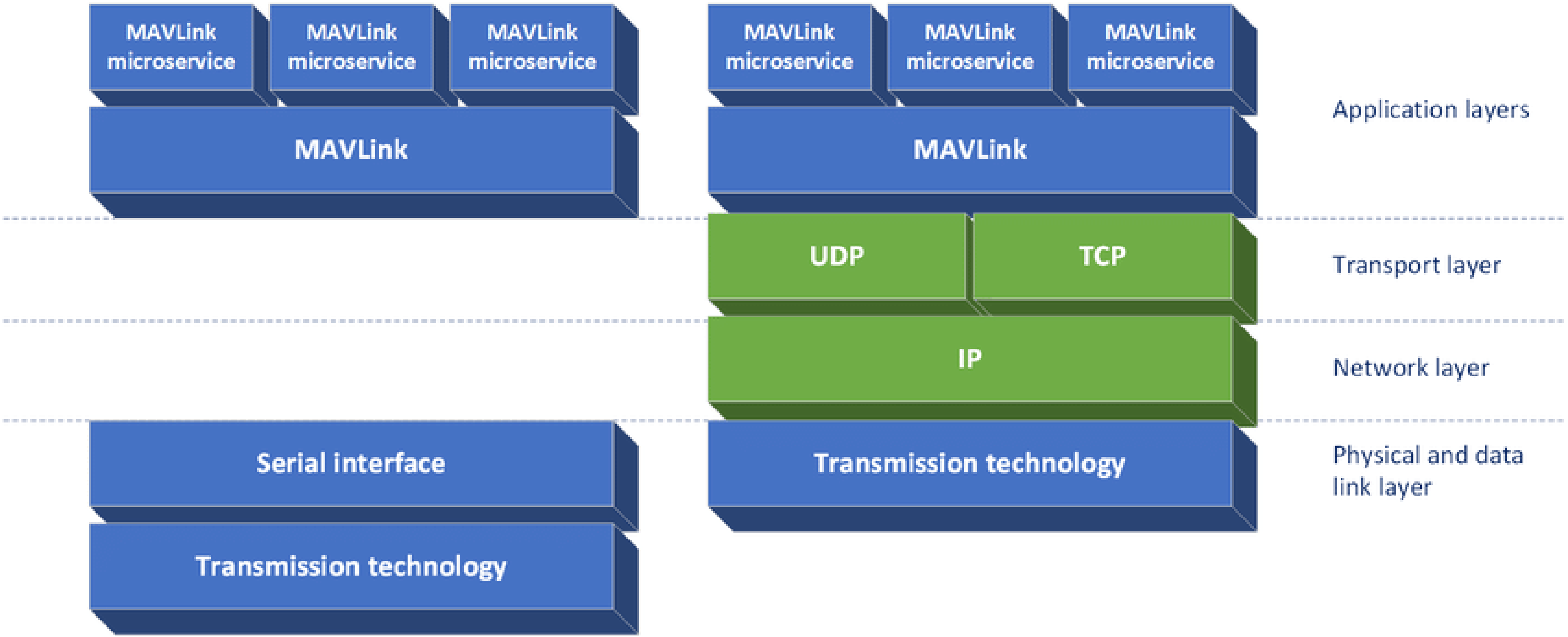
  MAVLink -- Micro Air Vehicle Message Marshalling Library.



Byte Index	Content	Value	Explanation
0	Packet start sign	v1.0: 0xFE (v0.9: 0x55)	Indicates the start of a new packet.
1	Payload length	0 - 255	Indicates length of the following payload.
2	Packet sequence	0 - 255	Each component counts up his send sequence. Allows to detect packet loss
3	System ID	1 - 255	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
4	Component ID	0 - 255	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
5	Message ID	0 - 255	ID of the message - the id defines what the payload "means" and how it should be correctly decoded.
6 to (n+6)	Data	(0 - 255) bytes	Data of the message, depends on the message id.
(n+7) to (n+8)	Checksum (low byte, high byte)	ITU X.25/SAE AS-4 hash, **excluding packet start sign, so bytes 1..(n+6)** Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from message fields. Protects the packet from decoding a different version of the same packet but with different variables).	

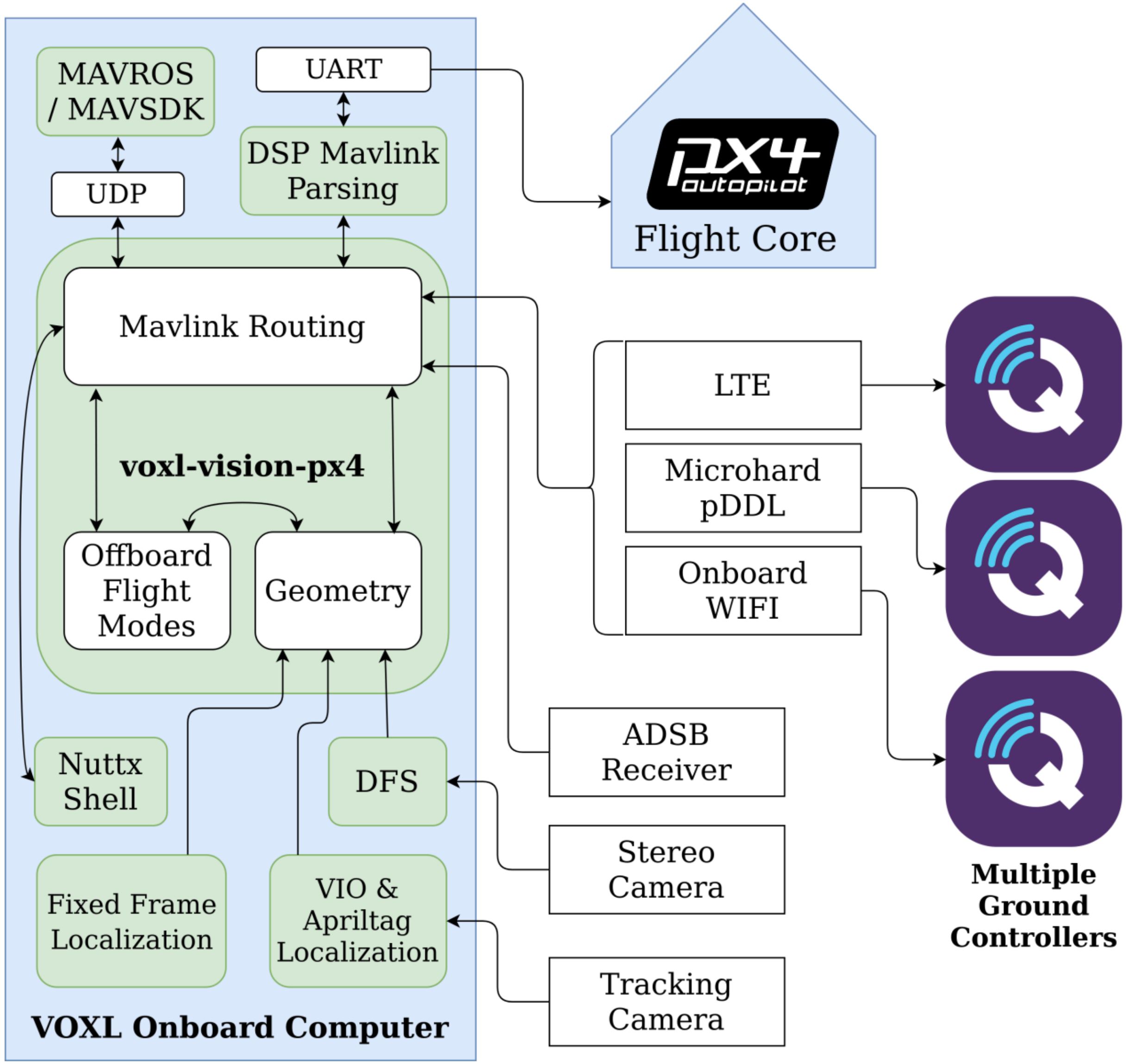
Communication entre le flight controller et ROS2

- Communication avec le drone



Communication entre le flight controller et ROS2

- Exemple d'utilisation



Communication entre le flight controller et ROS2

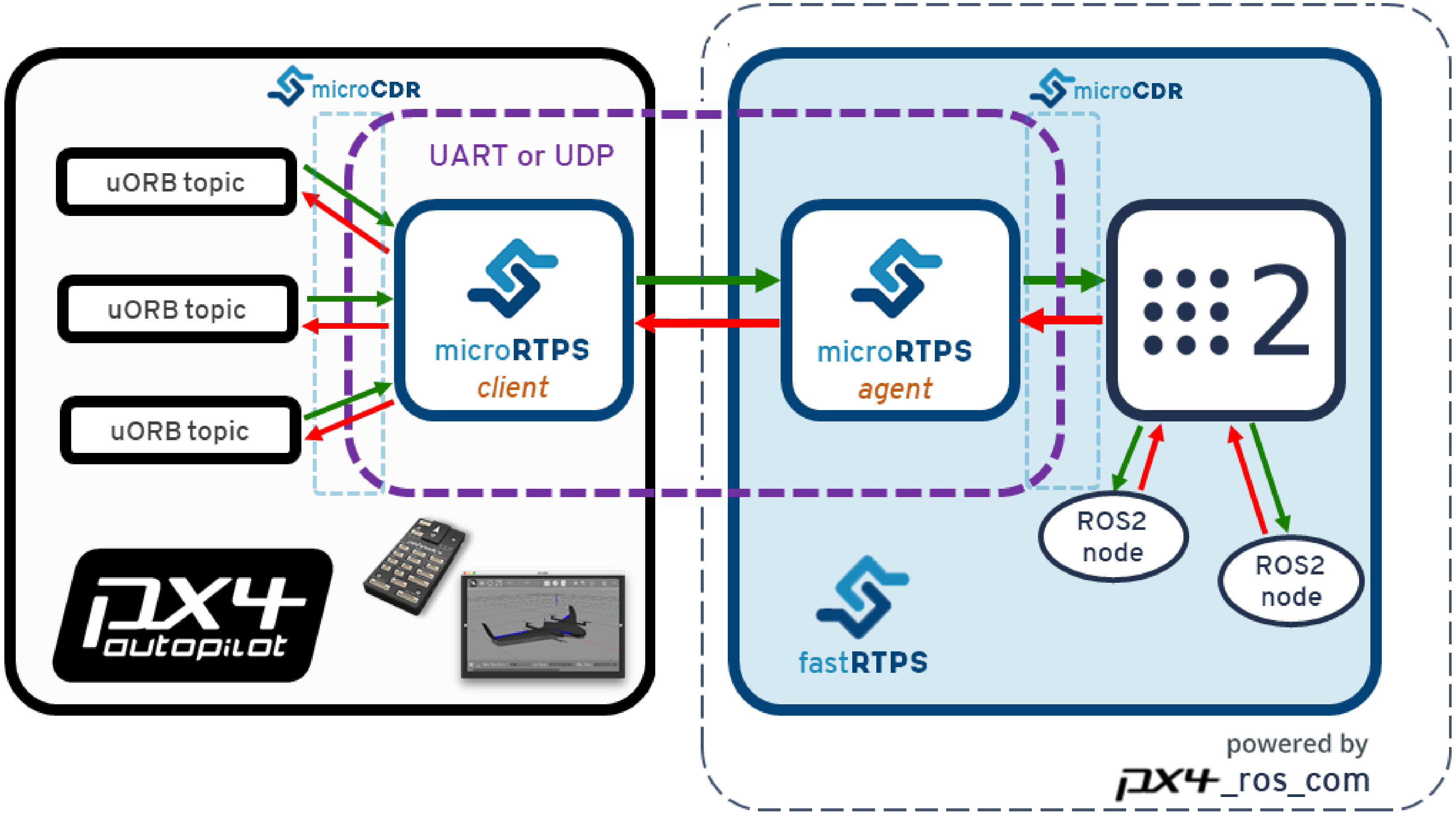
- Communication avec le drone

Middleware DDS (Data-Distribution Service)/RTPS (Real Time Publish Subscribe)

Product name	License	RMW implementation
eProsima *Fast DDS*	Apache 2	`rmw_fastrtps_cpp`
Eclipse *Cyclone DDS*	Eclipse Public License v2.0	`rmw_cyclonedds_cpp`
RTI *Connext*	commercial, research	`rmw_connext_cpp`
GurumNetworks *GurumDDS*	commercial	`rmw_gurumdds_cpp`

Introduction aux drones

Communication avec le drone



Communication entre le flight controller et ROS2

Quel système choisir ?

✓	MAVROS ✗
▪ Long-tested and industrial proven bridge between Mavlink and ROS;	▪ Not future proof – currently, no implementation going on to update the API and interface to support ROS 2.
▪ Allows parsing Mavlink messages to ROS standard messages;	▪ It's directly dependent on Mavlink and its interfaces, definitions and of course, limitations;
▪ Allows network rebroadcast of the data to and from other hosts.	▪ Does not tide directly to the PX4 internals, though losing granularity for introspection and control.

Communication entre le flight controller et ROS2

Quel système choisir ?

px4_ros_com

✓

- Take advantage of all the benefits of DDS implementation and direct integration with ROS 2;
- (Theoretically) faster throughput and lower latency over the link;
- Direct and more tide relation between the PX4 internals and the offboard components;
- Can be tide to other RTPS (DDS) participants which are not registered over ROS – example: MAVSDK.

✗

- Obliges to have a one-to-one conversion between the uORB messages and the ROS messages, meaning it's not parsed to standard ROS messages;
- For connecting to ROS (1), where most of the packages are still implemented, still requires a secondary bridge (`ros1_bridge`) so to be able to connect ROS (1) nodes with PX4.

23

Présentation de ROS2 dans le contexte des drones

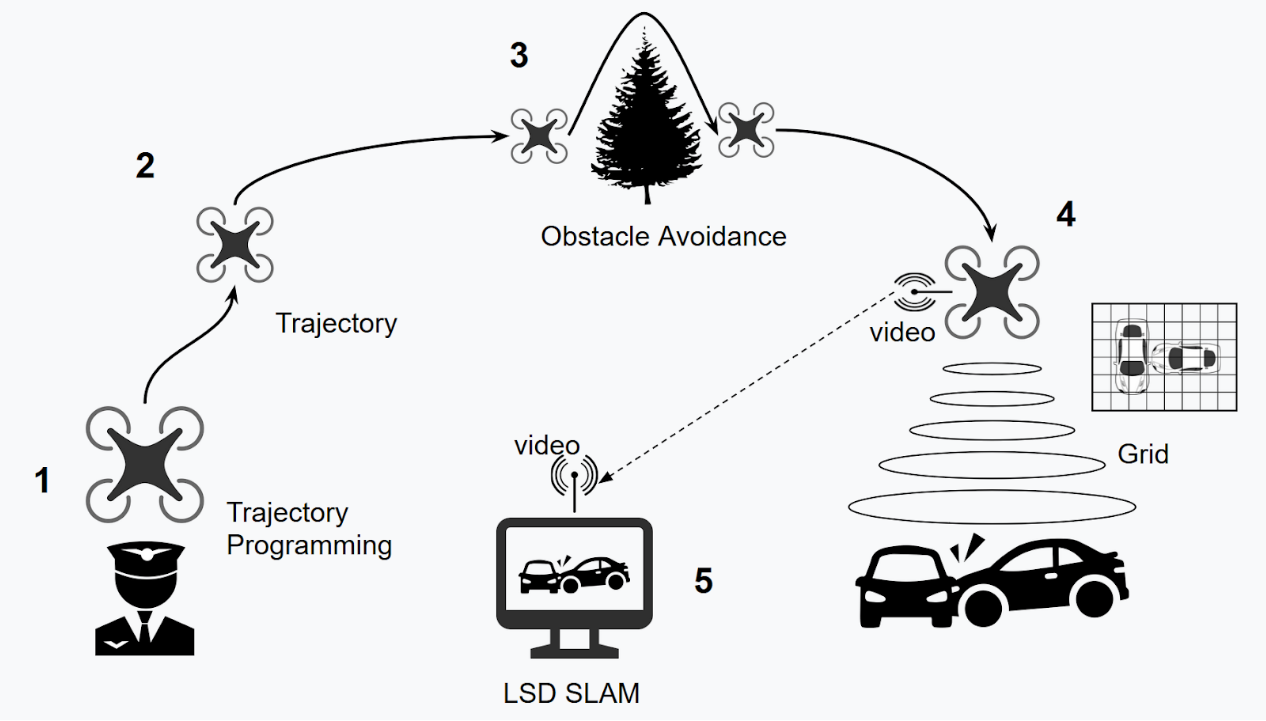
- Permet de créer des applications drones autonomes / avancées

  Utilisation de computer vision

  Génération de trajectoires

  Éviction d'obstacles

  SLAM



Présentation de ROS2 dans le contexte des drones

uORB Topics

TrajectorySetpoint (UORB message)

```
# Trajectory setpoint in NED frame
# Input to PID position controller.
# Needs to be kinematically consistent and feasible for smooth flight.
# setting a value to NaN means the state should not be controlled

uint32 MESSAGE_VERSION = 0

uint64 timestamp # time since system start (microseconds)

# NED local world frame
float32[3] position # in meters
float32[3] velocity # in meters/second
float32[3] acceleration # in meters/second^2
float32[3] jerk # in meters/second^3 (for logging only)

float32 yaw # euler angle of desired attitude in radians -PI..+PI
float32 yawspeed # angular velocity around NED frame z-axis in radians/second
```

VehicleGlobalPosition (UORB message)

```
# Fused global position in WGS84.
# This struct contains global position estimation. It is not the raw GPS
# measurement (@see vehicle_gps_position). This topic is usually published by t
# estimator, which will take more sources of information into account than just
# e.g. control inputs of the vehicle in a Kalman-filter implementation.
#

uint32 MESSAGE_VERSION = 0

uint64 timestamp         # time since system start (microseconds)
uint64 timestamp_sample         # the timestamp of the raw data (microseconds)

float64 lat         # Latitude, (degrees)
float64 lon         # Longitude, (degrees)
float32 alt         # Altitude AMSL, (meters)
float32 alt_ellipsoid       # Altitude above ellipsoid, (meters)
```

https://docs.px4.io/main/en/msg_docs/

Présentation de ROS2 dans le contexte des drones

Subscribe to topics

VehicleGlobalPosition (UORB message)

```
rmw_qos_profile_t qos_profile = rmw_qos_profile_sensor_data;
auto qos = rclcpp::QoS(rclcpp::QoSInitialization(qos_profile.history, 5), qos_profile);

subscription_ = this->create_subscription<px4_msgs::msg::SensorGps>("/fmu/out/vehicle_gps_position", qos,
[this](const px4_msgs::msg::SensorGps::UniquePtr msg) {
        std::cout << "\n\n\n\n\n\n\n\n\n\n";
        std::cout << "RECEIVED VEHICLE GPS POSITION DATA"   << std::endl;
        std::cout << "=================================="   << std::endl;
        std::cout << "ts: "      << msg->timestamp    << std::endl;
        std::cout << "lat: " << msg->latitude_deg  << std::endl;
        std::cout << "lon: " << msg->longitude_deg << std::endl;
        std::cout << "alt: " << msg->altitude_msl_m  << std::endl;
```

https://docs.px4.io/main/en/msg_docs/

Présentation de ROS2 dans le contexte des drones

- Publish to topics

TrajectorySetpoint (UORB message)

```
void OffboardControl::publish_trajectory_setpoint()
{
	TrajectorySetpoint msg{};
	msg.position = {0.0, 0.0, -5.0};
	msg.yaw = -3.14; // [-PI:PI]
	msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
	trajectory_setpoint_publisher_->publish(msg);
}
```

https://docs.px4.io/main/en/msg_docs/

Présentation de ROS2 dans le contexte des drones

- TD : Création de trajectoires

https://github.com/Kariboo-Corp/vol-drone-interieur