UNIVERSIDAD POLITÉCNICA DE MADRID

Thesis Title

by

Guillermo Echegoyen Blanco

A master thesis submitted in partial fulfillment for the
masters degree of Artificial Intelligence

in the

Escuela Técnica Superior de Ingenieros Informáticos
Departamento de Inteligencia Artificial

June 2018

Declaration of Authorship

I, AUTHOR NAME, declare that this thesis titled, 'THESIS TITLE' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Write a funny quote here."

If the quote is taken from someone, their name goes here

UNIVERSIDAD POLITÉCNICA DE MADRID

Abstract

Escuela Técnica Superior de Ingenieros Informáticos
Departamento de Inteligencia Artificial

Master Degree in Artificial Intelligence

by Guillermo Echegoyen Blanco

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

For/Dedicated to/To my...

Chapter 1

Introduction

[ToDo := Refs & Links]

In the following work an integration of a navigation system onto the Aerostack platform is presented.

1.1 Context

Aerostack is a framework for aerial robots, aimed at giving flight autonomy to some extent. It features a modular approach for the construction of behaviors that can be used to develop complex flights and automatic handling for certain situations such battery level or hardware conditions. It is the frame for the following work, which adds more autonomy through the integration of a octomap-based navigation system and a global planner. This provides a novel localization technique for the framework.

1.2 Motivation

So far, there exists only one simple geometry planner and an Aruco-based localization technique. In indoor environments, this system compels the need for environment preparation, the Arucos must be placed beforehand in well known localizations that must be hardcoded in the robot map. In this sense, there exists a need for a more robust, preparation-free localization system and accompanying planner. This work provides such an improvement with the introduction of a octomap-based navigation system and a global planner.

1.3 General Objective

This work aims at adding a novel navigation system to the Aerostack framework.

1.4 Specific Objectives

This section enumerates a comprehensive list of objectives.

1. Enrich the current navigation and localization systems.
2. Test and validate the new navigation and localization systems through simulated environments.

To achieve the first objective, the following additions to the framework are proposed:

- Add a robust planner based on a new navigation technique.
- Add a robust navigation technique through the use of octomaps.
- Add a robust localization technique based on octomaps.
- Add octomaps construction support through lidar.

1.5 Overview

[ToDo := Review when everything is finished]

This dissertation is organized as follows: ...

Chapter 2

Problem Description

In the present chapter the problem is presented, along with an introduction to the previous work. It is structured as follows: Section 2.1 introduces the Aerostack framework, Section 2.2 presents the context of the problem and the requirements a replacement should have. Section 2.3 describes deeply the improvements presented and the decisions taken to end in Section 2.4 with the description of the previous system.

2.1 The Aerostack Framework

Aerostack is an agnostic framework to build and design control architectures for aerial robotic systems. It provides some low level components as well as coordination processes and some planners. Figure 2.1 shows the general architecture of the framework.

2.2 Requirements

As of the second version of Aerostack, the only localization technique available is based on the recognition of a special type of marker called Aruco, first used for augmented reality applications. It is a fast and reliable technique to estimate the pose of the camera capturing the image. Altough this system works fine for many applications, it imposes the need of preparing the environment, placing this markers in a very precise way and annotating it's exact position before the experiments. While this might not be a problem in an augmented reality like scenario, when it comes to live localization in unknown environment it becomes useless. Hence, a new system for localization is required.

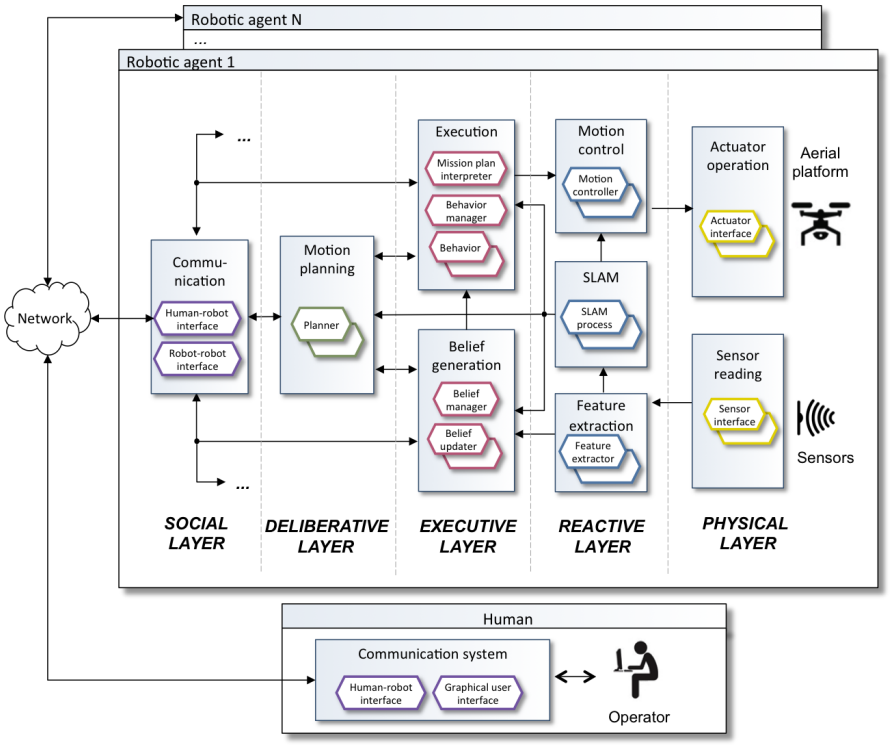

FIGURE 2.1: The Aerostack architecture

Along with the aforementioned localization technique comes the navigator which coordinates with a 2D geometric planner to accomplish the mission at hand. As the localization technique is to be changed, leading to a new way to perceive the environment, a new navigator and planner will be necessary.

2.3 Details of the New Features

A lidar is going to be used as the main source for localization. The module in charge of this part is already present in the Aerostack framework, but it is not being used. Based on lidar input an octomap is built. This octomap is then used by the navigator, along with the planner instructions to build a motion plan to execute.

The lidar + octomap functionality is provided by a module called hector_slam, developed at the Darmstadt University [3]. Taking the lidar's cloud of points it is able to reconstruct an octomap and then localize inside it (SLAM). The output of this module can be used to create a plan avoiding obstacles to reach a target point.

[ToDo := Review, NavStack & Global Planner]

2.4 Previous Work

By now, there is a process responsible for doing recognition of Aruco markers, it is connected to the robot camera and fetches data every n milliseconds, where n is a user defined constant. When an aruco is recognized, the pose of the robot can be estimated. Knowing the exact position of the marker enables the process to estimate the absolute position of the robot, leading to a high precision coordinates localization. While this approach has many advantages, it fails when the environment is not prepared beforehand.

To create a plan, a target position and the used Aruco markers are specified, then the motion planner creates a 2D plan to follow, using the Arucos to localize in the process.

Chapter 3

State of the Art

This chapter will review the classic methods used for localization and mapping, which is the core for any navigation technique, it starts describing the problems that arise both in localization and mapping. Continue with some of the most prolific solutions found to these problems to finish exposing the techniques and algorithms used by the Aerostack framework.

Localization is referred to all the techniques used to find the position in coordinates of an agent or object inside the world, relative to a reference frame. As far as it's absolute coordinates are known, anything can be used as reference frame, it is used as the coordinates system's centre. In an outdoors environment, the Earth could be the reference frame and the robot's coordinates can be acquired with Satellite systems, giving an absolute point inside the three dimensional space. It is desirable for these coordinates to be in a way that a computer can handle efficiently, tipically as two or three floating point numbers (although integer numbers are used sometimes too), depending on the number of dimensions used to represent the space. To save computational effort, the y axis could be unused in a wheeled robot.

Moving a robot avoiding possible obstacles through the space is tricky in itself, obstacles must be detected and handled correctly, moving objects can appear in the way, and so on. This alone does not provide any intelligence nor it helps planning, to aid in planning and moving smartly in the space, a map can be constructed while the localization is happening. The term mapping covers all the algorithms used to construct a map combining the data acquired from the many input sources a robot can have. Mapping opens the door for smart planning, along with many more advantages. A classic example is finding cycles in planned paths.

3.1 Localization

Localization techniques are divided into two groups: Outdoor and indoor techniques. The distinction comes from the fact that Satellite systems signals cannot go through walls. This fact has led to a whole new set of technologies and techniques that are able to localize in environments without an absolute reference of the world.

This section is organized as follows: First outdoor localization will be analyzed, with a brief review of it's core components, then various indoor localization techniques will be exposed, focusing on lidar based techniques.

3.1.1 Outdoor localization

By now, the most robust, reliable solution for outdoor localization is based on combining different sources of data. The mobile platform has drawn great attention over the past few years, pushing some of the largest companies in a shared effort to improve localization services while minimizing the impact over the battery's performance.

Satellite systems localization in mobile applications have certain drawbacks: The most obvious one is that acquiring and processing the signal wastes power, but also that in the case of civilian systems (such as GPS) the localization has precision of 10 meters for security reasons.

To aid both problems two new localization techniques were implemented:

1. GSM Localization. As every GSM antenna has a well known location stored in a database, one can localize trilaterating the near GSM antennas. Obviosly, this method is subject to GSM signal coverage.

2. WiFi Localization. This powerful method can serve both as an indoor and outdoor localization technique (see sect. 3.1.2). When a smarphone detects a WiFi hotspot, it sends it's BSSID along with the GPS coordinates if available to a centralized server. As this database grows the localization precision improves and every user can take advantage from it. This is scpecially useful in urban areas where stallite signal is poor or intermitent and improves as more and more users log information.

The best localization services can be provided with the conjunction of these three main techniques, which can be easily merged with an *Extended Kalman Filter*.

Although these techniques are widely used they are limited to mobile platforms, which are restricted both in sensorization and in processing capabilities. For robotic applications more sophisticated sensors are used, this includes depth cameras and lidars for the most part. Again, all the sensors output is merged to get the best estimates.

In robotics, it is usual that the localization we are interested in is relative instead of absolute. This is done to aid in locating near objects in the space relative to the robot but also to construct a map, the process of localization and mapping simultaneously is called *SLAM - Simultaneous Localization And Mapping* (see sect. 3.1.3).

3.1.2 Indoor localization

To localize in indoor environments many strategies can be followed, the general trend is to place different markers (active or passive) beforehand in well known locations. This markers are then recognized by the localizing device to know it's location. This *recognizable markers* can be anything, a Bluetooth beacon, a WiFi hotspot, etc.

Bluetooth beacons are specially crafted for this purpose as they can provide much more information. It has been extensively used in congresses and hotels to provide hosts with more information beyond localization, as services and timetables based on location.

In the case of WiFi hotspots localization is usually done by analyzing the signal strength and incoming angle. This method only works when the hotspots' location is known beforehand and is very prone to errors because the device must remain static in a certain angle.

From the computer vision perspective, visual markers can be placed too and processed by the device localizing and again, this requires preparation beforehand. One example of this setup are the well known Aruco

In many robotic applications like swarm robotics there is a necessity to track each member of the swarm in a closed, contained environment for this purpose an OptiTrack system can be used, it is a highly precise camera set that can track various markers (marked swarm members) and serve through the network. This setup is specially useful to monitor the swarm, enabling each member to access it's location.

3.1.3 SLAM

SLAM is the process of mapping and, at the same time localizing inside that map. This is specially useful in environments that are not prepared like the ones exposed previously,

enabling the robot to work on an unknown place without getting lost nor entering cyclic paths. SLAM techniques are crucial in any robot with some degree of autonomy, it makes the navigation possible.

There are three main variants here, the ones based on Extended Kalman Filters, the probabilistic ones and the ones based on Graph Optimization.

3.1.3.1 Extended Kalman Filters SLAM

Extended Kalman Filters (EKF) SLAM is the earliest technique developed for SLAM. EKF is a general technique to find the best estimate for the measuring variable based on the mean and covariance. In this case the inputs are the odometry used to estimate the robot position, features of the environment anchor the odometry measures and the robot motor system sensors (wheel decoders, etc) to estimate the change on the position. Then the objective is to find the best estimate for the current robot's position.

At the start of the process, the system's (0, 0) coordinate is established where the robot is, this is the most confident measure about the robot position. As the robot navigates the environment, succesive measures are taken and paired with the known landmarks, when a previously seen landmark is witnessed again, the position estimate is corrected with the covariance matrix, and the error correction is propagated along the previous estimates. In this way a long as the robot is navigating an sensing the landmarks, the position estimate improves.

Although this technique works well on small maps, it renders unusable for large maps, this happens because the covariance matrix used to correlate the position estimates grows cuadratically with the measures, making the memory footprint wildly large and the overall processing time very high. Some researchers have proposed an improvement over the EKF SLAM algorithm through submap decomposition [2, 4].

3.1.3.2 Particle Filters

Particle Filters are a probabilistic approach to position estimation, usually called Fast SLAM [5]. It uses various *particles* that represent the posterior probability of the true distribution of maps and possible paths. To do so it stands over a method called Rao-Blackwellization, that aids in dimensioning the number of particles needed to represent the map. Also, as conditional indepence is assumed between the observed landmarks, every landmark can be represented as N small Gaussians, which is linear, instead of exponential on the number of landmarks.

Although this method is easy to implement, is fast enough for real time applications with not very high demanding software and yields good results on small to medium maps, it suffers from the fact that lots of particles are needed to represent big maps, specially with multiple nested loops. Therefore, many improvements have been proposed, [1] for example uses ocucupancy grids instead of Gaussians.

3.1.3.3 Graph Optimization SLAM

The Graph Optimization SLAM techniques try to optimize a graphical model representing the landmark and robots locations. In this representation, each location is viewed a node in the graph, and the edge (called soft constraint) between two consecutive nodes is the captured odometry. The key intuition behind these methods is that at the end, the graph is sparse, because, each node will have just a few connections to other nodes. Also, at worst, the number of entries in the graph is linear in the time elapsed and in the number of nodes.

This is the most widely used approach because sparse linear optimization is in a very advanced stage, allowing for scalable, yet efficient implementations of the algorithm.

3.1.3.4 Lidar SLAM

Lidar is a well established laser range sensor that can be used for depth estimation. By doing fast sweeps in 360 degrees it can compose a depth map which can be used for SLAM.

Hector SLAM [3] is a technique developed in the Darmstadt University. It uses a lidar sensor to do a fast SLAM by matching rays along sweeps.

Along this work, the *hector_slam* ROS module will be integrated into the Aerostack framework, providing a robust SLAM technique ready to use for the drones equiped with lidar.

3.1.3.5 Visual SLAM

Using computer vision for SLAM have been a challange since it's conception, it raises the difficulty, specially for monocular cameras. Although many features can be extracted from images, it is not clear how to process nor store the data taking into account the full 6 degrees of freedom in a camera. All the parameters of the camera must be known

beforehand, depth cameras include a lot of noise and monocular cameras do not have scale.

In [6], ORB-SLAM is proposed. Intuitively, it creates ORB features from a visual input and stores it in a sparse matrix, then a matching process is launched to localize every feature, improving the localization along the way. It can work on Monocular (no scale), Stereo and Depth Cameras, giving extraordinary results.

Chapter 4

Implemented Modules

In this chapter, the technical goals of the project are introduced. Each developed module will be explained as well as it's requirements and it's specification details. For each module, it will be explored:

- The technical goals
- The problem to which the module provides a solution
- Some of it's properties (reusability, scalability ...)
- Integration with the current version of Aerostack

4.1 Technical Goals

In order for the Aerostack framework to localize with a different technique rather than visual markers a lidar sensor will be used. In this case, the *Hokuyo Eye* range sensor will be used, which is the *defacto* range sensor in this context. Also, the low level implentation provides a nice ros API that can be used to fetch data. To wrap all this functionallity we propose the implementation of a new high level behavior that coordinates all the framework with the lidar interface, providing a high level, standarized API for lidar-based localization.

As of the current version of Aerostack, navigation is done with a 2D probabilistic roadmap planner, the input for the planner is a predefined map, done by hand in the Graphical User Interface that Aerostack provides. This is a static map and goes against the nature of the any dynamically acquired mapping signal. To tackle this problem a new navigation behavior is proposed. This behavior will abstract the planner used

for each localization mode, providing a high level standarized API that can be used independently of the localization technique, replacing the old one.

4.2 Specification

Each implemented module should follow the specification imposed by the Aerostack framework. In Aerostack there are different types of processes providing structure and added functionallity. When a new process is created it should be decided whether to implement it as a plain, simple ros node, a robot process or a behavior process. Their differences are as follows:

- ROS Node: This is the standard way of adding modules in a ROS oriented architecture. A ros node is simply a process programmed in any of the programming languages supported by ros (C, C++, Python ...) that implements a task and is interfaced through the ros master server with named topics, services or both. A ros node can subscribe or publish topics and optionally, provide services, as many topics or services as it wants. These topics and services are nothing more than binded ports to the ROS master server, that works over TCP (normally) or UDP to distribute traffic. This is the implementation to follow when adding very low level modules, like platform drivers.

- Robot Process: A Robot Process is an abstraction provided by Aerostack, it serves mainly as a standarization layer, providing an interface for the rest of the architecture to be used. It provides three services to manage the process, one for stopping it, one for starting it and another one to check whether it is running or not, aditionally it emits an alive signal every second or an error signal when the thread crashes. It runs the inheritors' code inside a separate thread in order to monitor it. When adding a module that abstracts some low level APIs, like a visual marker processor, this is the class to inherit from.

- Behavior Process: This is the highest level of the hierarchy, inside the Aerostack framework there exists a process that coordinates all the behaviors, to do so, every behavior exposes an interface similar to the Robot Process and a configuration file that especifies the mid and low level processes the behavior depends upon (it's capabilities and incompabilities), amongst other parameters, in this sense, the behavior that provides localization based on visual markers depends on the visual marker processor to work. Formally, a behavior is just a high level process that monitors an algorithm: it runs the algorithm in a separate thread and emits the state and error signals, listening to *start/stop* events and acting accordingly over

the algorithm. When adding a high level functionallity, this is the class that should be inherited.

In a similar fashion to the visual marker localization behavior, the lidar localization behavior proposed will require three more processes: the slam process, an ekf that combines various signals and a localization technique selector, this is explained in detail in the corresponding behavior section (sect. 4.4).

The case of the navigation behavior is more complicated as it will need more modules to work correctly and will make use of other behaviors [**ToDo := Tell a little what modules does the navigator requires**]

4.3 Integration

To integrate each behavior, we will follow a bottom up procedure. This way, we will ensure that the processes the behavior depends on are working correctly inside the Aerostack and the error doesn't get masked with the behavior integration.

When integrating a new behavior some steps should be followed:

- Add the necessary mid and low level processes to the Aerostack and ensure they can be started automatically.
- Add the technical specification of the behavior to the behavior catalog. These are the capabilities and incompabilities of the behavior and should include the mid and low level processes previously mentioned so that they can be started automatically. In this step, the behaviors that are incompatible with the new one should be identified.
- Add the implementation of the behavior and test it with the Aerostack to ensure it can be started and that no incompatibilities arise.

The lidar-based localization behavior will provide a new localization mode, so it is reasonable to mark the rest localization behaviors as incompatible, also, a new localization method selector process will be added, this will ensure that when various localization techniques are to be used in the same mission, they can be easily toggled on and off. This will be detailed in the corresponding behavior section 4.4.

In the case of the navigation behavior, the rest of the navigation behaviors will be marked as incompatible, but as this behavior is intended to replace the old one, no selector will

be provided. Instead, this behavior will choose the data sources and plan with that data, providing an abstraction over the method used for localization and mapping.

4.4 Behavior Self Localize and Map by Lidar

The lidar range sensor outputs raytraces reflected over the near objects, in a way, it resembles a sonar sensor (that's way it's called lidar). Each raytrace, measures the distance from a concrete angle to a point at a certain distance, these measures then have to be converted in some way that can be used to map the environment and use this mapped environment to localize inside it (see section 3.1.3 for an in-depth explanation of SLAM). Refer to figure 4.1 for a visual representation of the behavior and it's subprocesses.

We will use a ros module implemented by the same laboratory as the SLAM node [3]: hector mapping. It will output the estimated localization along with the mapped environment. This localization will be merged with the measures from the rest of the sensors (namely odometry, IMU . . .) using an extended kalman filter (see section 3.1.3.1) to output a robust estimate of the robot's position inside the mapped world.

The process in charge of the EKF is called *droneRobotLocalization* and inherits from *Robot Process* class. It will listen for updates on the robot's pose (*hector_pose*) and the both the IMU and the odometry topics and output the estimated pose.

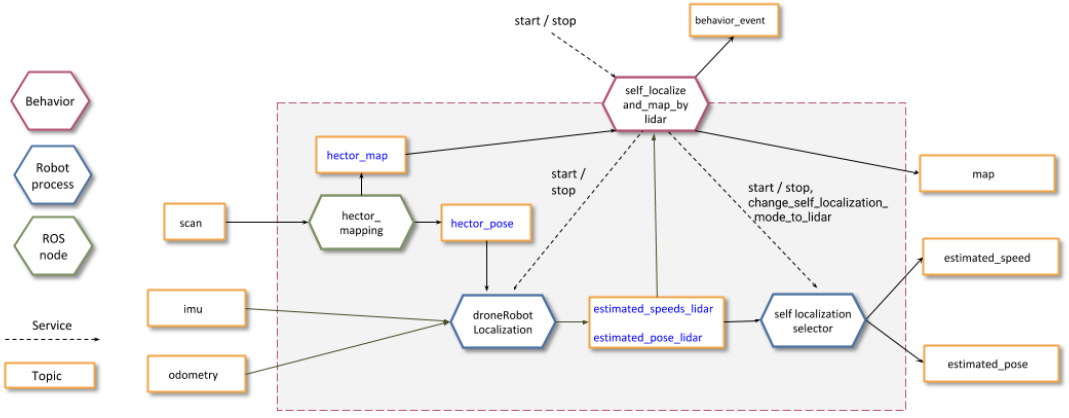

FIGURE 4.1: Behavior self localize and map by lidar architecture. *Hector mapping* is the SLAM module from [3]. *Drone Robot Localization* does the EKF. *Self Localization Selector* gives the localization based on the selected technique.

The estimated pose is then fed to the selector, which will toggle the localization technique. This implementation opens the door to new localization behaviors, a GPS based one for instance and also makes compatible the previous ones (visual markers based). It is easy to think of an scenario that requires both indoor and outdoor localization,

in such a mission, the localization technique in use should be toggled in order for the navigator to work properly. As can be seen in Fig. 4.1, this is a *Robot Process* with an added service to change the localization technique used.

Bellow there is a list of the inputs and outputs of this behavior:

[**ToDo := Set this up as a table??**]

- scan: This is the output of the lidar node, it is directly fed into the *hector mapping* node
- imu: This topic contains the measures from the *inertial measurement unit*
- odometry: This is a general topic with the measures from odometry
- map: This is the map as processed by the *hector mapping* node
- estimated_speed_lidar: This is the estimated speed from the whole behavior process, fed to the selector
- estimated_pose_lidar: This is the estimated pose from the whole behavior process, fed to the selector

This behavior monitors the correct working of the algorithm (*hector mapping*) by listening on the *map* topic, when it outputs strange or simply wrong data, an error is emitted.

In the configuration file of this behavior the localization by visual markers behavior will appear as incompatible. As for the capabilities, all of *hector mapping*, *drone robot localization* and *self localization selector* will figurate as capabilities, indicating that those processes should be started before this behavior.

4.5 Behavior Navigation Interface

This behavior should provide an API to do navigation tasks. It should be able to generate obstacle-free trajectories to any given point (when there exists one) and be able to move the robot along those trajectories.

The identified tasks for this behavior are as follows:

1. Given a point (or goal), execute the necessary motions to get the robot to that goal.
2. Given a path, execute the necessary motions to follow it until the path is finished.
3. Given a point (or goal), generate an obstacle-free path from the current robot's position to that goal.

This tasks can be directly mapped with processes. However, we will implement them as separate behaviors to provide more modularity and reusability.

As of the current version of Aerostack, there exists a behavior that executes the motion of going to a given point in the 2D map representation used by Aerostack. However, this behavior is not general enough to use a different map representation, so we will implement a new one capable of executing the motion in the new map format: occupancy grids (which is the format used in *hector mapping*).

The task of following a path can also be broken down to generate a series of points along that path and following each point. Following points blindly can be dangerous in environments whith moving obstacles, therefore a new behavior for this functionallity will be provided. This new behavior will receive a path, chop it in various points and start following them, testing before each motion if the path is obstacle-free. To do so, it will make use of the trajectory generator.

With all this information, the *behavior navigation interface* can be defined as the process in charge of coordinating all these new behaviors, it will receive a point and work in tandem with the other three behaviors to manage to go to that point without colliding with any obstacle.

The proposed names for each behavior are: *behavior go to point in occupancy grid*, *behavior follow path in occupancy grid*, *behavior generate path in occupancy grid*. Figure **ToDo := Generate figure** ilustrates the architecture followed for all these behaviors and the more general *behavior navigation interface*

In order to integrate this new behaviors with the current version of the Aerostack, the following topics will be created:

TODO := Table with all the generated topics and it's description

The following subsubsections explain each behavior in more detail.

4.5.1 Behavior Go to Point in Occupancy Grid

In order to move the robot from one point to another, this behavior will communicate the controller, providing it with the desired point. The controller will translate this point in the necessary motions and move the robot. Furthermore a *cancel motion* service will be provided to cancel the motion in course, giving grainer control to the navigation interface.

4.5.2 Behavior Follow Path in Occupancy Grid

This behavior will work in tandem with the go to point process, it will receive a path as a list of points and ask the *behavior go to point in occupancy grid* for each each point. Additionally, it will include the *cancel motion* service too.

4.5.3 Behavior Generate Path in Occupancy Grid

In this case we will employ a special planner provided as a ros package called *move base*, which is specially crafted for lidar interfaces. It accepts an occupancy grid map and the raytraces from the lidar and implements the planning algorithm. **TODO := Talk more over the move algorithm, explain the behavior monitoring.**

Appendix A

Appendix A

Bibliography

[1] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters". In: *IEEE Transactions on Robotics* 23.1 (Feb. 2007), pp. 34–46. ISSN: 1552-3098. DOI: 10.1109/TRO.2006.889486. URL: http://ieeexplore.ieee.org/document/4084563/.

[2] J.E. Guivant and E.M. Nebot. "Optimization of the simultaneous localization and map-building algorithm for real-time implementation". In: *IEEE Transactions on Robotics and Automation* 17.3 (June 2001), pp. 242–257. ISSN: 1042296X. DOI: 10.1109/70.938382. URL: http://ieeexplore.ieee.org/document/938382/.

[3] S Kohlbrecher et al. "A Flexible and Scalable SLAM System with Full 3D Motion Estimation". In: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. Nov. 2011.

[4] John J. Leonard, John J. Leonard, and Hans Jacob S. Feder. "A computationally efficient method for large-scale concurrent mapping and localization". In: (2000), pp. 169–176. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.122.4705.

[5] Michael Montemerlo et al. "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem". In: *IN PROCEEDINGS OF THE AAAI NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE* (2002), pp. 593–598. URL: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.2153.

[6] Montiel J M M Mur-Artal Raúl and Juan D Tardós. "ORB-SLAM: a Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163. DOI: 10.1109/TRO.2015.2463671.