Universitat de les Illes Balears

Turning a MAV into an Effective Tool for Vessel Inspection through a Reconfigurable Framework

Francisco Bonnin-Pascual, Alberto Ortiz,
Emilio Garcia-Fidalgo and Joan P. Company-Corcoles

Departament
de Ciències Matemàtiques
i Informàtica

TECHNICAL REPORT

January / 2018

A-01-2018

Turning a MAV into an Effective Tool for Vessel Inspection through a Reconfigurable Framework

Francisco Bonnin-Pascual[a,*], Alberto Ortiz[a], Emilio Garcia-Fidalgo[a], Joan P. Company-Corcoles[a]

[a]*Department of Mathematics and Computer Science, University of the Balearic Islands, Cra. Valldemossa km 7.5, 07122 Palma de Mallorca, Spain*

Abstract

Vessels constitute one of the most cost effective ways of transporting goods around the world. Despite the efforts, maritime accidents still occur, with catastrophic consequences. For this reason, vessels are submitted to periodical inspections for the early detection of cracks and corrosion. These inspections are nowadays carried out at a great cost. In order to contribute to make ship inspections safer and more cost-efficiently, this paper presents a novel framework to turn a Micro-Aerial Vehicle into a flying camera that can virtually teleport the human surveyor through the different structures of the vessel hull. The system architecture has been developed to be reconfigurable so that it can fit different sensor suites able to supply a proper state estimation, being at the same time compatible with the payload capacity of the aerial platform and the operational conditions. The control software has been designed following the Supervised Autonomy paradigm, so that it is in charge of safety related issues such as collision avoidance, while the surveyor, within the main control loop, is supposed to supply motion commands while he/she is concentrated on the inspection at hand. In this paper, we report on an extensive evaluation of the platform capabilities and usability, both under laboratory conditions and on board a real vessel, during a field inspection campaign.

Keywords:
Reconfigurable Framework, Micro-Aerial Vehicle, Vessel Inspection, Supervised Autonomy, Control Architecture, Sensor Fusion

1. Introduction

The importance of maritime transport for the international commerce is unquestionable. Different types of vessels are used depending on the kind of product that is to be carried: oil tankers, bulk carriers, container ships, etc. All of them can be affected by different kinds of defects that may appear due to several factors, such as structural overload, problems in the vessel design, the use of sub-standard materials/procedures, normal decaying of the metallic structures in the sea, etc. Regardless of its cause, cracks and corrosion are the two main defective situations that appear in vessel structures. Their presence and spread are indicators of the state of the vessel hull, so that an early detection can prevent major problems such as wreckages. For this reason, Classification Societies impose periodical inspection to assess the structural integrity of vessels.

Nowadays, to perform the inspection of a vessel, this must be situated in a dockyard (and sometimes in a dry-dock) where high scaffoldings are installed to allow the surveyors to reach all the plates and structures of the vessel. This procedure, together with the lost-opportunity costs due to the fact that the ship is not being operated, give rise to high expenses for the ship owner/operator. Furthermore, during this process, vessel surveyors may need to reach high-altitude areas or even enter into hazardous environments, putting at risk his/her own integrity.

In line with the aforementioned, the EU project INCASS[1] (finished in 2017) pursued to develop new

[*]Corresponding author: Tel.: +34-971-172-565;
Email address: xisco.bonnin@uib.es (Francisco Bonnin-Pascual)
[1]www.incass.eu

technological tools with the aim of contributing to the re-engineering process of vessel inspection. Among them, this paper focuses on an aerial robotic tool that has been developed for the visual inspection of the inner vessel hull. The idea behind this device is to allow the surveyor to perform a proper inspection from a safe and comfortable position.

Regarding the latter, the robotics literature contains a number of contributions for vessel hull inspection involving robotic platforms. The majority of the existing approaches make use of underwater vehicles to inspect the submerged part of the hull. Some of them are based on the use of Remotely Operated Vehicles (ROV) (see for example [16, 20]), while other approaches are based on the use of Autonomous Underwater Vehicles (AUV) which estimate their position with regard to the vessel hull using different devices and/or techniques. Apart from solutions based on free-floating AUVs (see for example [11, 22]), in this group we can also find some approaches of hull-crawling vehicles which are attached to the hull by means of suction (see [1, 19]). The robotics literature also reports on a reduced number of robotic platforms that operate magnetically attached to the vessel hull, what makes feasible the inspection above the water line (e.g. [3, 12]).

To the best of the authors' knowledge, the only contributions about flying robots specifically devised for vessel hull visual inspection result from our research. Our first attempt for vessel visual inspection using a MAV focused on providing a fully autonomous platform [4]. This robotic platform led to successful results in field tests performed in different types of vessels during the EU project MINOAS[2] [10]. Nevertheless, the usability of this platform was limited due to the way how inspections had to be performed. To carry out a mission, this had to be previously specified in a "mission description file" which consisted in a list of waypoints to attain and actions to perform. Despite this way of operation is suitable to sweep a vessel surface, e.g. a bulkhead, and take a picture, for example, every half a meter, it is not appropriate to make the vehicle attain a specific point in the vessel structure with unknown coordinates. Furthermore, during the field trials, some surveyors demanded the capability of flexibly manoeuvring the vehicle with some kind of remote control. Besides, since this autonomous system is based on a position control loop, issues in the position estimate, i.e. due to a malfunction of the laser scanner used for the perception of the surrounding structure, may put the platform in trouble or jeopardize the execution of the inspection mission.

This paper presents a novel approach for the visual inspection of vessels which intends to overcome the shortcomings of our previous platform. Results for preliminary designs of the new MAV can be found in [5], [7] and [21]. This paper focuses however on the next step of design and development, which consists in a reconfigurable framework intended to provide an existing MAV with the capabilities to become an effective and easy to use tool for the vessel inspection task. The resulting system is reconfigurable in the sense that it can incorporate different sensor suites depending on the payload capacity of the MAV and the operational conditions (such as the amount and kind of illumination or the presence of obstacles, e.g. consider the case of a cluttered environment). The present paper also provides an extensive evaluation of the performance and usability of the robotic device both under laboratory conditions and during a real inspection campaign on board a real vessel.

The paper is organized as follows: in Section 2, the system requirements are presented, including the requirements needed to accomplish the target tasks and also those necessary to improve the usability of the platform; Section 3 overviews the platform, introducing the key aspects of the approach and the operating paradigm; Section 4 reviews different sensor suites proposed to estimate the platform state estimation and perceive the environment; in Section 5, the control architecture design is detailed; Section 6 describes the pipeline that estimates the platform state from the sensor data; Section 7 provides the details for the implementation of the MAV; Section 8 reports on an extensive evaluation of the platform performance under laboratory conditions; Section 9 shows the performance and usability of the vehicle during an inspection campaign on board a real vessel; and, to finish, Section 10 draws some conclusions on the work described.

2. System Requirements

The system requirements have been defined taking into account the target task. The idea is to obtain an aerial robotic device to teleport the vessel inspector through the different structures of the vessel, so that

[2] `www.minoasproject.eu`

he/she can perceive an appropriate view of the hull state. The requirements to fulfil this task are:

1. the vehicle must allow a close-up view of the inspected surface,
2. the vehicle must obey the commands indicated by the user/surveyor,
3. the vehicle must allow reaching the highest structures of the vessel hull (notice that this requirement is not as obvious at it seems, since the robotic platform could be e.g. a magnetic crawler),
4. the vehicle must be able to operate inside the vessel hull, including rather narrow spaces, such as ballast tanks, and
5. the vehicle must be able to operate in dark areas, such as a ballast tank or a tanker cargo hold, where daylight can not penetrate.

Other requirements are defined to increase the usability of the platform and/or to reduce the mental workload of the user/surveyor who is performing the visual inspection:

6. the vehicle must implement self-preservation functions such as prevent collisions with the surrounding obstacles,
7. the vehicle must be operable by non-expert users who maybe have never used a robotic device, and
8. the vehicle should provide some autonomous behaviours to alleviate the inspection task to the surveyor, especially when performing repetitive operations or those prolonged in time.

3. System Overview

The aerial robotic tool has been designed to fulfil the system requirements presented in the previous section. To this end, the system architecture has been reconsidered from scratch. Regarding the vehicle configuration, we have chosen to use a multirotor device. This kind of vehicle, in its different setups (quadcopter, hexacopter, octocopter, etc.), has been widely used in the recent years for visual inspection tasks (see of example [15, 17, 25]). Multirotors present the advantage that they require simple rotor mechanics for flying control. Unlike single and double-rotor helicopters, multirotors use fixed-pitch blades and the vehicle motion is achieved by varying the relative speed of each motor to change the thrust and torque that they produce. Among them, we focus on those which weigh less than 2 Kg. The reduced size of these MAVs, together with their capabilities for hovering and Vertical Take-Off and Landing (VTOL), make them suitable for operating in confined spaces and close to structures, which is a crucial feature for being able to achieve close-up visual inspection.

With this aim, the vehicle is equipped with cameras to take high resolution pictures and videos from the vessel hull surface. The inspection in dark spaces, such as ballast tanks or closed cargo holds, is possible thanks to the use of high power LEDs that illuminate the inspected surface. All the pictures are tagged with the estimated pose of the vehicle to perform an effective inspection of the vessel and to allow revisiting the area if necessary. Pose estimation issues are explained in the following sections.

Operating a MAV in close proximity to a structure can be a challenging task due to complex environmental conditions and potentially poor situation awareness of the remote pilot. To reduce the mental workload of the pilot in these situations it is beneficial to give the vehicle its own artificial situation awareness. Following this idea, the system architecture has been designed around the Supervised Autonomy (SA) paradigm [9]. This defines a framework for human-robot interaction which aims at the alleviation of stress on human users through an appropriate level of instructions and feedback. In other words, the human user is not burdened with the complete control of the system, and hence he/she can concentrate on the task at hand. The SA framework comprises five concepts:

- *Self-preservation*, which refers to preserving the platform from anything that can jeopardize its integrity, such as a collision. The idea is that the required control issues are addressed by the robot itself.

- *Instructive feedback*, to provide the user with the same environment perception capabilities as the robot. For example, the system can provide the user with images of what the robot sees ahead, or the distance to the nearest obstacles at both sides of the robot.

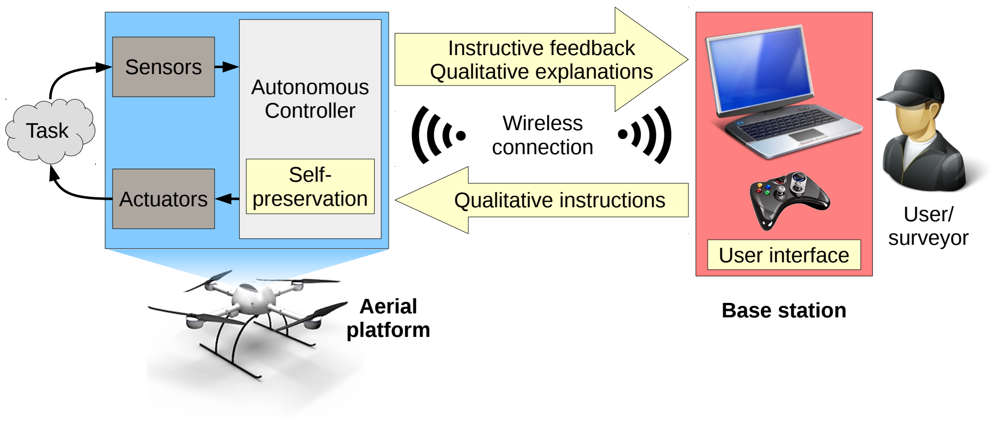


Figure 1: Overview of the inspection system designed around the Supervised Autonomy paradigm.

- *Qualitative instructions*, which are used to command the robot in an easily understood manner. For example, instructions such as "go ahead until you find an obstacle".
- *Qualitative explanations*, to describe to the user what is happening during the course of a mission using a language similar to the one employed for the qualitative instructions. For example, the robot can indicate that is "going forward" or inform about "obstacle detected".
- *User interface*, which is used to display the instructive feedback and allows the user to issue qualitative instructions.

To implement the SA framework, our solution has been designed around two separate agents. On the one hand, the *aerial platform*, which is fitted with several sensors and actuators, is in charge of all the control-related issues to successfully carry out the specified task. The autonomous controller is also in charge of the self-preservation of the platform. On the other hand, the *base station* is used by the user/surveyor to indicate the qualitative instructions to the aerial platform by means of some input device. At the same time, the base station is used to provide the user/surveyor with information about the mission's state and the MAV's situation, using instructive feedback and qualitative explanations. The communication between both agents is performed via a wireless connection. An overview of the system can be found in Fig. 1.

The vehicle is fitted with a suitable set of sensors to allow the platform to properly estimate its state and perceive its environment under the specific operational conditions that arise inside vessels. In particular, the vehicle can not use GNSS positioning systems due to the lack of line of sight with satellites. Furthermore, the sensor suite must include sensors to allow the vehicle operation in dark spaces, where daylight can not penetrate. Section 4 discusses about the different sensors that have been considered to be installed on-board the MAV.

The autonomous controller comprises a set of behaviours which are in charge of accomplishing the specified task while ensuring the platform self-preservation. For example, a behaviour is in charge of moving the platform as indicated by the user, another prevents collisions with the surrounding obstacles, another keeps a constant distance with the inspected surface, etc. The different behaviours developed are detailed in Section 5.4.

This design introduces the user/surveyor in the position control loop, allowing him/her to take the platform to the desired point while being assisted at all times by the control software. Furthermore, waypoint navigation is not used in this design and, hence, position estimation is not required for the control system. Instead, our approach is based on a velocity controller, what requires proper speed estimations.

4. Sensor Suite

The design of our inspection tool requires ensuring accurate estimation of speed (for the control software), as well as a position estimate (not so critical) to tag the pictures taken during a flight. A detailed description of the platform state, including the estimated velocity and position, is provided in Section 5.2.

As mentioned before, the need of flying inside closed spaces make impossible the use of GNSS systems such as GPS. Furthermore, the large dimensions of the holds and tanks inside vessels, together with the presence of traces of goods or rust particles, make unfeasible the use of motion tracking systems. Because of that, the vehicle state estimation must rely on on-board sensors. Among them, we focus on lightweight devices that can be carried by small UAVs as payload.

All MAVs are normally equipped with an IMU. This device usually comprises three accelerometers, three gyroscopes and a magnetometer. Using these sensors, the IMU can estimate the accelerations of the vehicle in the three axes (longitudinal, lateral and vertical), the three angular velocities around these axes, and the attitude of the platform. Despite they are widely used, IMUs can not be employed alone to estimate the platform velocity or position. Linear velocities are sometimes computed by integrating the accelerations measured with the IMU, but this just works for a short period of time (maybe a few seconds). Then, the inexactitudes in the acceleration measure, together with the effect introduced by the finite sampling frequency of the sensor, make the velocity estimation degenerate. For this reason, to obtain a proper velocity or position estimation, IMU data have to be combined with information provided by other sensors.

We propose three different sensor suites to be installed on the aerial platform, which are enumerated in the following sections.

Sensor suite 1 It is devised for small UAVs with a very limited payload. It is based on the use of velocity estimates regarding the floor and/or the front wall (the wall under inspection). These estimates are obtained using optical flow sensors which provide the velocity combining a camera and an ultrasound (US) range sensor. The camera is used to compute the optical flow while the range sensor is used to introduce the scale to the flow measures, to obtain speed values, as well as the distance to the wall. Two additional US range sensors are used to detect the obstacles at both sides of the platform. The height estimation is performed using an optical range sensor, instead of a US sensor because of the typically longer range offered by the former (in particular, longer than the downward-looking optical flow sensor). Finally, the pose of the platform is estimated using a forward-looking camera which provides images to feed a monocular SLAM algorithm. To summarize, the first sensor suite comprises:

- an IMU for attitude estimation,
- an optical flow sensor, comprising a camera and a US range sensor, looking forward,
- an optical flow sensor, comprising a camera and a US range sensor, looking downward,
- an optical range sensor looking downward,
- a US range sensor looking to the left,
- a US range sensor looking to the right, and
- a colour camera looking forward.

Since this sensor suite is based on the use of several cameras, it requires a sufficiently illuminated scene together with the presence of distinguishable points (known as features), to allow for a proper estimation of the vehicle motion and position. This lightweight sensor suite is therefore not adequate for dark environments.

Sensor suite 2 It is suitable for platforms with a larger payload. It is based on the use of a laser scanner for velocity estimation, obstacle detection and position estimation via SLAM. Compared to the first sensor suite, the optical flow and range sensors are removed, as well as the forward-looking camera for displacement estimation. The second sensor suite thus comprises:

- an IMU for attitude estimation,
- a laser scanner, and
- an optical range sensor looking downward.

The use of a laser scanner makes feasible the operation in dark or poorly textured environments, but requires the presence of, from time to time, changes in the structure, for a proper estimation of the MAV motion. For example, this sensor is affected by the so called "canyoning" effect, i.e. the miss-estimation of the displacement along a corridor or canyon due to multiple feasible matchings between consecutive laser scans.

Sensor suite 3 It is intended to provide a more robust system, suitable for flying in a larger variety of environments. It results from combining the first and second sensor suites so that both the optical flow sensors and the laser scanner are used to estimate velocity and position. The laser scanner is used as the main sensor, while the information provided by the optical flow sensors allows for a suitable estimation in non-structured environments or corridors, preventing miss-estimations such as the ones produced by the "canyoning" effect. To summarize, this last sensor suite comprises:

- an IMU for attitude estimation,
- an optical flow sensor, comprising a camera and a US range sensor, looking forward,
- an optical flow sensor, comprising a camera and a US range sensor, looking downward,
- a laser scanner, and
- an optical range sensor looking downward.

The different sensor suites will be referred to as SS1, SS2 and SS3 from now on. The way how the data provided by all the sensors is processed and combined to estimate the platform state is detailed in Section 6. Notice that SS2 and SS3 require an additional camera to perform the visual inspection of the vessel. This has not been included in the previous lists since it is not necessary for the estimation the platform state.

5. Control Architecture

The control architecture has been designed as a layered structure, so that each layer corresponds to a different control level. This architecture is shown in Fig. 2. The lowest layer of the architecture comprises the attitude and thrust controllers which supply the motors commands, while the mid-level layer consist of the height and velocity controllers. These two layers are detailed in Section 5.3. The high-level layer is in charge of executing the *MAV behaviours* module, comprising several robot behaviours. These behaviours collaborate to provide the middle layer with proper velocity commands. Notice that the control software has been designed following the SA paradigm, so that this last layer is in charge of the platform *self-preservation* and the fulfilment of the *qualitative instructions* given by the user/surveyor, as explained in Section 3. A description of all the behaviours and the way how they interact with each other can be found in Section 5.4.

Apart from the control layers, the *State estimation* module is in charge of processing and combining all the sensor data to estimate the platform state. The state estimate is used by the different control layers as seen in Fig. 2. The state estimation module is organized as a pipeline, as detailed in Section 6.

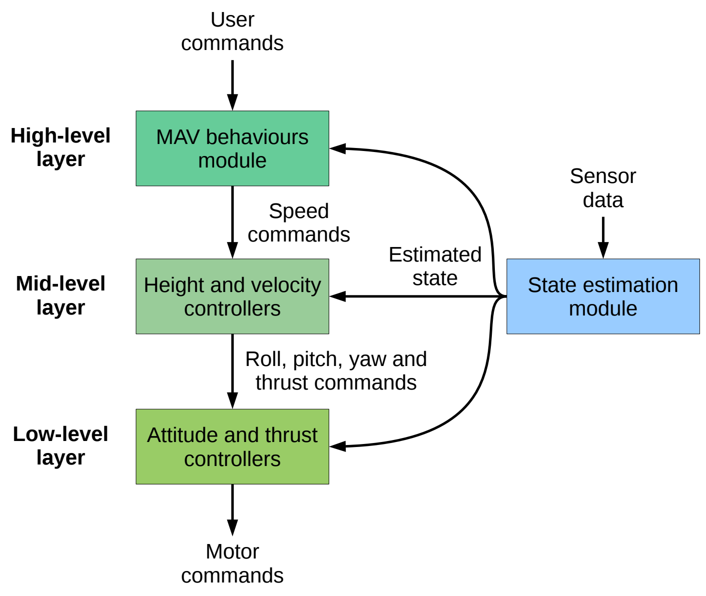


Figure 2: Control architecture.

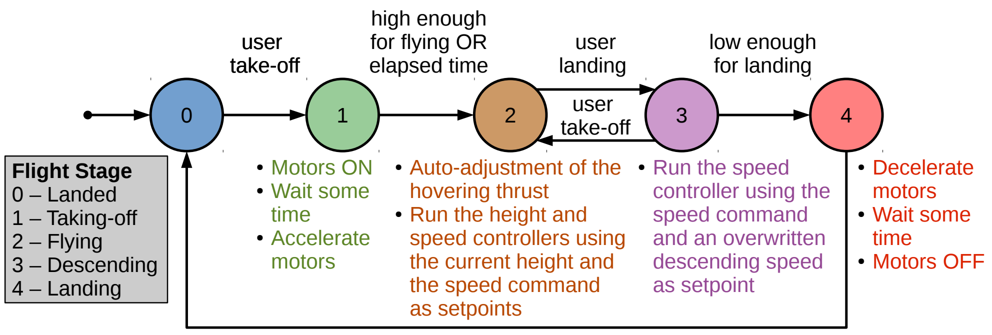

user take-off
high enough for flying OR elapsed time
user landing
user take-off
low enough for landing
0
1
2
3
4
Flight Stage
0 – Landed
1 – Taking-off
2 – Flying
3 – Descending
4 – Landing
• Motors ON
• Wait some time
• Accelerate motors
• Auto-adjustment of the hovering thrust
• Run the height and speed controllers using the current height and the speed command as setpoints
• Run the speed controller using the speed command and an overwritten descending speed as setpoint
• Decelerate motors
• Wait some time
• Motors OFF

Figure 3: Flight control state machine.

5.1. Flight Stages

Flight control is implemented as a finite state machine that comprises five states: *landed*, *taking-off*, *flying*, *descending* and *landing*. The transitions between states take place when particular conditions are met. For example, the system changes from *landed* to *taking-off* when the user starts the take-off manoeuvre from the user interface. Then, the motors start running and perform an acceleration ramp to elevate the vehicle from the floor. Some other transitions do not depend on the user commands but on sensor data and on the vehicle state. For example, the system changes from *taking-off* to *flying* when the platform height is above a certain value or after some time at a high level of motor thrust. The complete state machine is shown in Fig. 3.

When the system is in the *flying* stage, three controllers are in charge of tracking the speed command in the longitudinal, lateral and vertical axes. When the speed command in the vertical axis is zero, the height controller is enabled to provide the suitable command to the vertical speed controller in order to keep the current height. Furthermore, when the system enters the *flying* stage for the first time, an auto-adjustment of the hovering thrust is performed to stablish the suitable value for the specific air conditions. Details about the hovering thrust and the speed/height controllers are provided in Section 5.3.

The landing procedure is split into two stages. When the user starts the landing manoeuvre, the system

Table 1: MAV state. The first column indicates the modules or control systems which require each state variable.

	x	y	z	φ	θ	ψ	$\dot{x}$	$\dot{y}$	$\dot{z}$	$\dot{\varphi}$	$\dot{\theta}$	$\dot{\psi}$	$\ddot{x}$	$\ddot{y}$	$\ddot{z}$	d_b	d_f	d_l	d_r
Attitude contr.				×	×	×				×	×	×							
Velocity contr.							×	×	×				×	×	×				
Height contr.			×						×										
MAV behaviours																×	×	×	×
Image tagging	×	×	×			×													

Position: x, y and z.
Orientation: φ, θ and ψ.
Linear velocities: $\dot{x}$, $\dot{y}$ and $\dot{z}$.
Angular velocities: $\dot{\varphi}$, $\dot{\theta}$ and $\dot{\psi}$.
Linear accelerations: $\ddot{x}$, $\ddot{y}$ and $\ddot{z}$.
Distances to obstacles below, in front and at both sides: d_b, d_f, d_l and d_r

changes to the *descending* stage. Within this stage, the three speed controllers are still active and the vertical speed command is overwritten by a descending speed in order to reduce the flight height. The longitudinal and lateral commands are fed with the setpoint indicated by the MAV behaviours module, as in the *flying* stage, so that the platform still obeys the user commands, prevents collisions, etc. When the platform is close enough to the floor, it changes to the *landing* stage, which performs a deceleration ramp of the thrust and, finally, switches the motors off. The user can cancel a landing manoeuvre during the *descending* stage by starting a take-off manoeuvre. In that case, the system changes back to the *flying* stage.

5.2. Platform State

The pose of the aerial tool is determined by its position (x, y, z) and orientation (φ, θ, ψ) [3]. The latter is given using Euler angles, which are applied in the order yaw-pitch-roll (Z-Y-X). Linear velocities ($\dot{x}$, $\dot{y}$, $\dot{z}$) and accelerations ($\ddot{x}$, $\ddot{y}$, $\ddot{z}$) are defined with regard to the MAV body fixed coordinate frame. The same coordinate frame is used for the angular velocities ($\dot{\varphi}$, $\dot{\theta}$, $\dot{\psi}$).

After setting the coordinate frame conventions, the state of the aerial device can be defined. The height and velocity controllers, in the mid-level control layer, require the corresponding estimates of height (z) and linear velocities ($\dot{x}$, $\dot{y}$ and $\dot{z}$). Furthermore, the velocity controllers also require the linear accelerations ($\ddot{x}$, $\ddot{y}$ and $\ddot{z}$) to compute the roll, pitch and thrust commands, as explained in Section 5.3. Similarly, the orientation of the platform (φ, θ and ψ) and the angular velocities ($\dot{\varphi}$, $\dot{\theta}$ and $\dot{\psi}$) are required by the attitude controllers in the low-level control layer to compute the motor commands.

Regarding the high-level control layer, the MAV behaviours module requires the distance to the obstacles situated below (d_b), in front (d_f) and at both sides of the platform (d_l and d_r, for left and right respectively). The height estimation z is performed with regard to the take-off location, and may not coincide with the distance to the nearest obstacle situated below the platform d_b (see Section 6 for details).

Finally, the full position of the platform with regard to some agreed coordinate origin (typically the take-off location) is required to tag the pictures taken during an inspection mission. Thus, x and y estimates are also required. The full MAV state is defined in Table 1, indicating which module or control system requires each state variable.

5.3. Flight Control

This section focuses on the low- and mid-level control layers. The low-level layer is in charge of running the attitude and thrust controllers, i.e. this layer comprises the controllers in charge of keeping the desired

[3]ROS coordinate frame conventions are followed, see `www.ros.org/reps/rep-0103.html`

roll (φ_d), pitch (θ_d), yaw velocity ($\dot{\psi}_d$) and thrust ($\mathcal{T}_d$). Since these controllers are typically provided by the manufacturer as part of the platform firmware, they are not further discussed in this paper.

The mid-level control layer is in charge of tracking the desired linear velocity commands $\dot{x}_d$, $\dot{y}_d$ and $\dot{z}_d$ by providing the suitable attitude and thrust commands to the low-level controllers. Notice that, when a multirotor is tilted, i.e. rotated around the X and/or the Y axis, it suffers an acceleration towards that specific direction. The movement along the X axis is controlled through suitable pitch commands, while roll commands are used to control the movement along the Y axis. Regarding movements along the Z axis (changes in height), they can be controlled by means of suitable thrust commands.

Regarding the latter, three linear velocity controllers have been implemented in the form of the respective following three PID controllers:

$$\theta_d(t) = K_p^{\dot{x}}\,\mathcal{E}_{\dot{x}}(t) - K_d^{\dot{x}}\,\ddot{x} + K_i^{\dot{x}} \int_0^t \mathcal{E}_{\dot{x}}(\tau)\,d\tau, \tag{1}$$

$$\varphi_d(t) = K_p^{\dot{y}}\,\mathcal{E}_{\dot{y}}(t) - K_d^{\dot{y}}\,\ddot{y} + K_i^{\dot{y}} \int_0^t \mathcal{E}_{\dot{y}}(\tau)\,d\tau, \tag{2}$$

$$\mathcal{T}_d^*(t) = K_p^{\dot{z}}\,\mathcal{E}_{\dot{z}}(t) - K_d^{\dot{z}}\,\ddot{z} + K_i^{\dot{z}} \int_0^t \mathcal{E}_{\dot{z}}(\tau)\,d\tau, \tag{3}$$

where the outputs θ_d, φ_d and $\mathcal{T}_d^*$ are the desired pitch, roll and thrust, $\mathcal{E}_{\dot{x}}$, $\mathcal{E}_{\dot{y}}$ and $\mathcal{E}_{\dot{z}}$ are the errors in the three linear velocities, and K_p^{DOF}, K_d^{DOF} and K_i^{DOF} are the constants for the proportional, derivative and integral terms for the corresponding degree of freedom, namely $\dot{x}$, $\dot{y}$ and $\dot{z}$. Notice that the linear accelerations estimated by the IMU are involved in the derivative terms of the PIDs.

The desired thrust $\mathcal{T}_d^*$ is added to the thrust value necessary to compensate the weight of the platform, i.e. the thrust for hovering, $\mathcal{T}_h$. Thus, the final desired thrust value is obtained as

$$\mathcal{T}_d(t) = \mathcal{T}_h + \mathcal{T}_d^*(t). \tag{4}$$

The suitable value for the hovering thrust depends, obviously, on the vehicle weight, but also on the air density, which varies with the air temperature. For a proper configuration of this value, an auto-adjustment procedure is performed the first time that the MAV takes off. After changing to the *flying* stage, the MAV height is checked to be high enough to prevent perturbations due to the proximity to the ground. Then, the vehicle is left to free hover and the mean of the desired thrust is computed. During this process, the height controller will try to contribute to the initial $\mathcal{T}_h$ the suitable value to hover. After some seconds, $\mathcal{T}_h$ is overwritten with the computed mean, but limiting the update $\Delta\mathcal{T}_h$ to $\mathcal{T}_{h\_incr}$, to prevent oscillations. The process is repeated until $\mathcal{T}_h$ converges.

Regarding the height controller, it is activated when the desired vertical velocity $\dot{z}_d$ is zero. When this command becomes null, the platform height is saved as the desired height z_d, and used to compute the height error $\mathcal{E}_z$. PID control has also been used to implement this controller. The output of this PID is the desired vertical speed $\dot{z}_d^*$:

$$\dot{z}_d^*(t) = K_p^{z}\,\mathcal{E}_z(t) - K_d^{z}\,\dot{z} + K_i^{z} \int_0^t \mathcal{E}_z(\tau)\,d\tau. \tag{5}$$

Notice that, this time, the derivative term makes use of the estimated velocity in the vertical axis. Before being introduced in the vertical speed controller, the output of this PID is saturated by means of

$$\dot{z}_d(t) = \max(-\dot{z}_{d_M},\, \min(\dot{z}_{d_M},\, \dot{z}_d^*(t))), \tag{6}$$

where $\dot{z}_{d_M}$ is the maximum vertical velocity allowed. This is performed to limit the ascending/descending speed of the platform when it is trying to keep a certain height, as well as to reduce the effect produced by possible errors in the height estimation.

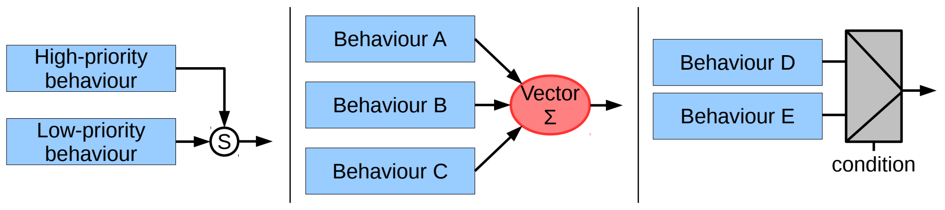


Figure 4: Behaviour combination mechanisms: (left) competitive mechanism using the subsumption architectural model for suppression, (middle) cooperative mechanism using motor schema for vector summation, and (right) selective mechanism by signals multiplexing.

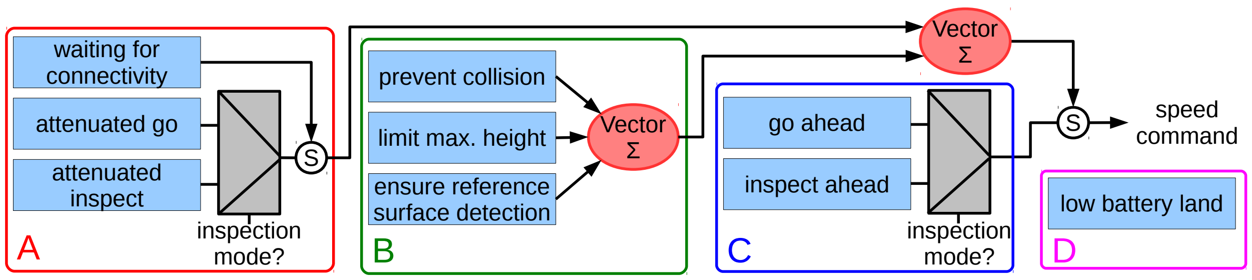


Figure 5: MAV behaviours: (A) groups behaviours to accomplish the user intention, (B) groups behaviours that ensure the platform safety within the environment, (C) groups behaviours that increase the autonomy level, and (D) groups behaviours oriented to check flight viability.

5.4. Behaviour-based Control

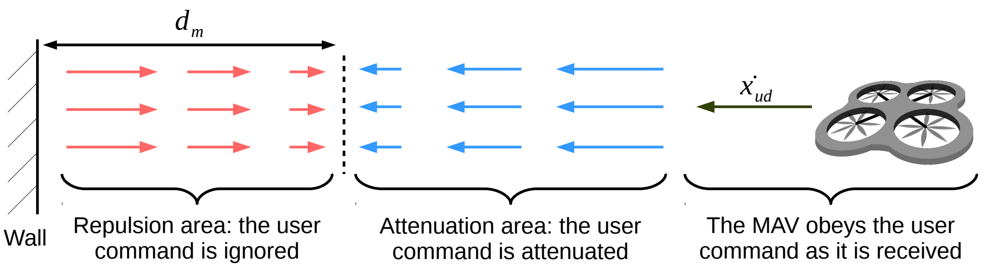
The high-level control layer executes the MAV behaviours module. Following the SA paradigm, this module comprises a set of robotic behaviours which are in charge of fulfilling the commanded task, indicated by the user/surveyor via *qualitative instructions*, while performing *self-preservation* tasks such as obstacle detection and collision avoidance. In other words, this module combines the user desired speed with the available sensor data through a reactive control strategy to provide the desired velocity command ($\dot{x}_d$, $\dot{y}_d$, $\dot{z}_d$).

The robot behaviours are organized in a hybrid competitive-cooperative framework. This framework makes use of the following combination mechanisms:

- a competitive mechanism to allow a higher priority behaviour to overwrite the output of a lower priority behaviour, which consists in using a suppression mechanism taken from the *subsumption* architectural model [2] (see Fig. 4 [left]);
- a cooperative mechanism to merge the output of several behaviours with the same priority level, which is performed through a *motor schema* [2], where all the behaviours involved supply each a motion vector, so that the final output is the weighted summation of all motion vectors (see Fig. 4 [middle]); and
- a selective mechanism to choose between the ouptput of two or more behaviours, i.e. a sort of multiplexer (see Fig. 4 [right]).

Figure 5 details our behaviour-based architecture, showing how the different behaviours are organized and how they contribute to the final speed command. The different behaviours are grouped depending on its purpose, setting up four general categories:

- *Behaviours to accomplish the user intention.* This group comprises the *attenuated_go*, the *attenuated_inspect* and the *waiting_for_connectivity* behaviours. The behaviour *attenuated_go* propagates the user desired speed vector command, attenuating it towards zero in the presence of close obstacles. In



d_m
$\dot{x}_{ud}$
Wall
Repulsion area: the user command is ignored
Attenuation area: the user command is attenuated
The MAV obeys the user command as it is received

Figure 6: Collision avoidance functionality for the MAV approaching a wall. This results from the joint actuation of the *attenuated_go/inspect* and the *prevent_collision* behaviours.

more detail, when the vehicle is moving towards an obstacle, the speed is reduced in accordance to the proximity to the obstacle. The speed is not attenuated when the user command moves the MAV away from the obstacle. By way of example, when the vehicle moves along the longitudinal axis obeying a user command $\dot{x}_{ud}$, the output of the *attenuated_go* behaviour $\dot{x}_{d\_ag}$ is computed as

$$\dot{x}_{d\_ag} = \min(\dot{x}_{ud},\, K_{ag}\, \dot{x}_{d_M} \cdot \max(0,\, d_f - d_m)), \tag{7}$$

where $K_{ag} \in [0, 1]$ is the attenuation factor, $\dot{x}_{d_M}$ is the maximum speed allowed along the X axis, d_f is the estimated distance to the nearest obstacle in front of the MAV and d_m is the minimum distance allowed to any obstacle. Notice that Eq. 7 limits the final speed command to the user desired speed, which in turn is also limited through the user interface.

The *attenuated_inspect* behaviour proceeds in the same way, being only activated in the so-called *inspection mode*. While in this mode, the vehicle moves at a constant and reduced speed (if it is not hovering) and user commands for longitudinal displacements or turning around the vertical axis are ignored. Furthermore, a PID controller, similar to that used for height control, is activated to provide the suitable velocity commands along the longitudinal axis ($\dot{x}_{d\_ai}$). In this way, during an inspection, the platform keeps at a constant distance and orientation with regard to the front wall, for improved image capture.

Finally, the *waiting_for_connectivity* behaviour sets zero speed (i.e. hovering) when the connection with the base station is lost. After some seconds, if the connection is not restored, this behaviour is also in charge of landing the platform.

- *Behaviours to ensure the platform safety within the environment.* This category includes the *prevent_collision* behaviour, which generates a repulsive vector to separate the platform from surrounding obstacles, whose magnitude increases as a function of proximity. By way of example, when an obstacle is detected in front of the platform, at a distance lower than the minimum allowed (d_m), the *prevent_collision* behaviour gives rise to the following output

$$\dot{x}_{d\_pc} = -K_{pc}\, \dot{x}_{d_M} \cdot \max(0,\, d_m - d_f), \tag{8}$$

where $K_{pc} \in [0, 1]$ is the repulsion factor. The joint actuation of this behaviour and the *attenuated_go/inspect* behaviours implements the collision avoidance functionality on-board the platform. Figure 6 illustrates this joint actuation for the case of the MAV approaching a wall.

A second behaviour called *limit_max_height* produces an attraction vector towards the ground when the vehicle is approaching its maximum flight height. This is computed as

$$\dot{z}_{d\_lmh} = -K_{lmh}\, \dot{z}_{d_M} \cdot \max(0,\, z - z_M), \tag{9}$$

where $K_{lmh} \in [0,1]$ is the attraction factor, and $\dot{z}_{d_M}$ and z_M are the maximum allowed values for the vertical speed and height.

A last behaviour called *ensure_reference_surface_detection* generates suitable attraction vectors that keep the platform close enough to at least one of the reference surfaces (the ground or the front wall), to ensure proper state estimations when using the optical flow sensors (see Section 6.1 for the details). Thus, if the vehicle requires the ground to estimate its estate (i.e. there is no wall in front of the MAV), an attraction vector towards this surface is applied when the distance d_b exceeds a maximum value d_{b_M}. The attraction vector is computed as

$$\dot{z}_{d\_ers} = -K_{ers}\,\dot{z}_{d_M} \cdot \max(0,\, d_b - d_{b_M}), \tag{10}$$

where $K_{ers} \in [0,1]$ is the attraction factor. Similarly, when the vehicle requires the front wall to estimate its estate (i.e. the ground is not detected by the bottom looking optical flow sensor), an attraction vector towards the inspected wall is applied. This is computed as

$$\dot{x}_{d\_ers} = K_{ers}\,\dot{x}_{d_M} \cdot \max(0,\, d_f - d_{f_M}), \tag{11}$$

where d_{f_M} is the maximum distance allowed regarding the inspected wall. Furthermore, in this situation, the commands for turning around the vertical axis are suppressed to keep detecting the wall in front of the platform. This is performed through a desired angular velocity that compensates the user rotation command $\dot{\psi}_{ud}$:

$$\dot{\psi}_{d\_ers} = -\dot{\psi}_{ud}. \tag{12}$$

- *Behaviours to increase the autonomy level.* This category comprises the behaviours that provide higher levels of autonomy to both simplify the vehicle operation and to introduce further assistance during inspections. The *go_ahead* behaviour is in charge of keeping the user speed command, i.e. the user does not need to reiterate the command all the time, until some obstacle is detected or a new desired speed is introduced by the user. This behaviour is of special interest when a large displacement has to be performed, for example, to go to the next wall to be inspected. An analogous behaviour called *inspect_ahead* is in use when the platform is flying in *inspection mode.* This is useful, for example, when surveying a large wall. Notice that the output of the behaviours in this category can be overwritten at any time by the behaviours in the previous mentioned categories.

- *Behaviours to check flight viability.* This group does not contribute to the speed command, but it is in charge of ensuring that the flight can start or progress at a certain moment in time. This group includes only one behaviour named *low_battery_land* that makes the vehicle descend and land when the battery is almost exhausted, i.e. its voltage is below a minimum value v_m.

As a final note, we would like to highlight that this set of behaviours has been designed having in mind the visual inspection application.

6. State Estimation

The estimation of the state is performed processing and combining the data provided by the different sensors. The state estimation module has been designed as a pipeline comprising several components which perform a specific task each. These components can be added or removed depending on the processes that need to be performed to get an estimate for one or more state variables from the data provided by a specific sensor belonging to the particular sensor suite employed on every occasion. Actually, there is a pipeline for every sensor suite. They will be described in the following sections, formalized through the following types of components:

- *Driver.* This component is used to communicate with a sensor device and to introduce the raw data into the pipeline.
- *Data filtering.* It provides different kinds of filters to block, restrict and/or smooth data. Some examples are the mean filter, the median filter and different versions of the Kalman Filter.
- *Data preparation.* This component eliminates/compensates undesired effects (such as biases or offsets) from the measured data. For example, a data preparation component can be used to eliminate the gravity acceleration from the linear acceleration measure captured with an IMU. Another component can be used to compensate the tilt (roll and pitch) of the MAV in order to obtain an accurate estimate of the distance to the ground from a bottom-looking range sensor data.
- *Data splitting.* This component allows splitting the information included in a data structure, so that several output structures are provided with a part of the information each. For example, a data splitting module can be used to divide an image into two sub-images, or to separate the different channels of a colour image.
- *Data processing.* This is the key component in charge of processing data to obtain useful information for the state estimation. This component can be used to implement processes such as odometers, SLAM methods, etc.
- *Data combination.* It is used to merge data from two or more inputs, usually fed by data processing components, to obtain a state estimation. This can entail some kind of process to select the suitable data among the input elements, according to some conditions.

As anticipated before, the different pipelines are detailed in the following subsections.

6.1. Sensor Suite 1

The state estimation pipeline designed for the SS1 is shown in Fig. 7. Seven *driver* components are used, one for each sensor: the IMU, two optical flow sensors, one optical range sensor, two US range sensors and one camera.

The *driver* component for the IMU is assumed to provide the orientation of the platform, the linear accelerations and the angular velocities. All these values are usually filtered on-board the IMU device, so that further filtering is not required. Nevertheless, the measured linear accelerations are affected by the gravity acceleration. To compensate this effect, a *data preparation* component is used, taking into account the platform roll and pitch, to compensate the corresponding value to each linear acceleration component.

Moreover, the linear acceleration measures are typically affected by some static bias. This is compensated in the same *data preparation* component, estimating a bias for each axis as the mean value of the first N measures after platform switch-on, and subtracting them from the gravity-compensated measures.

The *drivers* for the optical flow sensors provide both the velocity regarding the reference surface (the ground for the bottom-looking sensor and the front wall for the forward-looking sensor) and the distance to that surface (measured by means of the embedded US range sensor). A *data filtering* component is used to filter out the peaks (if any) in the measured distance, as well as to smooth the velocities. Firstly, a so-called peak filter is used to detect large changes in the measured distance. When this occurs, the last distance used is employed until detecting the end of the peak (i.e. the current distance becomes similar to the last used). If the peak has not finished after some time, it may indicate that this is due to a discontinuity in the reference surface (indeed it was not a peak), and the new measured distance is used. Notice that, while in normal operation, this filter does not introduce any delay into the distance measurement.

Secondly, a Kalman Filter (KF) is used to smooth the velocities estimated through optical flow measurement and also to estimate the normal speed with regard to the reference surface. Notice that, in SS1, the bottom-looking optical flow sensor provides the longitudinal and lateral velocities ($\dot{x}$ and $\dot{y}$), while the vertical velocity ($\dot{z}$) is estimated from the measured distance by the KF. For the case of the forward-looking optical flow sensor, it provides the lateral and vertical velocities ($\dot{y}$ and $\dot{z}$ in the body fixed coordinate frame),

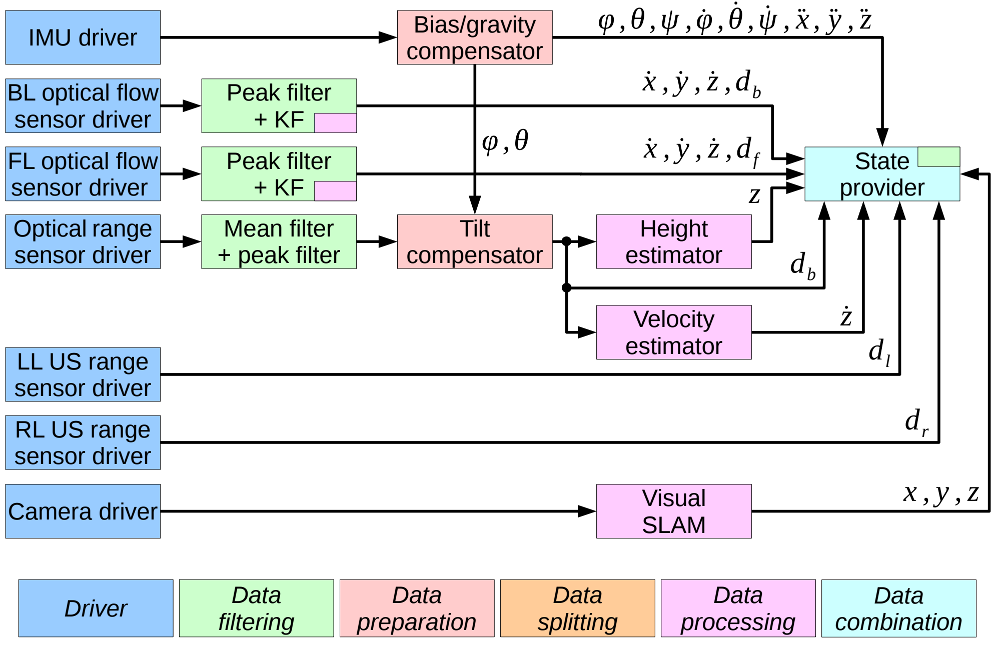

IMU driver
Bias/gravity compensator
$\varphi, \theta, \psi, \dot{\varphi}, \dot{\theta}, \dot{\psi}, \ddot{x}, \ddot{y}, \ddot{z}$
BL optical flow sensor driver
Peak filter + KF
$\dot{x}, \dot{y}, \dot{z}, d_b$
φ, θ
FL optical flow sensor driver
Peak filter + KF
$\dot{x}, \dot{y}, \dot{z}, d_f$
State provider
z
Optical range sensor driver
Mean filter + peak filter
Tilt compensator
Height estimator
d_b
Velocity estimator
$\dot{z}$
LL US range sensor driver
d_l
RL US range sensor driver
d_r
Camera driver
Visual SLAM
x, y, z
Driver
Data filtering
Data preparation
Data splitting
Data processing
Data combination

Figure 7: State estimation pipeline for the SS1. Sensor orientation: BL/bottom-looking, FL/forward-looking, LL/left-looking, RL/right-looking.

and the KF is used to estimate the longitudinal speed ($\dot{x}$). Due to the estimation of these state variables via software, i.e. not through a direct sensing device, the corresponding components in the pipeline can also be considered as *data processing*, as indicated in Fig. 7.

The optical range sensor is added to measure the distance to the floor with a detection range larger than the one provided by the US sensor embedded in the optical flow device. The data supplied by the corresponding driver is introduced in a third *data filtering* component. This makes use of an averaging filter, to remove the high-frequency noise, and a peak filter, as used for the distance provided by the optical flow sensors. The resulting distance is introduced in a *data preparation* component which compensates the MAV tilt, to obtain the distance to the floor. This is then introduced in two *data processing* components. On the one hand, the MAV height (z) is estimated in a so-called height estimator. This component keeps the values for the estimated heights of the platform and the floor, both initialized to zero. When a new distance measure d_b is received, this component computes the difference Δd regarding the previous distance received. If this value is below a certain threshold, it is considered a change in the flight height, and Δd is added to the estimated MAV height. If Δd is above the threshold, the distance change is considered to be probably due to a discontinuity in the floor, and the Δd is added to or subtracted from the estimated floor height, while the MAV height is preserved. Notice that both heights are always referenced to the take-off surface.

On the other hand, the tilt-compensated distance is also used to estimate the vertical speed $\dot{z}$. The corresponding component computes the instantaneous velocity by means of discrete differentiation. If the result is above a given threshold, probably due to a discontinuity in the floor surface or due to an error in the distance sensor, the velocity measure is considered incorrect and it is set to zero. The filtered velocity is finally introduced in a smoothing KF, as performed for the velocities provided by the optical flow sensors.

The drivers for the side-looking US range sensors provide the distance to the obstacles situated to the left (d_l) and to the right (d_r) of the platform. This values do not require any filtering nor preparation.

Finally, the camera driver supplies colour images from the environment situated in front of the MAV. These images are introduced into a *data processing* component which runs a SLAM algorithm. The output of this process is the position and orientation of the platform regarding the take-off location (further details are provided in Section 7.2).

Table 2: Selection of the source for the MAV velocities and height state variables, when using the SS1.

Mode	**Sensor availability**			**Sensor selection**			
	BL optF	**FL optF**	**optR**	z	$\dot{x}$	$\dot{y}$	$\dot{z}$
0	OK	N/A	OK	optR	BL optF	BL optF	optR*
1	OK	OK	OK	optR	BL optF	BL optF	FL optF
2	N/A	OK	OK	optR	FL optF*	FL optF	FL optF
3	N/A	OK	N/A	$\int \dot{z}$	FL optF*	FL optF	FL optF
-1	N/A	N/A	OK	optR	0	0	optR*
-2	N/A	N/A	N/A	$\int val'$	0	0	val''

BL optF refers to the bottom-looking optical flow sensor.
FL optF refers to the forward-looking optical flow sensor.
optR refers to the optical range sensor.
OK means that the sensor data is available.
N/A means that the sensor data is not available.
* indicates a derivation of a range measurement.
val' and val'' are positive values.

All the estimated state variables are introduced in a *data combination* component, the state provider. This component combines all the estimated state variables to build up the MAV state. The values for x and y are taken from the monocular SLAM algorithm and scaled using a λ factor since SS1 adopts a monocular approach. This factor is computed dividing the estimated height z, which is taken from the height estimator, by the scaled z provided by the SLAM method. The orientation (φ, θ, ψ), angular velocities ($\dot{\varphi}$, $\dot{\theta}$, $\dot{\psi}$), linear accelerations ($\ddot{x}$, $\ddot{y}$, $\ddot{z}$), and the distances d_f, d_l and d_r, are taken from their unique providers, as shown in Fig. 7. The distance d_b comes from the optical range sensor, while the one provided by the bottom looking optical flow sensor is just used to know whether this sensor is detecting the reference surface, as explained below.

Finally, the linear velocities ($\dot{x}$, $\dot{y}$, $\dot{z}$) result from combining the estimates resulting from the bottom-looking and forward-looking optical flow sensors, as well as by the optical range sensor (just for $\dot{z}$). The most suitable source is selected in every case depending on whether the reference surfaces are detected or not. The rules for this selection are detailed in Table 2.

Four different modes (modes 0 to 3) are defined depending on whether the front wall and/or the ground can be used as reference surface by the optical flow sensors (i.e. whether they are closer than the maximum detection range of the embedded US range sensor). Following these selection rules, the MAV velocities are preferably estimated based on optical flow measures, and only when these are not available, the system makes use of the values obtained differentiating distance measures. Notice that the flight height is not limited as long as there is a wall in front of the vehicle that can be used as reference surface by the forward looking sensor (mode 3 is used in that case). The detection of at least one reference surface (i.e the front wall or the ground) is guaranteed thanks to the *ensure_reference_surface_detection* behaviour (see Section 5.4). Nevertheless, in case this behaviour can not manage to achieve its goal, two additional error modes are defined. The first one (mode -1) is used when the optical range sensor is able to detect the ground, so that this is used to estimate both the height and the vertical velocity. The second error mode (mode -2) is used when no sensor can detect any reference surface. In that case, the height value is increased using a predefined ramp, while the vertical speed is set to a fixed positive value. This error mode makes the platform descend in such an emergency situation (the PID controllers for height and vertical speed will reduce the motor thrust trying to decrease the positive ascending speed).

The selected linear velocities are finally filtered in a KF, which is implemented in the same *data combination* component (see Fig. 7). This is used to combine the estimated linear velocities with the linear accelerations provided by the IMU.

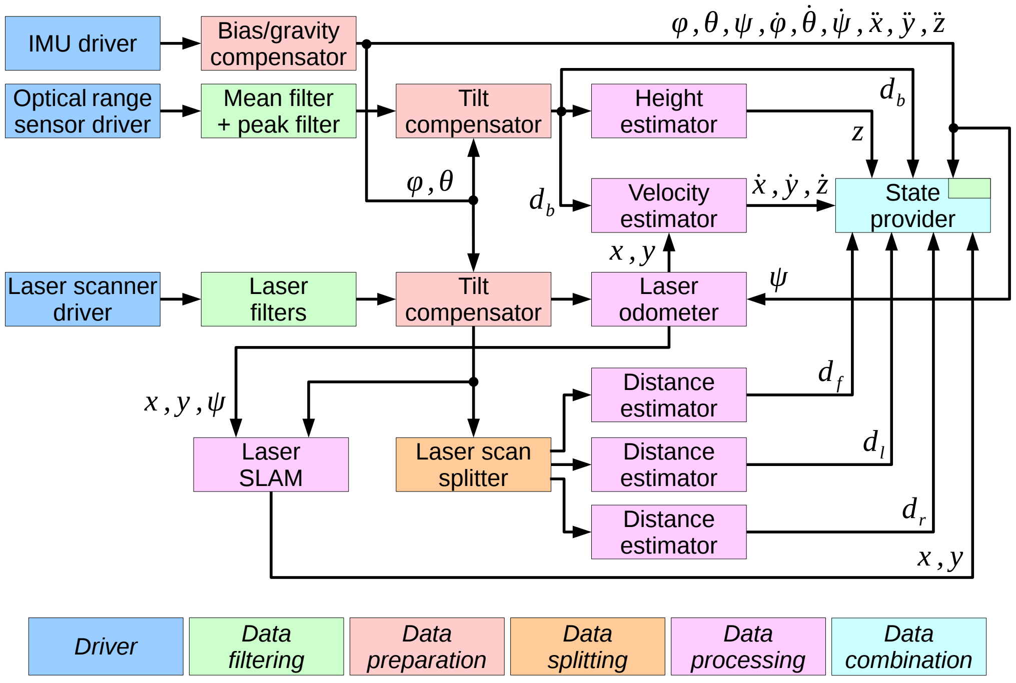


Figure 8: State estimation pipeline for the SS2.

6.2. Sensor Suite 2

Figure 8 shows the state estimation pipeline for the case of SS2. The IMU and the optical range sensor are used to estimate the same estate variables as for the SS1 pipeline, so that the same components are used.

The laser scanner driver provides the distances to the obstacles situated around the sensor. This array of distances, i.e. a laser scan, is introduced in a *data filtering* component which applies two filters. On the one hand, a filter is used to remove laser readings that are most likely caused by the veiling effect, which is produced when the edge of an object is being scanned. On the other hand, we apply a range filter to remove all measurements which are greater than an upper value or less than a lower value. This filter allows removing, for example, all laser beams which collide with some element of the MAV structure.

The filtered laser scan is introduced in a *data preparation* component which compensates the roll and pitch of the MAV, in order to obtain the orthogonal projection of the laser scan.

A laser-based odometer is used next to estimate first the 2D motion of the platform and, ultimately, its 2D location. This *data processing* component makes use of an Iterative Closest Point (ICP) algorithm to estimate the 2D transform T (comprising a translation and a rotation) necessary to match the current laser scan with the previous laser scan (or reference laser scan). In this algorithm, the rotation in yaw (ψ) provided by the IMU is used as initial guess for the transform. Furthermore, the reference laser scan is kept during several executions, and it is just updated when the platform performs a large displacement. This reduces the drift in the estimated pose produced by the noise in the scans, which can lead to non-zero transforms even when there is no displacement.

The resulting 2D location (x, y) is not used as the position estimate due to the inherent bias in dead-reckoning processes, but it is introduced in a 3-axis velocity estimator together with the estimated distance to the ground (d_b). This *data processing* component is analogous to the vertical speed estimator used in in the SS1, but defined for the three linear velocities. Like the one-dimensional version, this component includes a first step to filter out peaks in the computed speed, and a KF to smooth next the resulting signal.

A *data splitting* component is used to split the orthogonal laser scan into three segments, where each part comprises the beams providing information of the obstacles situated to respectively the left, in front

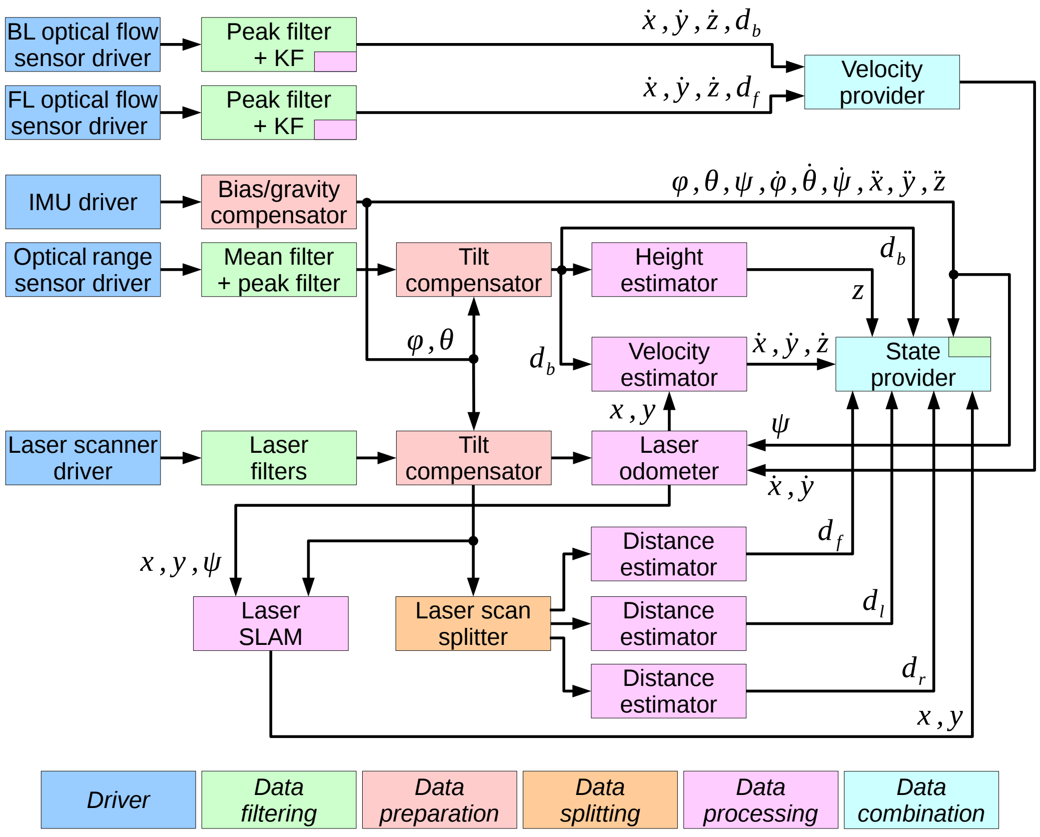


Figure 9: State estimation pipeline for the SS3.

and to the right of the platform. Each segment is next introduced in a *data processing* component which estimates the corresponding distances d_l, d_f or d_r as the minimum value among the readings.

As in the pipeline designed for the SS1, the 2D position of the platform $(x,\ y)$ is estimated in a *data processing* component which implements a SLAM process. This makes use of the tilt-compensated laser scan and the position estimated by the odometer to create a 2D map of the environment and to compute the drift-free position of the MAV. Further details are provided in Section 7.2.

Finally, a *data combination* component is used to collect and supply all the estimated state variables. Unlike the SS1, in this case, each state variable has a unique provider so it is not required to perform any kind of source selection. The same KF used for the SS1 is of application here to filter the linear velocities fused with the linear accelerations.

6.3. Sensor Suite 3

The pipeline for the SS3 is detailed in Fig. 9. This looks very similar to SS2, but including the information provided by the two optical flow sensors. This entails the use of the two *drivers* and the two corresponding *data filtering* components already used in the pipeline for SS1. The information provided by these two devices is merged into an additional *data combination* component. Within this, a selection of the suitable state variables describing the 2D velocity $(\dot{x},\ \dot{y})$ is performed following a subset of the rules defined for the SS1. These rules are detailed in Table 3.

The estimated 2D velocity is introduced, together with the estimated yaw, into the laser odometer. This is the key component within this pipeline, since it fuses the estimates provided by the optical flow sensors and the laser scanner. In this case, the odometer makes use of the estimated 2D linear velocities to get an initial estimate of the displacement of the MAV. This translation, together with the rotation indicated

Table 3: Selection of the optical flow data when using the SS3.

Mode	**Sensor availability**		**Sensor selection**	
	BL optF	**FL optF**	$\dot{x}$	$\dot{y}$
0	OK	N/A	BL optF	BL optF
1	OK	OK	BL optF	BL optF
2	N/A	OK	FL optF*	FL optF
-1	N/A	N/A	0	0

BL optF refers to the bottom-looking optical flow sensor.
FL optF refers to the forward-looking optical flow sensor.
OK means that the sensor data is available.
N/A means that the sensor data is not available.
* indicates a derivation of a range measurement.

by the IMU, is used to initialize the ICP algorithm. In this way, when the vehicle is flying in a poorly structured environment (such as a corridor or a single large wall without corners), where the ICP algorithm fails, the displacement can be successfully estimated thanks to the optical flow measurements. The rest of the pipeline is configured in the same way as for the SS2.

7. Implementation

This section provides details regarding the implementation of the aerial inspection tool. Section 7.1 tackles the physical realization of the aerial device, describing, by way of illustration, three different realizations, and showing the specific details for the integration of the different sensor suites considered, which result in a different configuration each. Section 7.2 provides details for the integration of the software corresponding to all the control and estate estimation systems/modules described previously.

7.1. Physical Realization of the Aerial Platform

For the physical implementation of the aerial device, we have used three different commercial multirotors produced by *Ascending Technologies* [4] (AscTec): the *Hummingbird*, the *Firefly* and the *Pelican*. These are electric-powered MAVs that fulfil the requirements regarding the vehicle configuration, capabilities (such as VTOL), size and weight, and hence they are suitable for flying in confined spaces or close to structures.

These platforms incorporate one IMU and two ARM7 processors. The primary ARM7, known as the Low-Level Processor (LLP), is in charge of executing the low-level control layer, comprising the attitude and thrust controllers. The LLP is also in charge of providing the inertial data from the IMU at 1 kHz. The secondary ARM7, the High-Level Processor (HLP), is left free so that the user can implement its own position/velocity controller. A serial connection is available to communicate both microcontrollers.

The Hummingbird is the smallest of the three platforms (see Fig. 10 [A]). This quadcopter has been used as test bench, so that all the algorithms to implement the different systems/modules within the control architecture have been firstly tested using this platform. Due to its limited payload (200 g), a laser scanner can not be carried by the Hummingbird, so that this platform can only fit the SS1. The optical flow sensors installed are the PX4Flow device developed within the PX4 Autopilot project [14]. Two MaxBotix US range sensors HRLV-EZ4 US are used estimate the distance to the obstacles situated to the left and to the right of the platform. As optical range sensor, we make use of an IR time-of-flight Teraranger One [24] which can detect an obstacle situated up to 14 m. The camera installed is a uEye UI-1221LE, while the on-board computer installed to execute the high-level control is a Commell LP-172 Pico-ITX board featuring an Intel Atom 2×1.86 GHz processor and 4 GB RAM. Due to the limited computation resources of this board, the visual-SLAM algorithm can not be executed on-board the Hummingbird.

[4] `www.asctec.de/en/`

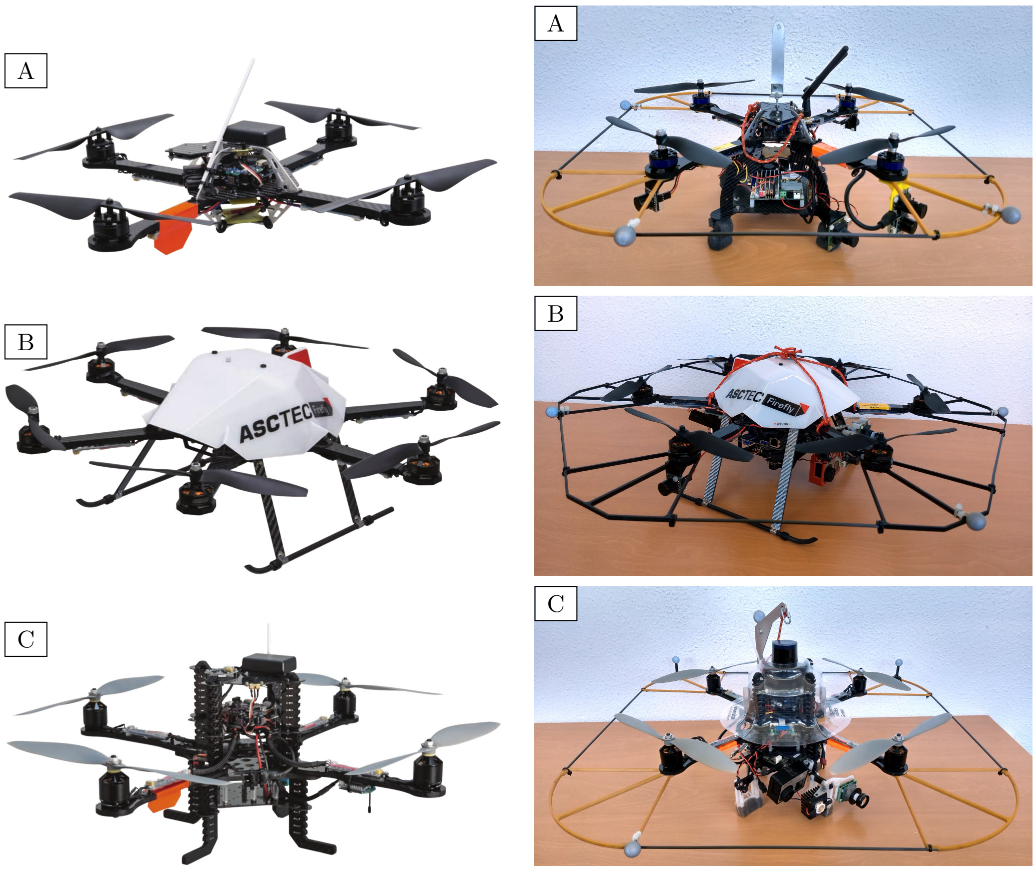


Figure 10: (Left) The aerial platforms as delivered by the manufacturer: (A) Hummingbird, (B) Firefly, and (C) Pelican. (Right) the platforms equipped with the sensor suites, on-board computers and cameras.

The SS1 has been also installed on-board the Firefly platform (see Fig. 10 [B]). This is a hexacopter with a higher payload capacity (600 g) that has been used to carry a more powerful on-board computer, which allows executing the visual-SLAM algorithm. This is an AscTec Mastermind board featuring an Intel Core 2 Duo SL9400 2×1.86 GHz processor and 4 GB RAM. Regarding the sensors, the same devices installed on the Hummingbird have been used for the Firefly. The optical range sensor has been changed by the Lidar-Lite laser range finder that provides range data up to 40 m. Additionally, this platform has been equipped with a GoPro Hero 4 camera to take first-person videos during the inspection mission.

Finally, the Pelican platform (see Fig. 10 [C]) is used to implement both the SS2 and the SS3. The larger payload capabilities of the Pelican (650 g) together with its layered structure, allows fitting the vehicle with a laser scanner and a powerful on-board computer. Regarding the laser scanner, a Hokuyo UST 20LX has been installed. This is a lightweight device (only 130 g) that can detect obstacles situated up to 20 m. The optical range sensor used is the Lidar-Lite also installed on-board the Firefly platform. Regarding the on-board computer, it is an Intel NUC board featuring an Intel Core i5-4250-U 2×1.3 GHz processor and 8 GB RAM. The vision system installed on-board the Pelican includes a PointGrey Chameleon3 USB 3.0 device and a GoPro Hero 4. Furthermore, this platform has been fitted with a high power LED to illuminate the inspected surface in dark environments. To implement the SS3, two PX4Flow sensors are added to the previous configuration. Figure 10 [C] shows the Pelican platform fitted with the SS2.

Regarding the base station, we have used a generic laptop featuring an Intel Core 2 Duo T6670 2×2.20 GHz processor and 4 GB of RAM. A joystick or gamepad is connected to this laptop to allow the user/surveyor to introduce the commands. The base station communicates with the on-board computers installed on the three MAVs via a WiFi connection. To this end, the involved machines can use both the 2.4 GHz and the 5 GHz bands.

7.2. Software Organization

To implement the inspection tool, we have developed software to be executed on the three different processing units/boards available in the robots: the secondary ARM7 processor (HLP, according to the manufacturer nomenclature), the on-board computer and the base station. The software for the HLP includes the implementation of the flying state machine, described in Section 5.1, and the mid-level control layer, comprising the height and velocity controllers. The HLP processor has been also programmed to execute the bias/gravity compensator component, included in the state estimation pipeline, to prepare the data supplied by the IMU (see Section 6.1 for details).

Both the on-board computer and the base station run Linux Ubuntu. The software developed for these machines has been programmed using the Robot Operating System (ROS) [5] [23]. Each component in the state estimation pipeline has been programmed as a ROS node (excluding the bias/gravity compensator). All these nodes, which are executed on the on-board computer, have been implemented following the specifications stated in Section 6. The laser odometer is an adaptation of [8].

Regarding the SLAM algorithms, we have integrated two existing solutions. On the one hand, the monocular version of the visual SLAM algorithm *ORB-SLAM* [18] has been integrated in the state estimation pipeline executed on-board the AscTec Firefly platform. On the other hand, the laser-based SLAM algorithm *GMapping* [13] has been integrated as part of the pipelines designed for the SS2 and the SS3, both using the laser scanner and executed on-board the AscTec Pelican.

The MAV behaviours module has been developed as another ROS node which comprises several functions to implement the different robot behaviours described in Section 5.4. This node receives the user commands and the state of the platform, and provides the final commands to be sent to the mid-level control layer, executed on the HLP. These commands are sent to an additional node which implements an interface between the HLP processor and the ROS software. In more detail, this node sends, through the serial communication with the HLP, the velocity, take-off and landing commands, while provides the other ROS nodes with the IMU data and information about the platform status: the flight stage, the linear acceleration biases, and the battery voltage.

A camera module has been implemented to manage the camera during an inspection. This consists in a ROS node which communicates with the camera driver. This module allows taking a single picture on demand, as well as taking a sequence of images at a specified frame rate. When taking a single image, this is sent to the base station for its visualization. Image sequences are stored on-board the MAV to reduce network traffic. The images in these sequences are tagged with the vehicle position, and represent the output of the inspection performed using our robotic device.

Table 4 shows measures for the CPU load and the memory usage, regarding the different processing boards/units installed on-board the MAVs. These measures are given both excluding and including the execution of the corresponding SLAM algorithm, which is the costliest process. As can be observed, when the SLAM algorithm is not considered, the SS2 pipeline executed on the Pelican platform requires more memory than the SS1, executed on the other platforms. This is due to the the different filtering and pre-processing stages required to prepare the data provided by the laser scanner. Nevertheless, when the SLAM methods are executed, the vision-based solutions included in the SS1 requires more memory to store all the data structures that this algorithm handles. The percentages provided in this table are illustrative, since the memory consumption will vary depending on the size of the map and the configuration of the algorithm parameters.

The base station (BS) essentially executes two functionalities: the management and sampling of the input device (e.g. joystick or gamepad), and the Graphical User Interface (GUI). Regarding the former, the BS provides the user commands to the different modules running on the aerial platform:

[5] `www.ros.org`

Table 4: CPU load and memory usage for the different processing boards. Metrics for the on-board computer provided excluding/including the execution of the corresponding SLAM method.

	HLP	**On-board computer**		
		Hummingbird	**Firefly**	**Pelican**
CPU load	62%	33%/–	7%/65%	6%/12%
Memory usage	–	15%/–	15%/66%	27%/40%

- the user desired velocities along the three axes, and the rotational velocity around the vertical axis, are fed into to the MAV behaviours module,
- the take-off/land commands are forwarded to the HLP interface node,
- the enable/disable *inspection mode* command is delivered to the MAV behaviours module,
- the command to keep the current speed (i.e. to activate the *go/inspect_ahead* behaviour) is supplied to the MAV behavious module, and
- the command to take a picture or to start/stop a sequence is issued to the camera module.

Regarding the GUI, following the SA paradigm, qualitative explanations are provided to indicate what is happening during the course of a mission. For example, the GUI indicates "going forward" when the *go_ahead* behaviour is enabled, or "low battery landing" when the corresponding behaviour is activated. The GUI is also used to provide instructive feedback including the distances to the obstacles situated around the platform, the flight height and the estimated velocities. When using the optical flow sensors (pipelines for SS1 and SS3), the GUI also indicates the mode used to combine the information provided by the two optical flow sensors (see Tables 2 and 3). Finally, the user interface is also used to show the images captured with the on-board camera when these are requested by the user. Different visualization tools included in ROS, such as *rqt_image_view* or *rqt_plot*, become useful when developing the GUI.

8. Experimental Evaluation

This section reports on the experimental assessment of the aerial robotic tool. Since different configurations using different sensor suites have been developed, this section firstly checks the flying capabilities of the different setups. In this regard, Section 8.1 presents several experiments performed to evaluate the state estimation and control performance in hovering and displacement manoeuvres. Secondly, Section 8.2 reports on the performance of the different robot behaviours, showing how each one contributes to the control and/or safety of the platform. In third place, Section 8.3 evaluates the usability of the platform is during an inspection mission. Finally, the localization methods performed while tagging the collected images is checked in Section 8.4. During the experiments, a motion capture system has been used to obtain the position, orientation and velocity of the platform. They all have been considered as the ground truth (GT) in all experiments.

8.1. Hovering and Displacement Capabilities

A first kind of experiments, assesses the hovering capability of the platform. This manoeuvre becomes a key component within the SA approach, as this is the reaction of the aerial platform while flying and waiting for new commands. In this regard, Fig. 11 shows some results obtained when using the SS1 fitted on-board the AscTec Firefly. This figure plots the histograms of the speed values during a 1-minute hovering manoeuvre, performed in each of the four estate estimation modes (see Table 2). Indeed, each plot compares the histogram of the speeds measured using the motion capture system (using a continuous line) with the values estimated using the optical flow sensors (using dashed lines). To facilitate the comparison, the histograms have been generated using the same quantization bins, and they are provided as a probability. As can be observed, all the histograms are approximately zero-centered, what indicates that the platform

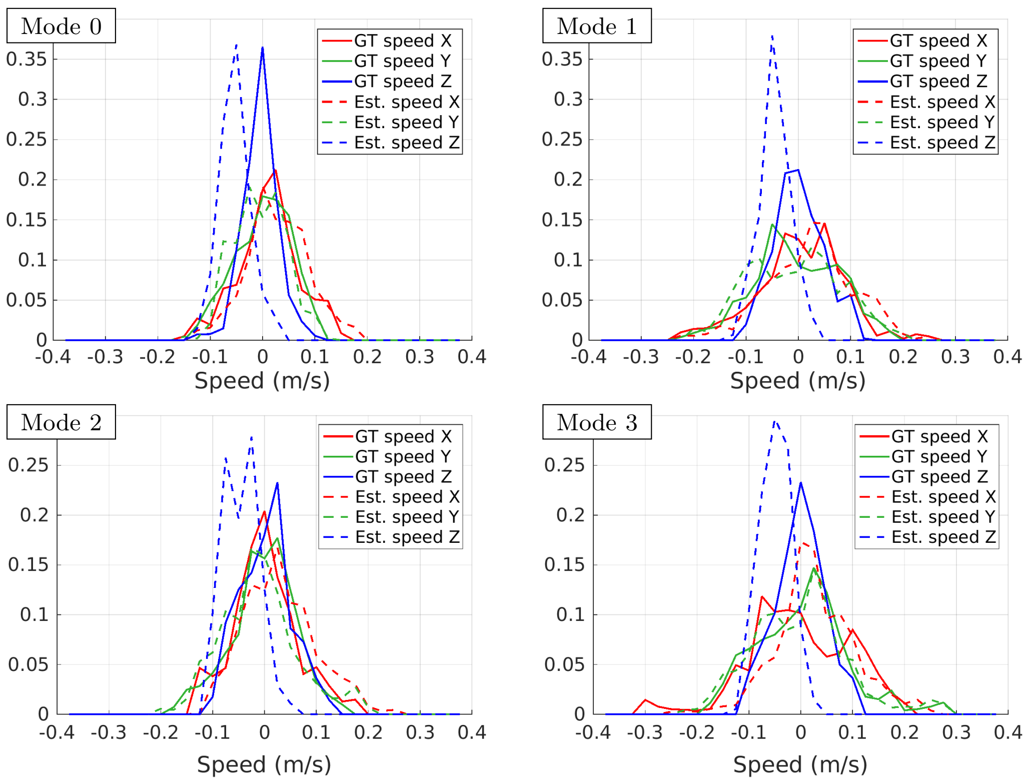


Figure 11: Normalized histograms of estimated speeds for 1-minute hovering flights performed using the SS1 and the different state estimation modes. Continuous lines are used for the data provided by the motion capture system (ground truth), while dashed lines are used for the values estimated by the aerial device. Experiments performed using the AscTec Firefly.

performs a suitable hovering using the different state estimation modes. These histograms also illustrate the quality of the on-board velocity estimations. Figure 12 shows the 3D position of the platform provided by the motion capture system during the four hovering flights. Notice that the deviations with regard to the first position which can be observed are normal since we do not apply position control but velocity control.

The hovering manoeuvre has been repeated using the laser scanner based system (i.e. the SS2) on-board the AscTec Pelican platform. The results are provided in Fig. 13. As already happened with the other platforms, the histograms resulting from the measured velocities are approximately zero-centered, and the displacement of the platform is pretty reduced.

In a second kind of experiments, the behaviour of the aerial devices has been evaluated while the user issues displacement commands. To be precise, in this case, the user/pilot is consigned to try to perform a square-like trajectory. We have proceeded in the same way as for the hovering experiments, so that four flights have been performed to evaluate the SS1 performance, using a different state estimation mode in each flight. For the first flight, the state estimation mode 0 has been used, so that the bottom-looking sensor has been utilized to estimate all the MAV velocities regarding the ground. The square-like trajectory has thus been performed in the XY plane. The rest of the flights, using the state estimation modes 1, 2 and 3, have been performed in front of a vertical wall, since this is required for the speed estimation. In these flights, the square has been performed in the YZ plane, i.e. parallel to the wall. Figure 14 shows the trajectories performed by the MAV as indicated by the motion tracking system. As can be observed, the user/pilot can easily perform the square-like trajectory parallel to the reference frame. Remember that the system lacks a position control loop, so that the human is in charge of the positioning of the MAV.

These experiments also allow checking the vehicle reaction to the user commands. In this regard, Fig. 15 shows the user commands sent to perform the square-like trajectories (blue), together with the estimated

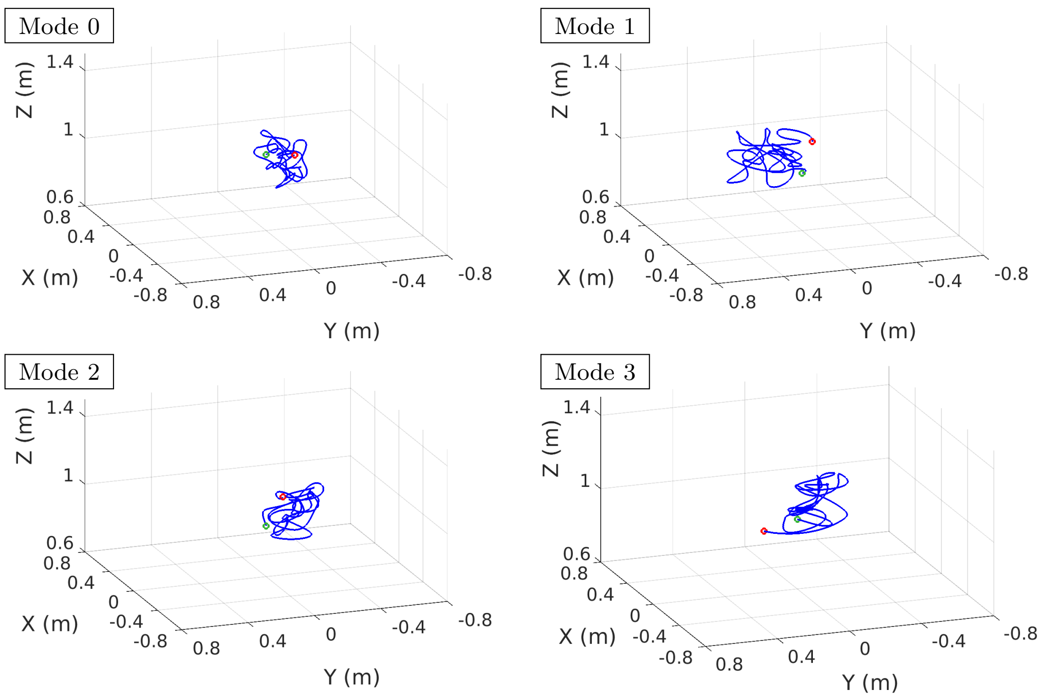


Figure 12: Plots of the position of the MAV indicated by the motion tracking system during 1-minute hovering flights performed using the SS1 and the different state estimation modes. Experiments performed using the AscTec Firefly. The green dot indicates the initial point, while the red dot indicates the final point.

velocities (red). Notice that, during these experiments, the vehicle has been operated far from obstacles, so that there are not attenuations nor repulsions, and the speed command provided by the MAV behaviours module coincides with the user desired speed. As can be observed in the plots, the estimated speeds follow the user desired speed, what indicates a successful operation of the velocity controllers. The plots also allow validating the suitability of the velocity estimation procedure, since the estimated velocities are compared with the velocities provided by the motion tracking system (green).

A similar experiment has been performed using the platform equipped with the SS2. In this case, the trajectory followed by the vehicle consists in two consecutive squares performed at different heights. The plots corresponding to this experiment are provided in Fig. 16.

A specific experiment has been carried out to assess the performance of the SS3 state estimation. This consists in flying the AscTec Pelican platform forwards parallel to a wall situated at its left. The vehicle has been displaced around 4 meters, and then it has been moved backwards approximately to the initial location. During this flight, all the data provided by the sensors comprising the SS3 have been saved. Then, several executions using the different state estimation pipelines have been carried out. Firstly, the SS1 and SS2 pipelines have been used to estimate the vehicle speed. Figure 17 [left] provides the results obtained in pink and black respectively. As can be observed, these approximately follow the ground truth value provided by the motion tracking system, indicated in green.

Then, the SS2 pipeline has been used once again, now limiting the laser scanner maximum range to 1 m (the sensor can detect obstacles at 20 m) in order to restrict the readings to the left wall, when estimating the vehicle velocities. In other words, the rest of the walls and structures in the laboratory are ignored by the aerial device. Under these conditions, the SS2, which relies solely on the laser scanner to estimate its longitudinal velocity, is not able to provide a correct speed estimation, as shown in Fig. 17 [left] in red. A last execution for the same sensor data has been performed using the SS3 pipeline. As can be observed in blue, the speed estimated by the laser odometer, when the optical flow data is used as initial guess, successfully approximates the ground truth speed.

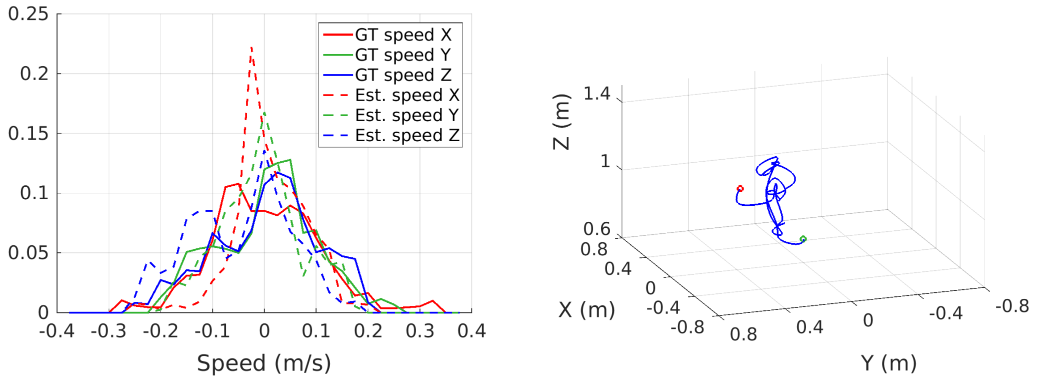


Figure 13: Results for a hovering flight using the SS2 on board the AscTec Pelican: (left) normalized histograms of estimated speeds (continuous lines are used for the data provided by the motion capture system, while dashed lines are used for the values estimated by the aerial device), (right) position of the platform (the green and red dots indicate the initial and final points respectively).

Figure 17 [right] provides an additional analysis of the results obtained with this experiment. In this figure, the estimated speeds have been integrated to obtain an estimate of the vehicle position along the X axis. As can be observed, when the laser scanner range is limited, the vehicle position indicated by the SS3 (blue) approximately matches the position indicated by the motion tracking system (green), while the displacement indicated by the SS2 (red) is clearly underestimated, as was expected.

8.2. Robot Behaviour Evaluation

Once we have assessed the flight capabilities of the MAVs equipped with the different sensor suites, we proceed to evaluate the performance of the robot behaviours. In the following, several experimental results are reported in this regard, where each behaviour is evaluated using only one of the MAVs (and a specific sensor suite). Similar results have nevertheless been observed for the other platforms.

In a first experiment, we check how the platform behaves in a situation of imminent collision. To do that, we move the Firefly platform equipped with the SS1 towards a wall. The plot for this experiment can be found in Fig. 18. The right plot shows how the longitudinal speed command provided by the MAV behaviours module ($\dot{x}_d$) coincides with a user command ($\dot{x}_{ud}$) of around 0.4 m/s until the wall in front of the vehicle becomes closer than 1.5 m (instant A), moment at which the user-desired velocity is attenuated by the *attenuated_go* behaviour making the speed command decrease in accordance to the closeness to the wall. When the wall becomes closer than 1 m (instant B), which is the minimum distance allowed (d_m), the user-desired speed is completely cancelled by the *prevent_collision* behaviour, and the platform stops. Notice that the user desired speed is around 0.4 m/s until instant C. The left plot provides the vehicle trajectory and the wall position, as captured by the motion tracking system. The actuation of the *attenuated_go* behaviour is indicated in a different colour (pink).

A second experiment, reported in Fig. 19 [right], checks the performance of the *go_ahead* behaviour. In this experiment, we have used the Pelican platform fitted with the SS2. At the beginning, the user indicates a longitudinal desired speed of 0.4 m/s and then activates the *go_ahead* behaviour (instant A). At this moment, in accordance to the behaviour definition, the speed command produced by the MAV behaviours module ($\dot{x}_d$) keeps at 0.4 m/s although the user-desired speed ($\dot{x}_{ud}$) returns to zero. This value is kept until the wall in front of the vehicle becomes closer than d_m (instant B), which is set to 1.2 m for this experiment. Then, the *prevent_collision* behaviour cancels the *go_ahead* command and stops the platform. This behaviour is also in charge of producing the negative speed command that separates the platform from the wall until it is again at the safe distance (instant C). Figure 19 [left] shows the vehicle trajectory, indicating in pink when the *go_ahead* behaviour is active.

In a third experiment, we check the performance of the *limit_max_height* behaviour. Figure 20 [right] shows how the Firefly platform equipped with the SS1 ascends at the user-desired vertical speed $\dot{z}_{ud}$ (and

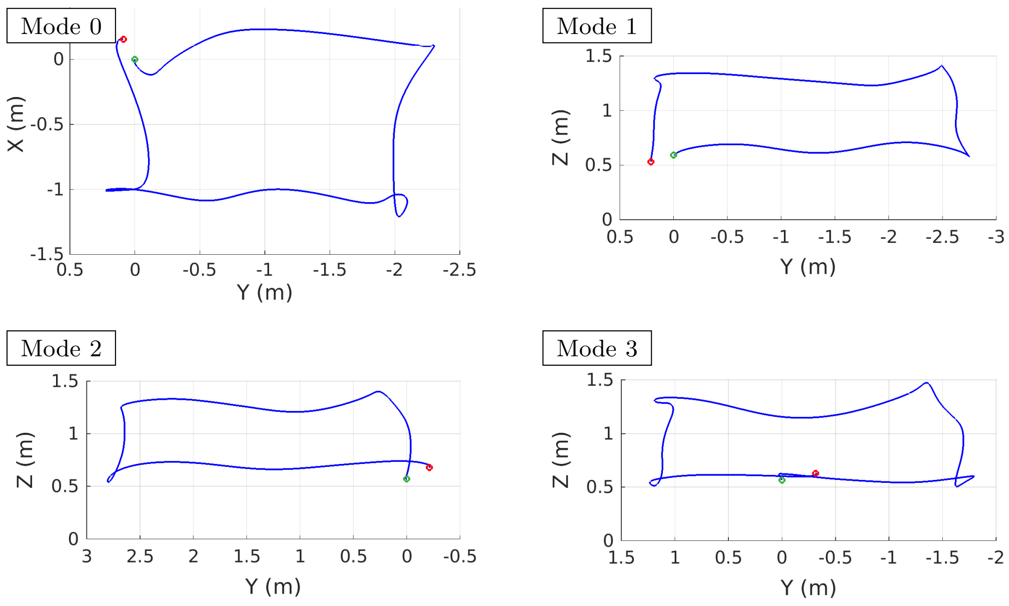


Figure 14: Plots of the trajectory of the MAV indicated by the motion tracking system during square-like flights performed using the SS1 and the different state estimation modes. Experiments performed using the AscTec Firefly. The green dot indicates the initial point, while the red dot indicates the final point.

the vertical speed command $\dot{z}_d$) until the platform reaches a height of 3 m (instant A), which was set as the maximum height for this experiment (z_M). From time instants A to B, the behaviour prevents the platform from going higher ignoring the vertical user-desired speed until it becomes zero (instant B). Next, the platform descends since the user asks for a negative vertical speed (instant C). Figure 20 [left] shows the vehicle trajectory, indicating in pink when the platform is at the maximum height and thus it is not allowed to go beyond.

Results for a fourth experiment are plotted in Fig 21 [right]. This case involves the *waiting_for_connectivity* behaviour and the Firefly platform. At the beginning of the experiment, the user orders a negative longitudinal speed to move the platform. During the displacement, the communication with the base station is lost (instant A), so that the user-desired speed signal $\dot{x}_{ud}$ is no longer available at the platform. As a consequence, the afore mentioned behaviour takes control and makes the vehicle hover while waits for a reconnection. After 5 seconds (instant B), the communication link has not been restored and the behaviour decides to make the platform land. Figure 21 [left] shows the vehicle trajectory, where different colours are used to indicate when the vehicle is receiving the desired command (red), when it is waiting and trying to reconnect (green), and when the vehicle performs the landing manoeuvre (blue).

Figure 22 corresponds to a fifth experiment, aiming at checking the performance of the *low_battery_land* behaviour. During this experiment, the Pelican is left hovering at almost 3 m until the battery voltage becomes lower than v_m, which is set to 10 V (instant A). At this moment, the behaviour takes control of the platform to make it land. The landing manoeuvre starts with a *descending* stage, to make the platform reduce its height up to 0.5 m, and finishes with the deceleration of the motors thrust (instant B).

Figure 23 [A] describes a sixth experiment in which the performance of the *ensure_reference_surface_detection* behaviour is assessed. This behaviour is only used with the SS1, so this experiment has been performed on the Firefly platform. The experiment starts with the platform flying at a certain distance from the front wall, so that the vehicle only makes use of the ground-looking optical flow sensor to estimate its velocity (state estimation mode 0). Within this mode, the vehicle is allowed to ascend (action a1) until the maximum distance to the ground, i.e. the reference surface, is attained (the maximum distance d_{b_M} was set to 1.5 m for this experiment). The next ascending order (action a2) is ignored. Next, the vehicle moves towards

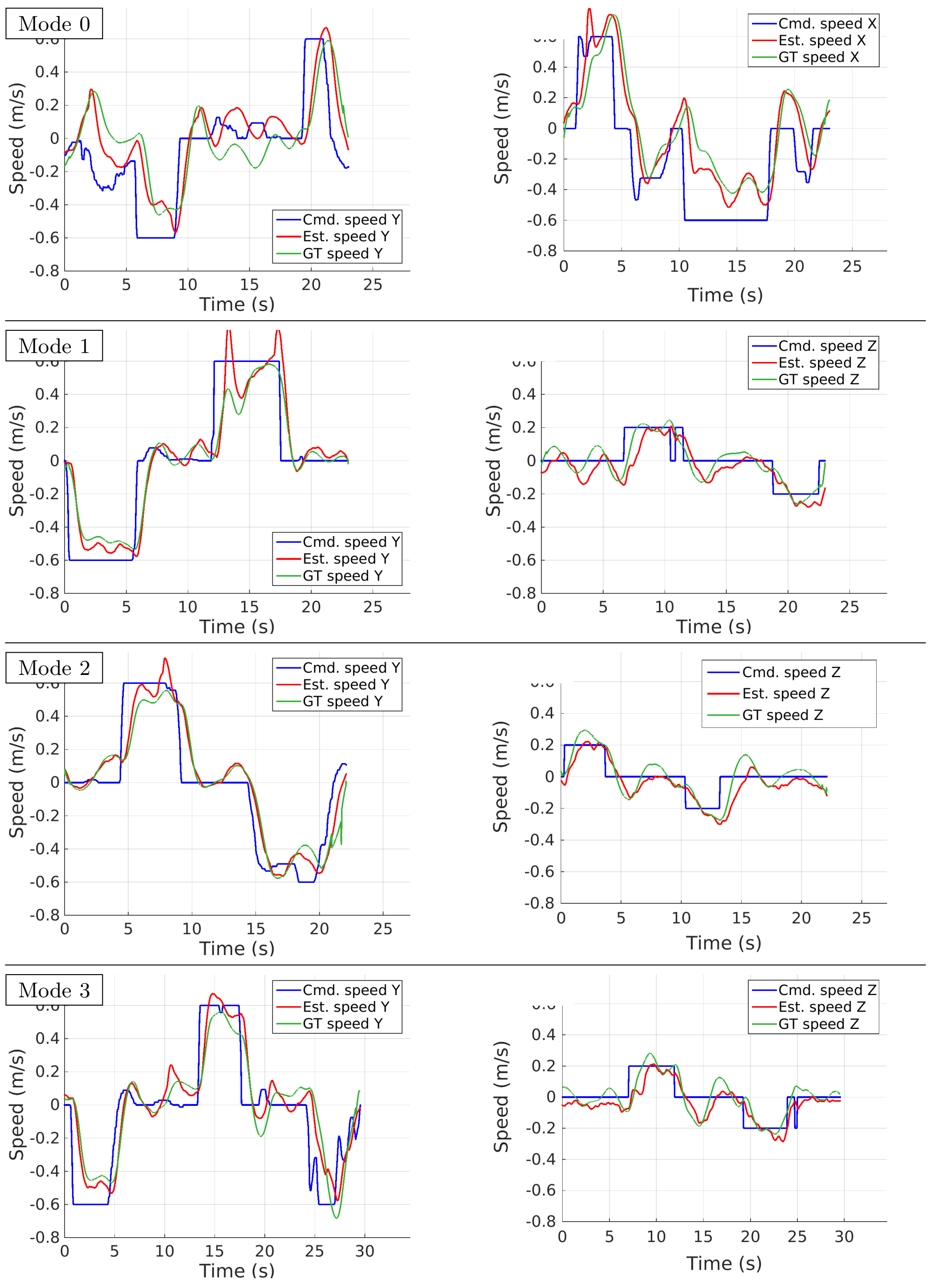


Figure 15: Plots of the speed of the aerial device while receiving commands to perform square-like trajectories. Experiments performed using the SS1 on the AscTec Firefly.

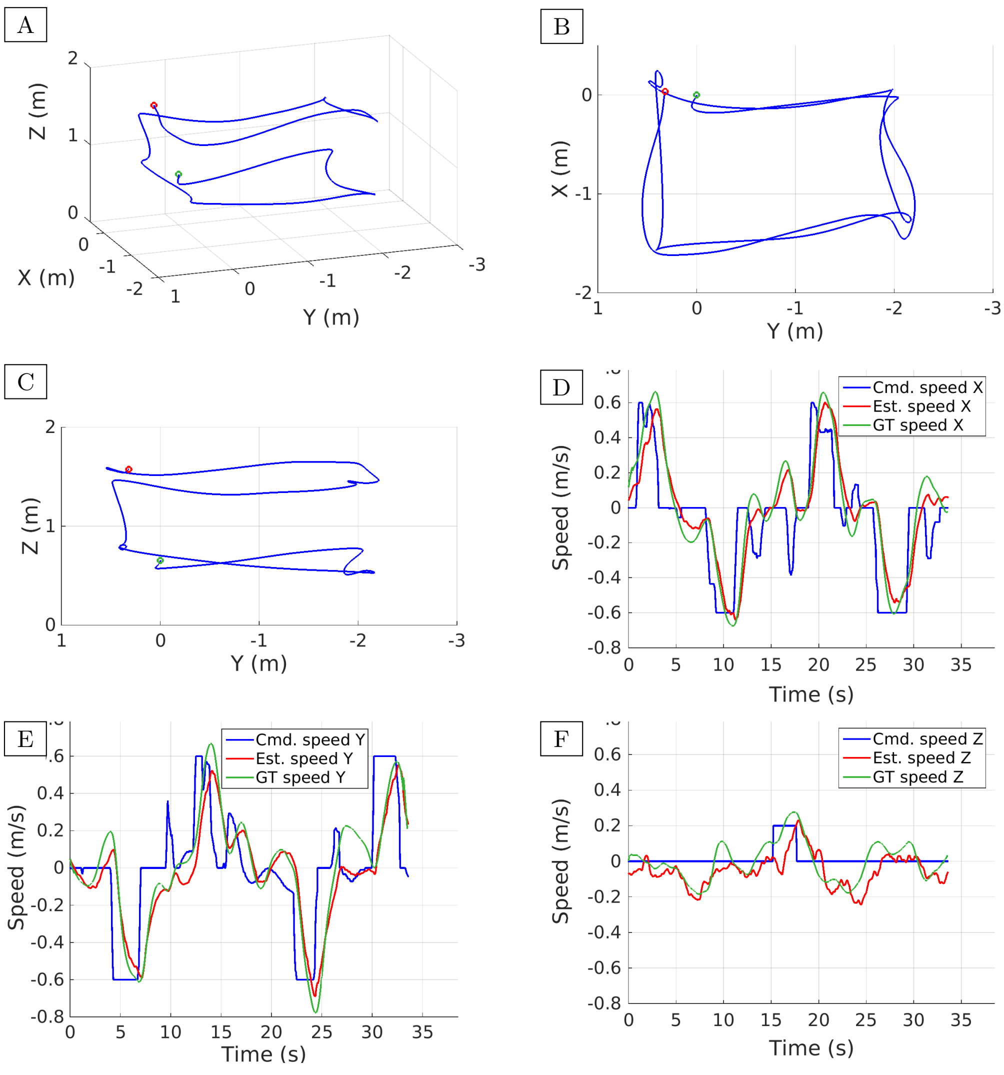
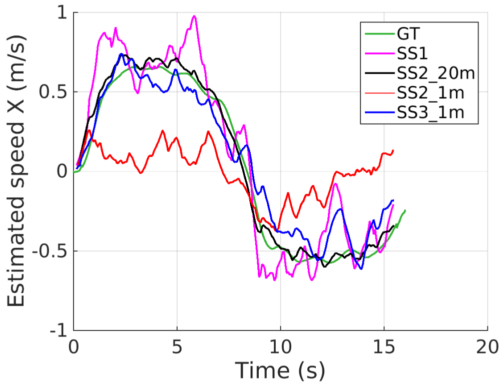


Figure 16: Results obtained with the AscTec Pelican fitted with the SS2 commanded to perform a double-square trajectory: (A) plot of the trajectory indicated by the motion tracking system (the green and red dots indicate the initial and final points), (B-C) 2D projections of the trajectory, (D-F) reactions of the MAV to the velocity commands in the tree axes.



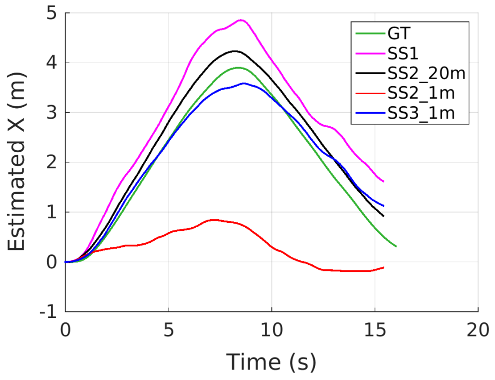
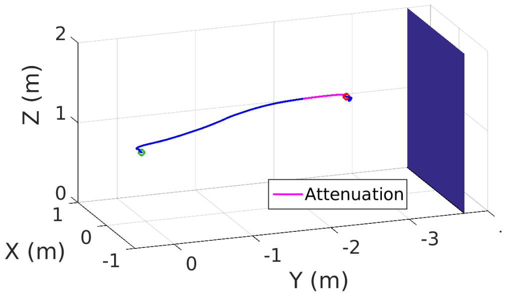


Figure 17: Results for a flight parallel to a wall using the SS3: (left) estimated speeds, (right) positions estimated via speed integration. The results are compared with the ground truth and the values obtained using the SS1 and SS2. Experiment performed limiting the laser scanner range to 1 m to force the situation inside the laboratory.



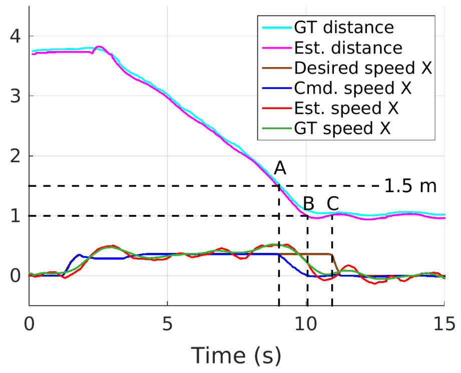
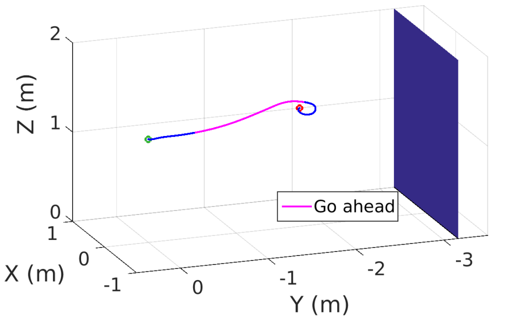


Figure 18: Performance of the *attenuated_go* and the *prevent_collision* behaviours: (left) vehicle trajectory and wall position indicated by the motion capture system, (right) the user-desired speed is obeyed (→A), it is attenuated (A→B) and cancelled to prevent an imminent collision (B→C) until the user-desired speed does become zero (C→). All units are in SI (m or m/s accordingly).



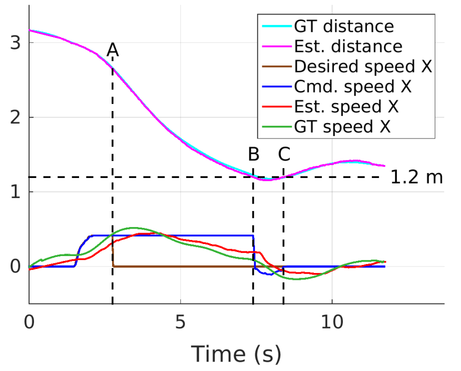


Figure 19: Performance of the *go_ahead* and the *prevent_collision* behaviours: (left) vehicle trajectory and wall position indicated by the motion capture system, (right) the user-desired speed is sustained while the wall is at enough distance (A→B), it is cancelled and even forced to be negative to prevent an imminent collision (B→C) until the platform is again at the safe distance (C→). All units are in SI (m or m/s accordingly).

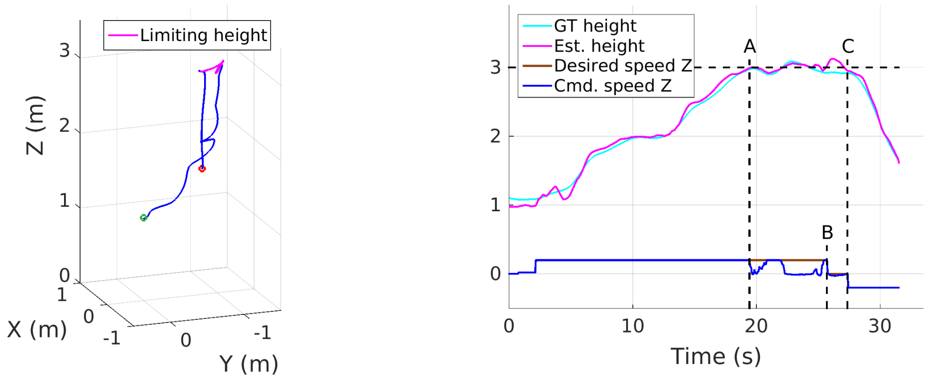


Figure 20: Performance of the *limit_max_height* behaviour: (left) vehicle trajectory indicating when the flight height is limited, (right) the user-desired vertical speed is obeyed until the platform reaches the maximum allowed height (→A), then the desired vertical speed is ignored (A→B) until it becomes zero (B→C), and finally the platform descends following again the desired speed (C→). All units are in SI (m or m/s accordingly).

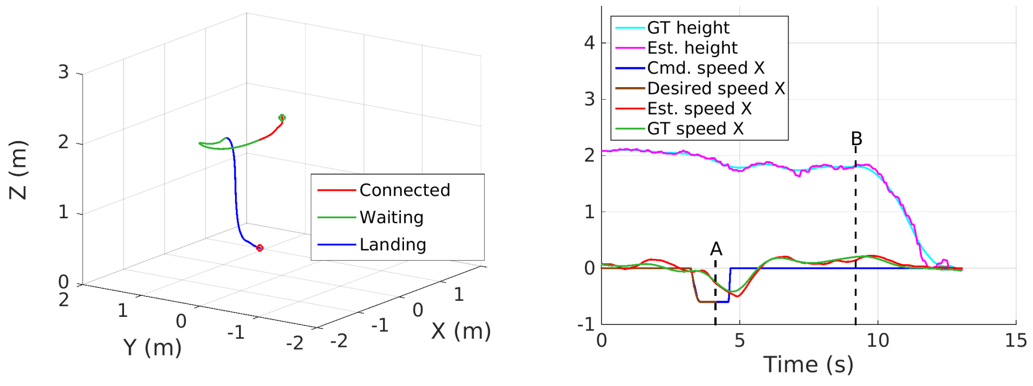


Figure 21: Performance of the *waiting_for_connectivity* behaviour: (left) vehicle trajectory indicated by the motion capture system, (right) the communication with the base station is lost during the flight (→A), what makes the behaviour force a hovering manoeuvre (A→B) while the vehicle tries to reconnect; after waiting for five seconds without success, the behaviour makes the platform land (B→). All units are in SI (m or m/s accordingly).

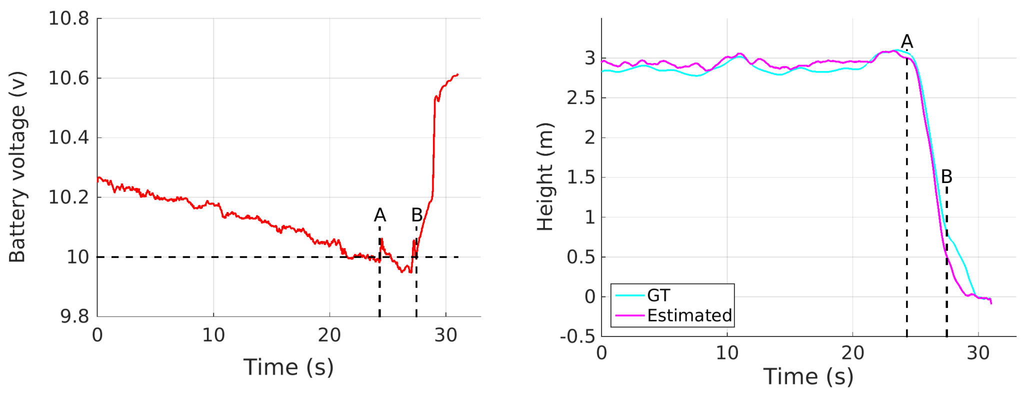


Figure 22: Performance of the *low_battery_land* behaviour: the platform hovers at 3 m until the battery voltage is below 10 V (→A), what makes the behaviour initiate the descending (A→B) and landing manoeuvres (B→).

the front wall (action a3) until the vehicle is close enough so as to also use this wall to estimate its state (state estimation mode 1). Once the vehicle is close to the wall, it moves upwards (action a4) until the

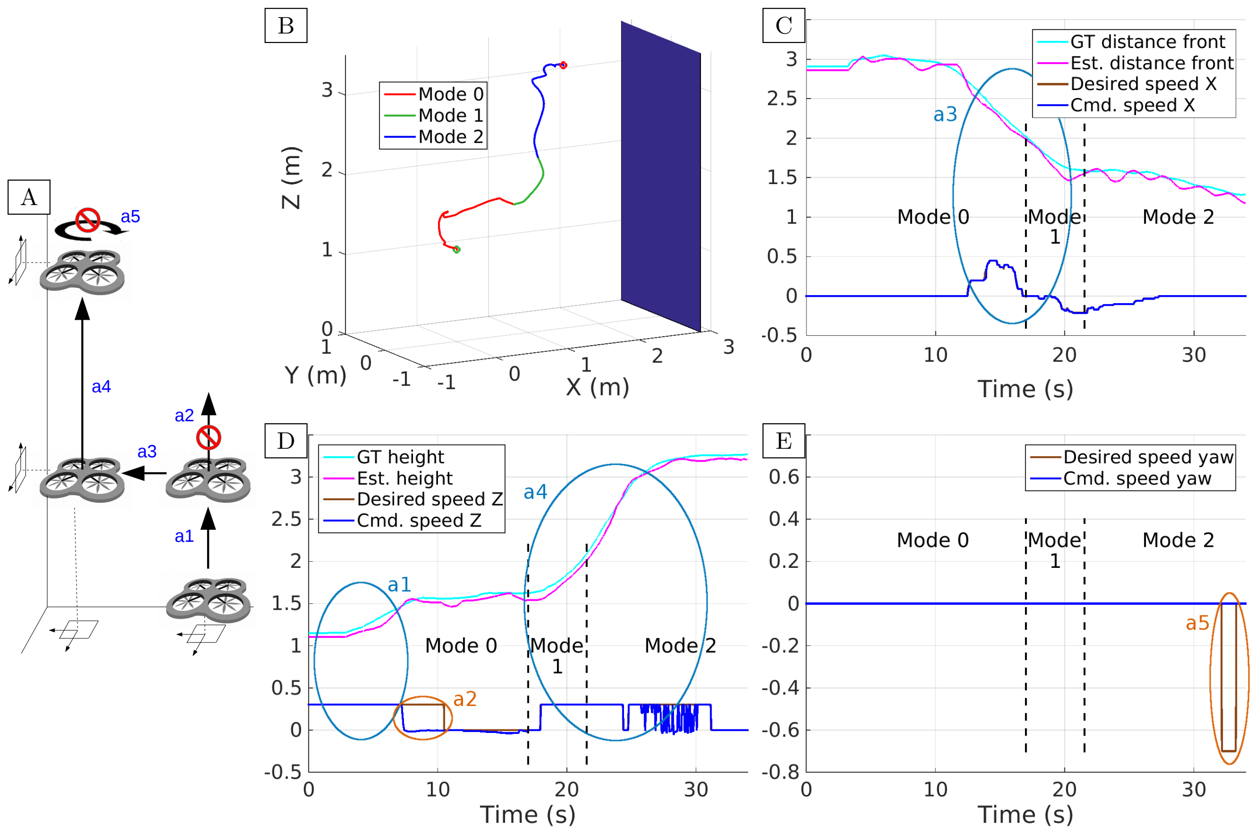


Figure 23: Illustration of the performance of the *ensure_reference_surface_detection* behaviour: (A) experiment performed, (B) trajectory followed by the MAV indicating the estimation mode used, (C-E) longitudinal, vertical and angular commands/displacements (see text for the explanation). All units are in SI (m, m/s or rad/s accordingly).

ground becomes too far for the ground-looking optical flow sensor (2 m for this experiment) and the front wall becomes the only reference surface (state estimation mode 2). Subsequently, the user tries to turn the vehicle to the right (action a5) but this action is ignored to ensure a proper reference surface detection. Figure 23 [B] shows the trajectory followed by the MAV, indicating the estimation mode used. Figures 23 [C-E] plot sensor data for the full operation, and for, respectively, the longitudinal, vertical and angular speeds. The distance to the front wall and the vehicle height are shown to make evident the corresponding motion.

8.3. *Illustration of an Inspection Mission*

In a last experiment involving specific behaviours, we show the performance of the robotic device during a typical inspection task. This experiment has been performed using the Pelican platform fitted with the SS2, and a 2.5×4 m canvas which was printed to simulate the metallic plates of a vessel wall. The operation starts when the user/surveyor makes the platform approach the wall to be inspected, i.e. the canvas. At more or less 1 m distance, the *inspection mode* is activated, and, hence, longitudinal motion as well as rotations in yaw are not allowed to ensure better image capture conditions. The operator next orders lateral and vertical motion commands to sweep the surface, while records an image sequence at 10 Hz. Figure 24 [A] shows the vehicle trajectory, indicating when the inspection mode is active. Figures 24 [B-C] illustrate the full operation for the longitudinal [B], lateral [C] and vertical [D] motions. These velocities are shown at the bottom of the plots, while distances to, respectively, the front wall/left wall/ground are shown at the top, to make evident the corresponding motion. Notice that, when the *inspection mode* is enabled (between instants A and B) the longitudinal user-desired speed is ignored, and a PID controller is in charge of keeping the distance to the inspected wall, while the user only has the option of selecting hovering or motion in the

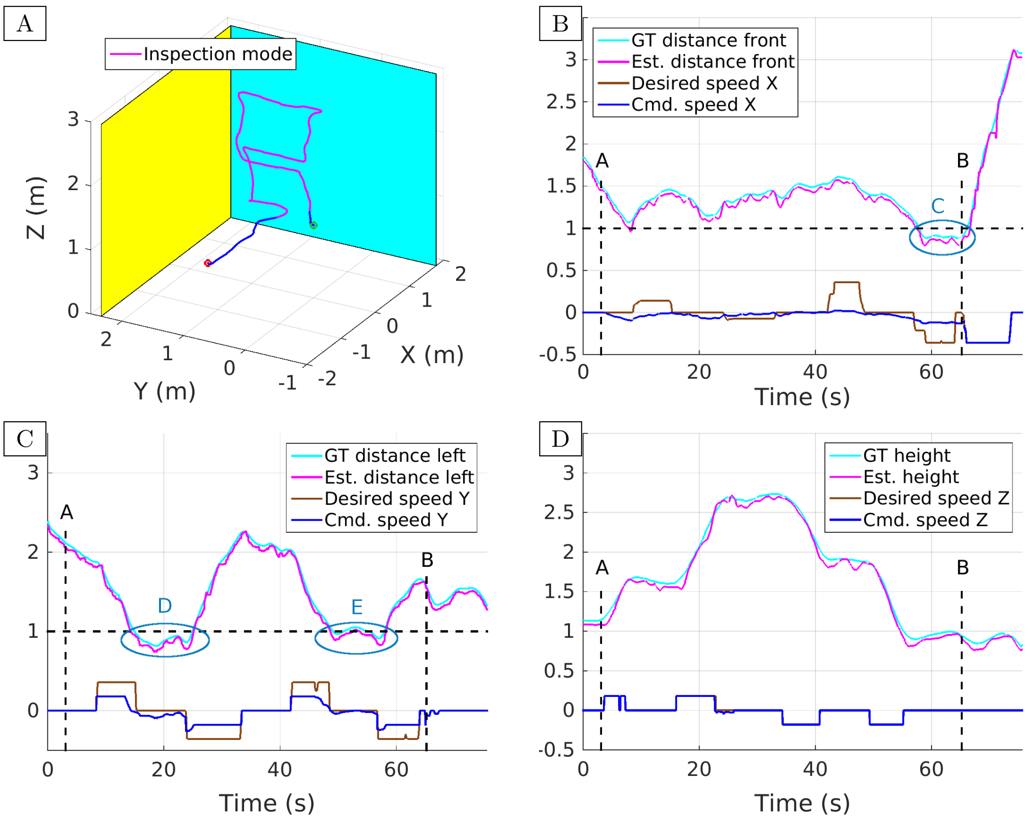


Figure 24: Performance of the robotic platform during an inspection task using the *inspection mode*: (A) walls and vehicle trajectory, indicating when the *inspection mode* is active, (B-D) longitudinal, lateral and vertical commands/displacements (see text for the explanation). All units are in SI (m or m/s accordingly).

vertical or lateral direction, but the speed command is set to ±0.2 m/s. The plots also show repulsive speed commands produced when the platform is below 1 m regarding the front or left wall (see instants C, D and E).

8.4. Position Estimation for Image Tagging

Finally, we have evaluated the SLAM algorithms as position estimators to tag the images taken with our aerial robotic tool, during an inspection mission. To do that, the positions provided by the two SLAM methods have been compared with the position indicated by the motion capture system. Two experiments have been performed to this end. The first one, reported in Fig. 25 [A], consists in a free flight including movements along the three axes. The second flight, reported in Fig. 25 [B], consists in sweeping a wall. As can be observed in the plots, the trajectories provided by both SLAM methods mostly coincide with the path provided by the motion tracking system.

To evaluate the position error of each SLAM method, we have computed the histogram of the differences regarding the ground truth values. These differences have been computed for the three axes separately. The resulting histograms can be found in Fig. 25. Plots [C] and [D] show the error histograms corresponding to the ORB-SLAM for the first and second experiment respectively, while plots [E] and [F] provide the values obtained employing the GMapping method, used as part of the SS2 pipeline. As can be observed, the position error is in all cases small and zero-centred.

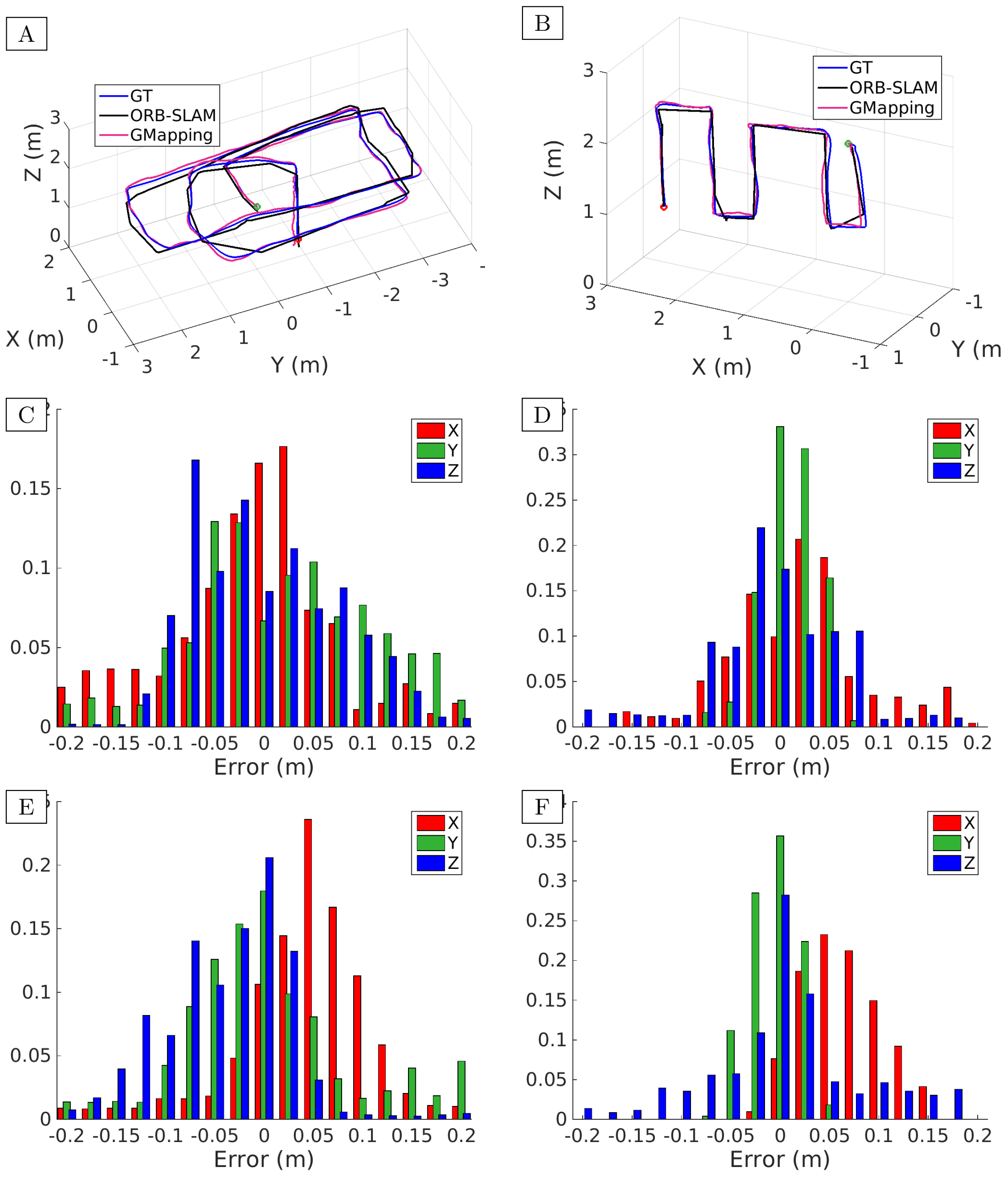


Figure 25: Performance of the SLAM algorithms in two different experiment: (A/B) trajectory estimated by the two SLAM algorithms, compared with the trajectory provided by the motion tracking system, (C/E) position errors for ORB-SLAM/GMapping for flight A, and (D/F) position errors for ORB-SLAM/GMapping for flight B.

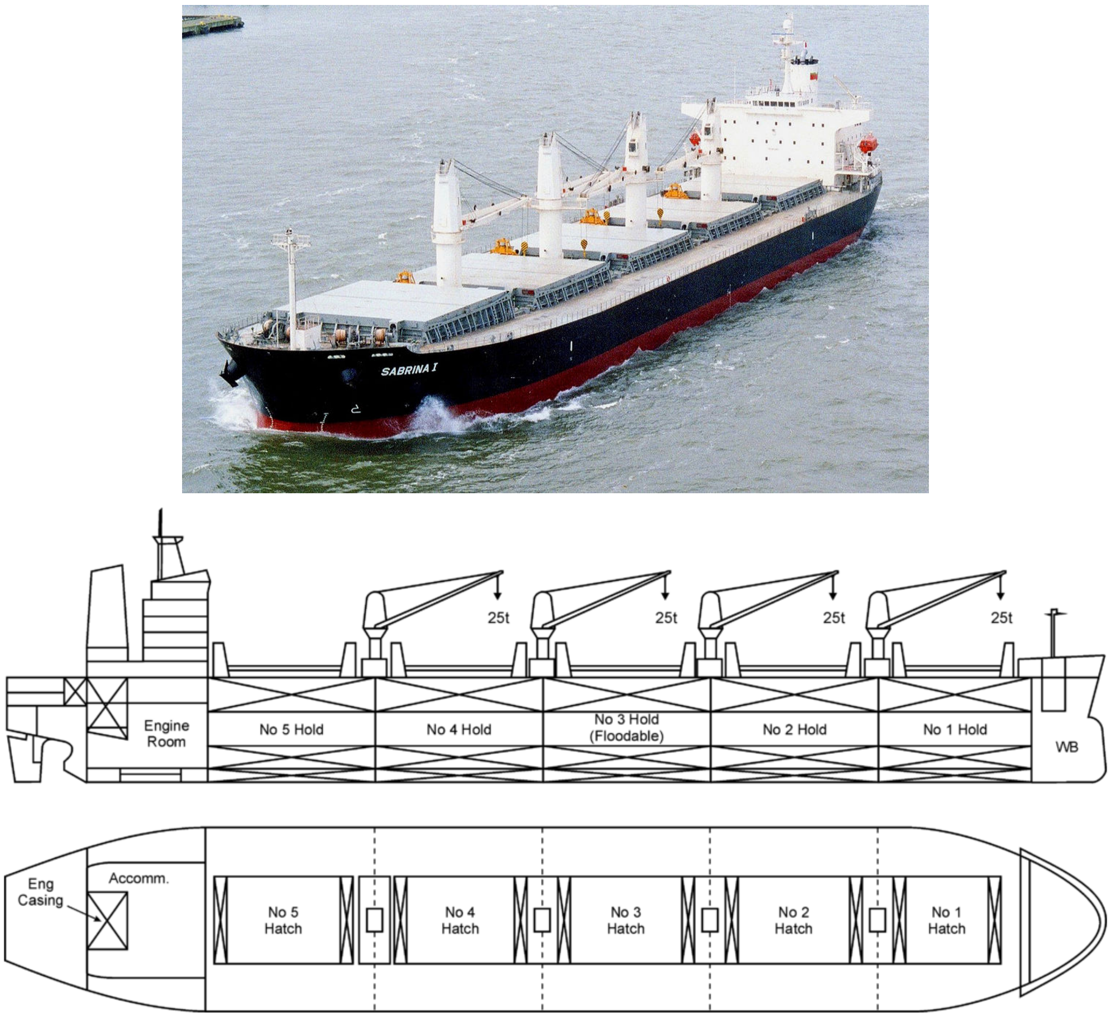


Figure 26: Handymax bulk carrier similar to the vessel visited during the field tests. Diagram by Rémi Kaupp taken from Wikipedia[6]

9. Field Tests

This section reports on the experimental results obtained from the robotic tool during the inspection of a real vessel. The field trials were performed on board a Handymax bulk carrier with deadweight tonnage above 45000 tons, and whose size was 190 m (length)×32 m (breadth)×16.5 m (height). A picture of this vessel can not be included for confidentiality reasons. Instead, Fig. 26 provides a general view and the plans corresponding to a vessel with the same characteristics. During the test campaign, the MAV was operated in three different compartments: the cargo hold #4 (see Fig. 27 [A]), the water ballast topside tank #3 (see Fig. 27 [B]) and the forepeak tank (see Fig. 27 [C]).

The operating conditions in each compartment were very different. On the one hand, the cargo hold was a very large compartment where the light could be relatively adjusted, since the hatch could be opened and closed. On the other hand, the forepeak and topside ballast tanks were narrow and dark spaces accessible through a manhole-sized entry point, so that the onboard LED had to be used to allow for a proper visual inspection.

[6]`https://en.wikipedia.org/wiki/Bulk_carrier`

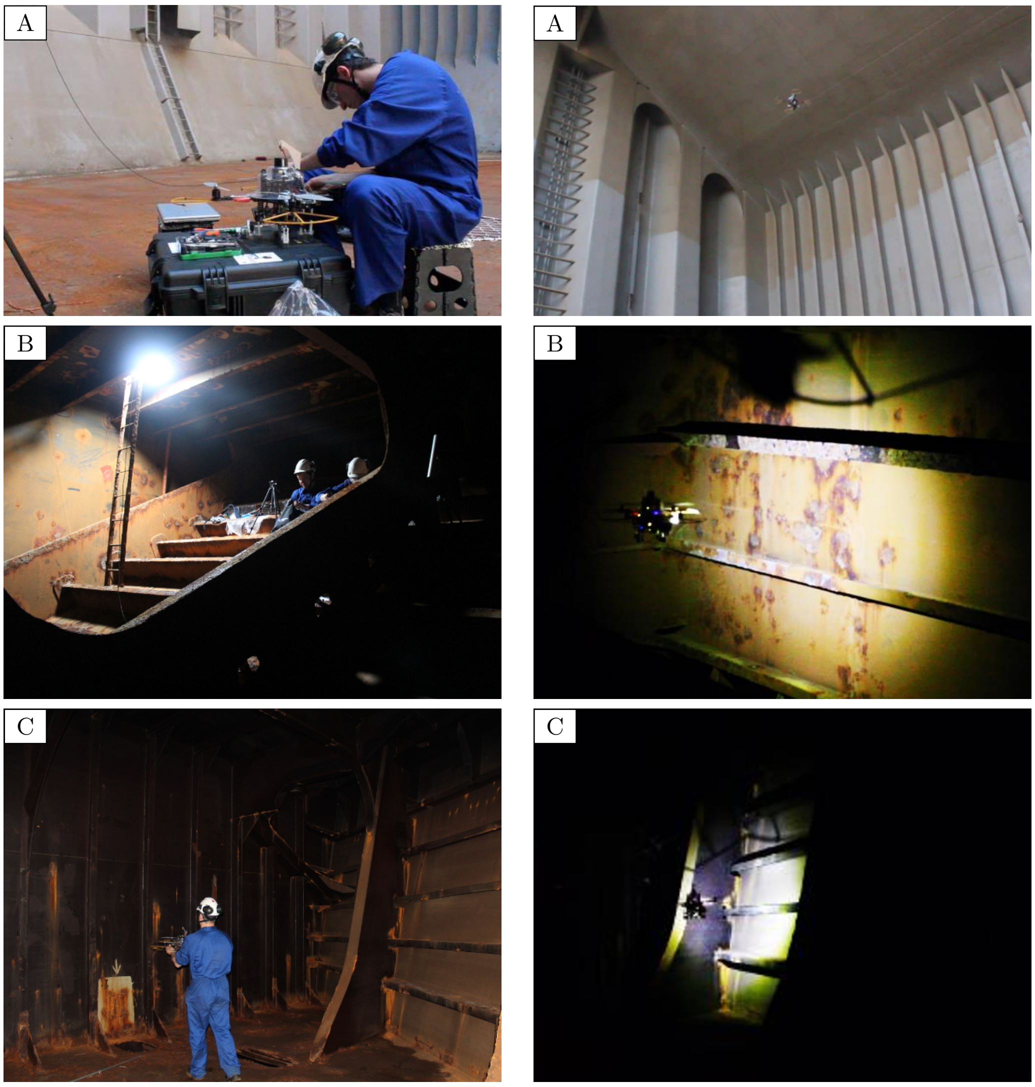


Figure 27: Some pictures to illustrate testing on board the bulk carrier: [A] cargo hold, [B] topside tank and [C] forepeak tank.

The MAV used during the field trials was the AscTec Pelican platform equipped with the SS2. This sensor suite, based on the use of a laser scanner, is suitable for flying in dark spaces (e.g. inside a ballast tank) where the optical flow sensors, employed in the SS1, can not operate. Furthermore, this vehicle is equipped with a high power LED to illuminate the inspected surface if necessary.

All the experiments were performed following the same procedure: (1) the vehicle is situated in a flat and obstacle-free area for the take-off, (2) the user sends the take-off command using a gamepad/joystick and the vehicle starts the flight, (3) the user approximates the platform to the area where the inspection has to take place, while the control architecture based on SA takes care of the platform preservation, (4) the user can optionally enable the *inspection mode* to make the vehicle move smoothly and keep at a constant distance to the inspected surface, (5) a sequence of pictures can be started when desired, (6) the user can command the platform along the lateral and vertical axes (also longitudinally if the *inspection mode* is not enabled) to perform the inspection, (7) the sequence of pictures can be stopped when desired, (8) the *inspection mode* is disabled (in case it was enabled), (9) the user commands the platform to an obstacle-free area for landing, and (10) the user issues the command for landing.

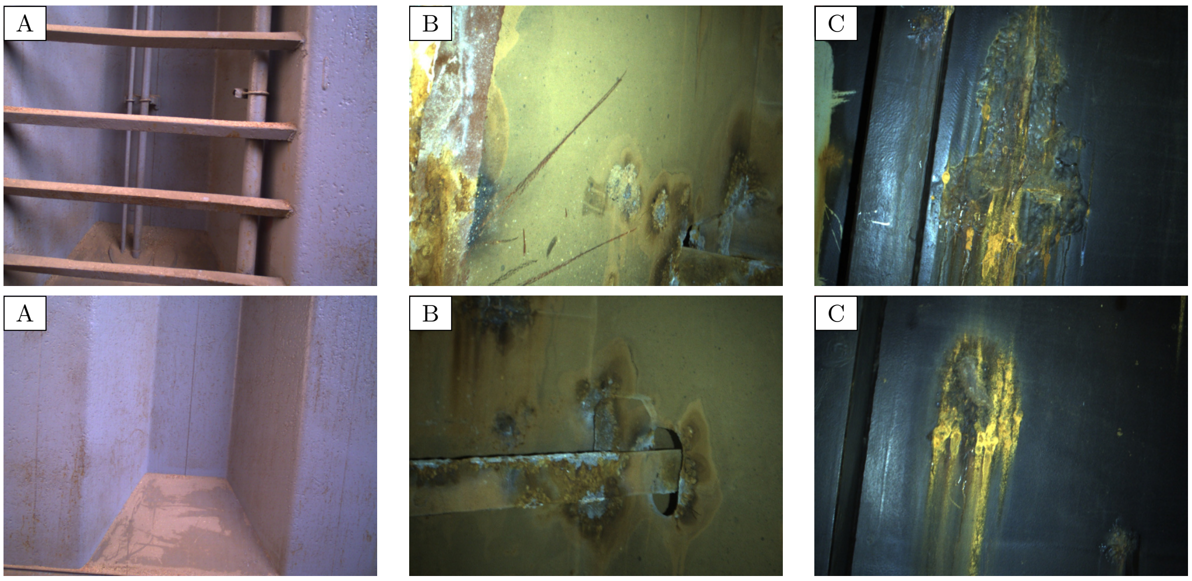


Figure 28: Images taken with the on-board camera while flying in the different compartments of the vessel: [A] cargo hold, [B] topside tank and [C] forepeak tank.

Figure 27 [A] shows some pictures taken during testing at the cargo hold. In a first session, flights took place in front of the aft bulkhead, frames #75-78, while in a second session, testing focused on the cargo web frames, starboard side, frames #78-90.

By way of illustration, Fig. 29 shows the paths estimated for two of these flights. In the two cases, paths were successfully estimated by means of the GMapping SLAM method, which makes use of the data provided by the laser scanner. Figure 30 shows a longer flight in front of a large wall, in which the robot flew from left to right and then back. On this occasion, the SLAM module got confused just before coming back, and, because of this, the path does not ends where it started. This error is probably due to the long distance to all the corners and lack of distinguishable structural elements inside the cargo hold. To show the actual path followed by the MAV, Fig. 30 [C-D] shows the first part of the flight, while Fig. 30 [E-F] shows the second part. Notice that this error in the position estimation does not compromise the platform nor the mission, since the control actions take place over the speed. Some of the images captured by the on-board camera during this flight can be found in Fig. 28 [A].

Figure 27 [B] shows some pictures of the experiments performed at the topside tank, which took place in front of frames #111-131. Unlike the cargo hold, this compartment is a confined space where the self-preservation capability, included in the SA framework, becomes critical. These tests also allowed us to check the capability of the platform to take pictures under low-light (hatchway open) and under completely dark (hatchway closed) conditions.

By way of illustration, Fig. 31 shows the paths estimated for two of the flights performed in the topside tank. In the first case, the vehicle was flying with some light available from the hatch. In the latter case, the hatch was closed, and hence the area was completely dark. In both cases, the SLAM method provided a successful position estimation. Some of the images captured by the on-board camera during this last flight can be found in Fig. 28 [B]. As can be observed, these are adequately illuminated thanks to the use of the high power LED installed in the MAV.

Finally, Fig. 27 [C] shows some pictures of the experiments performed at the forepeak tank. Testing took place among frames #215-225 in the upper stringer. All the experiments in this compartment were performed in complete darkness.

By way of illustration, Fig. 32 shows the paths estimated for two of the flights performed in the topside tank, while Fig. 28 [C] provides some of the images captured during the latter flight using the on-board

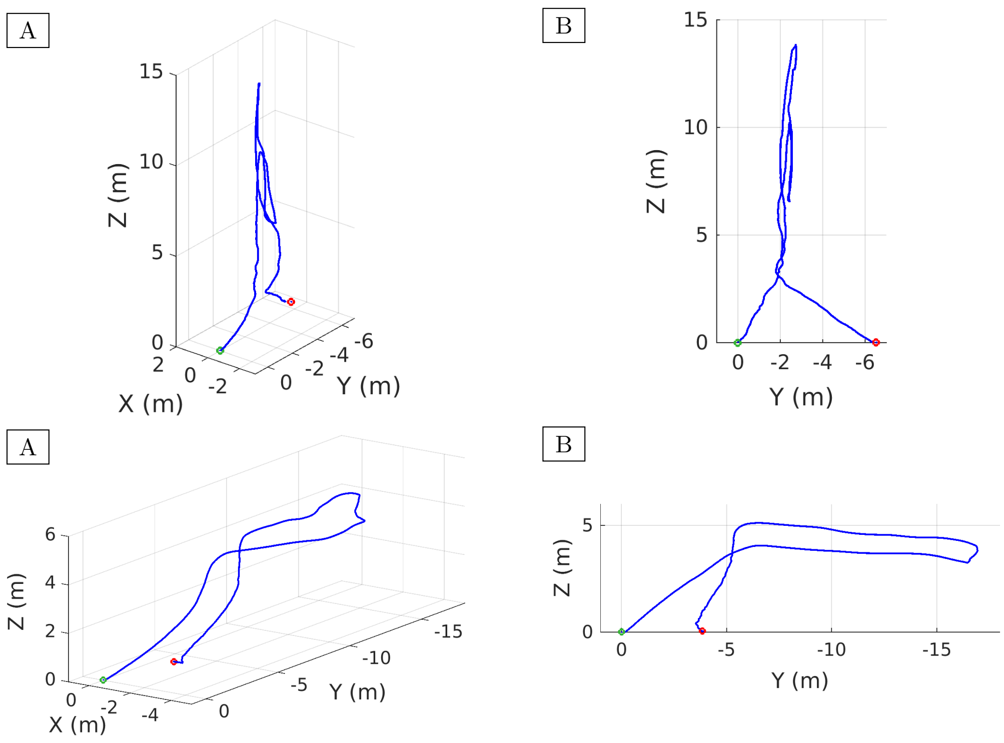


Figure 29: Estimated paths followed by the aerial robot during two flights in the cargo hold: (A) 3D plot of the trajectories, (B) 2D projection of the trajectories. The green and red dots indicate the initial and final points respectively.

camera. As happened in the topside tank, the self-preservation capability of the platform ensured an effective and safe operation, while the laser-based SLAM process supplied correct position estimations thanks to the well-structured environment. The pictures provided by the camera module were also good, thanks to the illumination available from the on-board LED.

The images taken during the inspection campaign were later analysed to detect the defective areas using the saliency-based defect detection method described in [6]. Examples of detection outputs can be found in Fig. 33. As can be observed, the detection method can successfully detect the corroded areas in the different vessel compartments, without any problem due to the illumination conditions or the motion of the platform.

10. Conclusions

A reconfigurable framework to turn a MAV into a useful tool for vessel visual inspection has been presented. Following the SA paradigm, this framework has been devised in order to obtain a robotic device that can be operated as a flying camera that takes care of all safety-related issues, while the surveyor can concentrate on the inspection task.

The system architecture has been devised to be reconfigurable in the sense that it can incorporate different sensor suites depending on the platform payload capabilities and the operational/environmental conditions. In this regard, three alternative sensor suites have been proposed and detailed, including the software components necessary for their suitable integration.

An extensive experimental evaluation has been performed to validate the capabilities and usability of different platform configurations, including tests performed under laboratory conditions and a real inspection campaign carried out on board a bulk carrier. The results obtained confirm that the system requirements have been successfully fulfilled. In particular, the images taken using the aerial robotic tool in *inspection*

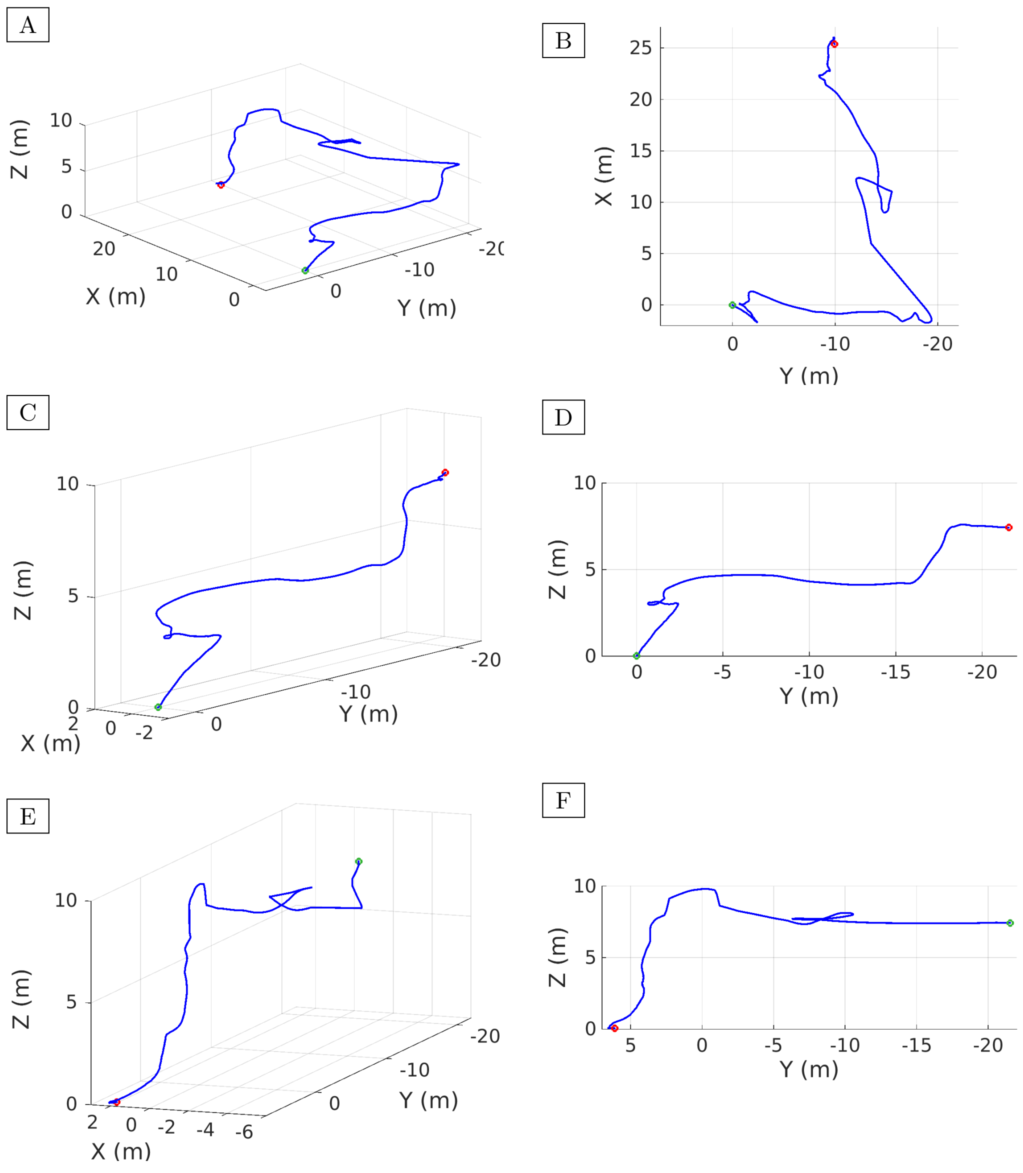


Figure 30: Erroneous estimation of the aerial robot path during a flight in the cargo hold: (A-B) 3D plot and 2D projection of the complete trajectory, (C-D) 3D plot and 2D projection of the first part of the flight, (E-F) 3D plot and 2D projection of the second part of the flight. The green and red dots indicate the initial and final points respectively.

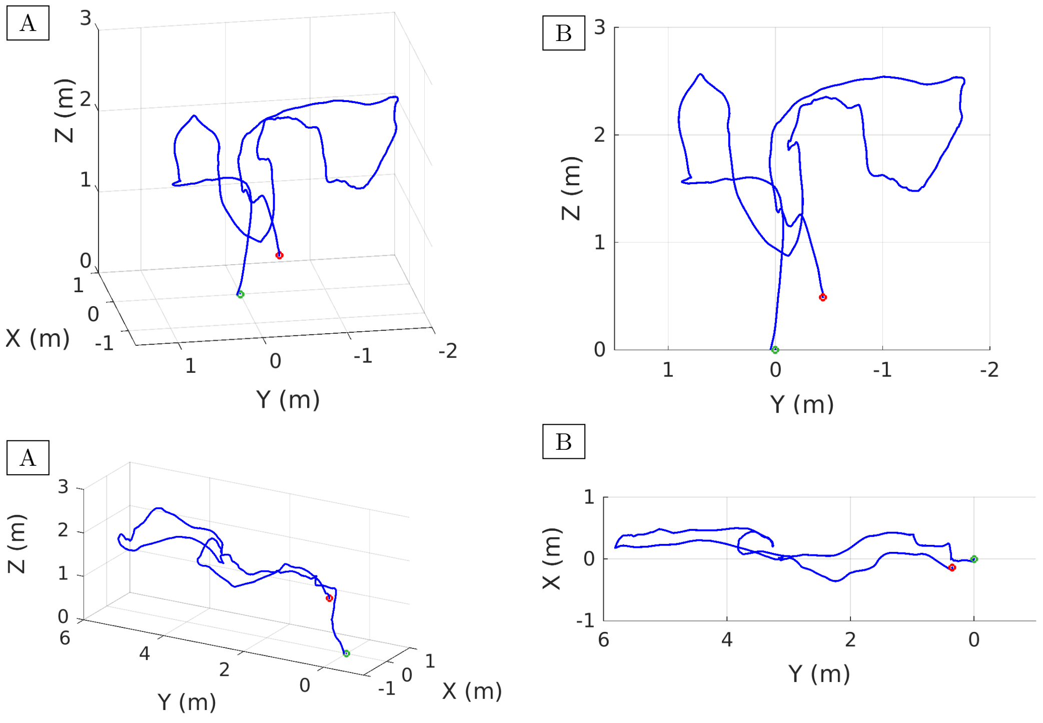


Figure 31: Estimated paths followed by the aerial robot during two flights in the topside tank: (A) 3D plot of the trajectories, (B) 2D projection of the trajectories. The green and red dots indicate the initial and final points respectively.

mode present good quality so that they can be used for a posterior inspection of defects by a human surveyor, or to feed defect detection algorithms to autonomously identify/locate the defective areas.

In comparison with the MAV presented in [4], the new approach allows introducing the human surveyor into the position control loop, what increases the platform usability and makes the vessel inspection more effective. At the same time, since its control architecture does not rely on precise position estimations (which may be difficult to obtain in certain scenarios inside real vessels), the new platform is more reliable and secure.

Acknowledgements

This work is partially supported by FEDER funding, by the European Social Fund through Ph.D. scholarship FPI10-43175042V (Conselleria d'Educació, Cultura i Universitats, Govern de les Illes Balears), by project nr.AAEE50/2015 (Direcció General d'Innovació i Recerca, Govern de les Illes Balears), by FP7 project INCASS (GA 605200) and by H2020 project ROBINS (GA 779776).

References

[1] Akinfiev, T. S., Armada, M. A., Fernandez, R., 2008. Nondestructive Testing of the State of a Ship's Hull with an Underwater Robot. Russian Journal of Nondestructive Testing 44 (9), 626–633.
[2] Arkin, R. C., 1998. Behavior-based Robotics. MIT Press.
[3] Bibuli, M., Bruzzone, G., Bruzzone, G., Caccia, M., Giacopelli, M., Petitti, A., Spirandelli, E., 2012. MARC: Magnetic Autonomous Robotic Crawler Development and Exploitation in the MINOAS Project. In: International Conference on Computer Applications and Information Technology in the Maritime Industries. pp. 62–75.

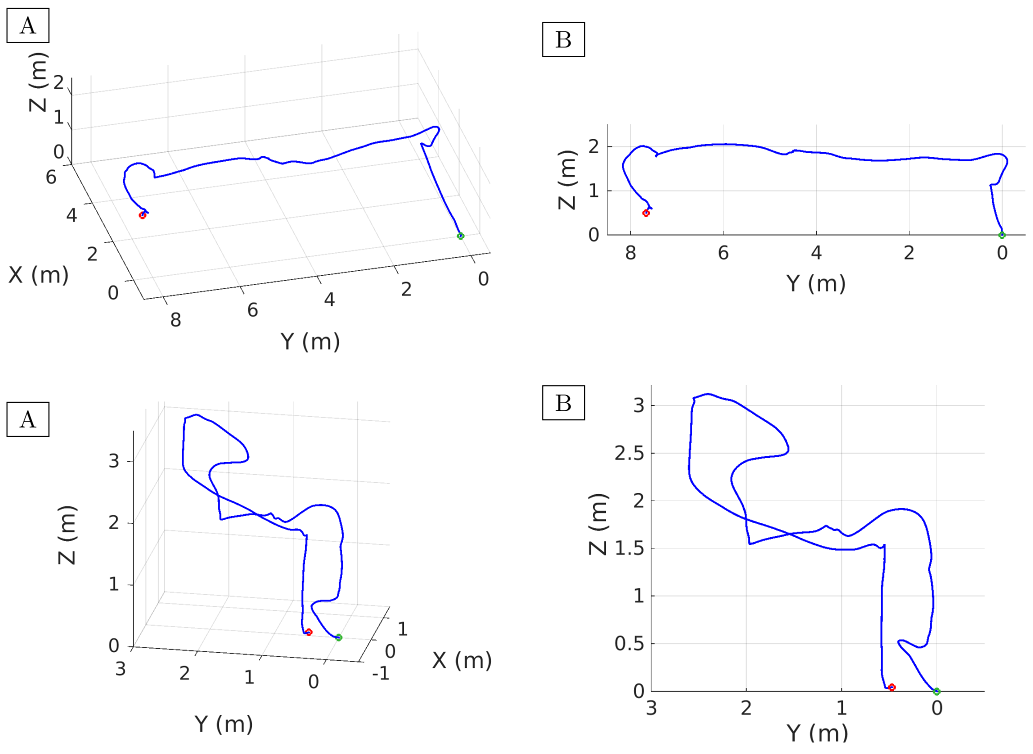


Figure 32: Estimated paths followed by the aerial robot during two flights in the forepeak tank: (A) 3D plot of the trajectories, (B) 2D projection of the trajectories. The green and red dots indicate the initial and final points respectively.

[4] Bonnin-Pascual, F., Garcia-Fidalgo, E., Ortiz, A., 2012. Semi-autonomous Visual Inspection of Vessels Assisted by an Unmanned Micro Aerial Vehicle. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 3955–3961.

[5] Bonnin-Pascual, F., Ortiz, A., 2016. A Flying Tool for Sensing Vessel Structure Defects using Image Contrast-based Saliency. IEEE Sensors Journal 16 (15), 6114–6121.

[6] Bonnin-Pascual, F., Ortiz, A., 2017. A Novel Approach for Defect Detection on Vessel Structures using Saliency-related Features. Ocean Engineering (In Press).

[7] Bonnin-Pascual, F., Ortiz, A., Garcia-Fidalgo, E., Company, J. P., 2015. A Micro-Aerial Platform for Vessel Visual Inspection based on Supervised Autonomy. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 46–52.

[8] Censi, A., 2008. An ICP Variant using a Point-to-Line Metric. In: IEEE International Conference on Robotics and Automation.

[9] Cheng, G., Zelinsky, A., 2001. Supervised Autonomy: A Framework for Human-Robot Systems Development. Autonomous Robots 10, 251–266.

[10] Eich, M., Bonnin-Pascual, F., Garcia-Fidalgo, E., Ortiz, A., Bruzzone, G., Koveos, Y., Kirchner, F., 2014. A Robot Application to Marine Vessel Inspection. Journal of Field Robotics 31 (2), 319–341.

[11] Ferreira, C. Z., Conte, G. Y. C., Avila, J. P. J., Pereira, R. C., Ribeiro, T. M. C., 2013. Underwater Robotic Vehicle for Ship Hull Inspection: Control System Architecture. In: International Congress of Mechanical Engineering.

[12] Fondahl, K., Eich, M., Wollenberg, J., Kirchner, F., 2012. A Magnetic Climbing Robot for Marine Inspection Services. In: International Conference on Computer Applications and Information Technology in the Maritime Industries. pp. 92–102.

[13] Grisetti, G., Stachniss, C., Burgard, W., 2007. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. IEEE Transactions on Robotics 23, 34–46.

[14] Honegger, D., Meier, L., Tanskanen, P., Pollefeys, M., 2013. An Open Source and Open Hardware Embedded Metric Optical Flow CMOS Camera for Indoor and Outdoor Applications. In: IEEE International Conference on Robotics and Automation. pp. 1736–1741.

[15] Huerzeler, C., Caprari, G., Zwicker, E., Marconi, L., 2012. Applying Aerial Robotics for Inspections of Power and Petrochemical Facilities. In: International Conference on Applied Robotics for the Power Industry. pp. 167–172.

[16] Ishizu, K., Sakagami, N., Ishimaru, K., Shibata, M., Onishi, H., Murakami, S., Kawamura, S., 2012. Ship Hull Inspection using A Small Underwater Robot With A Mechanical Contact Mechanism. In: IEEE/MTS OCEANS Conference. pp. 1–6.

[17] Morgenthal, G., Hallermann, N., 2014. Quality Assessment of Unmanned Aerial Vehicle (UAV) Based Visual Inspection

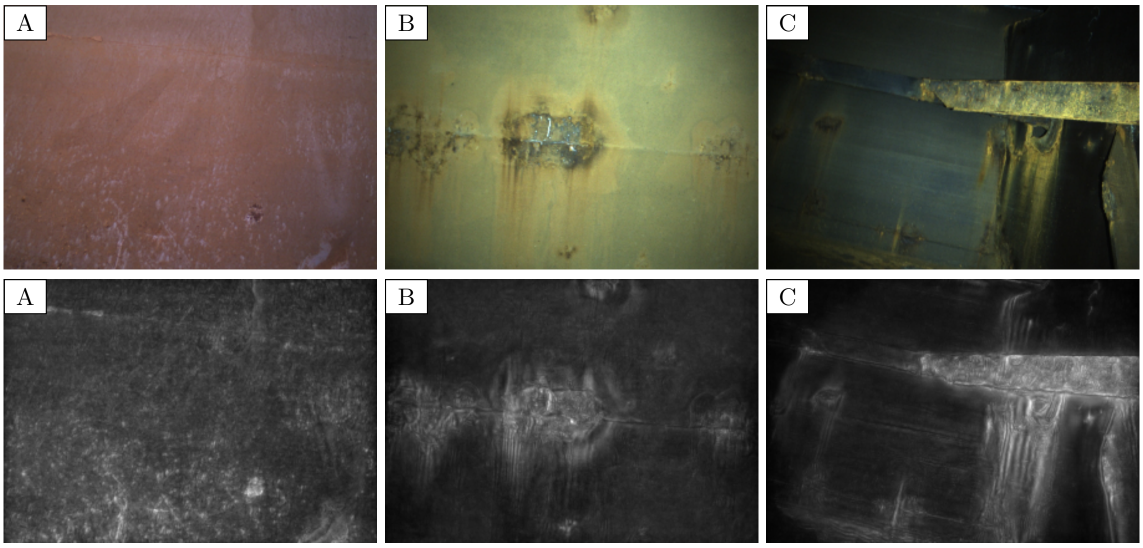


Figure 33: (Top) Images taken by the aerial inspection tool inside the vessel compartments: (A) cargo hold, (B) topside tank, and (C) forepeak tank. (Bottom) Defect maps resulting from the saliency-based defect detection algorithm described in [6].

of Structures. Advances in Structural Engineering 17 (3), 289–302.

[18] Mur-Artal, R., Montiel, J. M. M., Tardós, J. D., 2015. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. IEEE Transactions on Robotics 31 (5), 1147–1163.

[19] Narewski, M., 2009. Hismar - Underwater Hull Inspection and Cleaning System As a Tool for Ship Propulsion System Performance Increase. Journal of Polish CIMAC 4 (2), 227–234.

[20] Newsome, S. M., Rodocker, J., 2009. Effective Technology for Underwater Hull and Infrastructure Inspection. In: IEEE/MTS OCEANS Conference. pp. 1–6.

[21] Ortiz, A., Bonnin-Pascual, F., Garcia-Fidalgo, E., Company-Corcoles, J. P., 2016. Vision-Based Corrosion Detection Assisted by a Micro-Aerial Vehicle in a Vessel Inspection Application. Sensors 16 (paper nr. 2118).

[22] Ozog, P., Carlevaris-Bianco, N., Kim, A., Eustice, R. M., 2016. Long-term Mapping Techniques for Ship Hull Inspection and Surveillance using an Autonomous Underwater Vehicle. Journal of Field Robotics 33 (3), 265–289.

[23] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., 2009. ROS: an Open-Source Robot Operating System. In: ICRA Workshop on Open Source Software.

[24] Ruffo, M., Castro, M. D., Molinari, L., Losito, R., Masi, A., Kovermann, J., Rodrigues, L., 2014. New Infrared Time-of-flight Measurement Sensor for Robotic Platforms. In: IMEKO TC4 Int. Symposium and Int. Workshop on ADC Modelling and Testing. pp. 13–18.

[25] Sa, I., Hrabar, S., Corke, P., 2015. Inspection of Pole-Like Structures using a Visual-Inertial Aided VTOL Platform with Shared Autonomy. Sensors 15 (9), 22003–22048.