Home

Edit New Page

Martin Molina edited this page 28 minutes ago · 19 revisions

AEROSTACK

A Software Framework for Aerial Robotic Systems

Gregale

Version 1.0.0
Gregale Distribution

vision4UAV POLITÉCNICA

GPL V3

GOBIERNO DE ESPAÑA MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

What is Aerostack?

Edit New Page

Martin Molina edited this page on 20 Aug · 23 revisions

Aerostack is a 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, planning algorithms, etc.).

Aerostack 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.

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

Multipurpose software framework

Aerostack is versatile for building different system configurations with various degrees of autonomy:

- **From teleoperation to autonomous flight**. Aerostack can be used for teleoperation flights (with manual control) but it can also be used for building autonomous robot systems to perform aerial missions without operator assistance.
- **Single robots or multi-robot systems**. Aerostack can be used to fly a swarm of heterogeneous drones to perform multi-aerial-robot missions. It has been validated to operate with up to five drones simultaneously.
- **Flexible for different applications**. Aerostack can be used by system designers to develop their own systems in a wide range of applications. Aerostack provides languages and graphical tools to configure specific aerial missions.
- **Hardware independent**. Aerostack runs on conventional laptops and it has also run on onboard computers like mastermind or odroid XU3. Aerostack has been used in different aerial platforms, including, but not limited to:
    - Parrot AR Drone 1.0 & 2.0
    - Asctec Pelican
    - Mikrokopter Oktokopter
    - Pixhawk
    - Bebop

Operational features

The main operational features of Aerostack are the following:

- **Library of software components for aerial robotics**. Aerostack provides a library of software components for aerial robotics programmed in C++ and Python. The components are flight-proven and ready to use and the library is open to add new components. For example, these components use general algorithms of aerial robotics for motion control, feature extraction (e.g., with computer vision algorithms), self localization and mapping, motion planning, etc.
- **Compositional modeling**. Aerostack provides an architectural pattern with a combination scheme, based on the composition of configurable building blocks, to develop different specific architectures.

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
    - Configure Map
    - Specify Mission Tasks
    - Configure Launcher
- Execute the Mission:
    - Launch Real Flight
    - Practical Tips for Real Flights
- Mission examples:
    - Simulated Quadrotor
    - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
    - First prize at IMAV 2013
    - Stack pre-release 2013

Clone this wiki locally

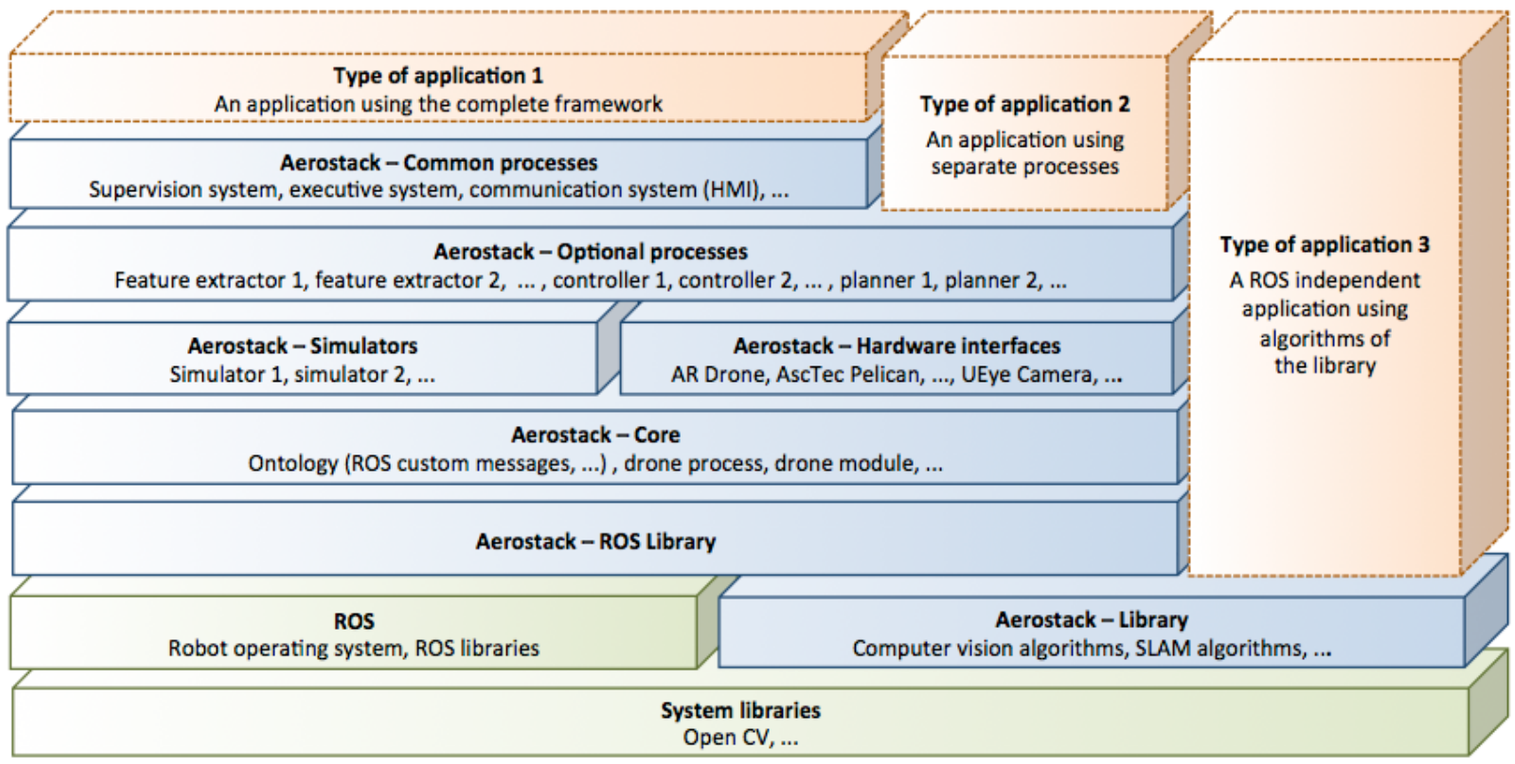
https://github.com/Vision4U

Clone in Desktop

- **Distributed processing**. Distributed processing is supported by the middleware ROS (Robot Operating System). Each software component can be executed in an on-board computer or in ground computers connected in a local area network (LAN) or in a wireless local area network (WLAN), communicated using ROS.

- **Open source**. License GPL v3.

- **Reusable components**. Aerostack uses ROS as the middleware, but we divided all the functionalities into non-ROS libraries and ROS-nodes. That means, all the functionalities and algorithm cores are coded without ROS, in a pure C++ or Python style.



Experience of Use

Aerostack is fully operative, validated by real flight tests and simulations for different types of hardware platforms. Aerostack has been used as a tool in the following situations:

- Develop new applications in aerial robotics in problems such as infrastructure/surface inspection and air quality analysis.
- Create and test new methods in aerial robotics such as the following: using computer vision to track any object with an autonomous UAS, using a coordinator to accomplish high-level missions requested by the user with an autonomous swarm of UAS, experimenting with Natural User Interfaces for Human-Drone Interaction using hand gestures, speech, body movements and visual cues, etc.
- Participate in UAS competitions such as IMAV 2013, and IARC 2014, IMAV 2016 with successful results.

Types of Applications

Developers can use Aerostack to fly UAV platforms in the following way:

- Perform tele-operated flights controlled by user commands using the Human Machine Interface. Examples of user commands are: take off, move forward, move backward, turn, move up, move down, land, etc. The user operates with the keyboard and the mouse of the ground station computer to control the flight of the UAV.

- Tracking a selected object by visual contact. The user can select the object to track by selecting camera images using the Human Machine Interface.

- Following a specific mission plan, defined by the user as a set of waypoints or a set of tasks to be performed in a specific spatial area, with certain limits defined by the user.

- Use a specific physical configuration that modifies significantly the size or weight of the UAV. Our flight controllers can be adapted by the user to the new configuration (in a XML file).

Developers who are familiar with software programming (e.g., C++ language) can use Aerostack to fly new UAV platforms with the following characteristics:

- A UAV with new specific types of sensors. The developer can reuse Aerostack but she/he needs to program and integrate new software modules to process the information from the new sensors.

- A UAV with different physical platform. The developer can reuse Aerostack for different UAV platforms, but she/he needs to program the interfaces between our architecture and the actuators of the new physical platform.

- A UAV with one or several software components that substitutes an existing software component (for example, a new localization and mapping). The developer can reuse Aerostack but she/he might need to substitute the core algorithm used by the previous component within the package of optional processes.

- A UAV with new functionalities that needs additional software components. The developer can reuse Aerostack but she/he needs to program and integrate the new software modules for the new functionalities.

- Developers can also reuse specific algorithms (e.g., perception algorithms or certain controllers) that are encapsulated in their corresponding C++ classes in Aerostack. Developers can reuse these C++ classes with their algorithms to be part of their own development.

Contributors

The main kernel of Aerostack was initially created and pre-released in 2013 by Vision4UAV (Technical University of Madrid - Spain) based on the experience of this group in aerial robotics and computer vision during more than 20 years. It was firstly introduced and demonstrated during the participation of Vision4UAV in the 2013 International Micro Air Vehicle Indoor Flight Competition IMAV2013 (Toulouse, France).

In 2015, with the contribution of other members of the Technical University of Madrid from the Department of Artificial Intelligence, the software was redesigned and extended to create the present version with the name of Aerostack (AErial RObotics STACK). This new version was the evolution from a previous control-oriented architecture to a more structured multi-layered architecture following hybrid paradigms (e.g., with deliberative and reactive layers) and social coordination. The first release of Aerostack (Version 1.0, Gregale distribution), as a mature documented product, was in June 2016.

Currently, a team from the Technical University of Madrid provides support for Aerostack. You can contact the team support at the following address:

- aerostack.vision4uav@gmail.com

The following list includes Aerostack contributors:

Aerostack administrators:

- Jose Luis Sanchez-Lopez
- Hriday Bavle

Research professors:

- Pascual Campoy
- Martin Molina
- Paloma de la Puente

Research assistants:

- Adrian Carrio
- Jean-François Collumeau
- Jesus Pestana Puerta
- Alejandro Rodriguez Ramos
- Carlos Sampedro

- Ramon Suarez-Fernandez

Software developers:

- Jorge Martin [2013-2014]
- Marcos Bernal [2014-2015]
- Yolanda de la Hoz [2014-2015]
- Enrique Ortiz [2015]
- Adrian Navarro [2015]
- Angel Luis Gonzalez [2015]
- Laura Garcia [2015]
- Oscar Fernandez [2015]
- Adrian Diaz-Moreno [2015-2016]
- David Palacios [2015-2016]
- German Quintero [2016]
- Guillermo de Fermín [2016-2017]
- Alberto Camporredondo [2016-2017]
- Carlos Valencia [2016-2017]
- Jorge Pascual [2017]
- Rafael Artiñano [2017]
- Daniel Rabasco [2017]

Funding Support

The development of Aerostack has been partially supported by the Spanish Ministry of Economy and Competitiveness through the project VA4UAV (Visual autonomy for UAV in Dynamic Environments) reference DPI2014-60139-R.

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Getting Started

Edit New Page

Martin Molina edited this page on 18 Jul · 7 revisions

This section shows with a simple example how to install Aerostack and control manually the flight of a drone.

1. Hardware Configuration

In this example we assume that the following hardware is used:

- An air vehicle Parrot AR Drone (other aerial platforms can be used with Aerostack).
- A computer with wifi connection.

2. Software Installation

Aerostack has been tested in a computer with the following software:

- Linux operating system Ubuntu 14.04 (64 bit).
- Robot Operating System (ROS), Jade distribution.

However Aerostack should also work on any other mainstream Linux distributions. The code requires a compiler that is compatible with the C++11 standard.

Aerostack requires additional libraries like: boost and ncurses (among others). Aerostack has ROS packages dependencies like: ardrone_autonomy, opencv2, roscpp, image_transport, sensor_msgs and std_srvs (among others). All the dependencies are automatically installed during the installation process.

The installation follows the same steps needed usually to compile a self-contained ROS stack:

1. Before the installation, you must read and accept the conditions defined by the software License.
2. Download the installation script file:

```
$ git clone https://github.com/Vision4UAV/Aerostack_installer ~/temp
```

3. Run it in a non-ARM architecture computer:

```
$ ~/temp/installation_script.sh
```

4. Re-open the terminal and Aerostack is ready to be used.

If you need to update the software:

1. Go to the Aerostack folder:

Pages 28

Introduction
- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components
- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]
- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]
- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information
- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

```
$ cd $AEROSTACK_STACK
```

2. Update the stack:

```
$ git pull origin master
```

3. Update submodules:

```
$ git submodule update --init --recursive
```

Sometimes, you will need to install new dependencies or update the old dependencies what is done by executing the following command:

```
$ $AEROSTACK_STACK/installation/installation_dep_script.sh
```

> NOTE: Be sure that you are connected to Internet when you compile Aerostack. This is because certain packages (e.g., opentld) need to be updated from remote repositories.

3. A First Flight

To launch and start running the system, the user has to open a new terminal and, then, perform the following tasks:

1. Make sure the drone is on, and reset it if needed.
2. Connect the computer to the drone wifi (e.g., ardrone2_002410).
3. Execute the ROS master node:

```
$ roscore
```

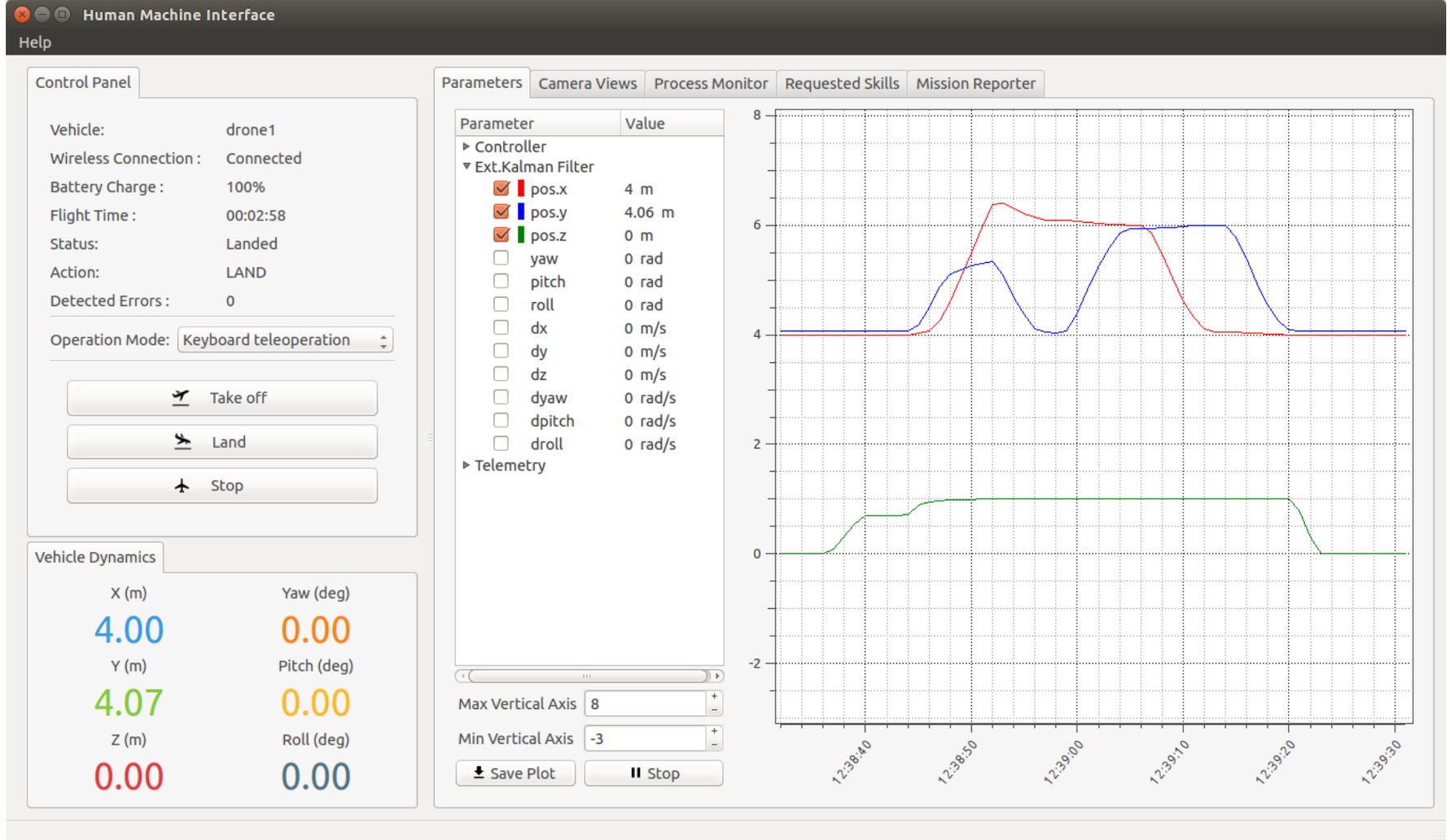
4. In a new terminal window, go to the directory which contains the launchers:

```
$ cd ${AEROSTACK_STACK}/launchers
```

5. Execute the script to launch all processes and the Human Machine Interface:

```
$ ./ardrone_basic.sh
```

Now, the installation is complete and it is possible to operate with the drone through the user interface controls (see details about the operation control in the chapter about the human machine interface).



To stop and terminate all program executions, it is necessary to perform the following steps in the terminal console:

```
$ cd ${AEROSTACK_STACK}/launchers
$ ./stop.sh
```

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

License

Edit New Page

Martin Molina edited this page on 6 Jun 2016 · 5 revisions

Aerostack - A Software Framework for Aerial Robotic Systems

Copyright (C) 2016 Technical University of Madrid

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Safety Warning and Disclaimer

You are using this software at your own risk. The authors decline any responsibility for personal injuries and/or property damage.

Some drones supported by this framework ARE NOT TOYS. Even operation by expert users might cause SERIOUS INJURIES to people around. So, please consider flying in a properly screened or isolated flight area.

Third-Party Libraries

Aerostack uses other third-party open-source libraries:

- Robot Operating System (ROS), license: `BSD` license, not included in the source of the stack.
- ardrone_autonomy ROS package (ardrone_autonomy), license: `BSD` license, not included in the source of the stack.
- Open Source Computer Vision library (OpenCV), license: `BSD` license, not included in the source of the stack.
- ArUco: a minimal library for Augmented Reality applications based on OpenCv (ArUco), original license: `BSD` license, included as source in the stack in the `lib_aruco` package.
- pugixml library (pugixml), original license: `MIT` license, included as source in the stack in the `lib_pugixml` package.
- newmat11 library, a matrix library in C++ (newmat11), original license: (newmat11 license), included as source in the stack in the `lib_newmat` package.
- other software libraries used by this stack also uses licenses similar to `BSD` and `MIT` licenses: ncurses and boost.

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

List of distributions

Edit New Page

Martin Molina edited this page on 8 Sep · 2 revisions

Distribution	Release date	Main features	Tested platforms	Requirements
Version 1.0 Gregale Distribution	June, 2016	Aerial robot library of software components (perception, control, planning, etc.), TML language for mission plan specification, human machine interface (HMI) for user-robot interaction.	Parrot AR Drone 1.0 and 2.0, Asctec Pelican, Mikrokopter Oktokopter, Pixhawk.	Ubuntu 14.04, ROS Jade
Version 2.0 Levant Distribution	Planned to be released in September or October 2017	Behavior-based execution system, graphical user interface (GUI) for environment maps and mission plans, Python API for mission plans.	Parrot AR Drone 2.0, Pixhawk, Parrot Bebop 2 (with SLAM-dunk).	Ubuntu 16.04, ROS Kinetic

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Aerostack Virtual Machine

Edit New Page

Jose Luis Sanchez Lopez edited this page on 7 Jul 2016 · 2 revisions

In the following link you can download a Virtual Machine (Virtual Box VM) with Aerostack installed: https://drive.google.com/open?id=0BziAte9fn_WzQk41cTJaYk85dlE

Features:

- Architecture: x64
- OS: Ubuntu 14.04
- ROS: Jade
- Aerostack

Host:

- Architecture: x64
- OS: Linux Ubuntu 14.04

User and password:

- User: aerostack
- Pass: paerostack

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Aerostack Architecture

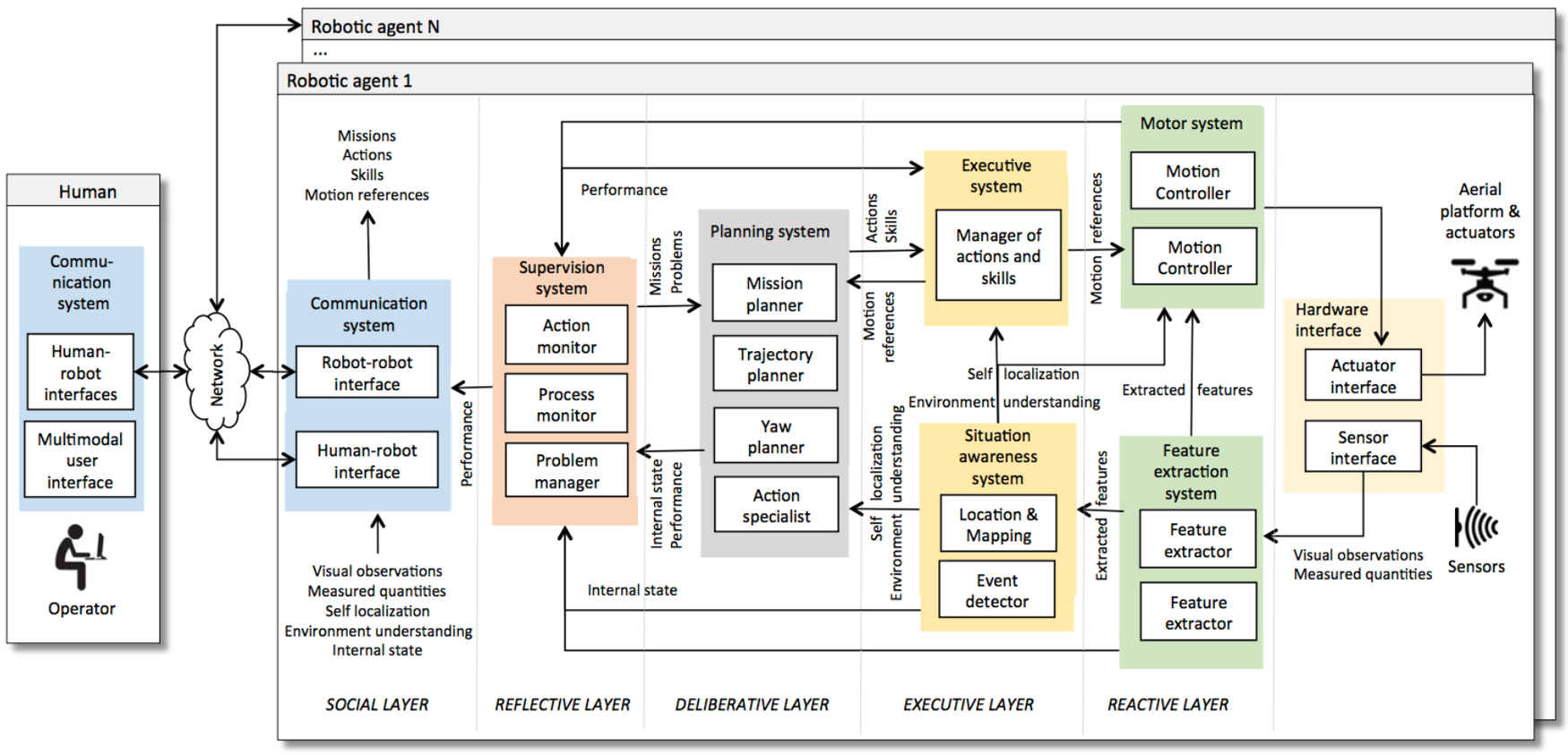
Edit New Page

Martin Molina edited this page on 18 Jul · 6 revisions

This section summarizes the main components of the Aerostack architecture and its structural organization.

Main Components

The next figure shows the main functional components of the architecture. This architecture follows the hybrid reactive/deliberative paradigm, i.e., an architecture that integrates both a deliberative and reactive approaches [Arkin et al., 1987]. Our design includes five layers: reactive, executive, deliberative, reflective and social. The first three layers correspond to the popular hybrid design known as the three layer architecture [Gat ,1998] that organizes algorithms according to whether they contain no internal state (reactive layer), contain state reflecting memories about the past (executive layer) or contain state reflecting predictions about the future (deliberative layer).



In particular, the Aerostack architecture includes the following layers [Russel, Norvig, 2003]: (1) reactive layer with low-level control with sensor-action loops; (2) executive layer (or sequencing layer) that accepts directives from the deliberative layer and sequences them for the reactive layer; this layer also integrates the sensor information into an internal state representation; and (3) the deliberative layer generates global solutions to complex tasks using planning.

The reactive layer is a sensor-action loop that includes perceptual schemas (in the feature extraction system) and controllers (in the motor system):

- Perceptual schemas may read simple states of sensors or may implement complex vision and pattern recognition algorithms. Examples of perceptual schemas are: read bumper, extract color, compute centroid of an image (white), recognize Aruco visual marker, detect power line tower, detect IARC ground robot, detect IARC obstacle, etc. A perceptual schema can integrate a set of perception procedures using different combination methods (e.g., fusion, sequence, etc.).
- Maneuver controllers typically implement combinations of Proportional-Integral-Derivative (PID) controllers (e.g., cascade controllers). For example, these type of controllers can accept orders about a desired value for a variable (position, speed, altitude, and yaw) in form of single commands or simultaneous commands that are translated into low level commands to be sent to the actuator interface. Controllers are used by the behavior manager. The behavior manager receives a high level command about a desired action (e.g., follow a trajectory, move to a desired position, etc.) and translates it into setpoints for controllers.

Pages 28

Introduction
- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components
- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]
- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]
- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information
- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Besides the mentioned three layers, the architecture includes other two, inspired by certain cognitive architectures in artificial intelligence. We include a self reflective layer [Singh, Minsky, 2005] to supervise the first three layers and include fault recovery actions. A social layer is also included with communication abilities, as it is proposed in multiagent systems and other architectures with social coordination (e.g., [Duffy, 2000]).

The architecture is also consistent with the usual components related to guidance, navigation and control of unmanned rotorcraft systems [Kendoul, 2012]. In particular, the Navigation System (NS) corresponds to our perception system and situation awareness system, the Guidance System (GS) corresponds to our executive system, planning system and supervision system and, finally, the Flight Control System (FCS) corresponds to our motor system.

Architecture Organization: Processes and Systems

Aerostack has been developed as a multi-process architecture supported by ROS (Robotic Operating System). The basic component of the architecture is the process. We distinguish between the intuitive meaning of a process (what represents) and its computational support (how it is implemented). In general, the intuitive meaning of a process is a series of actions taken in order to achieve a particular end. A process acts through time to change certain parameters of objects (e.g., certain physical quantities), consuming resources (e.g., time or space) to convert inputs into outputs. It is important to note that a process has a function, i.e., a purpose or practical use for which the process is designed or exists (the intention or the objective of the process).

The idea of process is an appropriate concept to describe the components of the architecture of Aerosack. It helps to divide the whole problem of automated support for UAVs into partial functional roles. Each process in Aerostack is named as an active processor with its main function, for example: mission planner (main function: planning a mission) or obstacle recognizer (main function: recognize obstacles).

The computational support of a process is designed as an atomic executable unit (a data processor) that receives input data and, as a result of a certain information processing, generates output data. A process in Aerostack corresponds to the concept of a ROS node. The idea of a process is also similar to the concept of an atomic functional block used in SysML with input/output ports. When Aerostack is running for a particular UAV, each process is an executable instance of a program running in a computer (each process can also call subprocesses, child computational processes).

Processes are grouped in systems. A system is a complex module that includes a set of interconnected processes that provides a common functionality. The idea of system is also similar to the concept of functional block (composite block) used in SysML, with input/output ports.

To consult the available processes provided by Aerostack, consult the following catalog:

- Catalog of Available Processes

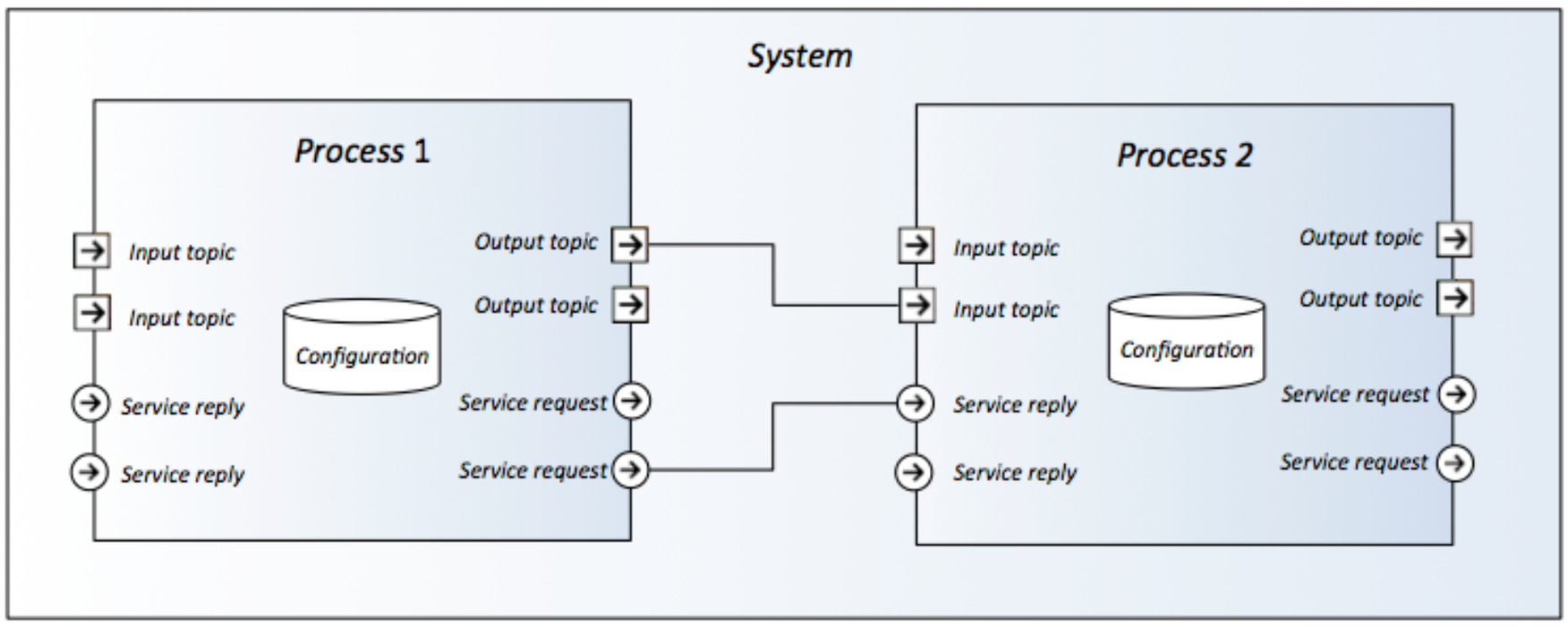
Inter-Process Communication

In general, the inter-process communication (IPC) in multitasking operating systems, where different processes run concurrently, admits different communication mechanisms such as pipes, message queues, semaphores, sockets, shared memory, etc. Aerostack uses communication methods provided by ROS:

- Topics. Processes can communicate input and output data using topics. According to ROS, topics are named buses over which processes exchange messages. In general, processes are not aware of who they are communicating with. Instead, processes that are interested in data read (or subscribe) the relevant topic and processors that generate data write (or publish) to the relevant topic. Processes can communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields. Messages can include arbitrarily nested structures and arrays. Files .msg files are simple text files for specifying the data structure of a message.

- Services. A service is a communication mechanism between processes based on a request-reply scheme. A service is defined by a pair of messages: one for the request and one for the reply.

Next figure shows a graphical notation to describe the inter-process communication of data processors of the Aerostack architecture. This notation is based partially on functional block diagrams and SysML (internal block diagrams).



References

- Arkin, R. C., Riseman, E. M., and Hansen, A. (1987): "AuRA: An Architecture for Vision-Based Robot Navigation," proceedings of the DARPA Image Understanding Work-shop, Los Angeles, CA, February, pp. 417–413.
- Duffy, B.R. (2001). "The Social Robot". Doctoral dissertation, Dept. of Computer Science, University College Dublin, Ireland.
- Gat, E. (1998): "On Three-Layer Architectures". In "Artificial Intelligence and Mobile Robots", (David Kortenkamp, R. Peter Bonnasso, and Robin Murphy, eds.) AAAI Press.
- Kendoul, F. (2012). Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. Journal of Field Robotics, 29(2), 315-378.
- Qi, X., Theilliol, D., Qi, J., Zhang, Y., Han, J., Song, D., Wang, L., Xia, Y. (2013): "Fault diagnosis and fault tolerant control methods for manned and unmanned helicopters: a literature review". Proceedings of the Conference on Control and Fault-Tolerant Systems (SysTol) (pp. 132-139), October 9-11, 2013, Nice, France. IEEE.
- Russel, S; Norvig, P (2003): "Artificial Intelligence: A Modern Approach (Second Edition)" Prentice Hall.
- Sadeghzadeh, I., Zhang, Y. M. (2011): "A review on fault-tolerant control for unmanned aerial vehicles (UAVs)". Infotech@ Aerospace, St. Louis, MO.
- Sanchez-Lopez, J.L.,; Pestana, J.; de la Puente, P.; Campoy, P. (2015): "Reusable Open-source System Architecture for the Fast Designing and Prototyping of Autonomous Multi-UAV Systems: Simulation and Experimentation". Journal of Intelligent and Robotic Systems.
- Sing P., Minsky M. (2005). "An Architecture for Cognitive Diversity". In "Visions of Mind". Edited by Darryl Davis. London: Idea Group.

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Catalog of Available Processes

Martin Molina edited this page on 29 May · 2 revisions

This catalog shows the available processes provided by Aerostack organized according to robot capabilities. The catalog also shows the launcher and the system corresponding to each process. In order to have a desired capability for a particular robot, the developer must select the launchers of processes described in this page.

Capability: Operation with an aerial platform

Option: Operation with Parrot AR Drone 2.0

Process	System	Launcher
Drivers platform	Hardware interface	driverParrotARDroneROSModule.launch

This option also requires the launcher:

`${AEROSTACK_STACK}/launchers/ardrone_launch/ardrone_indoors.launch`

> NOTE: Other aerial platforms not described here can also be used with Aerostack (e.g., Asctec Pelican, Bebop, etc.).

Option: Operation with a RViz-based simulator

Process	System	Launcher
Simulator	Simulation system	SimulatorRvizROSModule.launch

Capability: Aruco visual markers recognition

Process	System	Launcher
Aruco eye	Feature extraction	droneArucoEyeROSModule.launch

Capability: Self-localization

Option: Self-localization by Aruco visual markers

Required capability: **Aruco visual markers recognition**

Process	System	Launcher
Visual markers localizer	Situation awareness	droneVisualMarkersLocalizerROSModule.launch
Obstacle detector visual marks	Situation awareness	droneObstacleProcessor2dVisualMarksROSModule.launch
Obstacle distance calculator	Situation awareness	droneObstacleDistanceCalculationROSModule.launch
Self localizer	Situation awareness	droneEKFStateEstimatorROSModule.launch

Capability: Setpoint-based flight control

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Required capability: **Self-localization**

Process	System	Launcher
Basic actions controller	Motor system	droneMidLevelAutopilotROSModule.launch
Trajectory controller	Motor system	droneTrajectoryControllerROSModule.launch

Capability: Vision-based flight control

Process	System	Launcher
Tracker eye	Feature extraction	droneTrackerEyeROSModule.launch
Visual servoing control	Motor system	DroneIBVSControllerROSModule.launch
OpenTLD translator	Motor system	droneOpenTLDTranslatorROSModule.launch

Required launchers:

```
${AEROSTACK_STACK}/launchers/ardrone_launch/launch_files/opentld_for_IBVSController.launch
```

```
${AEROSTACK_STACK}/launchers/ardrone_launch/launch_files/opentld_gui_for_IBVSController.launch
```

NOTE: To activate this Feature for the AR Drone it is necessary to copy the files from the config folder:

`ardrone_camera_bottom.yaml`

`ardrone_camera_front.yaml`

to the folder in the home directory:

`.ros/camera_info/`

Capability: Path planning

Required capability: **Setpoint-based flight control**

Process	System	Launcher
Trajectory planner	Planning system	droneTrajectoryPlanner2dROSModule.launch
Yaw commander	Planning system	droneYawCommanderROSModule.launch

Capability: Action execution

Required capabilities: **Path planning**, **Setpoint-based flight control**, **Vision-based flight control**

Option: Execution of simple actions

Process	System	Launcher
Manager of actions	Executive system	droneManagerOfActionsROSModule.launch

Option: Execution of advanced actions

Process	System	Launcher
Behavior manager	Executive system	behavior_manager_process.launch
Manager of actions	Executive system	droneManagerOfActionsROSModule.launch
Action monitor	Supervision system	action_monitor_process.launch

Capability: Mission plan interpretation

Option: Mission schedule processor

Required capability: **Action execution (Option: Execution of simple actions)**

Process	System	Launcher
Mission planner	Planning system	droneMissionSheduleProcessorROSModule.launch

Option: Task-based mission plan interpreter

Required features: **Action execution (Option: Execution of advanced actions)**

Process	System	Launcher
Mission planner	Planning system	task_based_mission_planner_process.launch
Action specialist	Planning system	action_specialist_process.launch

Capability: User interface of ground control station

Option: Alphanumeric user interface

Process	System	Launcher
Graphical User Interface	Ground control system	droneInterface_jp_ROSModule.launch

Option: Graphical User Interface

Process	System	Launcher
Graphical User Interface	Ground control system	human_machine_interface_process.launch

Option: Speech-based communication

Process	System	Launcher
Speech module	Ground control station	droneSpeechROSModule.launch
Sound module	Ground control station	droneSoundROSModule.launch

Required launcher:

`${AEROSTACK_STACK}/launchers/sound_play.launch`

Option: Communication with leap motion

Process	System	Launcher
Leap motion	Ground control station	droneLeapMotionDroneControlUserInterfaceROSModule.launch

Capability: Swarm Coordination

Process	System	Launcher
Communication manager	Communication system	droneCommunicationManagerROSModule.launch

Capability: Process monitoring

This capability must be always active.

Process	System	Launcher
Process monitor	Supervision system	process_monitor_process.launch

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Human Machine Interface

Edit New Page

Martin Molina edited this page on 14 Jun 2016 · 1 revision

This page describes how the operator can interact with UAV platforms using the human machine interface (HMI). The HMI allows the interaction with the drone, observing states and dynamics. The HMI also presents graphical views and images to help the user to understand both the external and internal behavior of the drone.

The Human Machine Interface (HMI) offers to the drone operators a bi-directional communication with each aerial robotic agent. Both communication directions are equally important because:

- The communication from the robotic agent to the operator is needed to monitor its state and the performance of the requested mission. This will allow the operator to actuate in case of a need to redefine or abort the mission. A good user graphical interface simplifies this task.
- The communication from the operator to the robotic agent is needed if a non-rigid behavior is required to accomplish the goal, making it possible to redefine the mission and also to start, stop, abort the mission or repair certain unexpected malfunctions that may occur during the mission execution.

In general, the operator can use a graphical user interface to perform the following types of tasks:

- Specify the vehicle's behavior in advance (vehicle set up). Before a mission starts, the operator can specify what kind of behavior the drone must follow by selecting an operation mode (teleoperated, guided by mission plan), configure active capabilities and select a mission to be done.
- Monitor the vehicle's behavior during a complex mission. The operator can monitor the state and the performance of a requested mission. Here, a non-rigid behavior is required, making it possible to redefine the mission and also to start, stop, abort the mission or repair certain unexpected malfunctions that may occur during the mission execution. A good graphical user interface simplifies this task.
- Operate manually with simple movements. The operator can operate manually the vehicle using the mouse and the keyboard to send basic motion commands (move forwards, backwards, etc.) and emergency actions (land, stop, reset, etc.).
- Collect data for later use. The operator can also use the GUI to collect relevant data to be used later. This includes, for example, images (photographs, videos) or data from other types of sensors (lidar, temperature, etc.).

The HMI includes some procedures used in most of the standard user interfaces to operate with aerial vehicles. The HMI also includes other specific features. For example, when an operator monitors the vehicle's behavior during a mission, it is possible to distinguish between different levels of descriptions:

- *Mission level*. This level corresponds to the language used to describe the mission goals. Here the operator and the vehicle use the concepts such as the following: mission, name of the mission; task, a particular step of the mission (e.g., go to the building); skill, a capability of the vehicle (e.g., avoid certain types of obstacles, or interpret visual markers); action, an elementary task that can be performed (e.g., go to a certain point), and event, a situation that can require special attention.

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

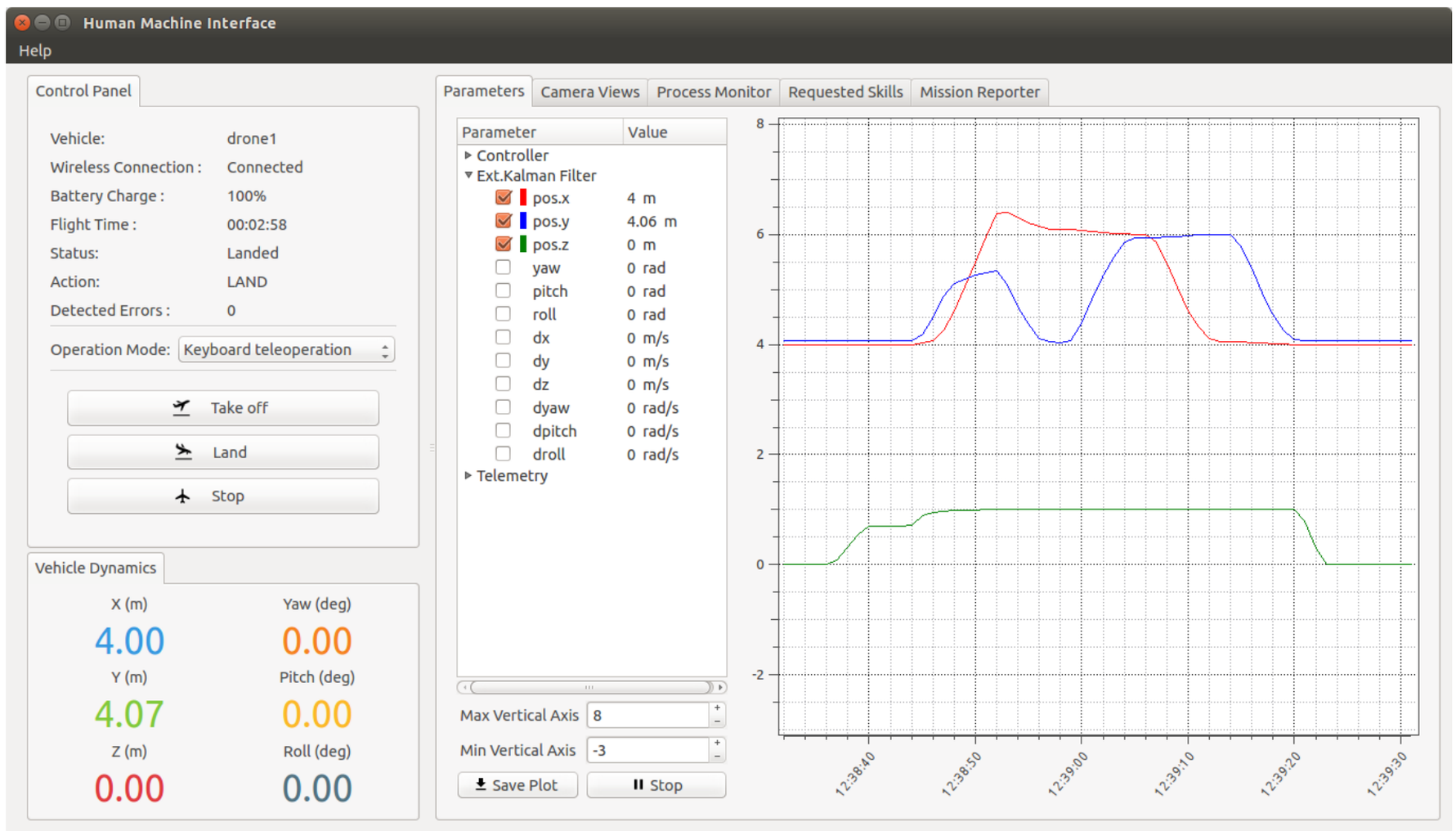
- *Operation level*. This level corresponds to observed facts or derived facts by the vehicle during its operation. Here the operator uses the following concepts: internal states, (e.g., battery state, armed or disarmed, flying or landed, state of wifi connection, etc.), parameters measured by sensors or computed by specific algorithms (pose, yaw, pitch, roll, etc.), and images obtained from cameras.

- *Software level*. This corresponds to the internal behavior of the vehicle, related to the execution of running processes and internal error messages.

The HMI presents information at the different levels. This requires the corresponding translation procedures and specific data structures. For example, the HMI keeps a relation between each skill (at mission level) and the corresponding set of running processes (at software level). Operators who are not familiar with software architectures can consult information at mission level, which is more abstract, avoiding specific details of software execution. On the other hand, operators who are interested in software debugging and maintenance can use the software level to get more details about the mission execution.

1. Windows Layout

The figure below shows a sample screen of the user interface with the windows layout with main parts: the control panel on the upper left side of the image, the dynamics viewer on the lower left side, the windows for detailed content on the right hand side of the screen with different tabs (parameters, the camera viewer or the performance monitor). At the top, there are drop-down menus (file, view, settings, etc.) to perform additional tasks related to standard procedures such as load and save files, help about user commands, etc. For example, in the settings menu the user can establish the connection between the GUI and the vehicle.



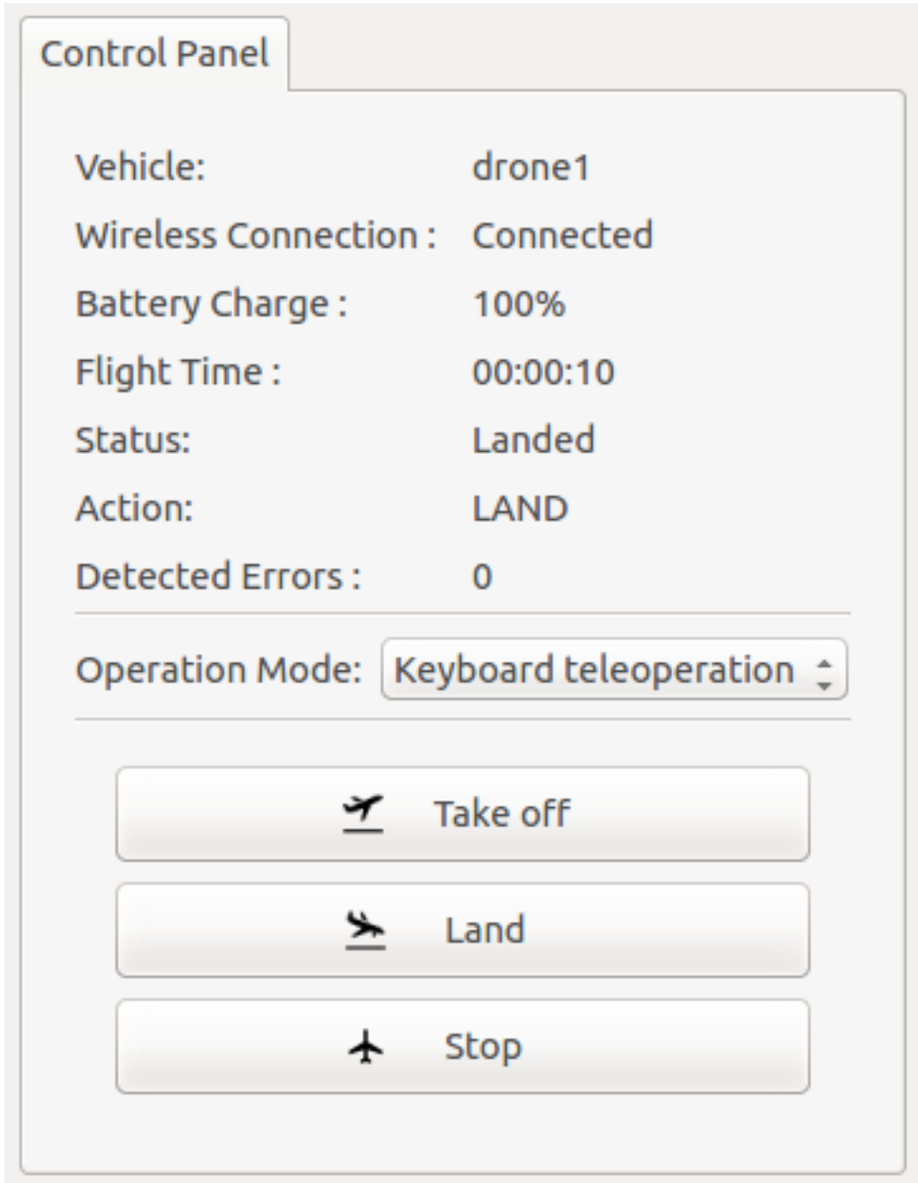
2. The Control Panel

The control panel shows the general state of the UAV and the main control commands. This panel provides a summary of information considered critical that needs to be permanently present on the screen to be easily available to the operator.

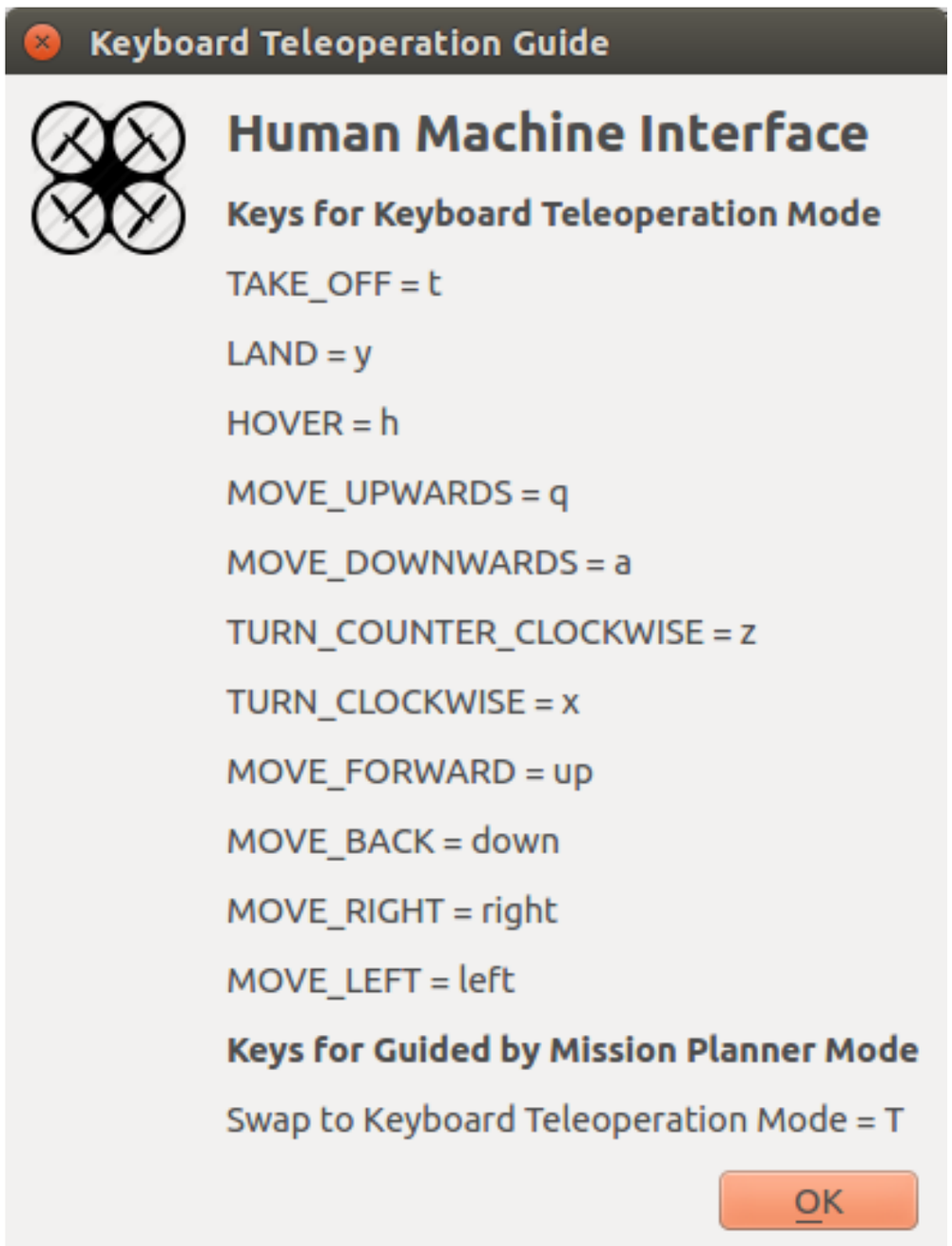
The control panel is divided into three main areas: general state of the vehicle, operation mode and operation commands. The general state of the vehicle shows information such as wireless connection (connected, disconnected), battery charge (percentage of remaining battery charge), flight time (total time the vehicle has been flying), current status (landing, taking off, , etc.), action (name of the current action) and detected errors (the number of detected errors detected during the mission).

In the central area, the operator can change the operation mode: keyboard teleoperation (the user controls the movements using the keyboard) or guided by mission planner (the mission planner takes the control of the movements).

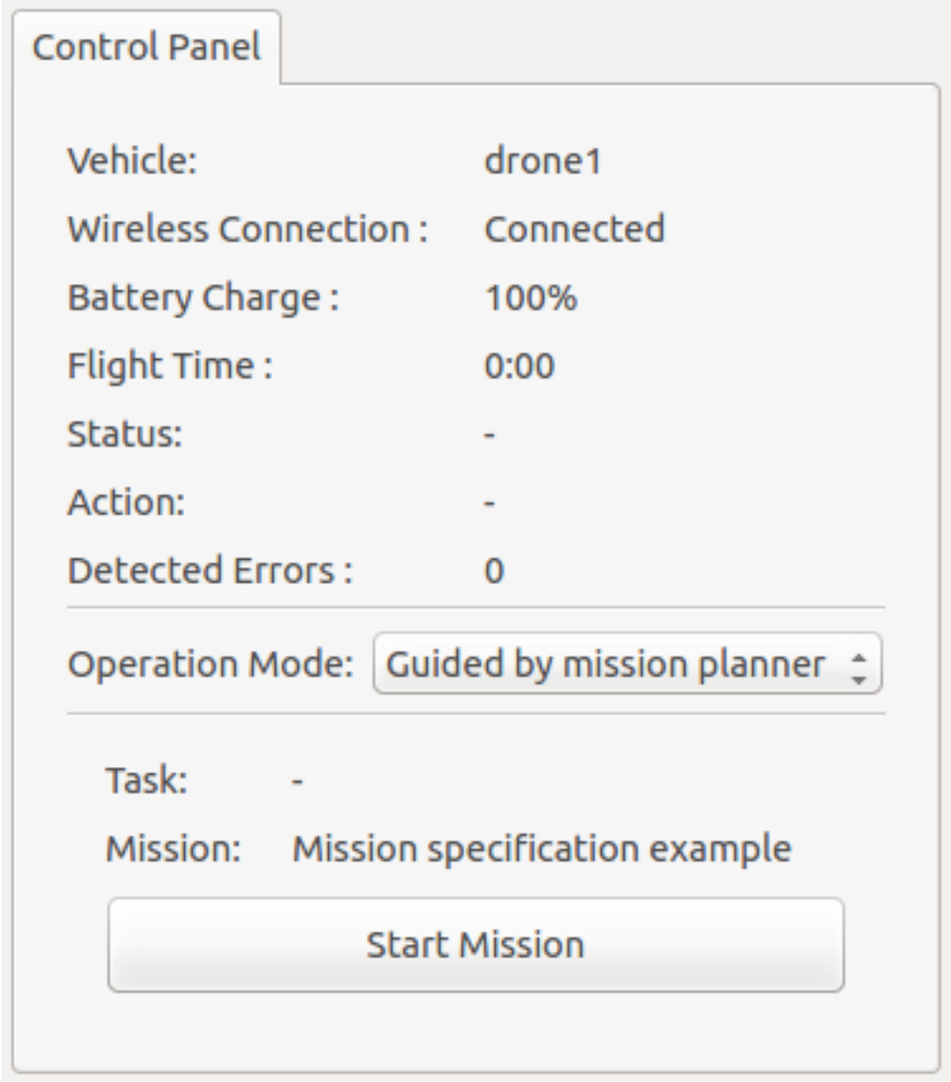
If the operation mode is keyboard teleoperation, the area for operation commands shows buttons for different actions: take off, land and stabilize (reset the vehicle position with zero values for rotation angles).



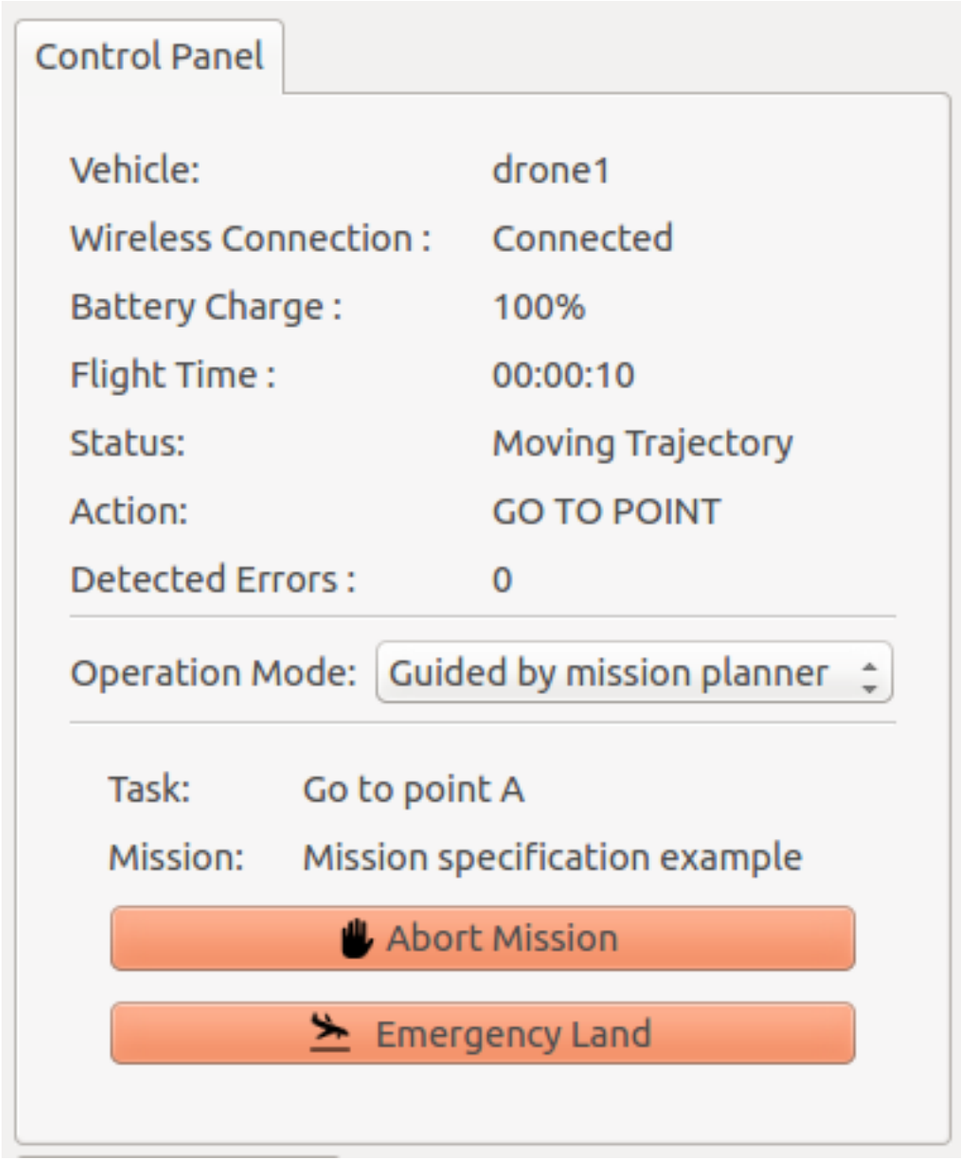
Besides these actions, the operator can control the movements using the following keys (this guide can be shown by clicking at the top of the screen, in the help menu):



If the operation mode is guided by the mission planner, the area for operation commands shows a button to start the mission and the mission name.

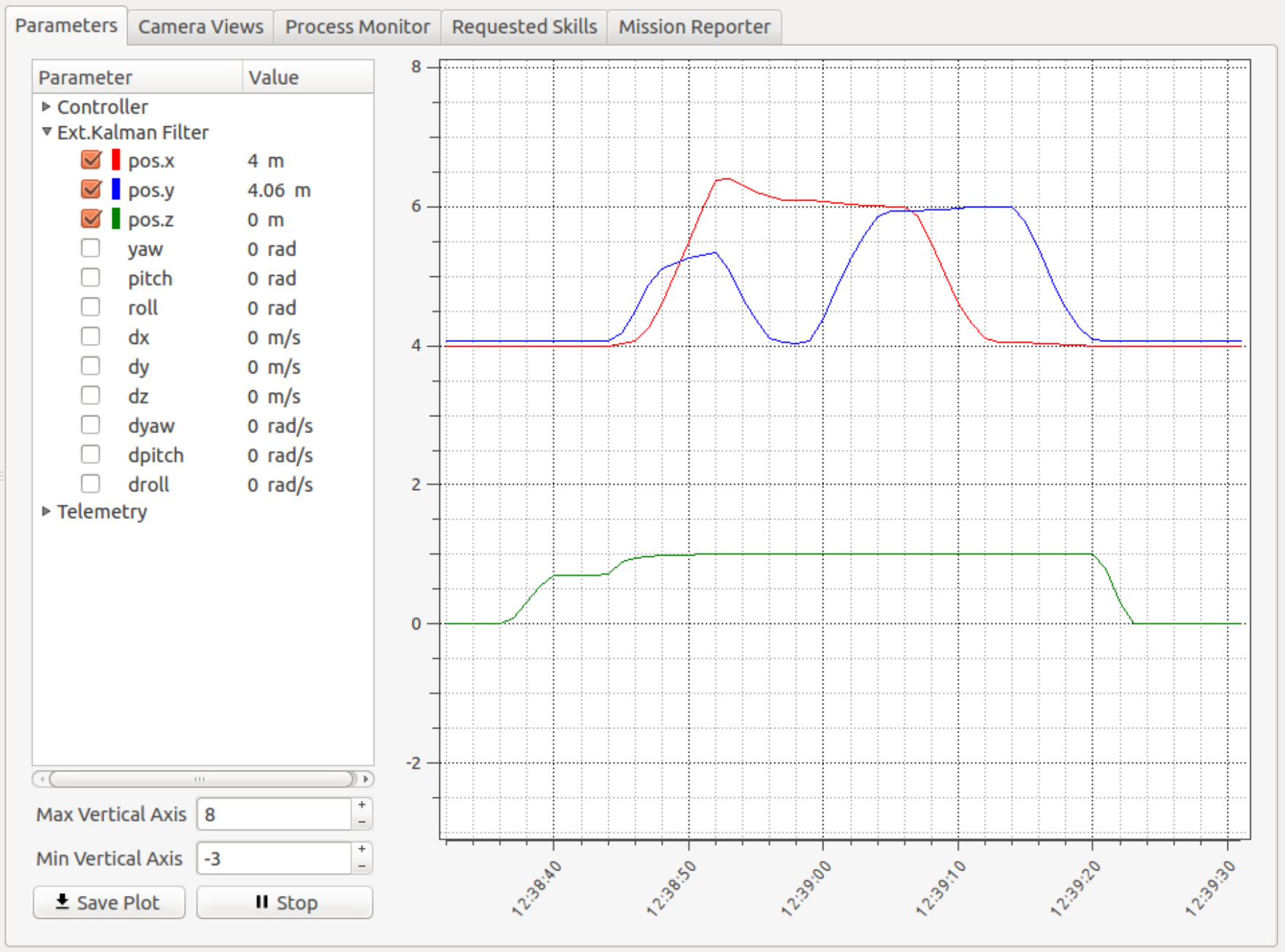


Once the operator starts the mission, this area of the window changes with a button that helps to abort the mission (and hover) or land the vehicle. This area also shows the current task in the mission.



3. The Parameter Viewer

In order to monitor the vehicle's behavior in detail during a mission, the operator can observe the values of numerical parameters. The operator can display multiple plots of the parameters selected in real time to analyze and compare their values.



This type of view is useful to verify the right operation of certain quantitative sensors. The view is also useful during the development of a particular vehicle configuration to select appropriate components and algorithms. For example, this window can be used to analyze the quality of different perception algorithms that estimate the vehicle's position.

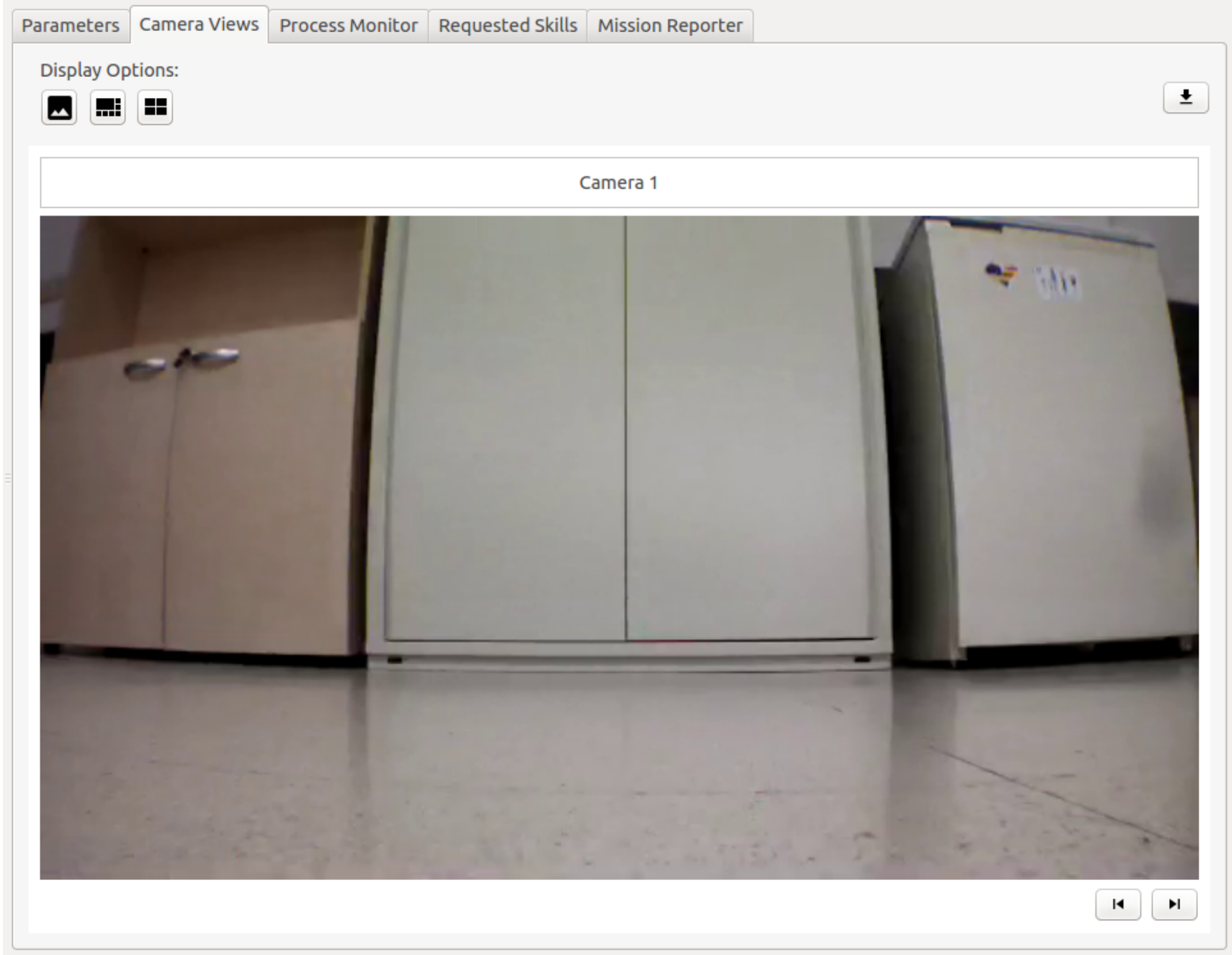
The window displays a plot for each parameter, identified by distinguishable colors. The graphic shows the evolution in a determined time interval, moving as time goes by from right to left. The vertical axis has been designed in a configurable way due to diversity of the parameters and the range of values. The maximum value can be set by a selector or directly zooming on the canvas.

As an additional feature, the window gives the possibility of saving the graphic plots for later use in various file formats (e.g., pdf, postscript).

4. The Camera Viewer

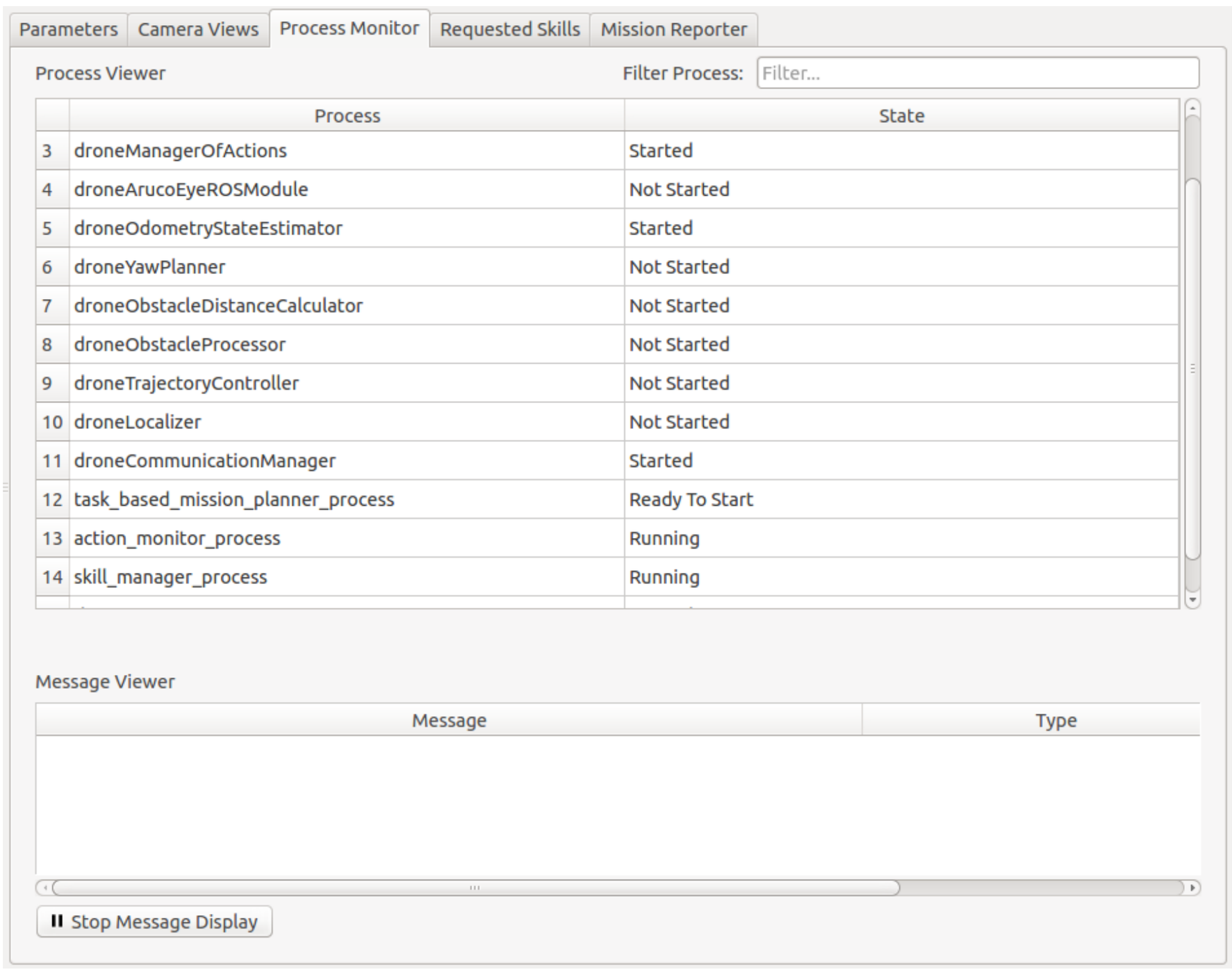
The camera viewer shows pictures and/or video images captured by the aerial vehicle during flight. The camera viewer displays a video streaming of the camera images.

Since the vehicle can use several cameras, the operator can select different display options to visualize the images. For example, the operator can select an option that displays only one camera (e.g., the frontal camera), an option that displays all cameras at the same time, or an option that displays a selected group of cameras.



5. The Process Monitor

The process monitor helps the user to supervise the current state of the software system (processes and error messages).



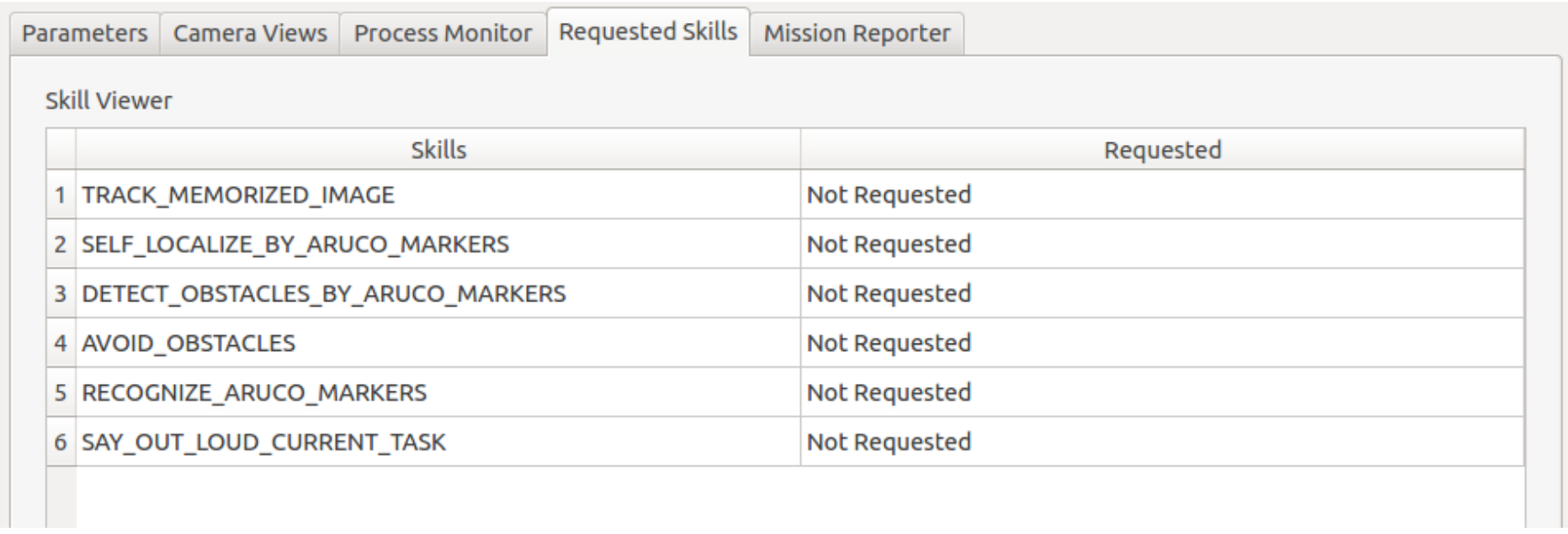
The process viewer (in the performance monitor window) helps the operator to monitor details of the internal behavior of the vehicle at software level. For example, the window displays a list of the existing processes and, for each one, the window shows its state (started, not started, etc.).

The process monitor also presents error messages with a temporal identifier or timestamp. The system uses this window to centralize and present in a uniform way to the operator all kind of software errors (invalid input data, time out, division by zero, etc.) This is very important to help operators who are familiar with the system architecture to diagnose and debug software errors for software development and maintenance tasks.

6. The Viewer of Requested Skills

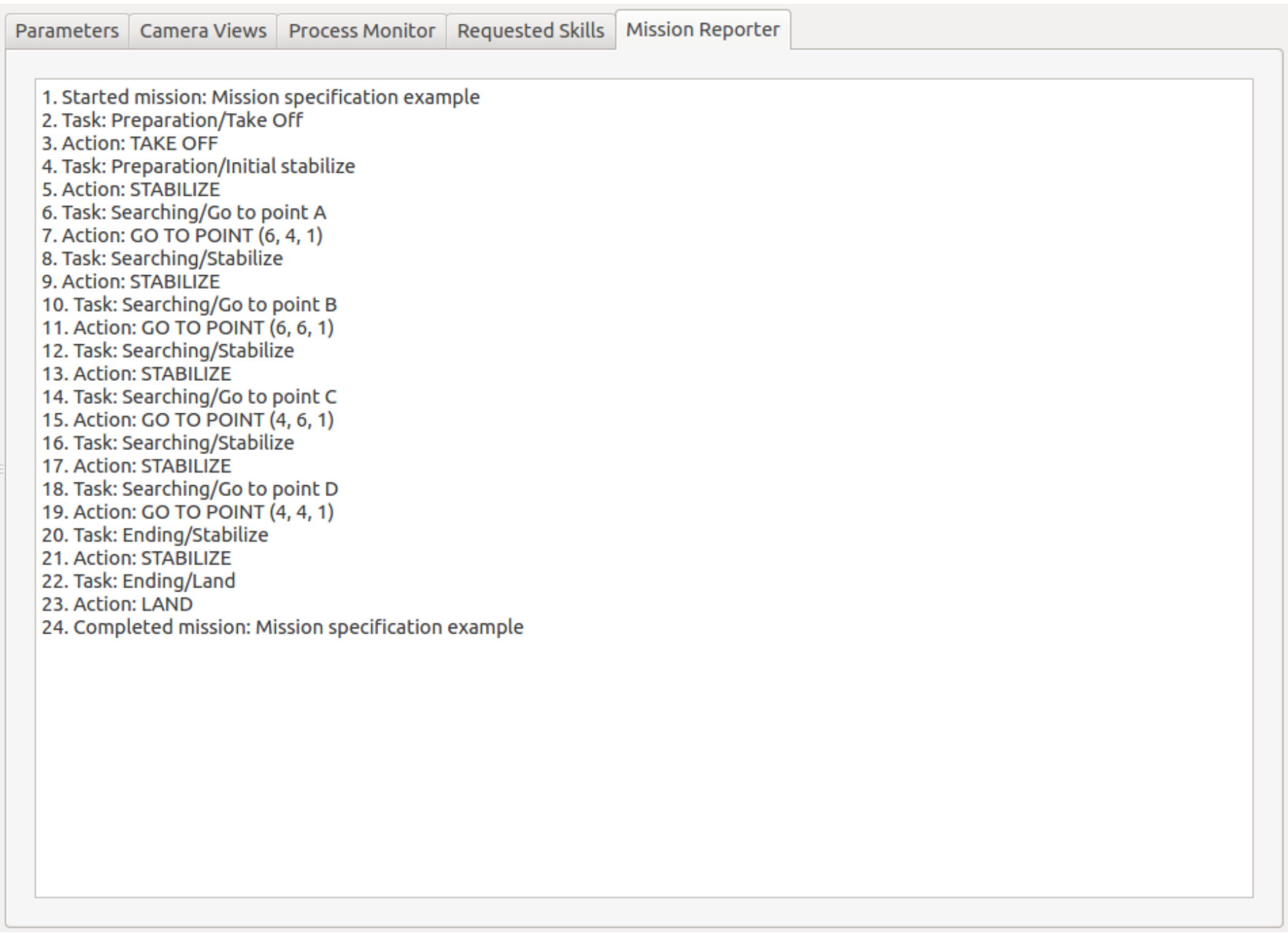
The operator can use the viewer of requested skills to consult and request the activation of specific skills. The GUI shows a list of available skills for the vehicle and indicates for each one its state.

For example, the operator can request a particular skills to be active (e.g., interpret Aruco visual markers) for a vehicle and can request another skills to be inactive (outloud speech the current task). The GUI translates each skill to the corresponding running processes.



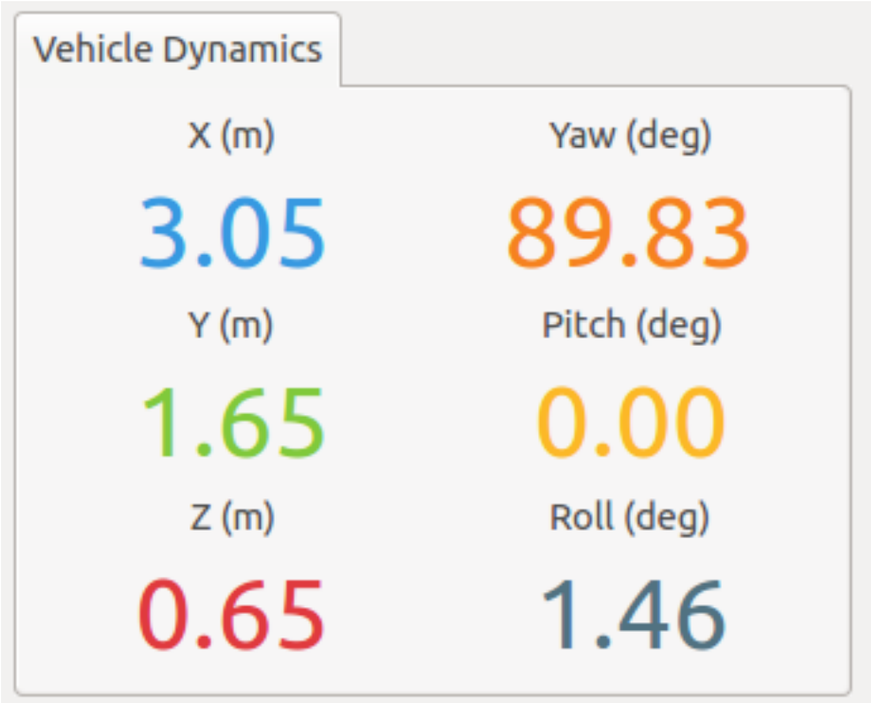
7. The Mission Reporter

The HMI includes a mission reporter. Using this window, the operator can see the sequence of tasks and actions followed by the vehicle during the execution of a mission.



8. The Dynamics of the Vehicle

The GUI includes a viewer that shows the dynamics of the vehicle. This can be used by the operator, for example, to better guide the vehicle during manual operations of the vehicle with the keyboard. The window presents the real time values of the position (x, y, z) and the angles corresponding to roll, pitch and yaw.



Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

TML Language

Edit New Page

Martin Molina edited this page on 15 Mar · 4 revisions

The operator can specify the tasks of a mission using the TML language (task-based mission specification language) specifically designed for Aerostack. TML uses XML syntax.

The following sections describe how to write in TML language a mission specification. As a complete example, a mission specification file is provided at the end of this document.

1. Mission

The mission is defined with the tag `<mission>` and the attribute `name`. For example:

```
<mission name= "example of mission">
</mission>
```

The body of the mission is separated in two main parts: (1) task tree and (2) event handlers.

2. Task Tree

The first part of the mission specification includes a list of tasks organized in a tree that are executed in sequence. Each task is specified using the tag `<task>` and the attribute `name`. For example:

```
<task name= "example of task">
</task>
```

The body of a task can include the tag `<action>` with the attribute `name` to specify an interpretable action (e.g., TAKE_OFF, LAND, etc.). The body of a task can also include one or several tags `<skill>` with the attribute `name` to specify the skills to be active. For example:

```
<task name="track image">
  <skill name="AVOID_OBSTACLES"/>
  <action name="WAIT">
      <argument name="duration" value="60"/>
  </action>
</task>
```

An action can include optionally arguments. Arguments use the tag `<argument>` and the optional attribute `name`. For example:

```
<task name="move to the first point">
  <action name="GO_TO_POINT">
      <argument name="coordinates" value="(3.0, 4.2, 1.5)"/>
  </action>
</task>
```

Tasks can be organized in a hierarchy. The body of a complex task (i.e., a task that use other tasks) can include skills that are active during the task execution. The body of a complex task does not include actions. Instead, the body includes one or several simpler tasks that are executed in sequence. For example:

```
<task name="move and wait">
  <task name="move to the first point">
    <action name="GO_TO_POINT">
```


Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

```
        <argument name="coordinates"  value="(3.0, 4.2, 1.5)"/>
      </action>
    </task>
    <task name="wait 20 seconds">
      <action name="WAIT">
        <argument name="duration" value="20"/>
      </action>
    </task>
  </task>
```

The linear sequence of task execution can be modified with repetitions and conditioned executions. The tag `<repeat>` and the attribute `times` is used to repeat a task several times. The tag `<condition>` and the attributes `parameter`, `comparison`, `value` are used to establish a condition to execute a task. The allowable values for the attribute comparison between numbers are {"equal", "less than", "less than or equal to", "greater than", "greater than or equal to", “not equal to”}. The comparison can be one of {"equal", "not equal to"} when the parameter is a string. The comparison can be "includes" when the parameter is a set of numbers and the value is a number. For example:

```
<task name="flip and move if green">
  <task name="flip three times">
    <repeat times="3"/>
    <action name="FLIP"/>
  </task>
  <task name="move to the first point">
    <condition parameter="OBSERVED_COLOR" comparison="equal" value="green"/>
    <action name="GO_TO_POINT">
        <argument name="coordinates" value="(3.0, 4.2, 1.5)"/>
    </action>
  </task>
</task>
```

3. Event handlers

The second part of a mission is the a list of event handlers. This part of the mission specification is defined with the tag `<event_handling>`:

```
<event_handling>
</event_handling>
```

Each event handler describes how to react to a particular event, while the mission planner is waiting the completion of an action. Events are defined with the tag `<event>` and the attribute `name`. Each event includes a list of conditions (in a conjunctive form), a list of actions and an ending step defined with the tag and the attribute `mode`. The allowable values for termination are {NEXT_TASK, REPEAT_TASK, ABORT_MISSION, END_MISSION}. For example:

```
<event_handling>

  <event name="time out since action started">
    <condition parameter="ACTION_FAILURE" comparison="equal" value="timeout"/>
    <termination mode="REPEAT_TASK"/>
  </event>

  <event name="low battery">
    <condition parameter="PERCENTAGE_BATTERY_CHARGE" comparison="less than" value="20"/>
    <action name="LAND"/>
    <termination mode="ABORT_MISSION"/>
  </event>

</event_handling>
```

Events can use a particular type of condition with the attribute `currentTask`. This is useful to express that the event happens while a particular task is being executing. The task is identified with a sequence of names of task-subtasks separated by "/". For example:

```
<event name="take off failure">
  <condition parameter="ACTION_FAILURE" comparison="equal" value="general"/>
  <condition currentTask="phase one/take off"/>
  <termination mode="ABORT_MISSION"/>
</event>
```

To know the available actions, skills and parameters for TML, consult the following lists:

- List of Actions for TML
- List of Skills for TML
- List of Parameters for TML

Publication

- Martin Molina, Adrian Diaz-Moreno, David Palacios, Ramon A. Suarez-Fernandez, Jose Luis Sanchez-Lopez, Carlos Sampedro, Hriday Bavle and Pascual Campoy (2016): "Specifying Complex Missions for Aerial Robotics in Dynamic Environments". International Micro Air Vehicle Conference and Competition, IMAV 2016. Beijing, China.(submitted)

Complete Example

This example is a complete specification of a mission in TML language.

```
<!--=====================================================
| FILE: mission_specification_file.xml
| BRIEF: A mission specification example.
| DETAILS:
|   This file includes basic movements to search in
|   an area and Aruco visual markers recognition.
|   The vehicle flips when an Aruco visual marker has
|   been found.
| AUTHORS:
|    Adrian Diaz
=====================================================-->
<mission name="Mission specification example">

<!--=====================================================
| TASK: Initial phase
=====================================================-->
  <task name="Initial phase">
    <task name="Taking off">
      <action name="TAKE_OFF"/>
    </task>
    <task name="Stabilizing at origin">
      <action name="STABILIZE"/>
    </task>
    <task name="Setting up HOME point ">
      <action name="MEMORIZE_POINT">
        <argument name="coordinates" label="HOME"/>
      </action>
    </task>
    <task name="Waiting 3 seconds">
      <action name="WAIT">
        <argument name="duration" value="3"/>
      </action>
    </task>
  </task>

<!--=====================================================
| TASK: Search object
=====================================================-->
```

```
  <task name="Searching Aruco identifiers">
    <skill name="RECOGNIZE_ARUCO_MARKERS"/>
    <task name="Moving to point A">
      <action name="MOVE">
        <argument name="coordinates" value="(1, 0, 0)"/>
      </action>
    </task>
    <task name="Stabilizing at point A">
      <action name="STABILIZE"/>
    </task>
    <task name="Moving to point B">
      <action name="MOVE">
        <argument name="coordinates" value="(0, 1, 0)"/>
      </action>
    </task>
    <task name="Stabilizing at point B">
      <action name="STABILIZE"/>
    </task>

    <task name="Moving to point C">
      <action name="MOVE">
        <argument name="coordinates" value="(-1, 0, 0)"/>
      </action>
    </task>
    <task name="Stabilizing at point C">
      <action name="STABILIZE"/>
    </task>
  </task>

<!--=====================================================
| TASK: Final phase
=====================================================-->

  <task name="Final phase">
    <task name="Going home">
      <action name="GO_TO_POINT">
        <argument name="coordinates" label="HOME"/>
      </action>
    </task>

    <task name="Final landing">
      <action name="LAND"/>
    </task>
  </task>

<!--=====================================================
| EXCEPTIONS
=====================================================-->
  <event_handling>
<!--=====================================================
| EVENT: Object found
=====================================================-->
    <event name="Object found">
      <condition parameter="RECOGNIZED_ARUCO_MARKERS" comparison="includes" value="0"/>
      <action name="STABILIZE"/>
      <action name="FLIP">
        <argument name="direction" value="FRONT"/>
      </action>
      <action name="STABILIZE"/>
      <termination mode="REPEAT_TASK"/>
    </event>

<!--=====================================================
| EVENT: Low battery
=====================================================-->
    <event name="Low battery">
      <condition parameter="percentageBatteryCharge" comparison="less than" value="20"/>
      <action name="LAND"/>
      <termination mode="ABORT_MISSION"/>
    </event>
```

```
  </event_handling>
</mission>
```

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

List of Actions for TML

Edit New Page

Martin Molina edited this page on 6 Jul 2016 · 2 revisions

Action `FLIP`

The vehicle performs a flip in a certain direction (to the front, to the back, to the right or to the left). The direction by default is to the front.

Arguments	Format	Example
direction	{front, back, left, right}	`<argument name="direction" value="back"/>`

Action `GO_TO_POINT`

The vehicle goes to a given point. The point can be expressed using absolute coordinates (argument "coordinates") or coordinates relative to the vehicle (argument "relative coordinates").

Arguments	Format	Example	Comments
coordinates	Tuple of 3 numbers x, y, z (meters)	`<argument name="coordinates" value="(1.23, 2, 0.5)"/>`	
relative coordinates	Tuple of 3 numbers x, y, z (meters)	`<argument name="relative coordinates" value="(1.23, 2, 0.5)"/>`	Not implemented yet

Action `LAND`

The vehicle descends vertically (through the z axis) until it touches the ground. It is assumed that the ground is static.

Action `MEMORIZE_POINT`

The vehicle memorizes the coordinates of the current point and assigns a label. The coordinates are absolute respects to the frame of reference.

Arguments	Format	Example
coordinates	Tuple of 3 numbers x, y, z (meters)	`<argument name="coordinates" label="point A"/>`

Action `MOVE`

The vehicle moves in a certain direction (forward, backward, right, left) a given number of meters.

Arguments	Format	Example	Comments
direction	{forward, backward, right, left}	`<argument name="direction" value="left"/>`	Not implemented yet
distance	Number (meters)	`<argument name="distance" value="0.5"/>`	Not implemented yet

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Action `ROTATE_YAW`

The vehicle rotates an angle in a the yaw axe. The specified angle can be relative (rotation angle) or absolute respect to a frame of reference (orientation angle).

Arguments	Format	Example	Comments
rotation angle	Number (degrees)	`<argument name="rotation angle" value="90"/>`	Not implemented yet
orientation angle	Number (degrees)	`<argument name="orientation angle" value="90"/>`	

Action `STABILIZE`

The vehicle stops completely in the air, cancelling all the movements, perturbations and turbulences that may affect its position.

Action `TAKE_OFF`

The vehicle takes off from its current location to the default altitude.

Action `WAIT`

The vehicle waits on the air for a specified number of seconds.

Arguments	Format	Example
duration	{Number (seconds)	`<argument name="duration" value="2.5"/>`

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

List of Parameters for TML

Edit New Page

Martin Molina edited this page on 31 May 2016 · 1 revision

Parameter	Values	Description	Comments
BATTERY_CHARGE_PERCENTAGE	Number	Percentage of battery charge (100% means full battery)	
ACTION_FAILURE	{timeout, general}	The action has failed because of a time out (value: timeout) or another unclassified failure (value: general)	Not implemented yet
RECOGNIZED_ARUCO_MARKERS	Set of numbers	Recognized Aruco visual markers	
CURRENT_TASK	String	Absolute name of a task within the task tree of the mission	

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

List of Skills for TML

Edit New Page

Martin Molina edited this page on 31 May 2016 · 1 revision

Skill	Description	Comments
AVOID_OBSTACLES	The vehicle is able to avoid obstacles	
RECOGNIZE_ARUCO_MARKERS	The vehicle is able to recognize the ID number of an Aruco marker	
DETECT_OBSTACLES_BY_ARUCO_MARKERS	The vehicle is able to detect obstacles identified by Aruco markers	Not implemented yet
SELF_LOCALIZE_BY_ARUCO_MARKERS	The vehicle is able to self localize by Aruco markers	Not implemented yet
SAY_OUT_LOUD_CURRENT_TASK	The vehicle says out loud the current task	Not implemented yet

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Configure Map

Martin Molina edited this page on 14 Jun 2016 · 1 revision

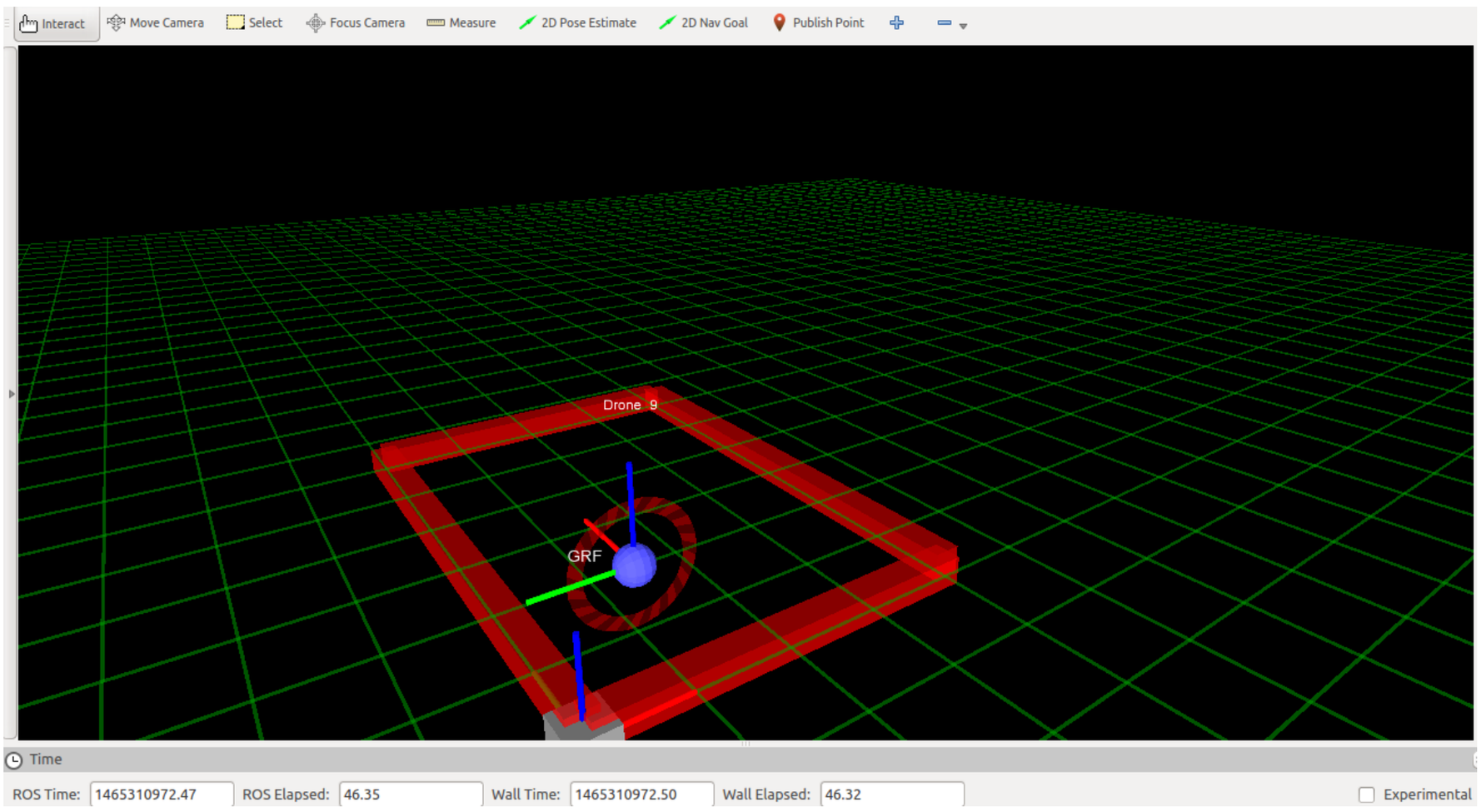
The environment map of the robot is specified in a series of configuration files which follow XML syntax. These files are located at `$AEROSTACK_STACK/configs/` within a folder selected by the operator called `drone{Id}` (`{Id}` = 00, 01, 02, ...). The `Id` parameter is the identification of the drone used for the mission, which is specified in the launch file used. To change the configuration of the environment map follow these steps.

1.Defining the Size of the Map

The size of the map is defined in `configFile.xml`. For example:

```
<map>
  <config>
    <dimensions>4.0 6.0</dimensions>
    <initPoint>0.0 0.0</initPoint>
  </config>
</map>
```

The `<dimensions>` parameter specifies the size (x, y) in meters of the perimeter of the flight zone. The `<initPoint>` specifies where the origin to the perimeter of the map will be. This parameter can be set in any value and has to be taken into account when placing the obstacles in the map. In this example the map size is 4m x 6m and the init point is set to (0,0).



2.Defining the Take-off Point

The take-off point should be specified in two files:

- the file `ekf_state_estimator_config.xml`
- the localization initial point in the file `param_localization_obs.xml`

For this example the start point in the x and y coordinates is (1m, 1m). Therefore, both files have this point set to (1, 1) but some changes must be made for each of these files.

2.1 EKF Take-off Point

Pages 28

Introduction
- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components
- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]
- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]
- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information
- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

The take-off point defined in `ekf_state_estimator_config.xml specifies the take-off point used by the (Extended Kalman Filter) EKF to start the estimation. This estimation is started with the drone in landed position so the height (z) is set to 0.

```
<take_off_site>
  <position>
    <x>1.0</x>
    <y>1.0</y>
    <z>0.0</z>
  </position>
  <attitude>
    <yaw>  90.00</yaw>
    <pitch> 0.00</pitch>
    <roll>  0.00</roll>
  </attitude>
</take_off_site>
```

Units:

- Units for tag `<position>` : meters
- Units for tag `<attitude>` : degrees (wYcPuR)

2.2 Aruco Visual Marker Take-off Point

The take-off point specified by the `param_localization_obs.xml` is used to determine where the Aruco visual marker localization start point is set in the map. This start point for the localization is done after takeoff and therefore is set to z = 0.7m.

```
<!-- Localization  -->
  <!-- Init pose  -->
  <x_0> 1.0</x_0>
  <y_0> 1.0</y_0>
  <z_0> 0.7</z_0>
  <yaw_0>  90.0</yaw_0>
  <pitch_0> 0.0</pitch_0>
  <roll_0>  0.0</roll_0>
```

Position is specified in meters and angles are given in degrees.

As you can see, both of these configuration files have initial values for Roll, Pitch and Yaw. Roll and Pitch will have a value of zero, since localization is started with these values. Nevertheless, the Yaw value is given according to the mission requirements of the user. If the user wishes to have the drone take-off facing the +x axis then this yaw value will be 0. In this example the dronewill take-off facing the +y axis so the rotation in yaw is +90deg.

3. Creating Obstacles and Virtual Walls

Walls and obstacles are defined in the `obstacle_config.xml` file. This file allows the creation of obstacles that can be real (to be detected by Aruco visual markers) or can be virtual.

Virtual walls cover the area of the map in order to restrain the movement of the drone to the desired map space. They can be created based on the size of the map. For example for a map of 4x6, the virtual borders can be defined as shown in the figure below.

```
<!-- borders (Virtual Obstacles): 10000-19999 -->
<rectangle description="bottom side">
  <id>10000</id>
  <centerPoint>2.0 0.0</centerPoint>
  <size>4.0 0.3</size>
  <yawAngle>0.0</yawAngle>
  <virtual description="define a virtual object">1</virtual>
  </rectangle>
```

```
<rectangle description="right side">
  <id>10001</id>
  <centerPoint>4.0 3.0</centerPoint>
  <size>0.3 6.0</size>
  <yawAngle>0.0</yawAngle>
  <virtual description="define a virtual object">1</virtual>
  </rectangle>

<rectangle description="up side">
  <id>10002</id>
  <centerPoint>2.0 6.0</centerPoint>
  <size>4.0 0.3</size>
  <yawAngle>0.0</yawAngle>
  <virtual description="define a virtual object">1</virtual>
  </rectangle>

<rectangle description="left side">
  <id>10003</id>
  <centerPoint>0.0 3.0</centerPoint>
  <size>0.3 6.0</size>
  <yawAngle>0.0</yawAngle>
  <virtual description="define a virtual object">1</virtual>
  </rectangle>
```

Besides specifying the boundaries of the map to be flown, the `obstacle_config.xml` file relates Aruco visual markers to obstacles in the map.

```
<pole description="A">
  <id>50002</id>
    <radius>0.15 0.15</radius>
    <aruco>
      <id aruco_id = "17"/>
      <id aruco_id = "18"/>
      <id aruco_id = "19"/>
      <id aruco_id = "20/>
    </aruco>
</pole>
```

As can be seen from the figure above, four Aruco markers (id: 17, 18, 19, 20) are used to define an obstacle, in this case a pole of 0.15 radius. With this file the user can specify which Aruco markers make up an obstacle and the geometry it can have.

4. Placing Aruco Visual Markers in the Map

The location of the visual markers used in the world is defined in the 'param_localization_obs.xml' file. In this file the user can specify the Aruco Slam details. Here the visual marker Ids that can be mapped by visual slam can be specified with `<VisualMarkersRange>`.

Also in this file the user can specify the fixed visual markers in the map.

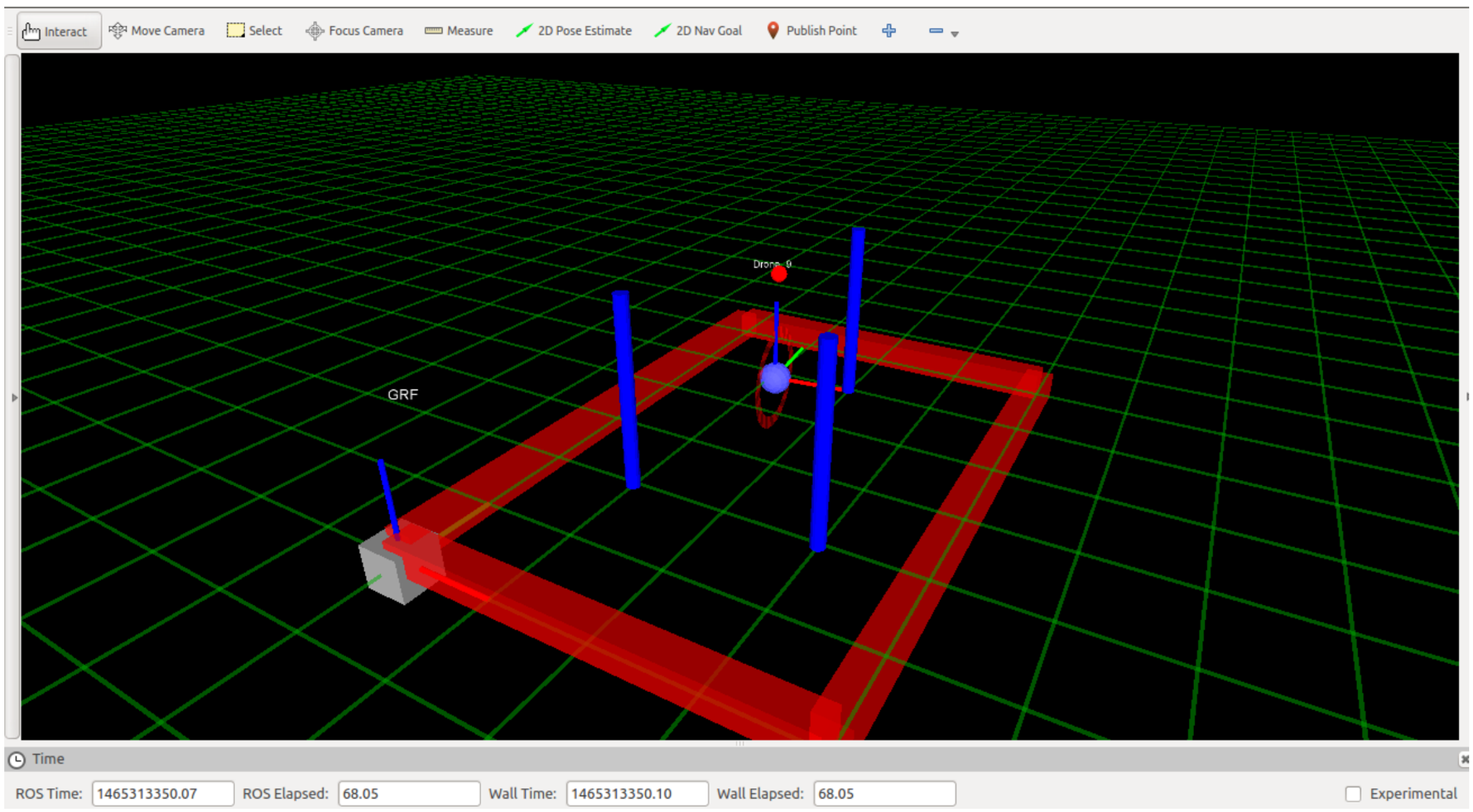
4.1 Visual Markers for Obstacles

This example uses three poles as known obstacles to navigate through and avoid. The location of each pole is given by the (x, y, z) coordinates of the center of the Aruco placed on the pole. Since the size of the poles used in this mission are 40cm in diameter, the location for the center of each Aruco is shown in the following example:

```
<!-- Pole 1 -->
<Aruco id="9"  x="3.0"  y="1.80" z="1.3" roll="0" pitch="0" yaw="-90"/>
<Aruco id="10" x="2.80" y="2.0"  z="1.3" roll="0" pitch="0" yaw="180"/>
<Aruco id="11" x="3.0"  y="2.20" z="1.3" roll="0" pitch="0" yaw="90"/>
<Aruco id="12" x="3.20" y="2.0"  z="1.3" roll="0" pitch="0" yaw="0"/>

<!-- Pole 2 -->
<Aruco id="13" x="1.0"  y="1.80" z="1.3" roll="0" pitch="0" yaw="-90"/>
```

```
<Aruco id="14" x="0.80" y="2.0"  z="1.3" roll="0" pitch="0" yaw="180"/>
<Aruco id="15" x="1.0"  y="2.20" z="1.3" roll="0" pitch="0" yaw="90"/>
<Aruco id="16" x="1.20" y="2.0"  z="1.3" roll="0" pitch="0" yaw="0"/>

<!-- Pole 3 -->
<Aruco id="25" x="2.0"  y="4.80" z="1.3" roll="0" pitch="0" yaw="-90"/>
<Aruco id="26" x="1.80" y="5.0"  z="1.3" roll="0" pitch="0" yaw="180"/>
<Aruco id="27" x="2.0"  y="5.20" z="1.3" roll="0" pitch="0" yaw="90"/>
<Aruco id="28" x="2.20" y="5.0"  z="1.3" roll="0" pitch="0" yaw="0"/>
```

Where the desired location of the obstacles is (3.0,2.0), (1.0,2.0) and (2.0, 5.0).



4.1 Visual Markers for Landmarks (for Self-Localization)

Besides the known pole location, landmarks are used for self-localization and are generally placed at the end of the map (e.g., where the mission of visual servoing will be performed). Each landmark carries two Aruco visual markers. If the landmark is placed on the y axis facing to the right, 0.11 needs to be subtracted from the y axis of the first aruco and 0.11 needs to be added to the y axis of the second aruco as shown in the figure below.

```
<!-- Landmark 1 -->
<Aruco id="61" x="0.0" y="3.89" z="1.3" roll="0" pitch="0" yaw="-45"/>
<Aruco id="62" x="0.0" y="4.11" z="1.3" roll="0" pitch="0" yaw="45"/>
```

If the landmark is placed on the x axis, facing the take-off point 0.11 needs to be subtracted from the x axis of the first aruco and 0.11 needs to be added to the x axis of the second aruco as shown in the figure below

```
<!-- Landmark 4 -->
<Aruco id="69" x="2.89" y="6.0" z="1.3" roll="0" pitch="0" yaw="225"/>
<Aruco id="70" x="3.11" y="6.0" z="1.3" roll="0" pitch="0" yaw="-45"/>
```

In this example, six landmarks are used:

```
<!-- Landmark 1 -->
<Aruco id="61" x="0.0" y="3.89" z="1.3" roll="0" pitch="0" yaw="-45"/>
<Aruco id="62" x="0.0" y="4.11" z="1.3" roll="0" pitch="0" yaw="45"/>

<!-- Landmark 2 -->
<Aruco id="63" x="0.0" y="4.89" z="1.3" roll="0" pitch="0" yaw="-45"/>
<Aruco id="64" x="0.0" y="5.11" z="1.3" roll="0" pitch="0" yaw="45"/>

<!- Landmark 3 -->
<Aruco id="65" x="0.89" y="6.00" z="1.3" roll="0" pitch="0" yaw="225"/>
<Aruco id="66" x="1.11" y="6.00" z="1.3" roll="0" pitch="0" yaw="-45"/>
```

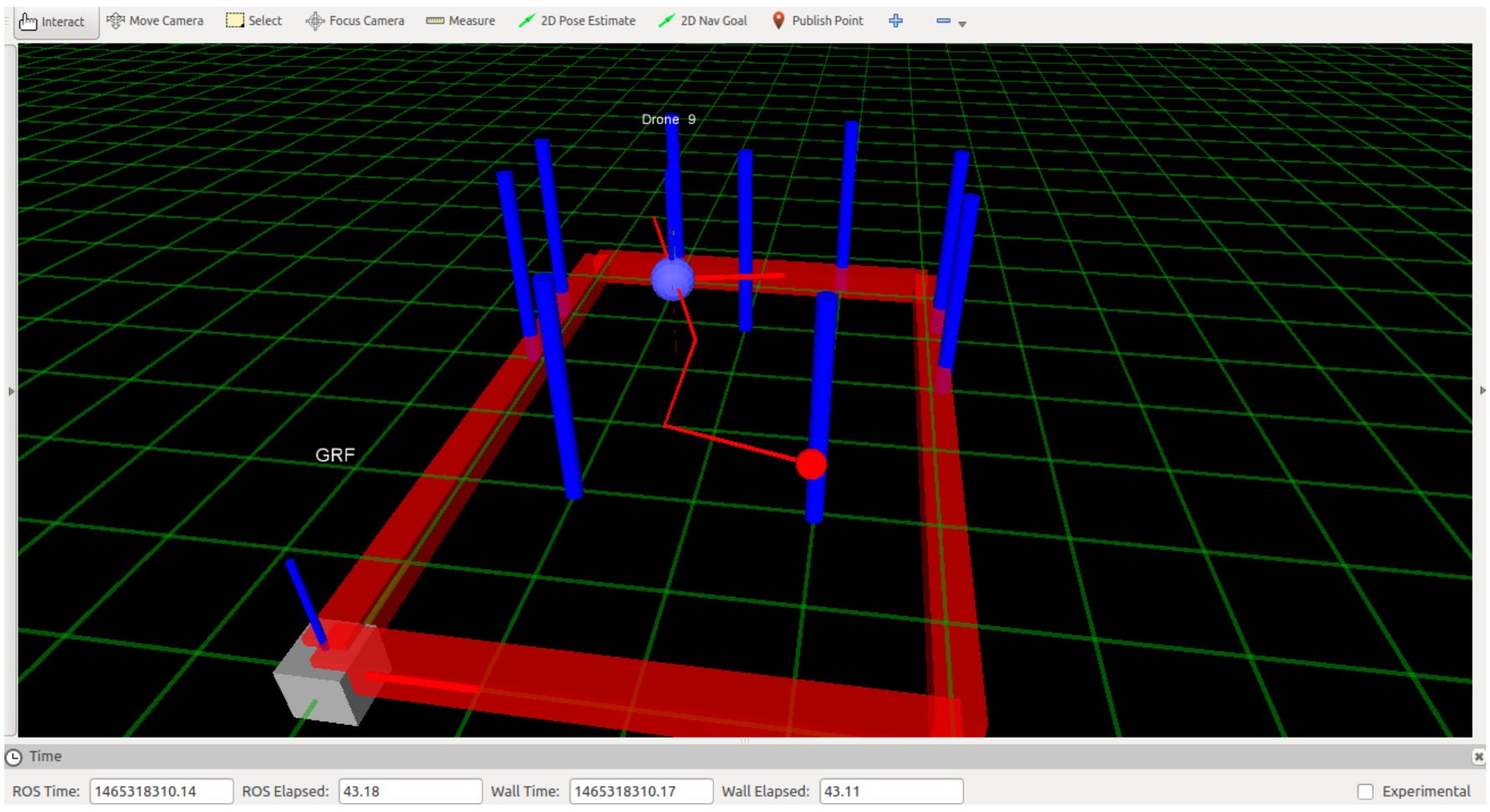
```
<!- Landmark 4 -->
<Aruco id="69" x="2.89" y="6.00" z="1.3" roll="0" pitch="0" yaw="225"/>
<Aruco id="70" x="3.11" y="6.00" z="1.3" roll="0" pitch="0" yaw="-45"/>

<!- Landmark 5 -->
<Aruco id="71" x="4.0" y="5.11" z="1.3" roll="0" pitch="0" yaw="225"/>
<Aruco id="72" x="4.0" y="4.89" z="1.3" roll="0" pitch="0" yaw="-45"/>

<!- Landmark 6 -->
<Aruco id="73" x="4.0" y="4.11" z="1.3" roll="0" pitch="0" yaw="135"/>
<Aruco id="74" x="4.0" y="3.89" z="1.3" roll="0" pitch="0" yaw="225"/>
```

5. Verify the Changes

After the environment is configured correctly, you can verify the changes using the RViz interface. For this purpose, run the following command:

```
$ ./rvizInterface_Test.sh drone{Id} localhost
```

It should presents a 3D image similar to this:



Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Specify Mission Tasks

Martin Molina edited this page on 14 Jun 2016 · 1 revision

Follow these steps to specify the tasks of a mission:

1. Edit the Mission Specification File

The task-based mission of the robot is specified in a XML file called `mission_specification_file.xml` (this is the default name, as it is established in the .launch file located within the `launch` directory in the repository `task_based_mission_planner_process`). This file must be located in `$AEROSTACK_STACK/configs/` within a folder selected by the operator called `drone{Id}` (`{Id}` = 00, 01, 02, ...).

Write your own mission specification file according to TML language.

2. Check Syntax Errors

To check if the file has syntax errors, launch the mission planner executing the following command:

```
$ roslaunch task_based_mission_planner_process
task_based_mission_planner_process.launch
```

If the xml file does not have any errors, the following message is presented:

```
File loaded without errors! Waiting to start ...
```


Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Configure Launcher

Martin Molina edited this page on 20 Aug · 4 revisions

The launcher is a file that specifies the set of processes that will be executed for a particular aerial robot system and and a specific mission to be done.

To configure a launcher, it is necessary to select a particular set of Aerostack processes. Each basic functionality (e.g., recognize ArUco visual markers, self-localization, mission planning, etc.) is provided by a set of processes. To configure the launcher, perform the following steps.

Go to the folder that contains examples of launchers. This folder includes .sh files that launch sets of processes for different architecture configurations:

```
$ cd ${AEROSTACK_STACK}/launchers
```

Select the .sh file that corresponds to an architecture similar to the architecture that you want to to create. Copy the selected file into a new .sh file. Edit the content of the new .sh file to add or remove processes. For example, to add the process task_based_mission_planner, add a new call to roslaunch such as the following:

```
gnome-terminal  \
...
--tab --title "Mission Planner" --command "bash -c \"
roslaunch task_based_mission_planner_process task_based_mission_planner_process.launch
--wait drone_id_namespace:=drone$NUMID_DRONE drone_id_int:=$NUMID_DRONE
my_stack_directory:=${DRONE_STACK};
                                                exec bash\"" \
...
```

To know the available functionalities and processes provided by Aerostack, see the following catalogue:

- Catalog of Available Processes

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Launch Real Flight

Edit New Page

Martin Molina edited this page on 14 Jun 2016 · 2 revisions

To launch a real flight using an AR Drone perform the following steps:

- First, connect the AR Drone to the computer using wifi.
- Launch the roscore

```
$ roscore
```

- Go to the folder that contains launchers. It includes different pre-configured .sh files that launch the set of processes for different mission types:

```
$ cd ${AEROSTACK_STACK}/launchers
```

- In this example we select the file *ardrone_basic.sh*. Write the following to launch the mission:

```
$ ./ardrone_basic.sh
```

Then you can use the human machine interface to start the mission (see Human Machine Interface). The AR Drone will execute the mission defined in the config files.

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Practical Tips for Real Flights

Martin Molina edited this page on 30 Jan · 4 revisions

This section includes a set of practical tips to take into account in real flights.

Multi-drone flight

In order to avoid drone collissions during the flight, you have to publish manually a message in the missionState topic after the complete initialization of Aerostack.

```
rostopic pub -1 /<DRONE_ID>/missionState std_msgs/Bool "data: true"
```

You must publish the previous message for each drone you are using.

Correct topics for Aruco-based localization

If the flight requires aruco-based positioning, be sure that you use the topics `ArucoSlam_EstimatedPose` and `ArucoSlam_EstimatedSpeeds` in your launcher, instead of `EstimatedPose_droneGMR_wrt_GFF` and `EstimatedSpeed_droneGMR_wrt_GFF`.

NOTE: If you are experiencing problems with the drone's localization, consider increasing the density of Aruco visual markers.

Points for the Trajectory Planner

The trajectory planner uses a set of randomly generated points that can be modified in the file `configFile.xml`. If you find that trajectory calculation is too slow you can try decreasing this value to reduce the calculation cost. However, keep in mind that decreasing it too much might cause the trajectory planner not to find trajectories in certain situations.

In the file *configFile.xml* located in the *configs/droneN* directory, you can specify the number of points used by the Trajectory Planner's algorithm to calculate trajectories. A high number will result in better trajectories, but will impact performance. To increase the probability of a good trajectory without too low performance, you can add a subzone with additional nodes for a difficult area. This can also work if you want to increase the probability of generating trajectories that pass through a certain area, since the Trajectory Planner favours trajectories with a higher number of nodes.

```
<config>
    <numNodes>3500</numNodes>
    <neigbourhood>5</neigbourhood>

    <subzone>
        <numNodes>500</numNodes>
        <initPoint>4.0 0.0</initPoint>
        <size>2.0 3.0</size>
    </subzone>
</config>
```

In this example, we add an extra 500 nodes to a 2m by 3m zone in (4, 0) where there is a narrow window that can make trajectory calculation more difficult.

Collect data about flight

To record the data of the flight you just have to run the following script before starting the mission.

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

```
#!/bin/bash

OUTFILE=$1
TOPICS="/public_event /rosout /rosout_agg /societyBroadcast /tf"

for i in $*; do
  if [[ "$i" != "$1" ]]; then
    DRONE_ID="drone$i"
    TOPICS="$TOPICS /$DRONE_ID/ArucoSlam_EstimatedPose   /$DRONE_ID/ArucoSlam_EstimatedPo
  fi
done

rosbag record -O $OUTFILE $TOPICS
```

Run the script like this:

```
./rosbag-multidrone.sh outfile.bag <DRONE_ID_1> <DRONE_ID_2> ... <DRONE_ID_N>
```

Where DRONE_ID is the number id of the drone.

Battery levels

Keep in mind that the lower the battery level is, the lower the power the drones' rotor will generate. In order to avoid that kind of problem, it is very recommended to change the battery when it has 30 percent or less.

Aerostack reboot

Whenever the drone is disconnected (because of a battery change or a similar reason), it is highly reccommended to restart Aerostack and roscore to prevent unexpected behaviour.

If any of the processes crash, it is necessary to restart Aerostack to ensure everything is loaded correctly.

OpenTLD GUI for Visual Servoing

When you are using the OpenTLD GUI you have to press the F5 button to refresh the image o the screen, select the area you want the drone to follow and then press and hold (just for few seconds) the ENTER key. Then the drone will start following the selected area.

Emergency mode

This state is used by the AR Drone to stop each of the rotors. This can be useful if the drone stops responding or it runs into an obstacle. To enter emergency mode you have two ways:

- Grab the drone with your hands and flip it so it faces downwards. Useful when the drone stops responding and you have no other way to land.
- Use the Command Line Interface and press the SPACE key.

To exit Emergency mode, you can press the SPACE key again. You will not be able to start a mission until you exit Emergency mode.

Avoid duplicate identifiers

It is important to check that identifiers for poles, walls or landmarks in the map are unique. If you use the same identifier for different objects in the map, the trajectory planner cannot work properly and could crash.

Efficient communication

To check if the connection to the drone is stable and fast enough, you can run the command `rostopic hz /drone0/camera/front/image_raw`. It should display frequencies of around 30Hz. If this is not the case, you should consider changing the drone's channel to a less occupied one (this can be done with a parameter in the launcher).

Visual servoing processes

If visual servoing produces unxpected behaviour, double check that the following processes have been started (with `rosnode list`):

- /droneN/droneIBVSController
- /droneN/open_tld_translator
- /droneN/ros_tld_tracker_node
- /droneN/ros_tld_gui_node
- /droneN/trackerEye

Distance between markers

The range for which the ARDrone's camera can detect Aruco markers is about 3 meters, so poles and landmarks should be placed in a way that the drone always detects at least 1 Aruco marker and can correctly estimate its position.

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Simulated Quadrotor

Edit New Page

Martin Molina edited this page on 14 Jun 2016 · 1 revision

Here you can find an example that describes how to perform a simulated mission. This example assumes that the vehicle is simulated.

This example uses the following configuration files:

```
./config/drone01/*
```

Execute the following launcher:

```
./launchers/simulated_quadrotor.sh
```


Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

AR Drone and Visual Markers

Martin Molina edited this page on 14 Jun 2016 · 2 revisions

Here you can find an example that describes how to perform a complete mission with a real flight. This example assumes that the self-localization uses Aruco visual markers.

This example uses the following configuration files:

```
./config/drone02/*
```

Execute the following launcher:

```
./launchers/ardrone_basic.sh
```


Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Setup the Communication Network

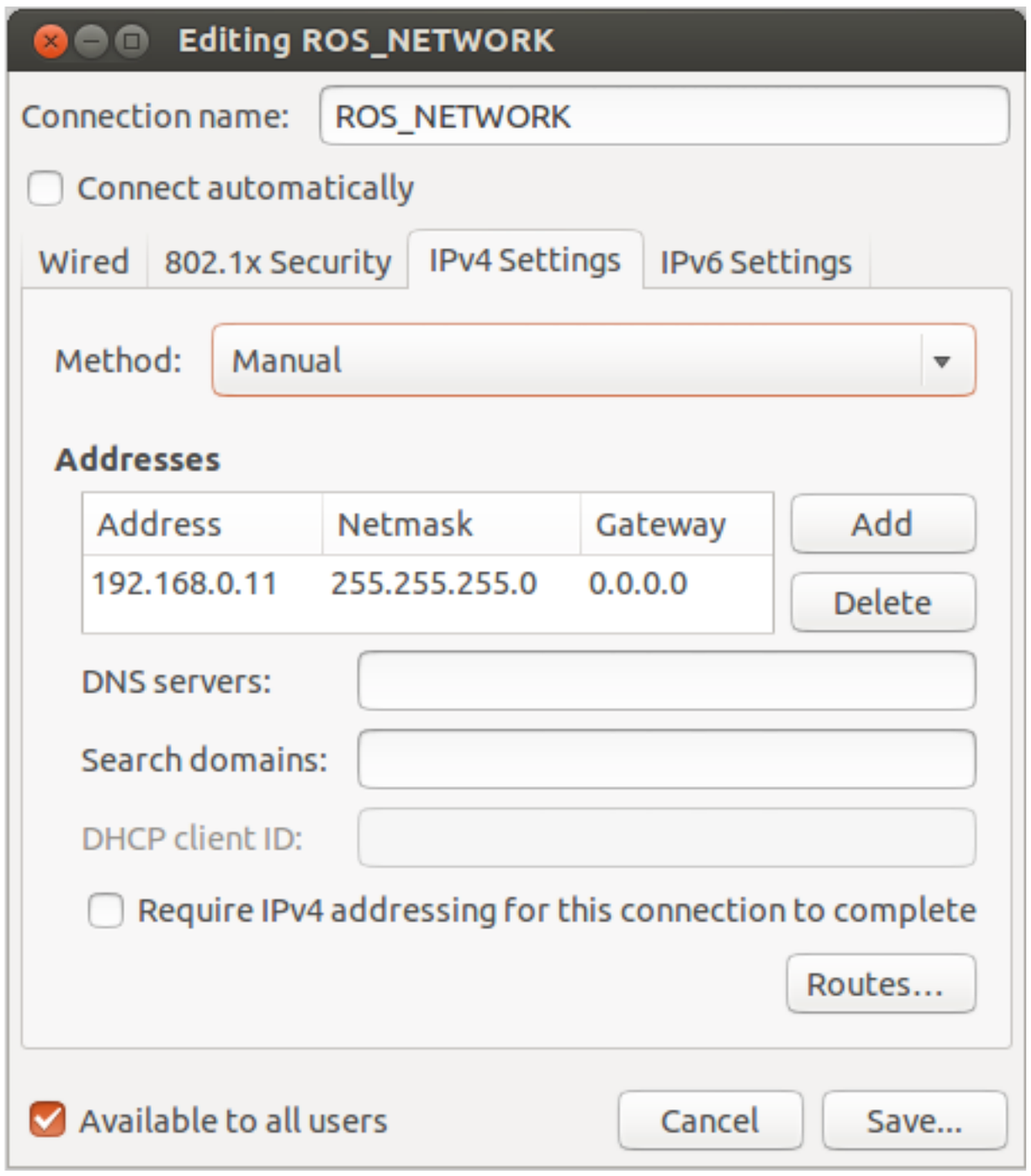
Martin Molina edited this page on 14 Jun 2016 · 1 revision

To setup the communication network, please read before the related ROS tutorial.

> NOTE: There is no need of a network setup when using an ARDrone as it can be connected directly to the groundstation using WIFI.

Then, follow these steps.

Configure a LAN network connection to communicate ground stations with the rest using ROS. The following image shows one of the used configurations:



Edit the hosts file, all the computers in the network have to be included in it. The IPs have to be the same that is specified in the configuration of each computer's ROS_NETWORK configuration. To edit the hosts file, run:

```
$ sudo gedit /etc/hosts
```

An example of the content of this file is the following:

```
127.0.0.1       localhost
127.0.1.1       jespestana-XPS

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# ROS_NETWORK, etc.
192.168.1.1     ardrone
192.168.0.10    CVG-JL-Toshiba
192.168.0.10    ROS_10
```

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

```
192.168.0.11    ROS_11
192.168.0.12    carrio-GT70
192.168.0.12    ROS_12
192.168.0.13    SATELLITE-P850
192.168.0.13    ROS_13
192.168.0.20    joselu-Aspire-E1-571
192.168.0.20    ROS_20
192.168.0.21    joselu-Aspire-E1-571-01
192.168.0.21    ROS_21
192.168.0.30    joselu-Q330
192.168.0.30    ROS_30
192.168.0.40    CVG-Desktop01
192.168.0.40    ROS_40
192.168.0.41    CVG-Desktop02
192.168.0.41    ROS_41
```

Finally, restart your network for the configuration changes to take place.

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Wi Fi Communication with Multiple Drones

Martin Molina edited this page on 30 Jan · 4 revisions

This section describes how to proceed to establish an adequate communication using Wi-Fi when there are multiple drones. There are two alternative options:

- One to one (each drone to one computer)
- Many to one (several drones to one computer)

One to one

In this case, you must have the same (or more) number of computers as drones. Each computer connects to a drone's Wi-Fi hotspot. Also, you must have a way of connecting computers to each other, for example a switch or a router. Keep in mind that there are two different subnets, one for the connection to the drone and another for the connection among computers.

After the computers are connected, you must edit the /etc/hosts file to add each computers hostname and their respective IPs.

NOTE: To avoid interferences make sure that the drones' hotspots are in different Wi-Fi channels. It also helps if there aren't too many other hotspots in those channels. To change a drone's channel you can use the following script:

```
SSID=$(grep ssid_single_player /data/config.ini | awk -F "=" '{print $2}')
ifconfig ath0 down
iwconfig ath0 mode master essid $SSID channel $2
iwconfig ath0 commit
ifconfig ath0 $1 netmask 255.255.255.0 up
```

To run this script login to the drone via telnet, create the script and execute it.

```
telnet 192.168.1.1
vi bin/set_IP_ch.sh #Copy the script and close vi
chmod +x bin/set_IP_ch.sh
set_IP_ch.sh 192.168.1.1 <CHANNEL>
```

NOTE: Since the drones are MAC-locked by default, when connecting a new computer to a drone, you should press the reset button on the drone to erase the previous computer's MAC address. The button is located under the battery, and you will need a paper clip or similar to press it.

Many to one

In this case, you should use a router or a computer with a wifi hostpot to which the drones will connect to. To do this, you have to edit the drone's configuration to connect to your router. For detailed instructions see: Multiple AR Drones

1. Edit the drone's pairing setup

First connect to the drone's hotspot, and login via telnet:

```
telnet 192.168.1.1
```

Now edit the following script:

```
vi bin/pairing_setup.sh
```

Pages 28

Introduction
- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components
- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]
- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]
- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information
- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

Look for this line and comment it:

```
iptables -A INPUT -m mac --mac-source $MAC_ADDR -j ACCEPT
```

Then change

```
iptables -P INPUT DROP
```

into

```
iptables -P INPUT ACCEPT
```

2. Add and execute the wifi setup script

Create the following file

```
vi bin/drone_wifi_config.sh
```

Add these lines

```
killall udhcpd
ifconfig ath0 down
iwconfig ath0 mode managed essid <ESSID>
iwconfig ath0 commit
ifconfig ath0 <IP_ADDRESS> netmask <NETMASK> up
```

Now make the script executable

```
chmod +x bin/drone_wifi_config.sh
```

Finally, exit your telnet session and execute the following command in your computer

```
echo drone_wifi_setup.sh | telnet 192.168.1.1
```

Your drone should now be connected to your router

NOTE: To insert text in vi, press 'i' before writing some text. To save and exit the file in vi, introduce the following sequence `ESC shift+z shift+z`

3. Edit the launchers

Edit the file $AEROSTACK_STACK/launchers/ardrone_indoors.launch and add `args="-ip $(arg drone_ip)"` to the first `<node>` tag. Now look for the ArDrone Autonomy section of your launcher and add the following parameter: `drone_ip:=$DRONE_IP`.

Ultrasound frequency

In order to avoid the drones repelling each other when they are in proximity, the following parameter should be modified in the *ardrone_indoors.launch* file located in *launchers/ardrone_launch*

```
<param name="ultrasound_freq" value="8" />
```

This parameter can be set to either 7 or 8. Drones with this parameter set to different values won't repel each other.

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Calibrate Controllers

Martin Molina edited this page on 18 Jul · 2 revisions

Calibrating controllers become very important step if user has added more payloads to the UAV or a UAV has been added which with the Aerostack was not used before. There are several controllers which are used the Aerostack and they are arranged in several layers. The first layer is the low level controller which is inside the autopilot of the UAV system. This low level controller can be tuned according to the instructions given by the people providing the autopilot system.

Above the low-level controller comes the mid-level controller which is present inside the Aerostack. This controller is used for take off, landing and hovering of the UAV. (Note: In case of the ARdrone, the mid-level controller provided by the ARdrone manufactures is being used and hence the mid-level controller provided by the Aerostack is not used). Above the mid-level controller is the high-level controller, using which the user can give speed commands, position commands as well as trajectories to the UAV.

Calibrating the Mid-Level Controller

As mentioned above the mid-level controller is used for take-off, landing and hovering of the UAV, and hence it is one of the critical controllers. If the mid-level controller is not tuned properly, there are high chances of the UAV crashing. Hence it is very important that is mid-level is tuned perfectly.

The mid-level controller is a combination of three different controllers namely:

- Altitude controller
- Hover controller
- Yaw controller

As the names suggest, the altitude controller is the one which is used for controlling the altitude of the UAV. The hover controller takes in input as vertical speeds and is used for making the UAV hover at a particular position. The yaw controller controls the heading of the UAV maintaining the current heading of the UAV.

The mid-level controller tuning parameters are stored inside $DRONE_STACK/configs/drone. The drone id (drone0, drone1, etc.) depends on the drone which the user is using. For example, the mid-level controller tuning file for Asctec Pelican UAV can be found inside $DRONE_STACK/configs/drone0, as drone0 is being used for Pelican. The name of the file which contains the tuning parameters is midlevel_autopilot.xml. Below is a step by step explanation of the midlevel_autopilot.xml.

```
<midlevel_autopilot>
  <ON_GROUND_ALTITUDE>  0.25  </ON_GROUND_ALTITUDE> <!-- m -->

  <TAKE_OFF>
    <TAKEOFF_SPEED>  0.6  </TAKEOFF_SPEED> <!-- m/s -->

    <TAKEOFF_FINAL_ALTITUDE> 1.1 </TAKEOFF_FINAL_ALTITUDE> <!-- m   -->

    <TAKINGOFF_ALT_SPEED_COS_GAMMA> 2.15 </TAKINGOFF_ALT_SPEED_COS_GAMMA>

    <MAX_TAKINGOFF_TIME_S> 5.0 </MAX_TAKINGOFF_TIME_S> <!-- s   -->
  </TAKE_OFF>

  <LANDING>
    <LANDING_SPEED> 0.40 </LANDING_SPEED> <!-- m/s -->
```


Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

```
    <ALTITUDE_FOR_LANDING_WITH_TIME> 0.40   </ALTITUDE_FOR_LANDING_WITH_TIME>  <!-- m   -

    <LANDING_WITH_TIME_MAX_SECONDS>     1.00 </LANDING_WITH_TIME_MAX_SECONDS>  <!-- s   -

    <LANDING_WITH_TIME_FIXED_THRUST_KG> 0.32 </LANDING_WITH_TIME_FIXED_THRUST_KG>  <!-- k
  </LANDING>
```

- ON_GROUND_ALTITUDE is the on ground altitude measured by the distance sensor when the UAV is landed.

The next parameters influence the take off of the UAV.

- TAKEOFF_SPEED influences how fast the UAV will take off. Ranging from 0.1 to 1.0. It is generally advised to keep to the takeoff speed parameter to 0.6 for a good take off.
- TAKEOFF_FINAL_ALTITUDE influences the final altitude the UAV is maintain after it taking off. In case PX4flow sensor is being used as the distance sensor the range is between 0.5 to 3.0.
- MAX_TAKINGOFF_TIME_S parameter defines the maximum time the midlevel controller will maintain its status at taking off after which it will change its status to hovering. In other words if the max taking off time is 5 secs, the maximum time the midlevel controller will allow the UAV to reach its final altitude is 5 secs after which the UAV will start hovering evern if it has not reached the final altitude.

The next set of parameters are for landing

- LANDING_SPEED parameter defines how fast the UAV will land. Advisable to keep it around 0.4 m/s.
- ALTITUDE_FOR_LANDING_WITH_TIME parameter defines the altitude at which the controller will give the final thrust for llanding the UAV. For example is the the parameter is 0.4 m the controller will give the final thrust after the UAV reaches 0.4 m.
- LANDING_WITH_TIME_MAX_SECONDS parmeter defines the time is seconds the controller will wait before give the final thrust.
- LANDING_WITH_TIME_FIXED_THRUST_KG parameter is the value of final thrust given in kgs for the UAV to touch the ground. It is influenced by the weight of the UAV, lighter UAV's around 2 kgs can have a this parameter around 0.32kgs and for heavy UAV's this parameter can be around 1.32 kgs. For more information on how to choose the landing thrust value user can refer to configs/drone0/midleve_autopilot.xml.

The next set of parameters are for setting the limits for the controller as can be seen in the table below

```
<MAX_ALTITUDE>
  <MAX_ALTITUDE>   3.00 </MAX_ALTITUDE> <!-- m   -->

  <MAX_ALTITUDE_DESCENDING_SPEED> 0.20 </MAX_ALTITUDE_DESCENDING_SPEED> <!-- m/s -->

  <MAX_ALTITUDE_TO_CONSIDER_UNKNOWN_STATE_AS_LANDED> 0.40 </MAX_ALTITUDE_TO_CONSIDER_UNKN
</MAX_ALTITUDE>


<CONTROLLER_SATURATIONS>
  <ML_AUTOPILOT_MAX_ALLOWED_YAW_SPEED> 30.0</ML_AUTOPILOT_MAX_ALLOWED_YAW_SPEED> <!-- deg

  <ML_AUTOPILOT_MAX_ALLOWED_ALTITUDE_SPEED>0.5 </ML_AUTOPILOT_MAX_ALLOWED_ALTITUDE_SPEED>
</CONTROLLER_SATURATIONS>
```

- MAX_ALTITUDE parameter as the the name says defines the maximum altitude that can be achieved by the UAV. Note: when using the px4flow sensor for altitude measurement, the height is always limited to 3 meters.
- MAX_ALTITUDE_DESCENDING_SPEED parameter defines the maximum descending speed which a UAV can achieve while landing.
- MAX_ALTITUDE_TO_CONSIDER_UNKNOWN_STATE_AS_LANDED parameter publishes the unknown state landed below a given altitude. The midlevel controller always publishes the states of the UAV where it is landed, taking off, flying etc, if in case the state is unknown the and the altitude is below.
- MAX_ALTITUDE_TO_CONSIDER_UNKNOWN_STATE_AS_LANDED parameter the midlevel controller will publish the state as landed.
- ML_AUTOPILOT_MAX_ALLOWED_YAW_SPEED parameter defines the maximum yaw speed for the UAV, recommended to keep it at 30 deg/s.
- ML_AUTOPILOT_MAX_ALLOWED_ALTITUDE_SPEED defines the maximum allowed altitude speed, right now the midlevel controller does not work properly with altitude speed more that 0.5 m/s.

The next set of parameters define the maximum and minimum tilt value, change in yaw value and the thrust value. This values remain constant to a specific autopilot of the UAV, which can obtained from the manufacturer of the UAV. In the below case the values are for the Asctec Pelican UAV obtained from the astec manufacturers.

```
<LL_AUTOPILOT>

  <LL_AUTOPILOT_COMMAND_TILT_MIN_VALUE>-51.175</LL_AUTOPILOT_COMMAND_TILT_MIN_VALUE> <!--

  <LL_AUTOPILOT_COMMAND_TILT_MAX_VALUE>+51.175</LL_AUTOPILOT_COMMAND_TILT_MAX_VALUE> <!--

  <LL_AUTOPILOT_COMMAND_DYAW_MIN_VALUE>-254.760</LL_AUTOPILOT_COMMAND_DYAW_MIN_VALUE> <!-

  <LL_AUTOPILOT_COMMAND_DYAW_MAX_VALUE>+254.760</LL_AUTOPILOT_COMMAND_DYAW_MAX_VALUE> <!-

  <LL_AUTOPILOT_COMMAND_THRUST_MIN_VALUE>0.000</LL_AUTOPILOT_COMMAND_THRUST_MIN_VALUE> <!

  <LL_AUTOPILOT_COMMAND_THRUST_MAX_VALUE>+32.000</LL_AUTOPILOT_COMMAND_THRUST_MAX_VALUE>

</LL_AUTOPILOT>
```

The next set of values define the altitude controller which is the first controller inside the midlevel controller as mentioned above.

```
<altitude_speed_controller>

  <FEEDFORWARD_M>  2.11  </FEEDFORWARD_M> <!-- kg  -->

  <THRUSTER_GAIN> 0.3065625 </THRUSTER_GAIN> <!-- 1p1/kg, pelican: 1/(32/9.81)-->

  <GAIN_P>        0.10      </GAIN_P> <!-- Pelican: 0.1, 1p1/(m/s)-->

  <GAIN_I>        0.08      </GAIN_I> <!-- Pelican:(GAIN_P / 1.25),1p1/(s*m/s)-->

  <GAIN_D>        0.00      </GAIN_D> <!-- Pelican:(GAIN_P * 0.00),1p1*s/(m/s)-->


  <ALT_SPEED_MEASURE_SATURATION_FACTOR> 3.0</ALT_SPEED_MEASURE_SATURATION_FACTOR>

  <COMMAND_UPPER_LIMIT>  0.33  </COMMAND_UPPER_LIMIT>  <!-- Pelican:(          450.0 / 4

  <COMMAND_LOWER_LIMIT> -0.11  </COMMAND_LOWER_LIMIT>  <!-- Pelican: ( (-1.0) *  150.0 /
```

```
  <SUBSAMPLING_DIVIDER>  6      </SUBSAMPLING_DIVIDER> <!-- Pelican: 6 -->

</altitude_speed_controller>
```

- FEEDFORWARD_M parameter is in kilograms the value at which the UAV can manitain its altitude. It can be calculated as follows. The user first must know the value in the terms units of the radio at which the UAV holds altitude, this can be found out manaully flying the UAV. For example in case of pelican it was found that with the current payload it holds altitude at 2560 counts using the radio, and hence FEEDFORWARD_M is: 2650 * (32/9.81) * (1/4095) = 2.11 kgs, where 32: LANDING_WITH_TIME_FIXED_THRUST_KG, 4095: The maximum thrust in units of the radio (counts), 9.81: Acceleration due to gravity.
- THRUSTER_GAIN parameter can be calculated as follows: 1/(32/9.81) where 32: LANDING_WITH_TIME_FIXED_THRUST_KG, 9.81: Acceleration due to gravity.
- GAIN_P parameter is the most important parameter, should be started with a smaller value like 0.01, until the UAV does not oscillate in the altitude.
- GAIN_I parameter should be increased from a smaller value like 0.01 until the UAV reaches its setpoint which is 1.1 meters in this case.
- GAIN_D parameter influences how fast the UAV reaches the given the setpoint. Higher values can make it very responsive and can be kept to very less value or zero as in case of pelican.
- ALT_SPEED_MEASURE_SATURATION_FACTOR parameter deprecated.
- COMMAND_UPPER_LIMIT and COMMAND_LOWER_LIMIT set the maximum and minimum saturations for the altitude controller.
- SUBSAMPLING_DIVIDER currently not used.

The next set of parameters define are used for tuning the hover controller.

```
<hover_controller>

  <GAIN_P>     9.00 </GAIN_P>      <!-- [deg / (m/s)]  -->

  <GAIN_I>     1.80 </GAIN_I>      <!-- Ti = 5s, (GAIN_P / 5.0)  -->

  <GAIN_D>     0.50 </GAIN_D>      <!-- -->

  <MAX_OUTPUT> 12.00 </MAX_OUTPUT> <!-- [deg] -->

  <ALTITUDE_FOR_NOMINAL_D> 0.70 </ALTITUDE_FOR_NOMINAL_D> <!-- [m]   -->

</hover_controller>
```

- GAIN_P parameters as for all the PID controllers is most important, higher the gain_p the more the oscillations UAV about a given position. Gain_p parameter should be set a value where the UAV is not too sluggish or responsive at a desired position.
- GAIN_I parameter makes the UAV maintain its desired setpoint position. In this case the gain_I should be gain_p/5.
- GAIN_D parameter parameter influences how fast the UAV can reach the desired attitude, hence higher values tend to make the UAV more responsive.
- MAX_OUTPUT parameter defines the minimum and maximum saturations for the hover controller.
- ALTITUDE_FOR_NOMINAL_D is the altitude value upto which the gain_d is considered by the controller after which the gain_d becomes zero.

The next set of values are for the yaw controller which is currently a PD controller.

```
<yaw_controller>

  <GAIN_P> 5.00 </GAIN_P> <!-- [deg/s / (deg)]  -->

  <GAIN_I> 0.00 </GAIN_I> <!--  -->

  <GAIN_D> 0.50 </GAIN_D> <!-- (0.1 * GAIN_P) -->

</yaw_controller>
```

- GAIN_P parameter can be tuned the same way as in the above controllers.
- GAIN_D parameter is 0.1*gain_p

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Add a New Camera

Martin Molina edited this page on 19 May · 2 revisions

Aerostack supports all types of ueye cameras with usb. There are ROS packages in the Aerostack for the ueyecameras which take the data from the camera and then publishes the camera data using the suitable config files provided by the user. Adding a new camera to the Aerostack is relatively easy and can be done in the following steps.

1. Installing the ueyecamera driver
2. Getting the camera_id
3. Generating a suitable config file
4. Generating a camera calibration file
5. Launching the ueyecamera node

1. Installing the Ueyecamera Driver

The most important thing in order to add a new camera to the stack is installing the drivers for the ueyecamera. Note that installing the ueyecamera drivers is mandatory when installing the Aerostack because without the ueyecamera drivers the Aerostack will never be build. But just in case if the ueyecamera drivers are not installed here is how it can be done.

The link to download the ueyecamera can be found here

https://en.ids-imaging.com/download-ueye-lin64.html

The user has to install the drivers for linux. The installing instructions for installing the ueyecamera drivers is properly mentioned in a text file which comes along with the drivers.

2. Getting the camera_id

After installing the ueyecamera drivers, the user has to check the camera_id of the connected ueyecamera. Which can be done as follows. Open the Gnome terminal and type the command ueyecameramanager and a window will open showing all the connected cameras to the computer.

The figure below shows an example of the ueyecameramanager connected with one camera and with all its properties. The camera_id can then be seen under the cam ID section.

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

3. Generating a Suitable Config File

The config file is an important file, using which the user can define all the properties of the camera, for example the parameters like the brightness, frame rate, exposure and many more. In order to set these parameters the camera needs to be opened first. It can be done by right clicking on the available camera in the screen shown in the above figure.

After opening the camera, all the camera parameters can be set as desired by the user. After setting the desired camera parameters, the user then has to save the config file with the desired parameters. The config file has to be saved inside DRONE_STACK/Configs/drone_id. The number of the drone_id depends on the drone which the user is using. For example if the user is using the camera with Asctec Pelican the config file needs to be saved in drone0, as drone0 is currently being used for the Astec Pelican.

Also there are some default config files for the ueye cameras which are already there inside DRONE_STACK/Configs/drone_id. small_camera_RGB24_HUB.ini is one such config file which is already being generated with certain camera parameters. It completely depends on the user to choose as to which camera config is the best.

4. Generating a Camera Calibration File

The camera calibration file contains the necessary camera calibration data like the focal length, projection matrix etc of the camera. The camera calibration file can be generated using the ROS camera calibration package which can be found in the link below:

http://wiki.ros.org/camera_calibration

After generating the suitable calibration file for the camera it can then be saved inside the same folder DRONE_STACK/Configs/drone_id. Where again drone_id depends on the drone the user is using. An example of the calibration file can be found below.

```
image_width: 640
image_height: 360
camera_name: front
camera_matrix:
  rows: 3
  cols: 3
  data: [466.6953647276067, 0, 306.515467461629, 0, 466.1550228548235,
169.6544700845251, 0, 0, 1]
distortion_model: plumb_bob
```

```
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.3088667172761168, 0.1060937937543472, -0.001639357143514502,
0.0004536735112699787, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [391.9715881347656, 0, 301.7997751755956, 0, 0, 442.0516967773438,
167.6912864963015, 0, 0, 0, 1, 0]
```

Note that the format of the calibration file should be same as the above file or the ueye package in the Aerostack will fail to start. The camera calibration file has to be saved with the same camera name which is will be used by the user in the launch file to launch the camera. For example, if the name of the camera is front, the name of the calibration file has to be front.yaml.

5. Launching the Ueyecamera Node

The user has to make a suitable launch file in order to launch the ueyecamera with the proper config file and the calibration file as well as with the proper camera name and camera_id. An example of such a launch file can be found below.

```
<!-- launches: ueye_cvg node -->
<launch>
    <arg name="camera_id_num" default="1"/>
    <arg name="config_file" default="small_camera_RGB24.ini"/>
    <arg name="camera_name_str" default="front" />

    <arg name="drone_id_int" default="0" />
     <arg name="my_stack_directory" default="$(env DRONE_STACK)" />

    <group ns="drone$(arg drone_id_int)">
     <node name="cvg_ueye_cam$(arg camera_id_num)_$(arg camera_name_str)" pkg="driver_cam
      <param name="camera_ident_number" value="$(arg camera_id_num)" type="int"/>
      <param name="configuration_file" value="$(arg my_stack_directory)/configs/drone$(ar
      <param name="camera_name"   value="$(arg camera_name_str)" type="string"/>
      <param name="topic_name" value="camera/$(arg camera_name_str)"  type="string"/>
      <param name="droneId" value="$(arg drone_id_int)" type="int"/>
      <param name="stackPath" value="$(arg my_stack_directory)" type="string"/>
      <param name="cam_intr_filename"  value="$(arg my_stack_directory)/configs/drone$(ar
     </node>
   </group>
</launch>
```

The above launch file launches the ueyecamera node in the Aerostack with parameters and files as explained below:

- The first argument in the launch file is the camera_id where the user inputs the camID seen from the ueyecameramanager. In this example the camera_id is 1.
- The second argument defines the name of the config file, which is small_camera_RGB24.ini in this case above.
- The third argument is the camera name which is front in the above case. The fourth and the fifth argumets defines the drone_id being used and the environment for the Aerostack.

After all the arguments the ueyecamera ROS node is launched with the name "cvg_ueye_cam$(arg camera_id_num)_$(arg camera_name_str)". Hence here in the above case the name of the ueyenode would be cvg_ueye_cam1_front.

After launching the node, the parameters for the node are set:

- The first parameter is the camera_id.
- The second parameter is the assigns the camera config file, which takes the value "$(arg my_stack_directory)/configs/drone$(arg drone_id_int)/$(arg config_file)". In the above it is
- $DRONE_STACK/configs/drone0/small_camera_RGB24.ini
- The third parameter is the name of the camera, in the above case the camera name is front.
- The fourth parameter defines the topic name which contains the camera data, which takes the camera/$(arg camera_name_str). In the above case it is camera/front.
- The fifth and the sixth parameters are the drone_id and the stack_path.
- The seventh parameter is the assigns the camera calibration file, which takes the value "$(arg my_stack_directory)/configs/drone$(arg drone_id_int)/$(arg camera_name_str).yaml". In the above case the calibration file is assigned as follows
  $DRONE_STACK/configs/drone0/front.yaml"

The above launch file can be launched using the roslaunch command assigning all the camera arguments. For example:

```
roslaunch driver_camera_ueye_ROSModule driver_camera_ueye_ROSNode
--wait camera_id_num:=2
         camera_name_str:="front"
         config_file:="small_camera_RGB24_HUB.ini"
```

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Add a GPS Sensor

Martin Molina edited this page on 19 May · 2 revisions

This section describes with an example how to add a new sensor to a UAV platform using the Aerostack. There might be several sensors that can be added to the Aerostack according to its usage, the important thing to keep in mind is that when adding a new sensor to the stack the sensor should publish its data in ROS and there should be an appropriate translator/interface that receives this sensor data. This translator or interface then converts the data in the units and conventions required by the stack and then publishes it to the appropriate module of the stack. In the following sections an example of adding a GPS sensor to the Aerostack is presented.

This example illustrates adding a particular type of GPS sensor shown in the figure below. GPS is the most commonly used sensor with UAV's, since it can give a good position estimation of the UAV with reference to a global frame of reference, the only drawback of GPS is that is cannot be used indoors or in cluttered environments for example in places with a lot of trees or buildings.

The position estimated by the GPS always has a drift, hence GPS data has to be given to the extended kalman filter in the Aerostack in order to obtain a better pose estimation of the UAV with respect to a given frame of reference.

The following steps have to performed in order to add a GPS sensor in the Aerostack

1. A GPS sensor with USB connection sending NMEA sentences (shown in figure below).
2. A ROS package that listens to this NMEA sentences and then converts them to latitude and longitude values.
3. A ROS package which converts this latitude and longitude values into proper x, y and z coordinates in Universal Transverse Mercator coordinate system (UTM coordinate system).
4. A ROS package to convert these x, y and z values in UTM coordinate system to appropriate coordinate system required by the Aerostack. This package is called the translator or the interface for the GPS sensor.
5. Adding few changes in the extended kalman filter package to subscribe to the x, y, z values published by the GPS interface package.

A detailed explanation of all this steps is given below step by step.

1. Choosing a GPS Sensor

The first and the most important part in choosing a GPS is that it should publish NMEA sentences, because all the packages developed in ROS for GPS modules assume that the GPS module sends NMEA sentences.

One can also use the GPS data given by a GPS which is attached to the autopilot of UAV. In this case, the user has to hack the autopilot to publish the appropriate GPS data for the following package which will be subscribing to this data. The figure below shows the USB GPS being used in this example which can be bought from a local electronics store at a low cost.

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

2. ROS Module to Get the Data from the GPS Sensor

There is already a package available in ROS which listens to the NMEA sentences given by the GPS sensor and converts them into appropriate latitude and longitude values. The ROS package can be found under this link:

https://github.com/ros-drivers/nmea_navsat_driver

A launch file can be made to launch the node of this package which is as follows:

```
<!-- Launches: nmea_navsat_driver_node -->
<launch>

  <arg name="drone_id_namespace"  default="drone0" />
  <arg name="drone_id_int"                default="0" />
  <arg name="my_stack_directory"     default="$(env DRONE_STACK)" />

  <group ns="$(arg drone_id_namespace)">

        <node name="nmea_navsat_driver_node" pkg="nmea_navsat_driver" type="nmea_serial_d

        <param name="port"    value="/dev/ttyUSB0"  type="string" />
        <param name="baud"   value="4800"               type="int"   />
        </node>
 </group>
</launch>
```

This file launches the 'nmea_serial_driver' node with the name 'nmea_navsat_driver_node' from the package 'nmea_navsat_driver'. The param port defines the port at which the GPS is connected and the baud which should always be kept at 4800 in case of this GPS.

3. ROS Module Which Provides Position in UTM Coordinates

This is also a standard ros package which can be downloaded from github:

https://github.com/ktossell/gps_umd/tree/master/gps_common

This package converts the latitude and longitude values provided by the ROS Module in the previous section into x, y and z position in UTM coordinates.

A launch file can be made to launch the node of this package which is as follows:

```
<!-- Launches: gps_umd_node -->
<launch>

  <arg name="drone_id_namespace"  default="drone0" />
  <arg name="drone_id_int"                default="0" />
  <arg name="my_stack_directory"     default="$(env DRONE_STACK)" />

   <group ns="$(arg drone_id_namespace)">
```

```
        <node name="gps_umd" pkg="gps_common" type="utm_odometry_node" output="screen" res

        </node>
    </group>
  </launch>
```

This file launches the 'utm_odometry_node' with a name 'gps_umd' from the package 'gps_common'.

4. ROS Interface/Translator for the Received GPS Data

This package converts the position data given by the utm_odometry_node into data which is required by the Aerostack. At first, for the Aerostack the X axis should always be pointing towards the north and the Y axis towards the east but in case of utm_odometry_node the X axis points towards the east and Y axis points towards the north. Hence when this package receives data from the utm_odometry_node it first changes the axis as shown in the following code below. This piece of code is a callback function in this package which is called when this package receives the data from the utm_odometry_node.

```
void GPSOdomROSModule::GPSOdomCallback(const nav_msgs::Odometry &msg)
{
        //Asynchronous module with only one callback!
        if(!run())
        return;

        GPS_pose.x = msg.pose.pose.position.y;
        GPS_pose.y = msg.pose.pose.position.x;
        GPS_pose.z = msg.pose.pose.position.z;

        ConvertToLocal(&GPS_pose);

        publishGPSValue();
        return;
}
```

Where:

- GPS_pose.x is Pose in x axis which will be published to the Aerostack,
- msg.pose.pose.position.y is Pose in y axis which was published by the utm_odometry_node,
- The convention goes similar for other axis.

The function ConvertToLocal(&GPS_pose) takes the current pose given utm_odometry_node and substracts it from the pose at the initial point (Initial pose is the posewhich was sent by the GPS was got connected). For example, if the GPS is connected at a particular point the utm_ododmetry_node publishes data in x = 44545 y = 4482, z = 650, but since this being the initial point the data should always be x = 0, y = 0 and z = 0, and hence the function ConvertToLocal(&GPS_pose) converts all the start point data to zero.

```
bool GPSOdomROSModule::GlobalToLocal(droneMsgsROS::dronePose *current)
{
        static droneMsgsROS::dronePose map_origin;
        static bool first_pose_received = false;

        bool initial_pose = !first_pose_received;

        if(initial_pose)
        {
        map_origin   = *current;
        map_origin.x = map_origin.x;
        map_origin.y = map_origin.y;
        map_origin.z = map_origin.z;
```

```
        first_pose_received = true;

        ROS_INFO("INITIAL data (%.3f, %.3f, %.3f), map origin (%.3f, %.3f, %.3f)",
        current->x, current->y, current->z,
        map_origin.x, map_origin.y, map_origin.z);
        }

        current->x -= map_origin.x;
        current->y -= map_origin.y;
        current->z -= map_origin.z;

        return initial_pose;
}
```

The variable map_origin has the data when the GPS started sending data. The current data is then subtracted from the map_origin data to get distances with respect to the start point.

5. Publishing the GPS Data to Aerostack

The translated data given by a sensor now has to be processed by a suitable module in the Aerostack. In case of the GPS data the x, y and z positions need to be given to EKF_Module to get a better pose estimate from it.

But before giving the gps data to the EKF it first needs to be converted to the frame of reference used by the EKF. The frame of reference used by the EKF can be seen inside documention/Coordinate_frames/General_Frames_Field.svg. As can be seen from the figure the EKF takes measurements in Fdrone__Lmrt wrt to FEKF Frame. This fixed EKF frame always aligns itself to the heading of the UAV when the EKF is started and then the heading of the fixed EKF frame remains constant.

The GPS Frame of reference is always fixed with x pointing towards the north and y pointing towards the east, and hence by using the yaw_degree given by the magnetometer the GPS fixed frame can be aligned to the EKF fixed frame as soon as the EKF is started. Below is a code showing how this reference frame conversion is being done.

```
        float x_drone,y_drone;

        GlobalFrame(0,0) = x_gps;
        GlobalFrame(1,0) = y_gps;
        GlobalFrame(2,0) = z_gps;

        RotationMat(0,0) = cos(yaw_drone);
        RotationMat(1,0) = -sin(yaw_drone);
        RotationMat(2,0) = 0;

        RotationMat(0,1) = sin(yaw_drone);
        RotationMat(1,1) = cos(yaw_drone);
        RotationMat(2,1) = 0;

        RotationMat(0,2) = 0;
        RotationMat(1,2) = 0;
        RotationMat(2,2) = 1;

        BodyFrame = RotationMat*GlobalFrame;

        x_drone = BodyFrame(0,0);
        y_drone = BodyFrame(1,0);
```

The yaw_drone parameter is taken only once when the EKF is started or it is reset, and during rest of time it remains constant with the first value when EKF was started. The x_drone and y_drone are positions converted with respect to EKF frame of reference and are then provided to the EKF and if everything else is correct the user should obtain a pose from the EKF which is fused with the GPS data. Note that in the example above only x and y data from the GPS are given to the EKF the z data is not used as it has a lot of noise.

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Use Aerostack for a New Physical Platform

Martin Molina edited this page on 19 May · 2 revisions

Adding a new physical platform basically means adding a new autopilot system which is not supported by the Aerostack. Sometimes the autopilot comes with a fixed platform like the Ardrone or sometimes it is only the autopilot and the user has to build the platform for the autopilot like Pixhawk. The following example demonstrates how the Pixhawk autopilot was added to the Aerostack.

1. Basic Steps for Setting up Communication Between Aerostack and Pixhawk

When adding a new physical platform the most important thing to have a ROS package of the autopilot being added to the Aerostack. This ROS Package is used for communication between the Autopilot with the Aerostack. As mentioned before the Aerostack is modular and hence in order to communicate with the Aerostack the Autopilot ROS package needs an Interface in order to convert the data coming from the Autopilot in units required by the Aerostack and also to convert the commands sent by the Aerostack in units required by the Autopilot. If the user is successful is making an interface for the Autopilot, the rest of the code for the controllers remains the same, and then its just a matter of tuning the controllers to fly the Autopilot with the Aerostack.

In case of Pixhawk, the ROS package used for communication is the Mavros which is a MAVLink extendable communication node for ROS. An Interface called the driverPixhawkROSModule is then made for the Mavros in order for it to communicate with the Aerostack.

The basic things that the Aerostack requires from any autopilot are the IMU data (rotation angles) and the battery voltage data and for sending commands from the Aerostack, the autopilot should except roll, pitch, dyaw (change in yaw) and throttle commands.

2. Getting the Rotation Angles from the Pixhawk Using Macros

The Aerostack accepts rotation angles in NED (North, East and Down) frame of reference for more information about the frame of reference used in Aerostack please refer to documentation/coordinate_frame/Multirotor_coordinate_frames.txt. It requires the rotations in euler angles and the angles should be in degrees and not in radians.

Mavros on the other hand uses ENU (East, North and Up) frame of reference. Please refer to the following wiki article for more information https://en.wikipedia.org/wiki/Axes_conventions. Also mavros publishes rotations in quaternion.

Hence the driverPixhawkROSModule converts the rotations published by mavros into suitable units required by the Aerostack. Below is a piece of code on how it is done

```
 //deg,  mavwork reference frame
 geometry_msgs::Vector3Stamped RotationAnglesMsgs;

 tf::Quaternion q(msg.orientation.x, msg.orientation.y,
                  msg.orientation.z, msg.orientation.w);

//make a matrix m of quaternion q
tf::Matrix3x3 m(q);


//convert quaternion to euler angels
 m.getEulerYPR(yaw, pitch, roll);
```


Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

```
        RotationAnglesMsgs.vector.x = (double)(+1)*(roll)*180/M_PI;
        RotationAnglesMsgs.vector.y = (double)(-1)*(pitch)*180/M_PI;
        RotationAnglesMsgs.vector.z = (double)(-1)*(yaw)*180/M_PI;

        //converting the yaw which is in the range from -phi to +phi to 0 to 360
       degrees
        if( RotationAnglesMsgs.vector.z < 0 )
        RotationAnglesMsgs.vector.z += 360.0;

        publishRotationAnglesValue(RotationAnglesMsgs);
```

The above code takes the rotations given by mavros in quaternion converts them into roll, pitch and yaw angles. This angles are then converted from ENU frame of reference to NED frame of reference and also converted from radians to degrees. The yaw angle which is in the range of -180 to +180 degrees is then converted in the range of 0 to 360 degrees. These angles are then finally published to the Aerostack.

3. Getting the Battery Voltage from the Pixhawk Using Macros

Mavros provides the current battery voltage using which the current battery percentage can be calculated. This percentage is then displayed on the user interface used for flying, which can be very important during a mission or normal flight to determine if the mission or flight can be carried out with a given percentage of battery. The code below determines how to convert the battery voltage to percentage required by the Aerostack.

```
        droneMsgsROS::battery BatteryMsgs;

        // Read Battery from navdata
        BatteryMsgs.header   = msg.header;
        // minimum voltage per cell used by the "digital battery checker" is
       3.55 V
        // msg.voltage is in V
        float battery_voltage = (msg.voltage);

       //Battery used in this case is 4 cell battery
        BatteryMsgs.batteryPercent = ((battery_voltage - 14.8)/(16.8-14.8))*100;

        publishBatteryValue(BatteryMsgs);
```

The above code is used to convert the battery voltage to battery percentage for a 4 cell battery and hence the lower voltage limit is 14.8V and higher voltage limit is 16.8V. If using a 3 cell battery the lower voltage limit should be 11.0V and higher voltage limit should be 12.6V.

4. Sending Commands to the Pixhawk Using Macros

There are several ways of sending commands to the pixhawk using mavros, by giving a certain position using position setpoints or directly setting the attitude and altitude using attitude setpoints.

This example uses roll, pitch, yaw and throttle commands from the Aerostack and sends to the pixhawk using mavros using setpoint_raw/attitude topic.

The Aerostack produces commands for any autopilot in terms of roll, pitch, dyaw (change in yaw) and throttle and in NED frame of reference.

The mavros on the other hand accepts commands in terms of roll, pitch, yaw and throttle but the roll, pitch and yaw have to be in quaternion and in ENU frame of reference. The throttle also has to be in the range from 0 to 1 counts. The driverPixhawkROSModule does all this conversions and finally sends the commands to the pixhawk using mavros.

The code below shows how commands are sent using mavros to the pixhawk.

```
//message type for sending commands to mavros
 mavros_msgs::AttitudeTarget attitude_msg;

 //adding the current yaw angle to the dyaw command
 yaw_command = yaw_angle + msg.dyaw;

 //convert the values in radians and from NED frame(Aerostack) to ENU
frame (Mavros)
 roll_command  = (+1)*msg.roll*(M_PI/180.0);
 pitch_command = (-1)*msg.pitch*(M_PI/180.0);
 yaw_command   = (-1)*yaw_command*(M_PI/180.0);

 //convert euler angles to quaternion but in eigen form
 quaterniond =  mavros::UAS::quaternion_from_rpy(roll_command,
                                  pitch_command, yaw_command);

 //convert quaternion in eigen form to geometry_msg form
 tf::quaternionEigenToMsg(quaterniond, orientation);

 attitude_msg.orientation = orientation;

 attitude_msg.thrust = msg.thrust * THRUST_SCALE;

 drone_CtrlInput_publisher.publish(attitude_msg);
```

In the above code the dyaw command is first converted into yaw by adding the current yaw angle to it because the mavros needs commands in yaw and not in dyaw. The code then converts rotation angles from degrees to radians and then the angles to quaternions.

Then code then converts the throttle in the range defined in the Aerostack to the range of 0 to +1 required by the mavros by using the parameter THRUST_SCALE. The value of the THRUST_SCALE depends on the throttle range defined in midlevel controller in the Aerostack.

For example if the throttle is in the range of 0 to 32 Newtons in the Aerostack (midlevel controller), then the THRUST_SCALE has to be 1/32 in order to convert the throttle command in the range of 0 to +1 for macros.

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com

Publications

Edit New Page

Martin Molina edited this page on 19 May · 9 revisions

The functionality of Aerostack is explained in the following papers. Please, refer to them in your work.

Core Papers

- J.L. Sanchez-Lopez, M. Molina, H. Bavle, C. Sampedro, R. A. Suarez-Fernandez, P. Campoy. (2017). A Multi-Layered Component-Based Approach for the Development of Aerial Robotic Systems: The Aerostack Framework. Journal of Intelligent & Robotic Systems, 1-27.
- J. L. Sanchez-Lopez, R. A. Suarez-Fernandez, H. Bavle, C. Sampedro, M. Molina, J. Pestana, P. Campoy (2016): AEROSTACK: An Architecture and Open-Source Software Framework for Aerial Robotics. ICUAS 2016, Arlington, USA.
- J.L. Sanchez-Lopez, J. Pestana, P. de la Puente, P. Campoy (2015): "A reliable Open-source System Architecture for the Fast Designing and Prototyping of Autonomous Multi-UAV Systems: Simulation and Experimentation". Journal of Intelligent and Robotic Systems. Online: Oct. 2015. Springer Netherlands. Online ISSN: 0921-0296. DOI: 10.1007/s10846-015-0288-x.

Other related papers:

- J. L. Sanchez-Lopez, J. Pestana, P. de la Puente, R. Suarez-Fernandez, P. Campoy. A System for the Design and Development of Vision-based Multi-robot Quadrotor Swarms. 2014 International Conference on Unmanned Aircraft Systems (ICUAS'14). Orlando (FL, USA). May 27-30, 2014. Pp: 640-648. DOI: 10.1109/ICUAS.2014.6842308.

Mission Planning

- M. Molina, A. Diaz-Moreno, D. Palacios, R. A. Suarez-Fernandez, J.L. Sanchez-Lopez, C. Sampedro, H. Bavle, P. Campoy (2016): Specifying Complex Missions for Aerial Robotics in Dynamic Environments. International Micro Air Vehicle Conference and Competition, IMAV 2016. Beijing, China.
- C. Sampedro, H. Bavle, J. L. Sanchez-Lopez, R. A. Suarez-Fernandez, A. Rodriguez-Ramos, M. Molina, and P. Campoy (2016): A Flexible and Dynamic Mission Planning Architecture For UAV Swarm Coordination. ICUAS 2016, Arlington, USA.

Controllers

- J. Pestana, I. Mellado-Bataller, C. Fu, J. L. Sanchez-Lopez, I. F. Mondragon, P. Campoy. A General Purpose Configurable Controller for Indoors and Outdoors GPS-Denied Navigation for Multirotor Unmanned Aerial Vehicles. Journal of Intelligent and Robotic Systems. Jan. 2014. Vol 73, Issue 1-4, pp. 387-400. Springer Netherlands. Print ISSN: 0921-0296. Online ISSN: 1573-0409. DOI: 10.1007/s10846-013-9953-0
- J. Pestana, I. Mellado-Bataller, J. L. Sanchez-Lopez, C. Fu, I. F. Mondragon, P. Campoy. Floor Optical Flow Based Navigation Controller for Multirotor Aerial Vehicles. Robot 2013: First Iberian Robotics Conference (ROBOT 2013). Madrid (Spain). Nov. 28-29, 2013. Book title: Robot 2013: First Iberian Robotics Conference. Book subtitle: Advances in intelligent Systems and Computing. Vol. 253. Published year: 2014. Pp: 91-106. DOI: 10.1007/978-3-319-03653-3_8. Print ISBN: 978-3-319-03652-6. Online ISBN: 978-3-319-03653-3. Publisher: Springer International Publishing.

Pages 28

Introduction

- What is Aerostack?
- Getting Started
- License
- List of distributions
- Aerostack Virtual Machine

Components

- Aerostack Architecture
- Catalog of Available Processes
- Human Machine Interface
- TML Language

Tutorials [Beginners]

- Prepare the Mission:
  - Configure Map
  - Specify Mission Tasks
  - Configure Launcher
- Execute the Mission:
  - Launch Real Flight
  - Practical Tips for Real Flights
- Mission examples:
  - Simulated Quadrotor
  - AR Drone and Visual Markers

Tutorials [Advanced]

- Setup the Communication Network
- Wi Fi Communication with Multiple Drones
- Calibrate Controllers
- Add a New Camera
- Add a GPS Sensor
- Use Aerostack for a New Physical Platform

More information

- Publications
- Vision4UAV:
  - First prize at IMAV 2013
  - Stack pre-release 2013

Clone this wiki locally

https://github.com/Vision4U

Clone in Desktop

- J. Pestana, I. Mellado-Bataller, C. Fu, J. L. Sanchez-Lopez, I. Fernando Mondragon, P. Campoy. A General Purpose Congurable Navigation Controller for Micro Aerial Multirotor Vehicles. 2013 International Conference on Unmanned Aircraft Systems (ICUAS'13). Atlanta, Georgia (USA). May 28-31, 2013. DOI: 10.1109/ICUAS.2013.6564733. Print ISBN: 978-1-4799-0815-8. Publisher: IEEE. Pages: 557 - 564

Human Robot Interaction

- R.A. Suarez Fernandez, J. L. Sanchez-Lopez, C. Sampedro, H. Bavle, M. Molina, and P. Campoy (2016): Natural User Interfaces for Human-Drone Multi-Modal Interaction. ICUAS 2016, Arlington, USA.

Visual Servoing

- J. Pestana, J. L. Sanchez-Lopez, S. Saripalli, P. Campoy. Computer Vision Based General Object Following for GPS-denied Multirotor Unmanned Vehicles. 2014 American Control Conference (ACC 14). Portland (OR, USA). Jun. 4-6, 2014. Pp: 1886-1891. ISSN: 0743-1619. DOI: 10.1109/ACC.2014.6858831.

- J. Pestana, J. L. Sanchez-Lopez, S. Saripalli, P. Campoy. Vision based GPS-denied Object tracking and Following for Unmanned Aerial Vehicles. The 2013 (11th) IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR 2013). Linköping (Sweden). Oct 21-26, 2013. Best Paper Award Finalist. Pages: 1-6. Print ISBN: 978-1-4799-0879-0. DOI: 10.1109/SSRR.2013.6719359

Competitions

IARC 2014:

- J. L. Sanchez-Lopez, J. Pestana, J.-F. Collumeau, R. Suarez-Fernandez, P. Campoy, M. Molina. A Vision Based Aerial Robot solution for the Mission 7 of the International Aerial Robotics Competition. Unmanned Aircraft Systems (ICUAS), 2015 International Conference on. Denver, (CO, USA). 9-12 June 2015. Pp: 1391 – 1400. Print ISBN: 978-1-4799-6009-5. DOI: 10.1109/ICUAS.2015.7152435

IMAV 2013:

- J. Pestana, J. L. Sanchez-Lopez, P. de la Puente, A. Carrio, P. Campoy. A Vision-based Quadrotor Multi-Robot Solution for the Indoors Autonomy Challenge of the 2013 International Micro Air Vehicle Competition. Journal of Intelligent and Robotic Systems.

- J. Pestana, J. L. Sanchez-Lopez, P. de la Puente, A. Carrio, P. Campoy. A Vision-based Quadrotor Swarm for the participation in the 2013 International Micro Air Vehicle Competition. 2014 International Conference on Unmanned Aircraft Systems (ICUAS'14). Orlando (FL, USA). May 27-30, 2014. Pp: 617-622. DOI: 10.1109/ICUAS.2014.6842305.

- J. L. Sanchez-Lopez, J. Pestana, P. de la Puente, A. Carrio, P. Campoy. Visual Quadrotor Swarm for the IMAV 2013 Indoor Competition. Robot 2013: First Iberian Robotics Conference (ROBOT 2013). Madrid (Spain). Nov. 28-29, 2013. Book title: Robot 2013: First Iberian Robotics Conference. Book subtitle: Advances in intelligent Systems and Computing. Vol. 253. Published year: 2014. Pp: 55-63. DOI: 10.1007/978-3-319-03653-3_5. Print ISBN: 978-3-319-03652-6. Online ISBN: 978-3-319-03653-3. Publisher: Springer International Publishing.

Contact

We thank and welcome any suggestion or comment about Aerostack. For any question or bug report you can read and/or write at the issues page.

You can also contact the team support at the following address: aerostack.vision4uav@gmail.com