

Autonomous Navigation Behaviors for an Aerial Robotics Software Framework

Guillermo Echegoyen Blanco
Technical University of Madrid
2019



Overview

- 1. Introduction
- 2. Problem Description
- 3. State of the Art
- 4. Implementation
- 5. Experiments
- 6. Conclusions
- 7. References

Introduction

Autonomous Navigation

Move a drone through an automatically-generated, obstacle-free trajectory to a given point.

- Planning
- Localization
- Mapping

Introduction # Objectives

As a **general goal** we want to:

- Construct an indoor *Autonomous Navigation* interface inside Aerostack using a **lidar** sensor.

We propose the use of different state of the art algorithms following the Aerostack framework. The **specific goals** are:

- Contribute to the research community
- Test implementation suitable for real time aerial robotics applications.

Problem Description

Problem Description # Aerostack

General purpose, modular software framework for aerial robotics.

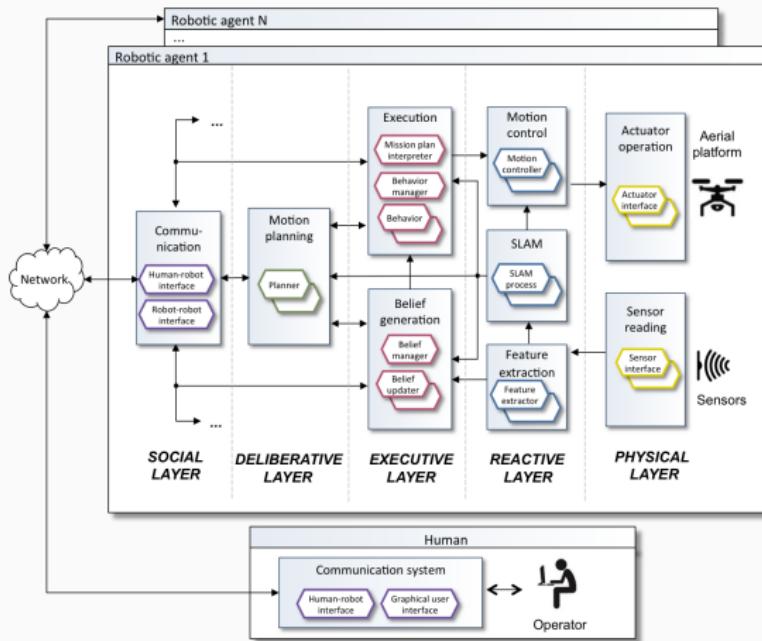


Figure 1: The Aerostack architecture

Problem Description # Description

Aerostack

Current version features:

- Planner: 2D geometric.
- Localization: Odometry, Aruco (*requires preparation*)
- Mapping: Hardcoded in the robot.

Problem Description # Improvements

To provide an **autonomous, lidar** based navigation interface we propose:

- A module to localize and map using **lidar** sensor.
- A module to generate obstacle-free trajectories using that map.
- A module to follow a given trajectory.

State of the Art

State of the Art # Localization

Usually based on trilateration.

Outdoor Localization

External sources:

- GSM Antennas
- Satellites

Indoor Localization

Landmarks, beaconing (can provide more information)

- Bluetooth
- WiFi

In general, preparation of the environment is required.

State of the Art # SLAM

With Simultaneous Localization and Mapping, the robot constructs a map and localizes inside it at the same time.

Does not require environment preparation.

- EKF Slam
- Particle Filters
- Graph Optimization
- Lidar Slam
- Visual Slam

State of the Art # Lidar Slam

Use a **lidar** sensor to do SLAM (Hector Slam [**hector slam**]).

- Ray traces alignment as a Gauss-Newton minimization problem.
- Multi resolution occupancy grid map.
- Already available as ROS node.
- Suitable for real time
- Reliable enough

State of the Art # Hector Slam

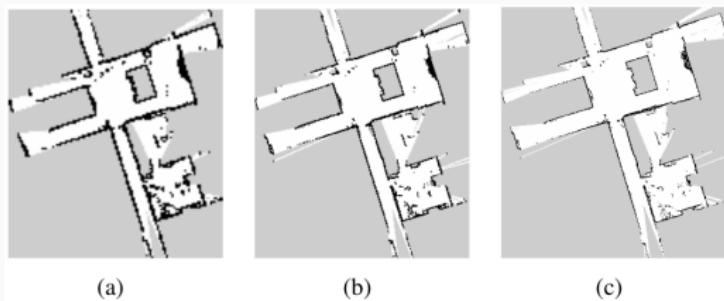


Figure 2: Multiresolution representation of the map



Figure 3: Vehicle experiments: Learned map during a competition

Both images taken from [\[hector`slam\]](#)

State of the Art # Planning

Lots of planning algorithms:

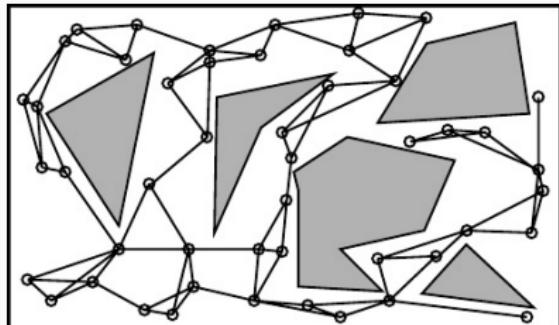


Figure 4: Probabilistic roadmaps
(taken from [choset2005a])

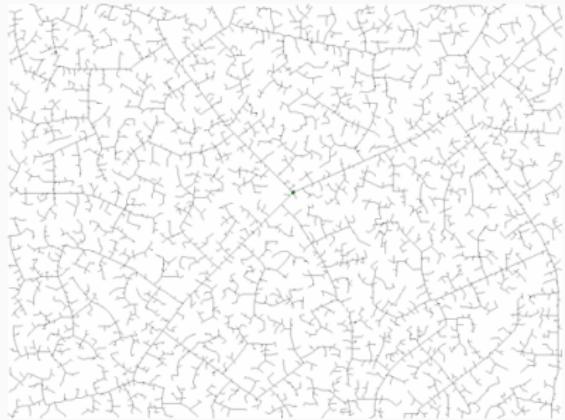
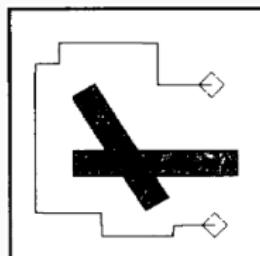


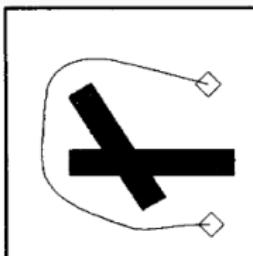
Figure 5: Rapidly exploring
Random Trees (taken from
[wikipedia·rrts·web])

State of the Art # Elastic Band Planner

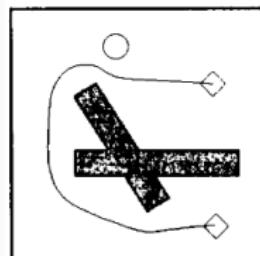
Already available as a ROS module (**move base**)



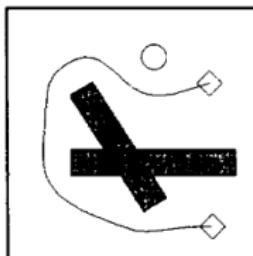
a)



b)



c)



d)

Pre planned path

- Forces applied
- Handle moving obstacles

In c) and d) an obstacle moves and the path is deformed.

Figure 6: Elastic Band with a moving obstacle (taken from [eband])

Implementation

Implementation # ROS

Robot Operating System. Widely used in robotics research.

- Pub-Sub architecture (topics & services).
- Multiple programming languages (C, C++, Python...)
- Logic encapsulation

Implementation # Aerostack

Aerostack features:

- ROS node
- Robot processes.
- Behavior processes.

Implementation # Behavior System

The Behavior Manager coordinates the execution of all the behaviors. A behavior:

- Is the highest level abstraction component.
- Encapsulates an algorithm.
- Listens to *start* and *stop* events and emits *error* events.
- Can have *exclusive constraints* and dependencies on other processes.

Implementation # Navigation Interface

Therefore, the following components should be provided as behaviors or robots processes:

- **Behavior** to localize using lidar.
 - Behavior *Self Localize And Map by Lidar*
 - Hector Slam ROS module (monitored by ↑)
- **Behavior** to navigate the drone to a certain pose.
 - Behavior *Go To Point*
 - Behavior *Follow Path*
 - Behavior *Generate Path*
- **Robot process** to plan obstacle-free trajectories using lidar.
 - *Path planner robot process*
 - Move base ROS module (monitored by ↑)

Implementation # Implemented Behaviors

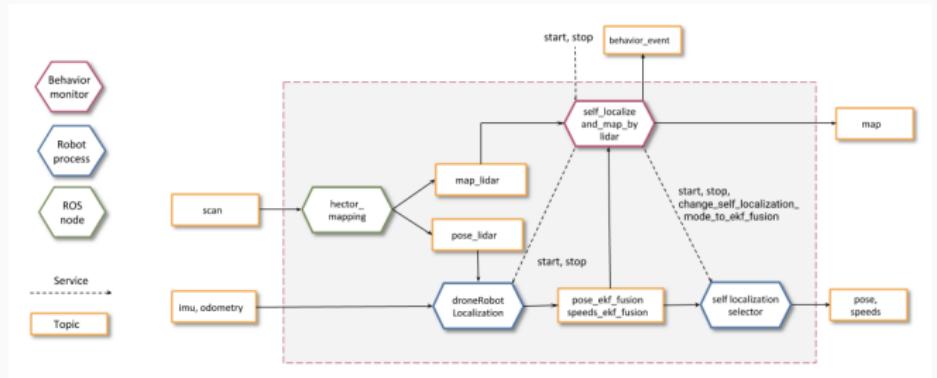
Implementing each behavior separately enables:

- Different, granular control over functionality
- Testability of components
- Separation of functionality.

Implementation # Behavior Self Localize And Map by Lidar

- Monitors the *Hector Slam* ROS module
- Applies EKF to: IMU, Odometry, SLAM
- Instructs the Aerostack for localization and mapping.

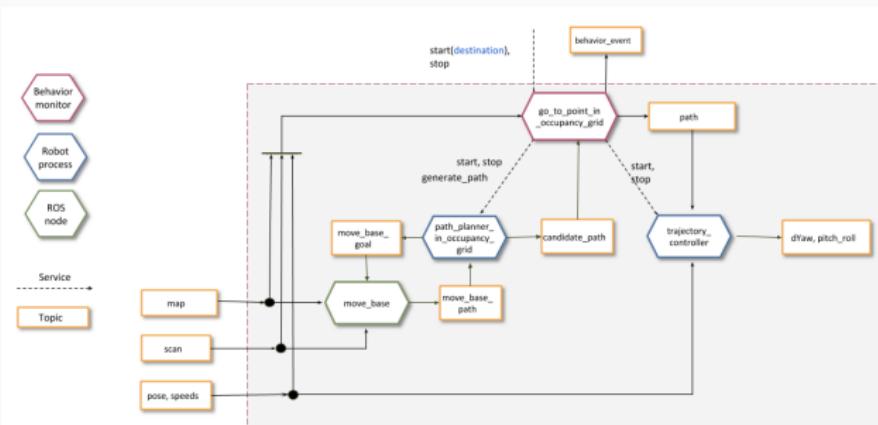
```
#-----  
# SELF_LOCALIZE_AND_MAP_BY_LIDAR  
#-----  
# behavior: SELF_LOCALIZE_AND_MAP_BY_LIDAR  
timeout: 120  
processes:  
  - hector_mapping  
  - droneRobotLocalization  
  - self_localization_selector  
#-----  
# Self-localization behaviors are mutually exclusive  
#-----  
# mutually_exclusive:  
  - SELF_LOCALIZE_BY_ODOMETRY  
  - SELF_LOCALIZE_BY_VISUAL_MARKERS  
  - SELF_LOCALIZE_AND_MAP_BY_LIDAR
```



Implementation # Behavior Go To Point in Occupancy Grid

Given a point:

- Gets an obstacle-free trajectory.
- Instructs the trajectory controller to move the drone.
- Replans if an obstacle arises in the way.

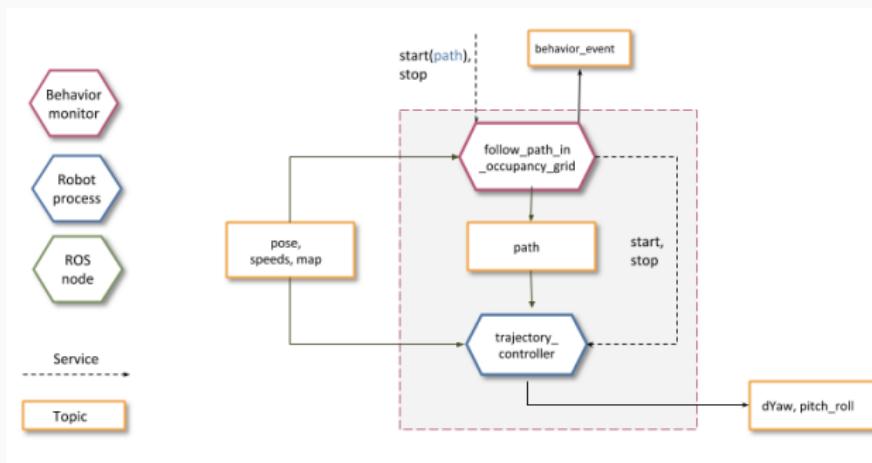


```
#
# GO_TO_POINT_IN_OCCUPANCY_GRID
#
behavior: GO_TO_POINT_IN_OCCUPANCY_GRID
timeout: 240
processes:
- move_base
- path_planning_in_occupancy_grid
- droneTrajectoryController
arguments:
- argument: coordinates
  allowed_values: [-100,100]
  dimensions: 4
#
# Motion behaviors are mutually exclusive
#
mutually_exclusive:
- TAKE_OFF
- LAND
- KEEP_HOVERING
- GO_TO_POINT
- ROTATE
- KEEP_MOVING
- FOLLOW_OBJECT_IMAGE
- GO_TO_POINT_IN_OCCUPANCY_GRID
- FOLLOW_PATH_IN_OCCUPANCY_GRID
-
mutually_exclusive:
- GENERATE_PATH_IN_OCCUPANCY_GRID
- FOLLOW_PATH_IN_OCCUPANCY_GRID
- GO_TO_POINT_IN_OCCUPANCY_GRID
```

Implementation # Behavior Follow Path in Occupancy Grid

Given a path:

- Instructs the trajectory controller to move the drone.
- No obstacle avoidance is provided.



```
#-
# FOLLOW_PATH_IN_OCCUPANCY_GRID
#-
- behavior: FOLLOW_PATH_IN_OCCUPANCY_GRID
timeout: 240
processes:
  - path_planning_in_occupancy_grid
  - move_base
  - droneTrajectoryController
arguments:
  - argument: path
  dimensions: 1

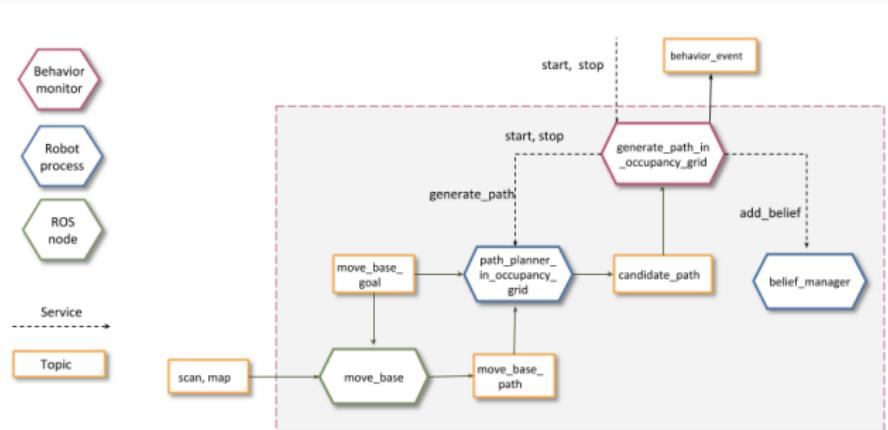
#-
# Motion behaviors are mutually exclusive
#-
- mutually_exclusive:
  - TAKE_OFF
  - LAND
  - KEEP_HOVERING
  - GO_TO_POINT
  - ROTATE
  - KEEP_MOVING
  - FOLLOW_OBJECT_IMAGE
  - GO_TO_POINT_IN_OCCUPANCY_GRID
  - FOLLOW_PATH_IN_OCCUPANCY_GRID
- mutually_exclusive:
  - GENERATE_PATH_IN_OCCUPANCY_GRID
  - FOLLOW_PATH_IN_OCCUPANCY_GRID
  - GO_TO_POINT_IN_OCCUPANCY_GRID
```

Implementation # Behavior Generate Path in Occupancy Grid

Given a point:

- Gets an obstacle-free trajectory.
- Stores the trajectory in the Belief Memory

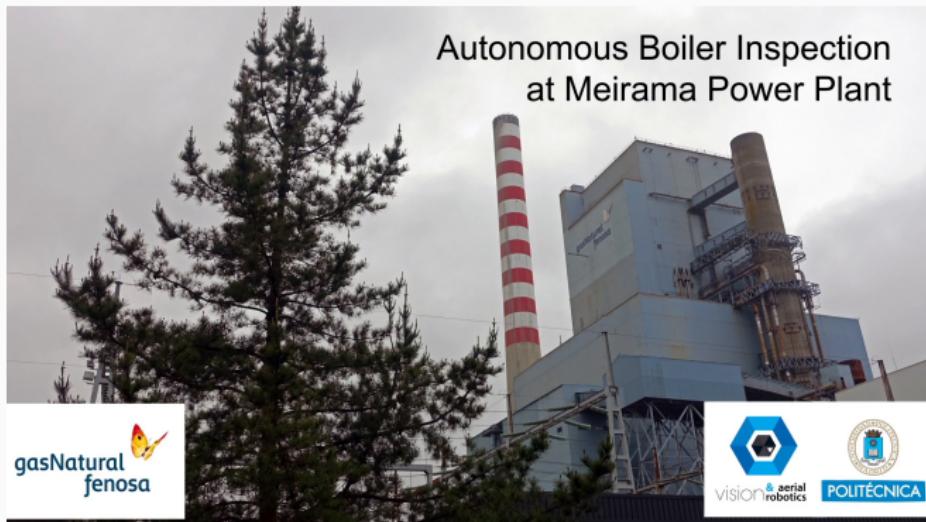
```
#-----
# GENERATE_PATH_IN_OCCUPANCY_GRID
#-----
- behavior: GENERATE_PATH_IN_OCCUPANCY_GRID
  timeout: 120
  processes:
    - move_base
    - path_planning_in_occupancy_grid
  arguments:
    - argument: coordinates
      allowed_values: [-100,100]
    dimensions: 3
```



Experiments

Experiments

Boiler inspection mission



Experiments # Experiment Details

Boiler mission both in simulation and real flight:

Simulation

- Gazebo + RotorS simulator
- Hummingbird Drone
- 16x16x57 - WxDxH mts

Real Flight

- Sports Center
- DJI 100 Matrice Drone
- 10x25x14 - WxDxH mts

Experiments # Simulated Experiment

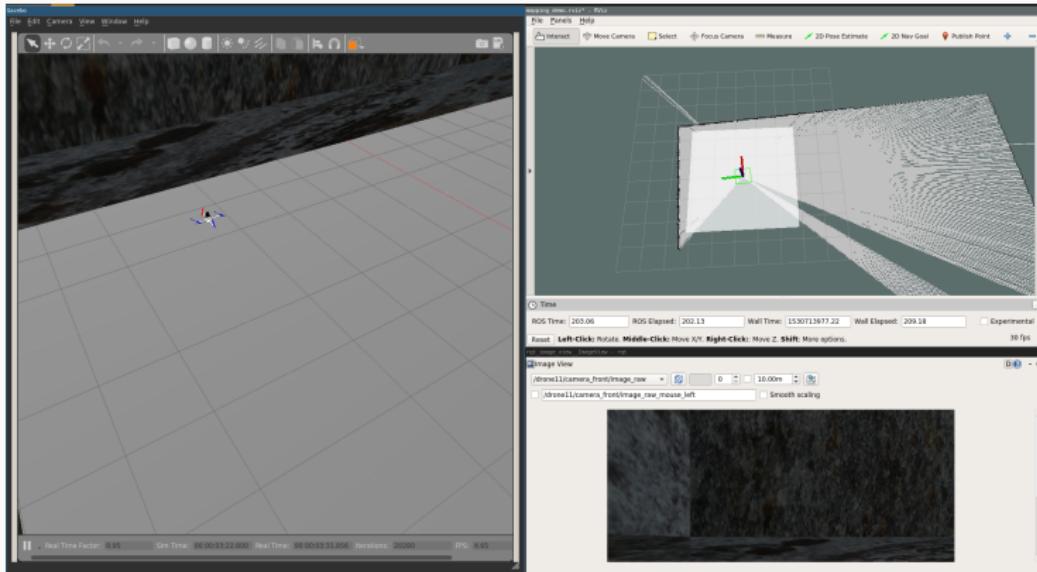


Figure 7: Simulated boiler environment

Experiments # Real Experiment

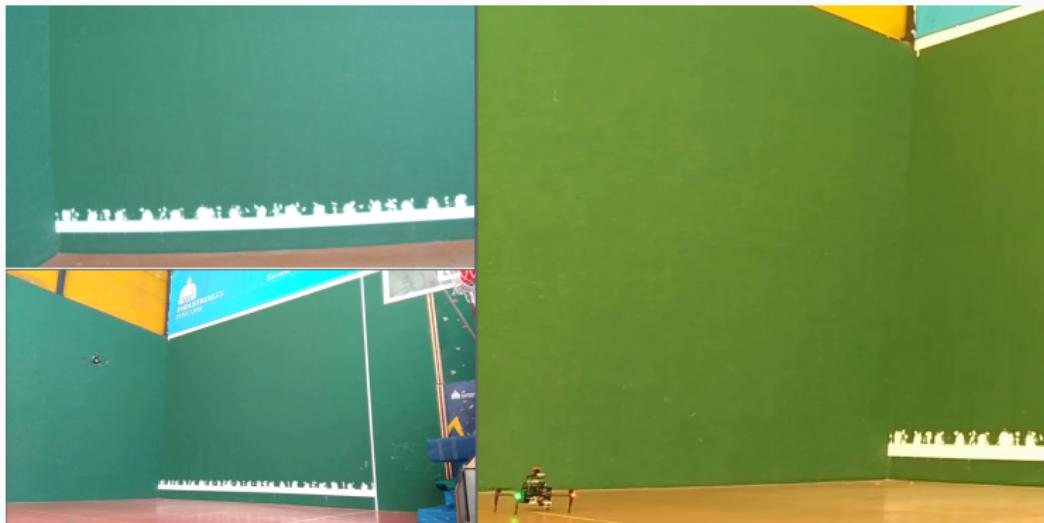


Figure 8: Real experiment on the Sports center ([video](#))

Experiments # Data

Simulation (10 tests, 6 points each)

#	Correct	Avg. Total Point	Avg. Time
<i>Go to Point</i>	41/60 (68%)	1.82 (\pm 0.52)	11.43 (\pm 3.12)
<i>Follow Path</i>	39/60 (65%)	1.97 (\pm 0.59)	12.21 (\pm 3.58)
<i>Generate Path</i>	60/60 (100%)	0.20 (\pm 0.00)	1.20 (\pm 0.01)

Real Flight (3 tests, 6 points each)

<i>Go to Point</i>	16/18 (88%)	0.94 (\pm 0.23)	5.97 (\pm 1.38)
--------------------	-------------	--------------------	--------------------

Experiments # Discussion

- Much room for improvement
- Localization is not accurate enough (lots of timeouts)
- Real flight is more stable & fast
- Simulation is more tested

Conclusions

Conclusions

With ...

- Improve Localization
- Lower timeout
- Extend testing

... an **autonomous navigation can be achieved.**

Questions?

References

References i