Hands on Aerostack2

Developing a basic swarm application from scratch

Miguel Fernandez-Cortizas, Rafael Perez-Segui, 19 - Jul 2024 Deflt

Agenda

1. What is Aerostack2?
    a. Which were our requirements?
    b. Our Architecture
    c. Use Cases

2. Hands on example (Tutorial)
    a. Formulating a problem
    b. Implementing the solution
    c. Validation on simulation.
    d. Real World deployment.

What is Aerostack2 ?

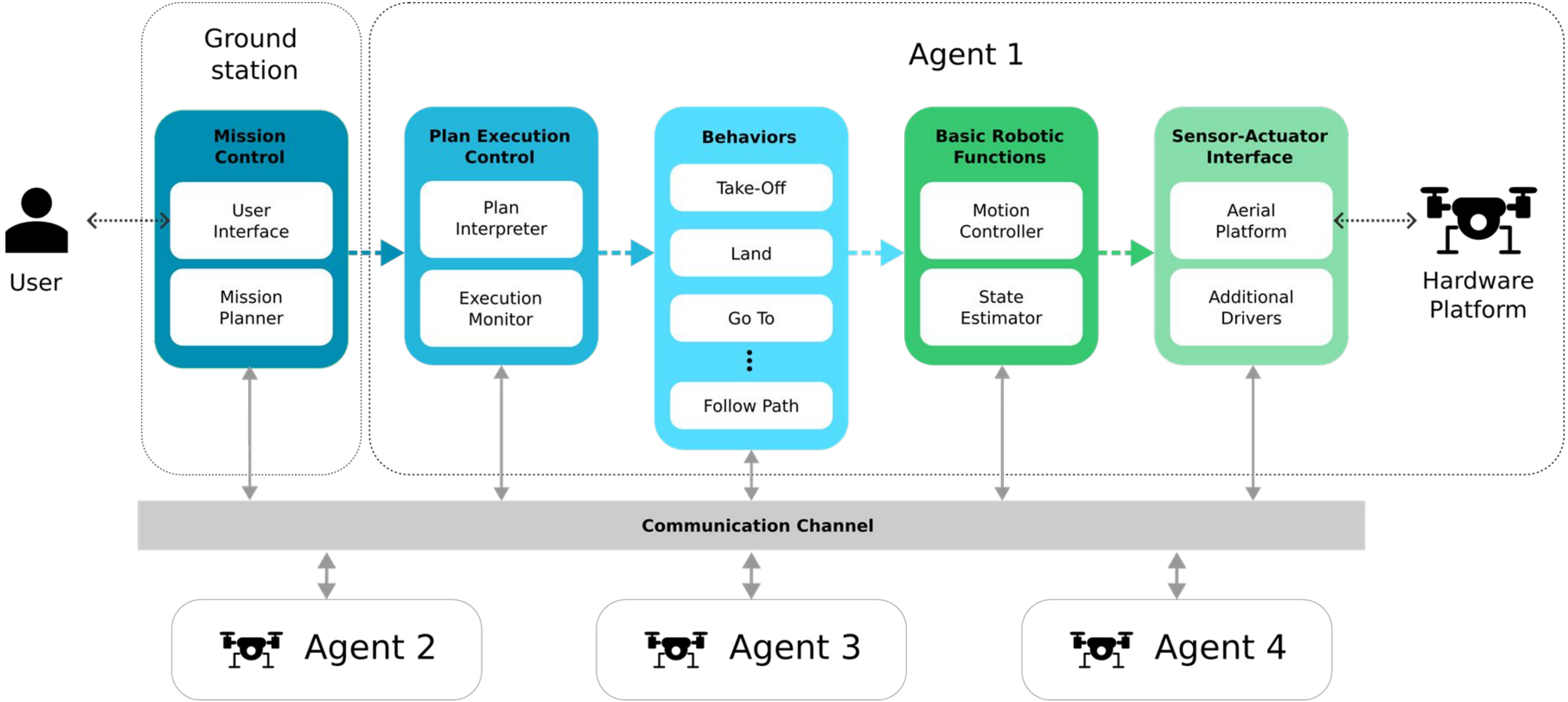
Aerostack2 is an open-source software framework that helps developers design and build multi robot aerial robotic systems. It is designed with ROS 2 and its part of their ecosystem.

It is an evolution of the former Aerostack framework developed and used succesfully in the Computer Vision & Aerial Robotics (CVAR) Group since 2016.

Which were our requirements?

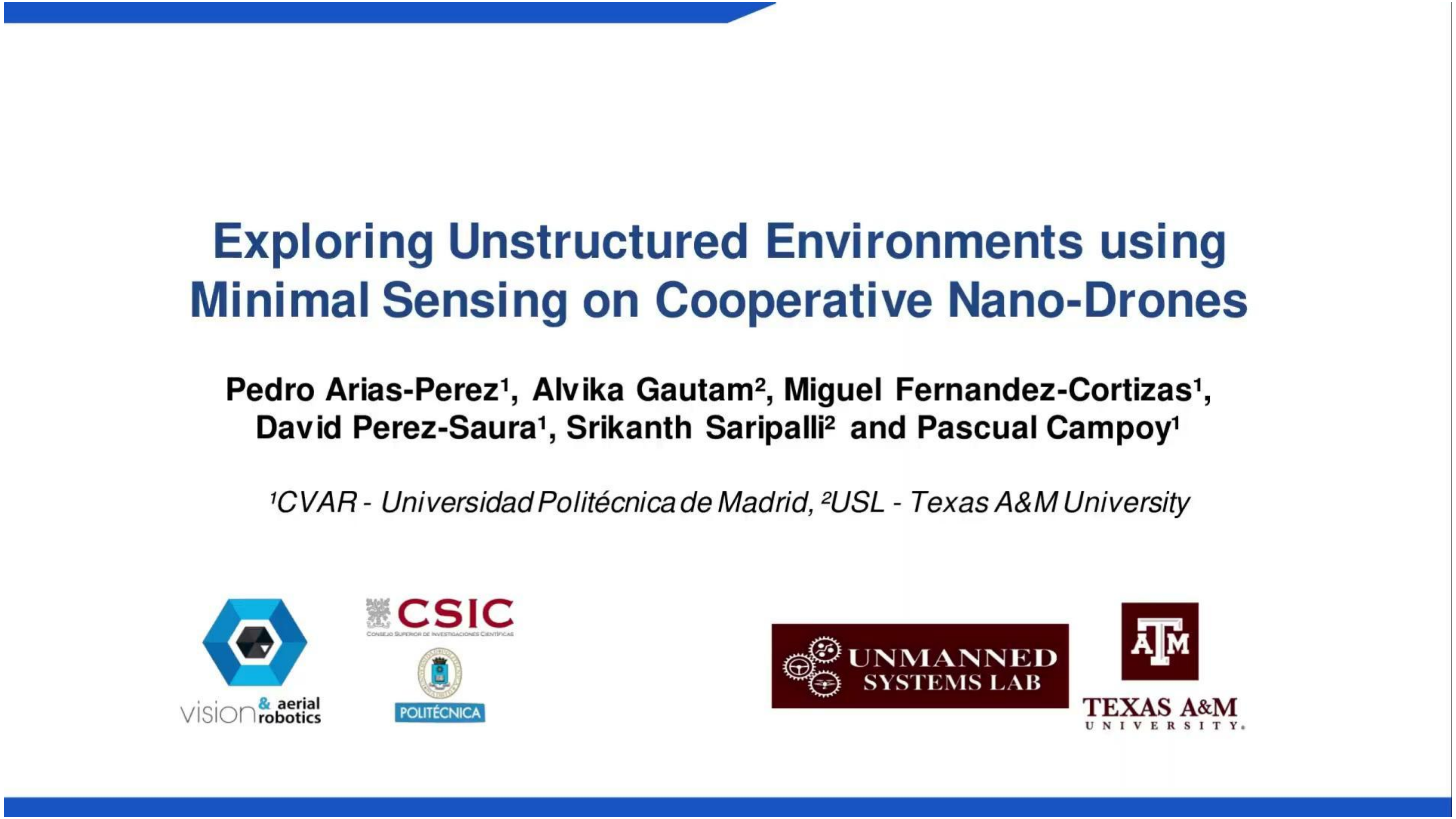
1. Modularity and flexibility
2. Support a variety of platforms
3. Handle multiple drones simultaneously
4. Create new missions easily
5. A safe system development framework
6. Support indoor & outdoor flights operations

Architecture



Use Cases:

Use Cases:



Exploring Unstructured Environments using Minimal Sensing on Cooperative Nano-Drones

Pedro Arias-Perez[1], Alvika Gautam[2], Miguel Fernandez-Cortizas[1], David Perez-Saura[1], Srikanth Saripalli[2] and Pascual Campoy[1]

[1]CVAR - Universidad Politécnica de Madrid, [2]USL - Texas A&M University

ARIAS-PEREZ, Pedro, et al. Exploring Unstructured Environments using Minimal Sensing on Cooperative Nano-Drones. *arXiv preprint arXiv:2407.06706*, 2024.

Use Cases:

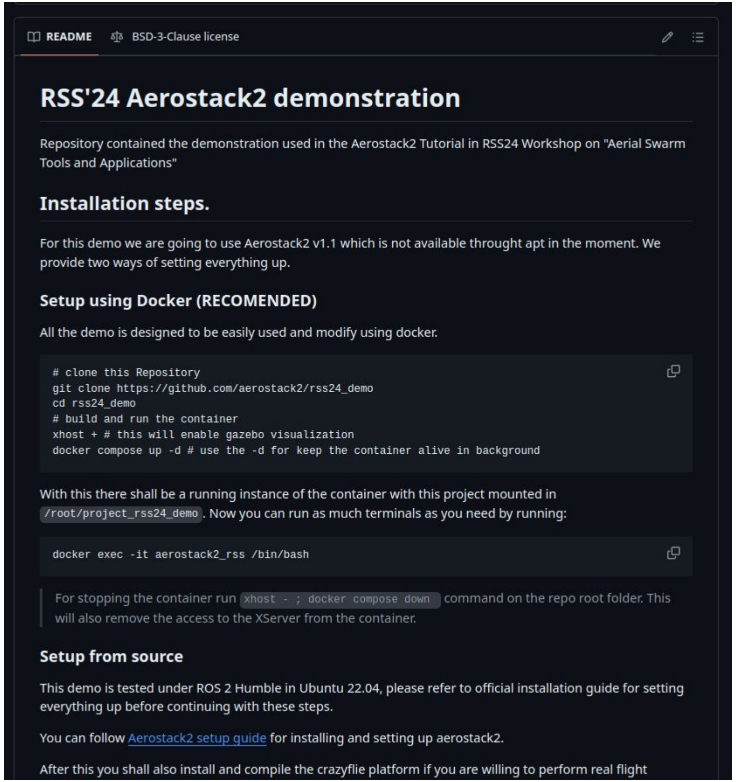
Hands on:
Swarm convoy formation

Problem formulation

Develop an aerial system composed of 1 Leader drone and N follower drones that follows the leader in a convoy-like fashion.

1. Drone N shall follow drone N - 1 trajectory
2. Drones shall keep a horizontal security distance (Avoiding one on top of other)
3. The leader drone can follow a predefined mission or being teleoperated.

Project code available



RSS'24 Aerostack2 demonstration

Repository contained the demonstration used in the Aerostack2 Tutorial in RSS24 Workshop on "Aerial Swarm Tools and Applications"

Installation steps.

For this demo we are going to use Aerostack2 v1.1 which is not available throught apt in the moment. We provide two ways of setting everything up.

Setup using Docker (RECOMENDED)

All the demo is designed to be easily used and modify using docker.

```
# clone this Repository
git clone https://github.com/aerostack2/rss24_demo
cd rss24_demo
# build and run the container
xhost + # this will enable gazebo visualization
docker compose up -d # use the -d for keep the container alive in background
```

With this there shall be a running instance of the container with this project mounted in `/root/project_rss24_demo`. Now you can run as much terminals as you need by running:

```
docker exec -it aerostack2_rss /bin/bash
```

> For stopping the container run `xhost - ; docker compose down` command on the repo root folder. This will also remove the access to the XServer from the container.

Setup from source

This demo is tested under ROS 2 Humble in Ubuntu 22.04, please refer to official installation guide for setting everything up before continuing with these steps.

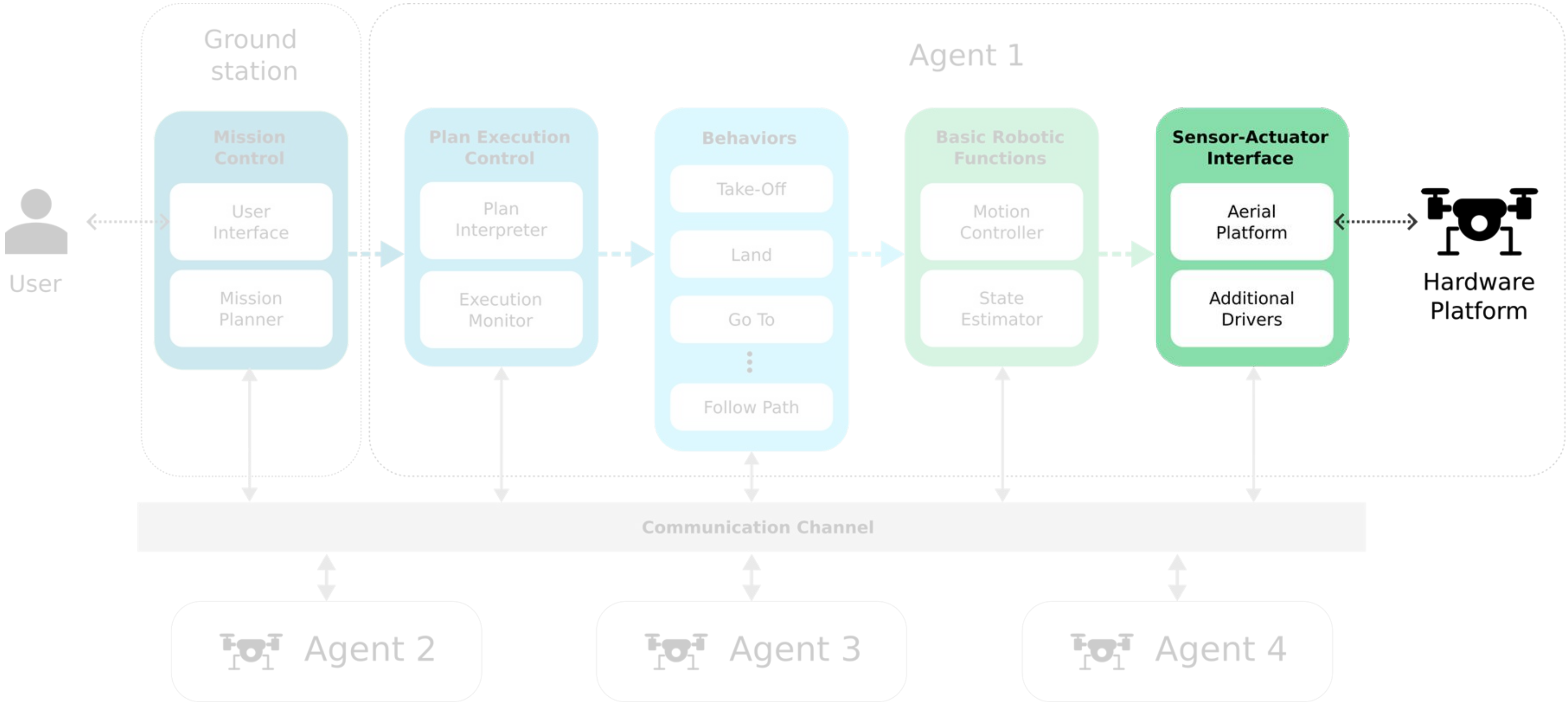
You can follow Aerostack2 setup guide for installing and setting up aerostack2.

After this you shall also install and compile the crazyflie platform if you are willing to perform real flight

https://github.com/aerostack2/rss24_demo

Step by Step Approach: Selecting components

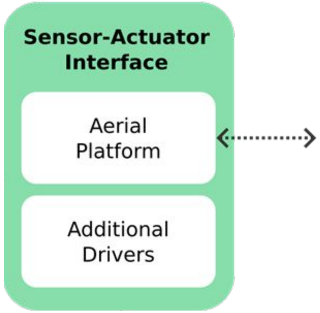
Aerial Platform

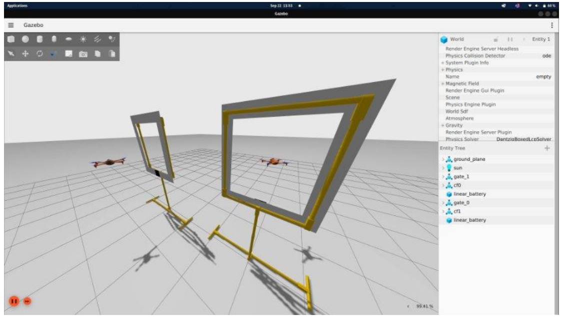


Aerial Platform

Interface between an specific aerial platform and Aerostack2.

- Sensor Measurements
- Actuator commands
- Flight status



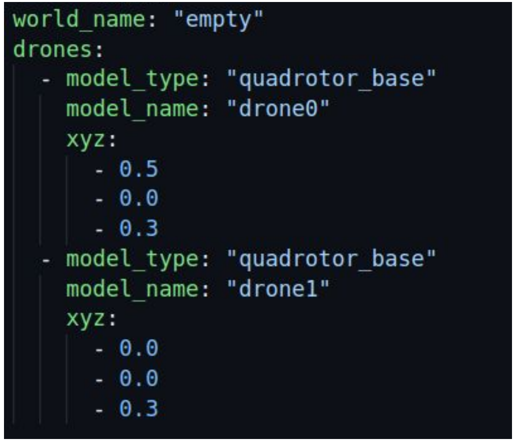


World — Entity 1
Render Engine Server Headless
Physics Collision Detector: ode
System Plugin Info
Physics
Name: empty
Magnetic Field
Render Engine Gui Plugin
Scene
Physics Engine Plugin
World Sdf
Atmosphere
Gravity
Render Engine Server Plugin
Physics Solver: DantzigBoxedLcpSolver

Entity Tree
- ground_plane
- sun
- gate_1
- cf0
- linear_battery
- gate_0
- cf1
- linear_battery

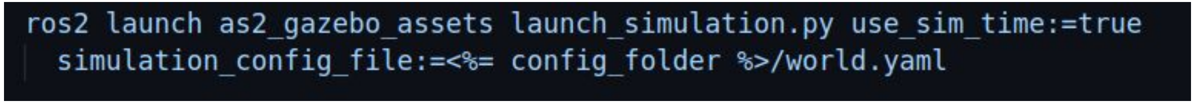
Gazebo Platform

1. Specify a simulation_config_file (world.yaml)



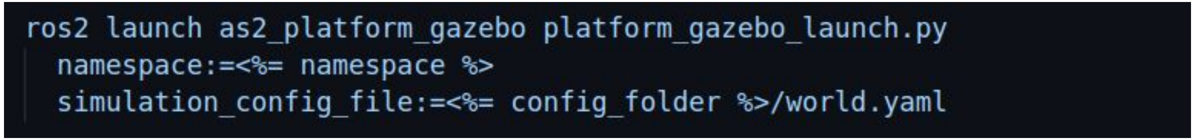
```yaml
world_name: "empty"
drones:
  - model_type: "quadrotor_base"
    model_name: "drone0"
    xyz:
      - 0.5
      - 0.0
      - 0.3
  - model_type: "quadrotor_base"
    model_name: "drone1"
    xyz:
      - 0.0
      - 0.0
      - 0.3
```

2. Launch simulation **one time** for start GZ Simulation

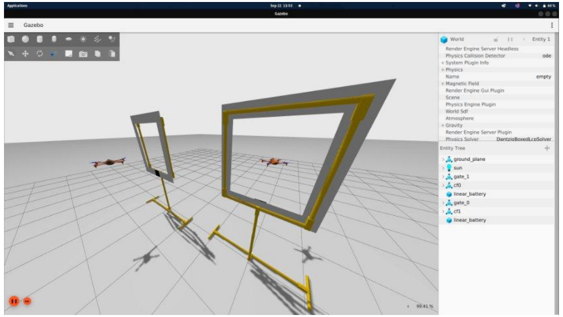


```
ros2 launch as2_gazebo_assets launch_simulation.py use_sim_time:=true
  simulation_config_file:=<%= config_folder %>/world.yaml
```

3. One platform launch per drone (namespace = model_name)



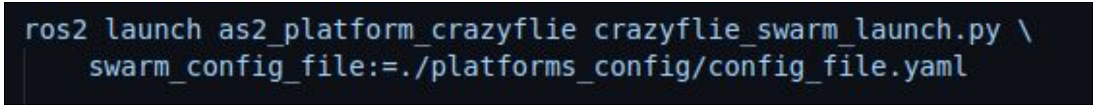
```
ros2 launch as2_platform_gazebo platform_gazebo_launch.py
  namespace:=<%= namespace %>
  simulation_config_file:=<%= config_folder %>/world.yaml
```



World — Entity 1
Render Engine Server Headless
Physics Collision Detector: ode
System Plugin Info
Physics
Name: empty
Magnetic Field
Render Engine Gui Plugin
Scene
Physics Engine Plugin
World Sdf
Atmosphere
Gravity
Render Engine Server Plugin
Physics Solver: DantzigBoxedLcpSolver

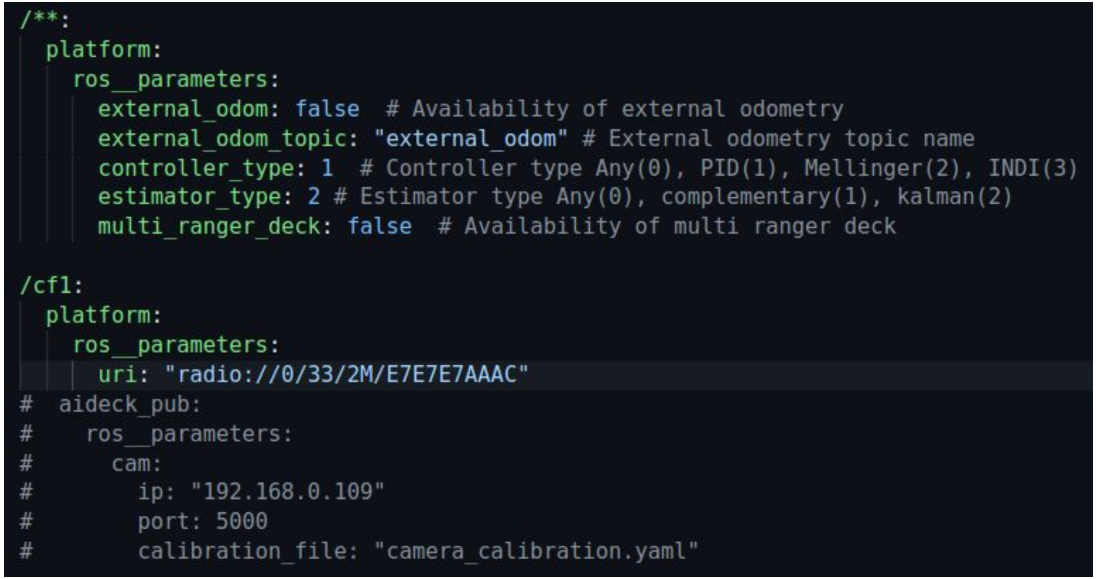
Entity Tree
ground_plane
sun
gate_1
cf0
linear_battery
gate_0
cf1
linear_battery

Crazyflie Platform



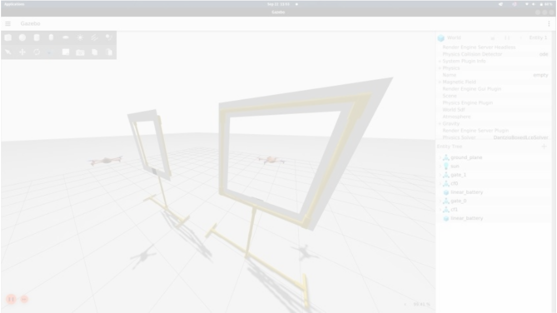
```
ros2 launch as2_platform_crazyflie crazyflie_swarm_launch.py \
    swarm_config_file:=./platforms_config/config_file.yaml
```

1. Only launch one for all the crazyflies (antenna resource)
2. Specify a swarm_config_file indicating URIs

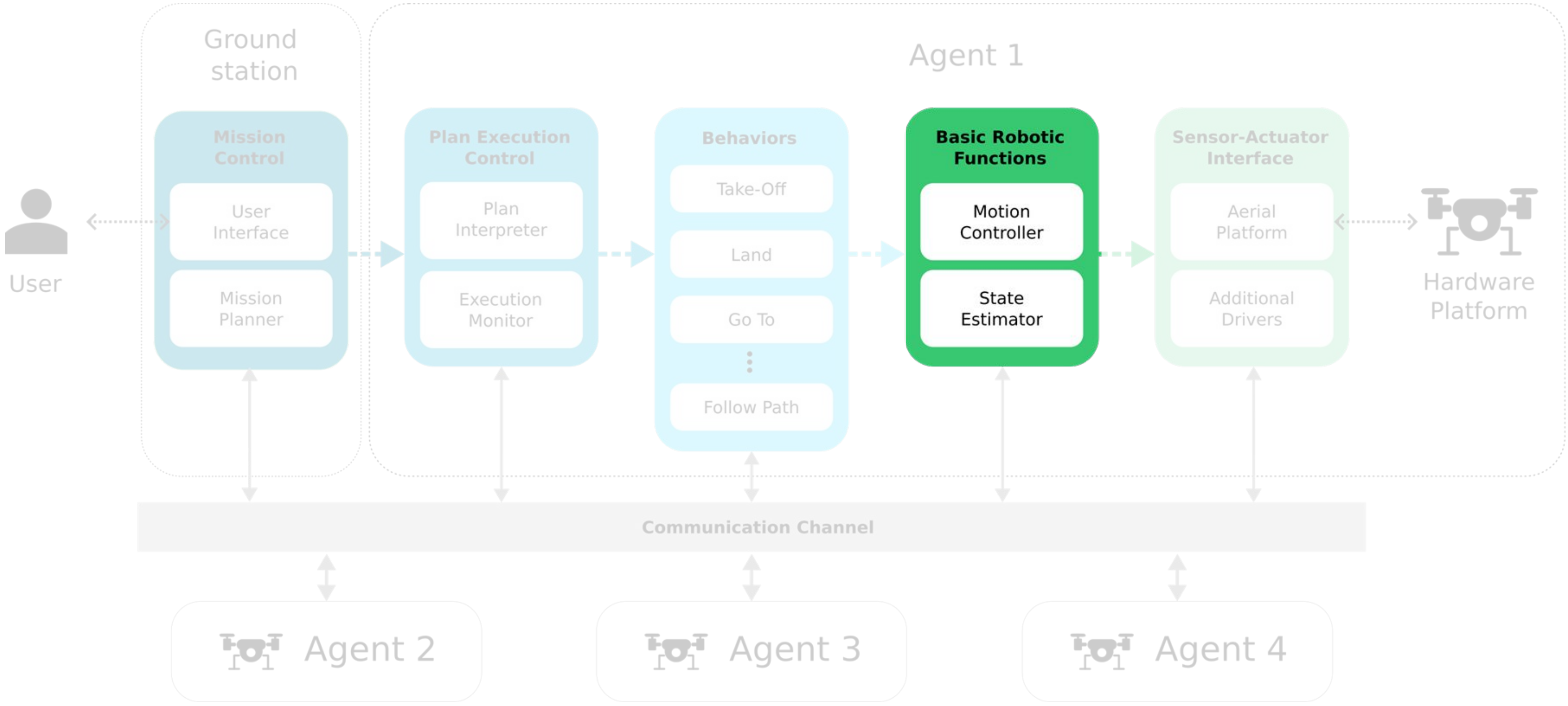


```
/**:
  platform:
    ros__parameters:
      external_odom: false  # Availability of external odometry
      external_odom_topic: "external_odom" # External odometry topic name
      controller_type: 1  # Controller type Any(0), PID(1), Mellinger(2), INDI(3)
      estimator_type: 2 # Estimator type Any(0), complementary(1), kalman(2)
      multi_ranger_deck: false  # Availability of multi ranger deck

/cf1:
  platform:
    ros__parameters:
      uri: "radio://0/33/2M/E7E7E7AAAC"
#  aideck_pub:
#    ros__parameters:
#      cam:
#        ip: "192.168.0.109"
#        port: 5000
#        calibration_file: "camera_calibration.yaml"
```



Basic Robotic Actions



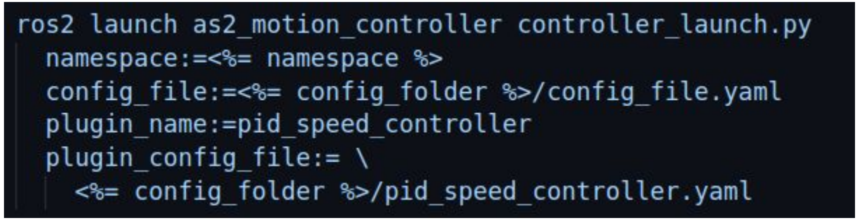
Basic Robotic Actions

Motion controller:

Receives motion references and converts them into actuator commands to the platform.

Can load different plugins with their specific configurations.

For this application we will use **pid_speed_controller**



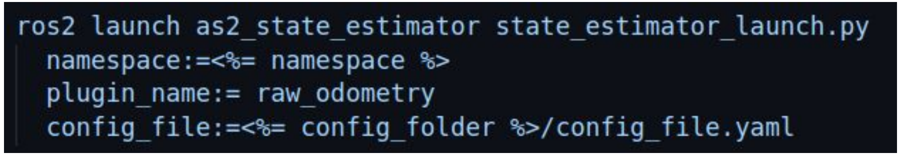
```
ros2 launch as2_motion_controller controller_launch.py
  namespace:=<%= namespace %>
  config_file:=<%= config_folder %>/config_file.yaml
  plugin_name:=pid_speed_controller
  plugin_config_file:= \
    <%= config_folder %>/pid_speed_controller.yaml
```

State Estimator:

Provides an state estimation of the drone based on different sensory inputs and generates a TF-Tree.

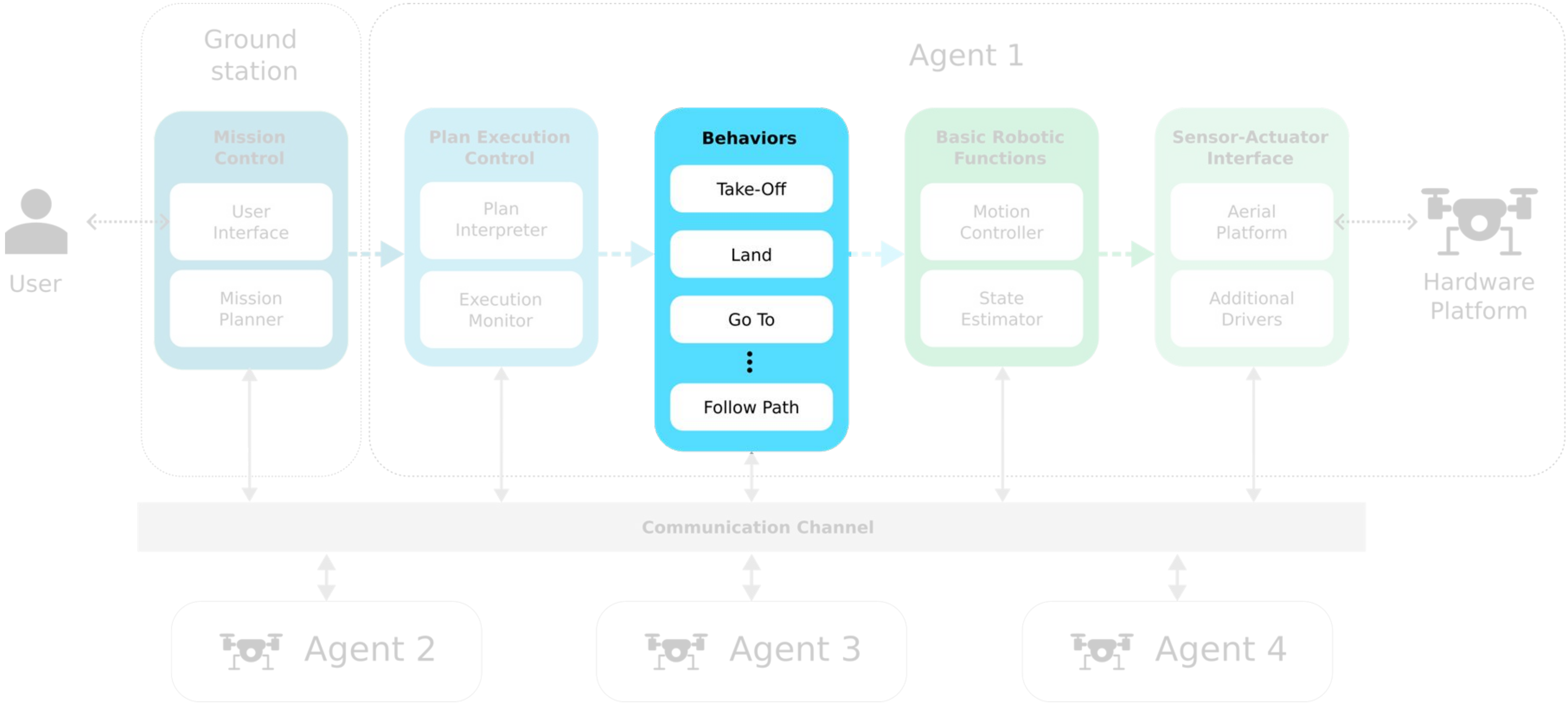
Can load different plugins with different configurations.

For this application we will use **ground_truth** (for simulation) and **raw_odometry** (for real world)

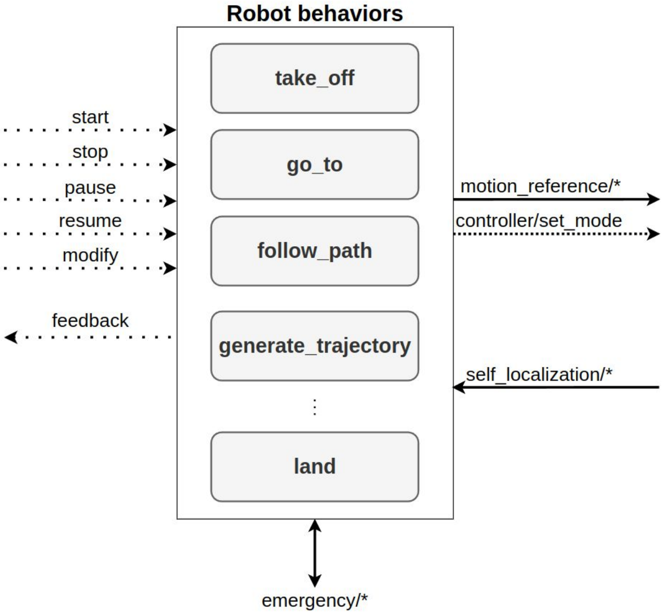


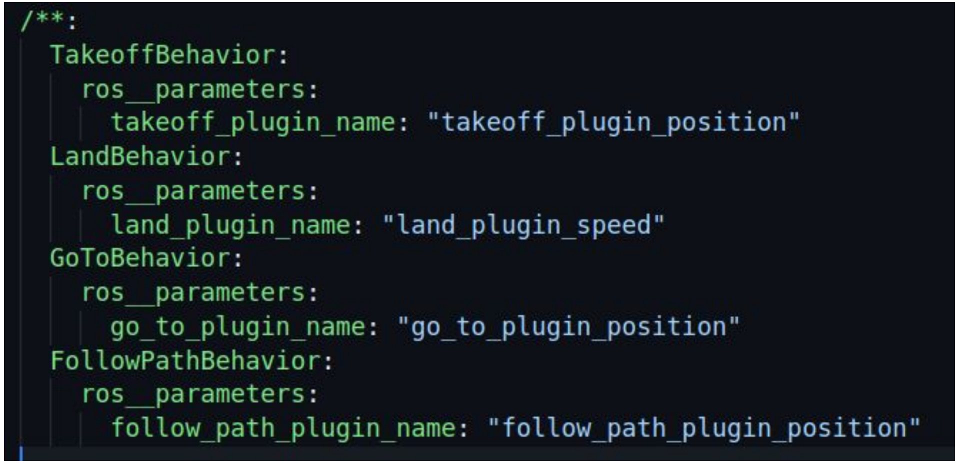
```
ros2 launch as2_state_estimator state_estimator_launch.py
  namespace:=<%= namespace %>
  plugin_name:= raw_odometry
  config_file:=<%= config_folder %>/config_file.yaml
```

Behaviors

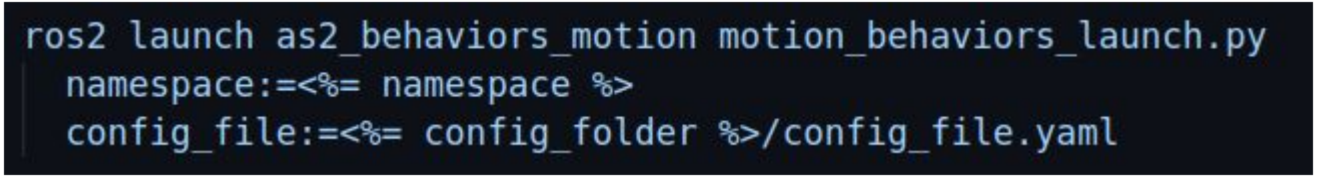

Ground station
Agent 1
User
Mission Control
User Interface
Mission Planner
Plan Execution Control
Plan Interpreter
Execution Monitor
Behaviors
Take-Off
Land
Go To
⋮
Follow Path
Basic Robotic Functions
Motion Controller
State Estimator
Sensor-Actuator Interface
Aerial Platform
Additional Drivers
Hardware Platform
Communication Channel
Agent 2
Agent 3
Agent 4

Behaviors



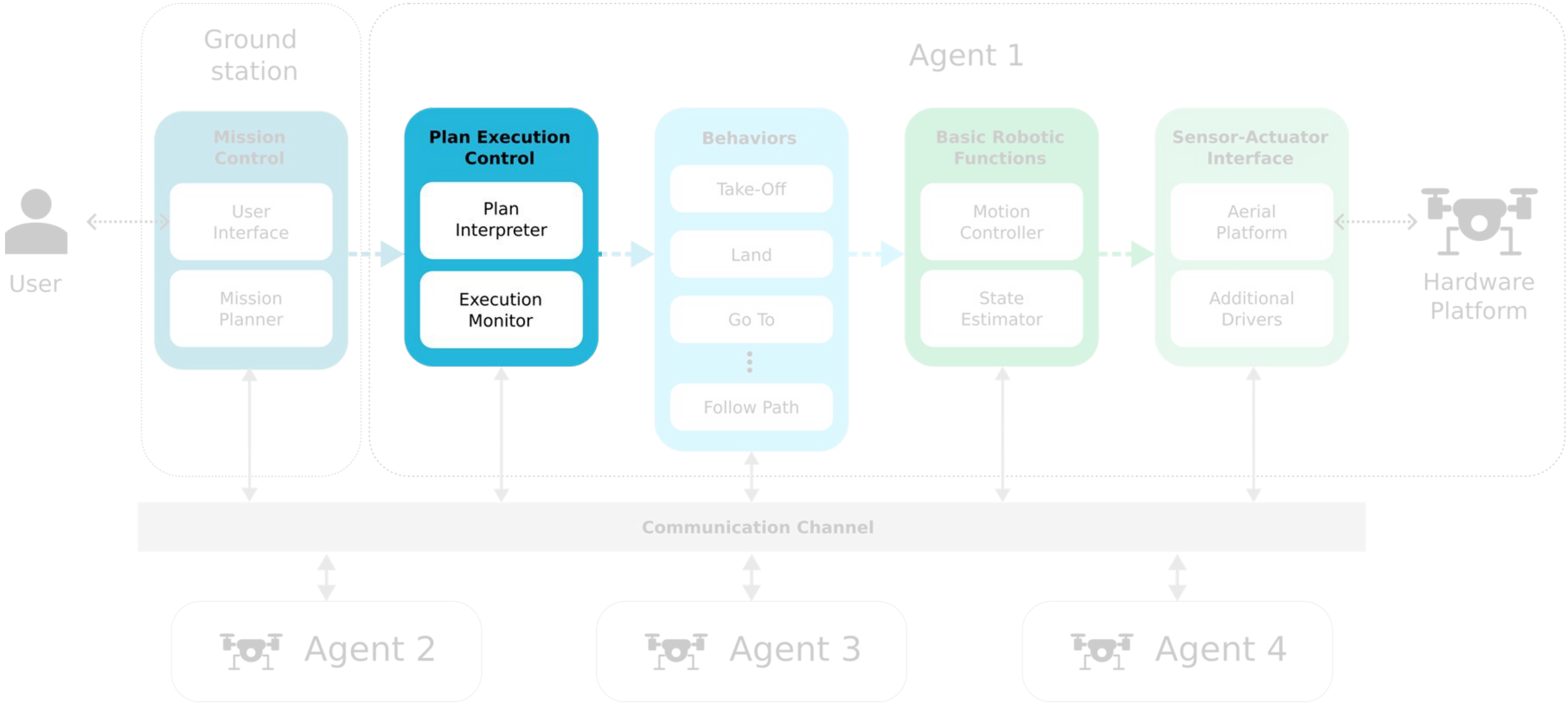


```
/**:
  TakeoffBehavior:
    ros__parameters:
      takeoff_plugin_name: "takeoff_plugin_position"
  LandBehavior:
    ros__parameters:
      land_plugin_name: "land_plugin_speed"
  GoToBehavior:
    ros__parameters:
      go_to_plugin_name: "go_to_plugin_position"
  FollowPathBehavior:
    ros__parameters:
      follow_path_plugin_name: "follow_path_plugin_position"
```



```
ros2 launch as2_behaviors_motion motion_behaviors_launch.py
  namespace:=<%= namespace %>
  config_file:=<%= config_folder %>/config_file.yaml
```

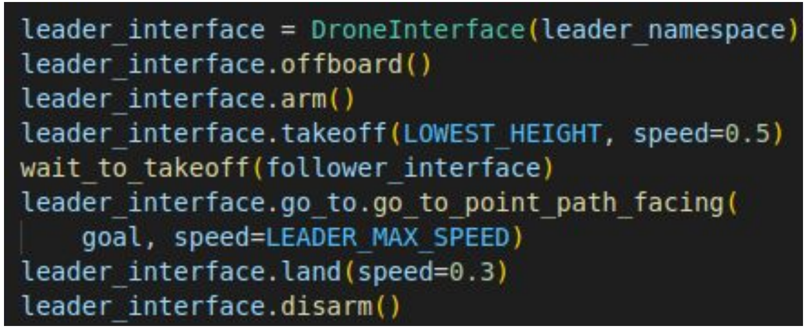
Plan control



Mission plan (Followers)

Leader mission:

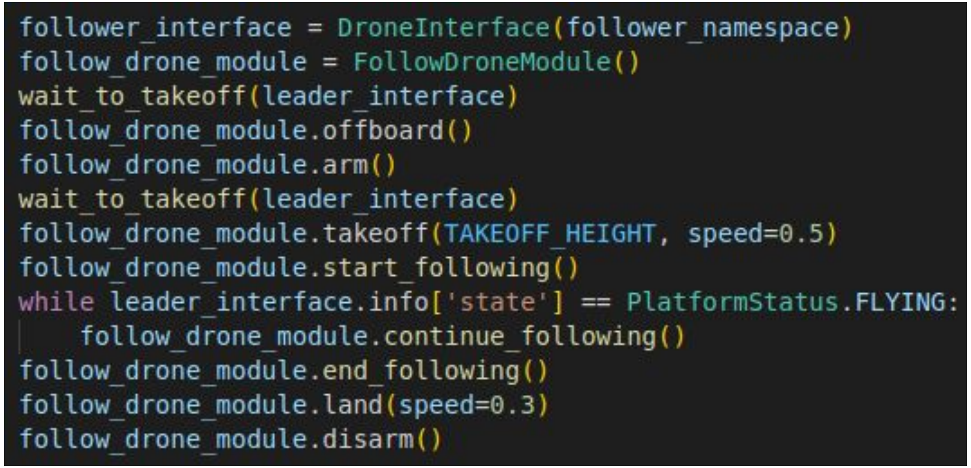
Use Python API to call behaviors.



```
leader_interface = DroneInterface(leader_namespace)
leader_interface.offboard()
leader_interface.arm()
leader_interface.takeoff(LOWEST_HEIGHT, speed=0.5)
wait_to_takeoff(follower_interface)
leader_interface.go_to.go_to_point_path_facing(
    goal, speed=LEADER_MAX_SPEED)
leader_interface.land(speed=0.3)
leader_interface.disarm()
```

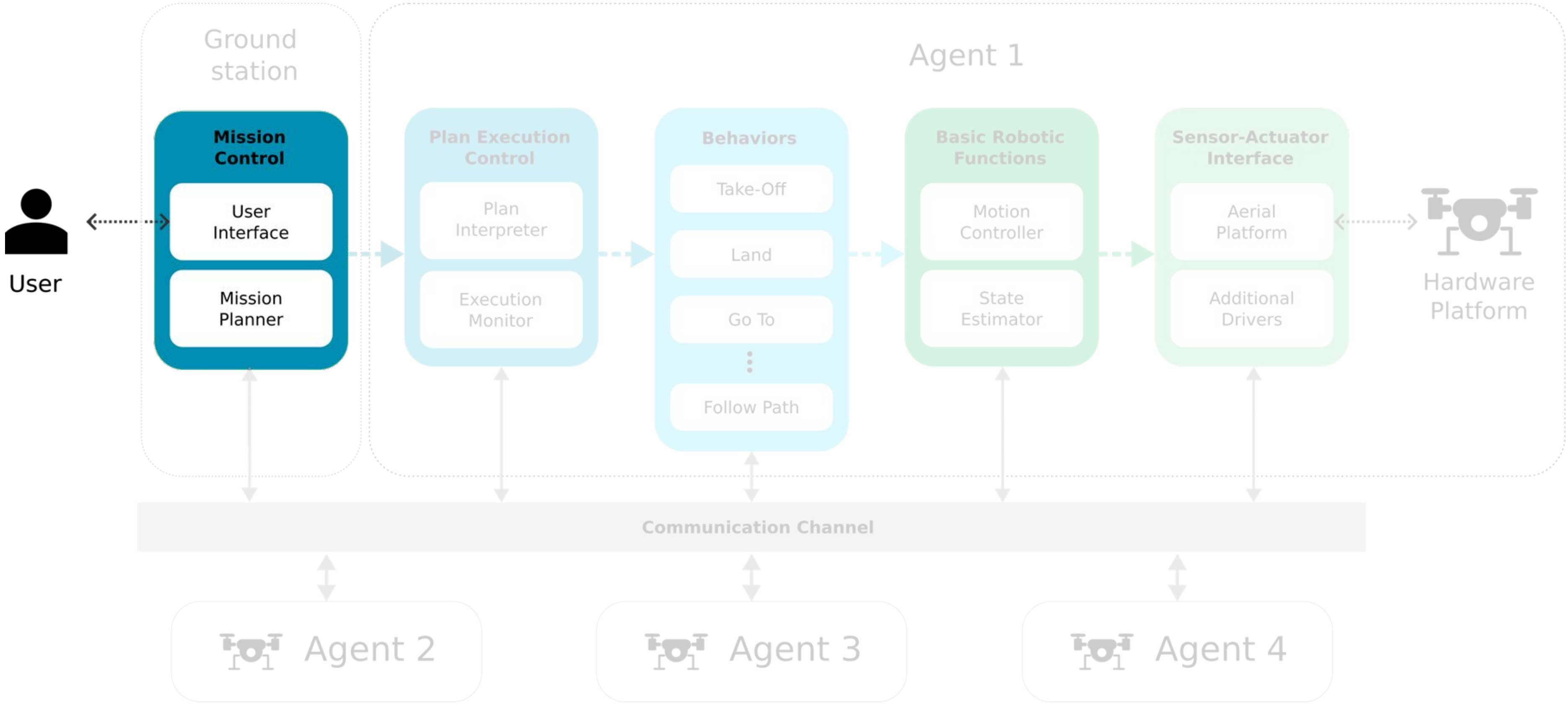
Follower mission:

Use Python API to call behaviors and FollowDroneModule



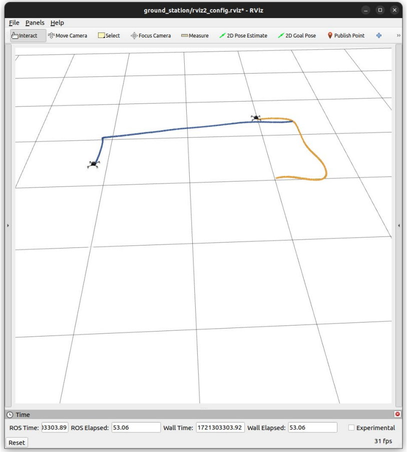
```
follower_interface = DroneInterface(follower_namespace)
follow_drone_module = FollowDroneModule()
wait_to_takeoff(leader_interface)
follow_drone_module.offboard()
follow_drone_module.arm()
wait_to_takeoff(leader_interface)
follow_drone_module.takeoff(TAKEOFF_HEIGHT, speed=0.5)
follow_drone_module.start_following()
while leader_interface.info['state'] == PlatformStatus.FLYING:
    follow_drone_module.continue_following()
follow_drone_module.end_following()
follow_drone_module.land(speed=0.3)
follow_drone_module.disarm()
```

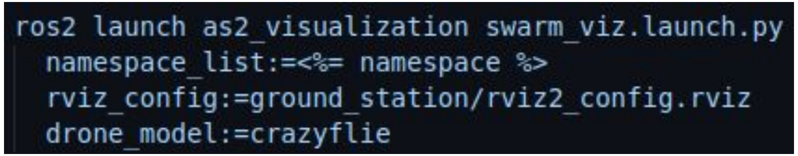
Mission Monitoring



Mission Control

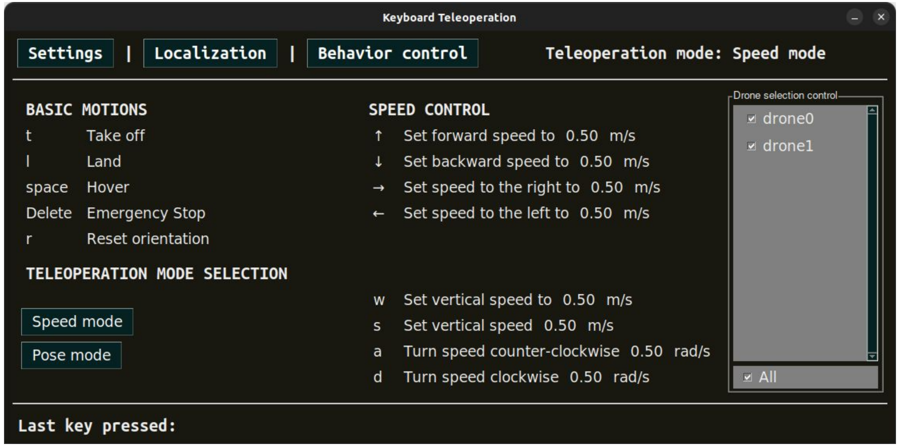
AS2_VIZ (monitoring)





```
ros2 launch as2_visualization swarm_viz.launch.py
  namespace_list:=<%= namespace %>
  rviz_config:=ground_station/rviz2_config.rviz
  drone_model:=crazyflie
```

Keyboard teleoperation



Keyboard Teleoperation

Settings | Localization | Behavior control   Teleoperation mode: Speed mode

BASIC MOTIONS

t	Take off
l	Land
space	Hover
Delete	Emergency Stop
r	Reset orientation

TELEOPERATION MODE SELECTION

Speed mode

Pose mode

SPEED CONTROL

↑	Set forward speed to 0.50 m/s
↓	Set backward speed to 0.50 m/s
→	Set speed to the right to 0.50 m/s
←	Set speed to the left to 0.50 m/s
w	Set vertical speed to 0.50 m/s
s	Set vertical speed 0.50 m/s
a	Turn speed counter-clockwise 0.50 rad/s
d	Turn speed clockwise 0.50 rad/s

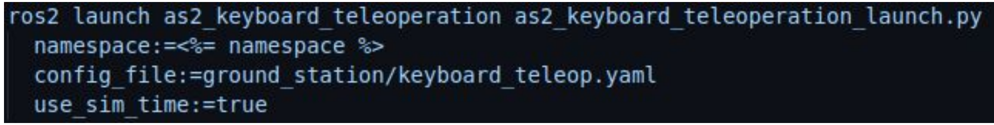
Drone selection control

☑ drone0

☑ drone1

☑ All

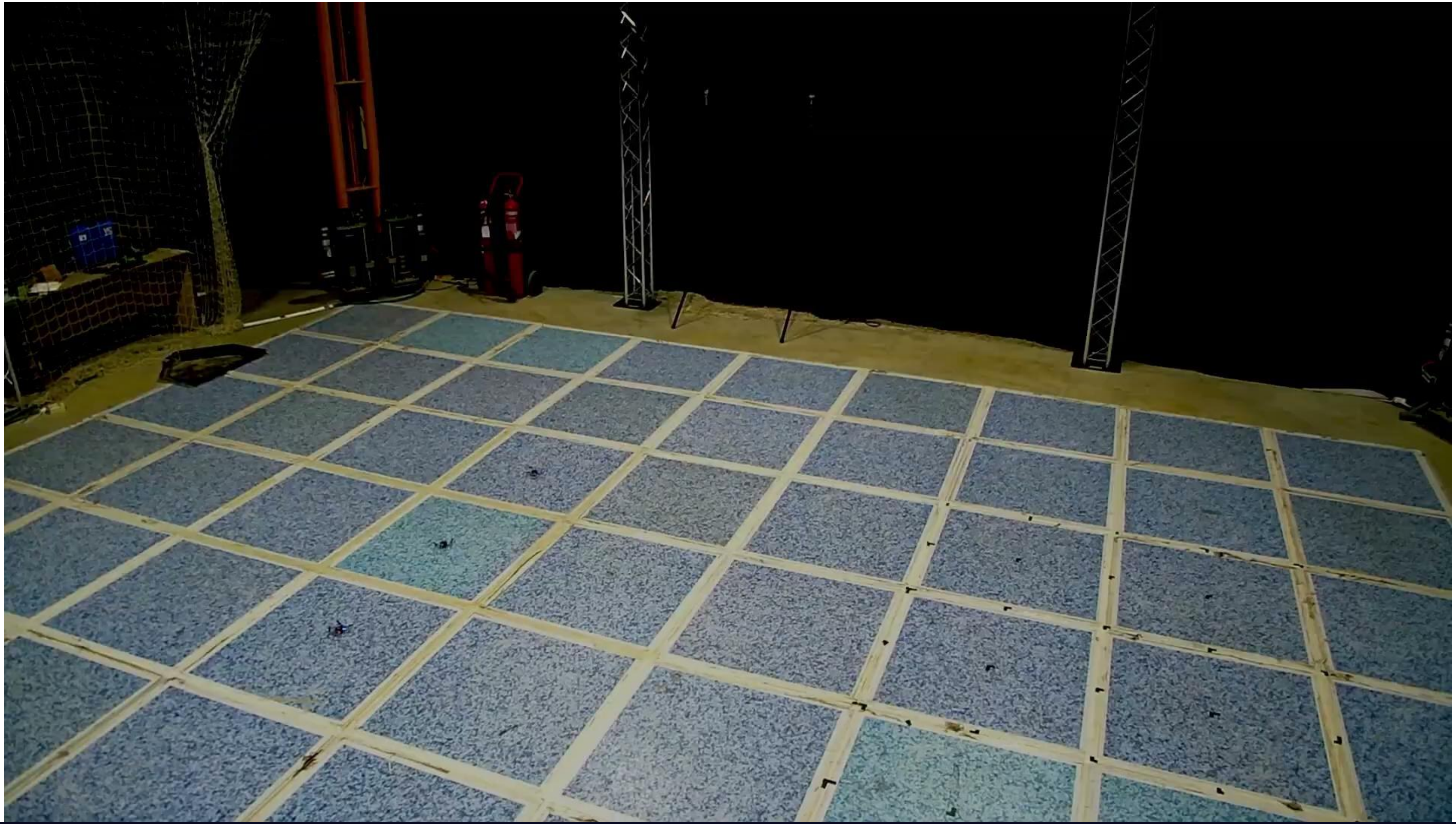
Last key pressed:



```
ros2 launch as2_keyboard_teleoperation as2_keyboard_teleoperation_launch.py
  namespace:=<%= namespace %>
  config_file:=ground_station/keyboard_teleop.yaml
  use_sim_time:=true
```

Launch on simulation

Real World Experiment



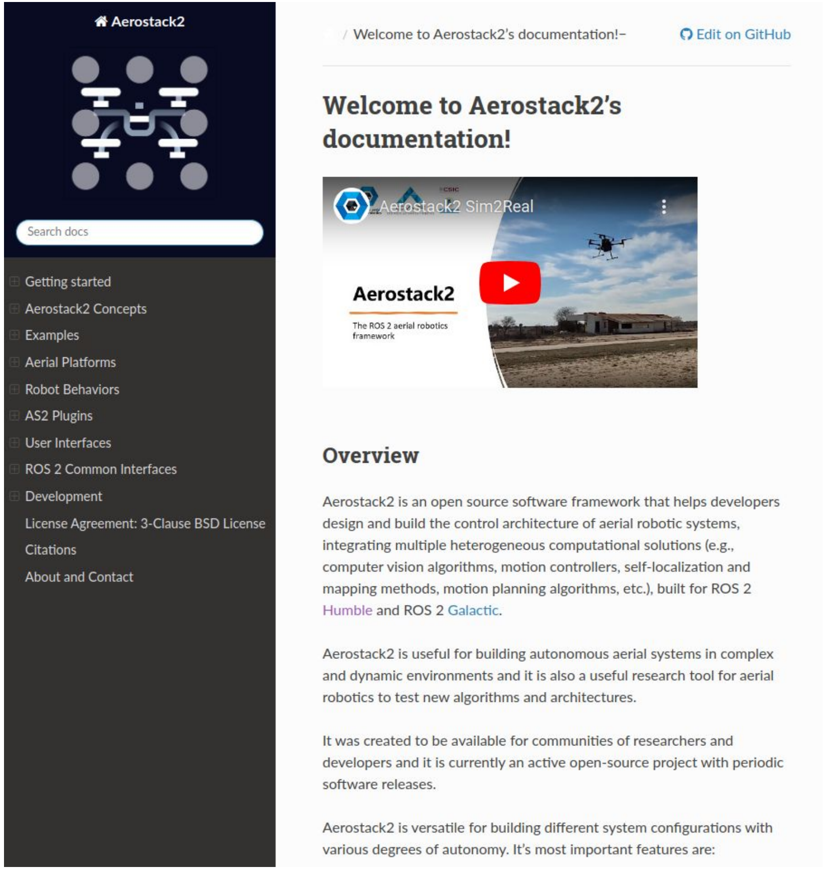
Even harder !!

More info

We are about to release v1.1 with several improvements:

- More platforms (DJI, Mavlink)
- Path planners
- Map generators
- …

Stay tuned for more info !



Welcome to Aerostack2’s documentation!

Overview

Aerostack2 is an open source software framework that helps developers design and build the control architecture of aerial robotic systems, integrating multiple heterogeneous computational solutions (e.g., computer vision algorithms, motion controllers, self-localization and mapping methods, motion planning algorithms, etc.), built for ROS 2 Humble and ROS 2 Galactic.

Aerostack2 is useful for building autonomous aerial systems in complex and dynamic environments and it is also a useful research tool for aerial robotics to test new algorithms and architectures.

It was created to be available for communities of researchers and developers and it is currently an active open-source project with periodic software releases.

Aerostack2 is versatile for building different system configurations with various degrees of autonomy. It’s most important features are:

Aknowlegments

Main contributors:

- Miguel Fernandez-Cortizas
- Pedro Arias-Perez
- Rafael Perez-Segui
- Javier Melero-Deza
- David Perez-Saura
- Martin Molina
- Pascual Campoy

MINISTERIO DE CIENCIA E INNOVACIÓN
Plan de Recuperación, Transformación y Resiliencia
Financiado por la Unión Europea NextGenerationEU
AGENCIA ESTATAL DE INVESTIGACIÓN

Please consider to star and contribute to our project on GitHub!

Thanks for your attention