Personal and Ubiquitous Computing
https://doi.org/10.1007/s00779-019-01356-4

ORIGINAL ARTICLE

Simulation in real conditions of navigation and obstacle avoidance with PX4/Gazebo platform

Jesús García[1] · Jose M. Molina[1]

Received: 6 May 2019 / Accepted: 29 November 2019

Abstract
In the future, UAVs should be a part of the IoT ecosystems. Integration of sensors onboard allows to enrich information stored in the cloud and, at the same time, to improve the capacities of UAVs. Developing new sensors and the integration in UAV architecture could improve control functions. Design of future UAV systems requires from advanced tools to analyze the system components and their interaction in real operational conditions. In this work, authors present an approach to integrate and evaluate a LIDAR sensor and the capacity for improving navigation and obstacle avoidance functions in simulated situations using a real UAV platform. It uses available software for mission definition and execution in UAVs based on PixHawk flight controller and peripherals. The proposed solution (a general method that could be used to integrate other kind of sensors) shows physical integration of the main types of sensors in UAV domain both for navigation and collision avoidance, and at the same time the use of powerful simulation models developed with Gazebo. Some illustrative results show the performance of this navigation and obstacle avoidance function using the simulated sensors and the control of the real UAV in realistic conditions.

Keywords UAVs sensor processing · Data analysis · System design · Gazebo simulation

1 Introduction

IoT technology have had a fast evolution in recent years. The goal of this paper is to show the capacity of UAVs (unmanned aerial vehicles) to integrate heterogeneous sensors to acquire data from the environment to be used as a source of data for IoT environments and to control the UAV. IoT platforms store the information collected in an ecosystem in the cloud, and then model the behavior of sensors (in order to be used in future simulations) or adjust some parameters of decision functions. In this paper, integration of new sensors in a UAV is analyzed, from simulated conditions to the real world. UAVs must safely execute monitoring missions, acquiring data from these new sensors that will be stored in the cloud and could be used to improve some control functions.

✉ Jesús García
jgherrer@inf.uc3m.es

Jose M. Molina
molina@ia.uc3m.es

1 Applied Artificial Intelligence Group (GIAA), Universidad Carlos III, Madrid, Spain

UAVs flying autonomous missions must be controlled based on data streams from electro-mechanical sensors and local or global positioning systems. The controller usually is an embedded microcontroller with appropriate interfaces to all vehicle components. In this work, PixHawk/PX4 has been selected as hardware/software platform able to integrate navigation and detection sensor data together with software modules to process the stream of data with fusion algorithms [1].

Fusion of complementary sensors is essential for navigation, a very extended area due to the ubiquity of GPS and the availability of inertial sensors based on inexpensive MEMS components [2–4]. Other options for non-dependence on the GPS signal involve the deployment of autonomous localization systems such as the recognition of the environment by artificial vision [5] or location by means of electromagnetic beacons [6], with the associated cost of developing a complementary infrastructure.

Complementary to navigation technologies, the use of lasers in combination with other detection sensors (sonar, radar, video), allows the extension of autonomy to safer navigation with obstacle avoidance. In the UAVs, the integration requirements (consume, weight, dimensions) are much more restrictive than general case but, even so, it is a line in continuous development [7–9]. Range sensors are very useful for a wide

Published online: 02 January 2020

variety of UAV applications, such as the measurement of altitude, obstacle detection, and surface mapping [10–12]. This information is sent to the base station and stored in the cloud as the rest of IoT sensors.

There are several types of distance sensors, and each of them has its advantages, limitations, and restrictions:

- LIDAR (light detection and ranging) determines the distance to an object or surface with a pulsed laser beam.
- SoNAR (sound navigation and ranging) is a technique that uses the propagation of sound to navigate, communicate, or detect objects. It uses sound impulses.
- RaDAR (radio detection and ranging) is based on radio waves with pulsed emissions or frequency modulation in order to detect objects based on receiving the reflected signals in the obstacles.

The aforementioned ranging technologies pose different limitations, with the common aspect of providing measurements of distance to obstacles or structures in the environment. Choosing the right sensor depends on measurement purpose of the mission. For instance, Google's autonomous vehicle use a LIDAR sensor as the primary sensor to assess the situation around the car when it is in autonomous driving mode. Conversely, the company Tesla combines information from multiple cameras, ultrasonic and Radar sensors in cars equipped with the autopilot system, because this technology can detect objects in complex environments such as snow, rain, and dust, while a LIDAR could have problems in these conditions.

The ultrasound systems have been proposed mostly for short-range detection and indoor navigation, avoiding collision with walls and near objects in the environment. This kind of environment information is stored and could be useful to generate maps, identify objects and, in general, to be fused with data from other sensors of the IoT platform to obtain a general view of the environment.

The acquired information is useful to some control functions of the UAV, for example, obstacle avoidance function. Obstacle avoidance algorithms is one of the main goals of vehicle autonomy research, and a good methodology requires software simulation to carry out tests without incurring risks for people and devices before having an operational solution. The hardware platform used in this work is PixHawk/PX4 and in this developed environment, Gazebo is a powerful 3D simulation environment for autonomous vehicles that is particularly suitable for testing object avoidance. Gazebo can be used with software in loop (SIL) and hardware in loop (HIL) design.

The validation has been carried out using incremental testing of the developed systems; the research group has several multirotor drones (a hexacopter and quadrocopters) mounted using individual pieces and PixHawk as a flight controller (Figure 1). Both drones could be handled manually from radio control and through missions defined by the QGroundControl tool. For experiments, all information from onboard sensors is collected for each mission and stored in a SD card incorporated in vehicle for data processing and visualizing.

This work is based on a real UAV that has been used to perform predefined flight missions. Predefined missions allow to integrate information obtained from the environment in the IoT ecosystem, UAV perform a predefined trajectory, and the information acquired is easily stored related to waypoints that the whole systems should surveillance. This UAV platform has been optimized for navigation in certain missions following the methodology proposed in [13] to select parameters and quality metrics. The current work is an extension to integrate and test the obstacle avoidance function based on a LIDAR sensors and running in the same hardware together with the navigation. This paper presents in section 2 the selected platform, design tools, and environment for real and simulation experimentation. Section 3 explains how LIDAR is modeled in Gazebo simulation environment, where real platform and environment is modeled. Section 4 explains the implementation of the integrated system with the main functions (navigation and obstacle avoidance) and how simulation is used to provide data input to the controller of the real UAV. The integration of available and developed modules makes this function possible, integrated as an embedded application which process the range data to control the navigation in hazardous situations, as illustrated in realistic simulated conditions presented in section 5. Finally, section 6 closes the paper with the main conclusions extracted from this work.

2 PixHawk and PX4 UAV frameworks

2.1 Architecture

PX4 is a powerful platform to implement UAV systems based on an open source autopilot [14] . It runs in the PixHawk flight

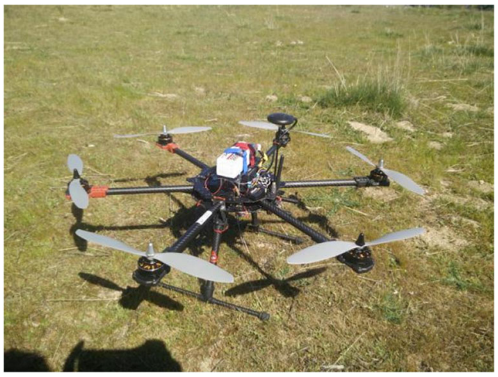


Fig. 1. Multirotor UAV available in GIAA research group

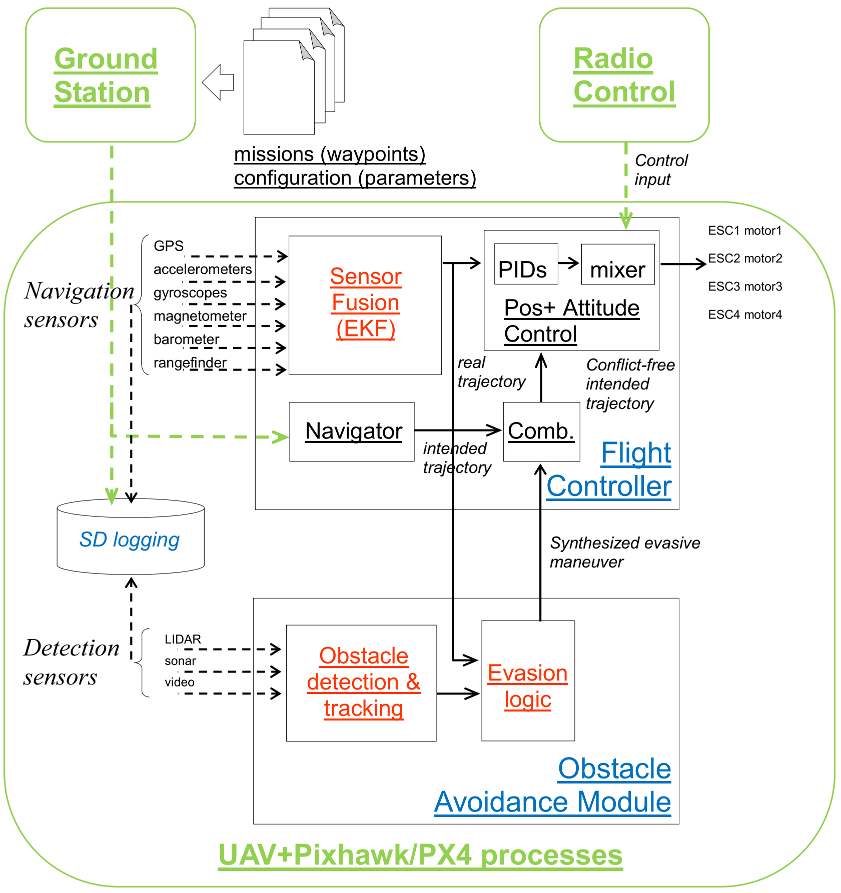


Fig. 2 Architecture of the integrated solution

controller, an open-hardware microprocessor designed to implement autopilot solutions, which integrates two boards, PX4FMU (flight management unit) and PX4IO (input/output). Among the integrated sensors, PixHawk contains

- L3GD20H gyroscope at 760 Hz
- LSM303D magnetometer at 100 Hz
- MPU-6000 accelerometer at 1000 Hz
- MS5611 barometer at 1000 Hz

The control software of PixHawk is PX4, a real-time operating system based on NuttX[1] and consists of two main layers: PX4 flight stack, with the embedded applications for UAV control, and PX4 middleware. This last layer is the interface that allows the flow of data from more sensors to applications through a publish/subscribe system called uORB, which makes the data coming from the sensors available to the applications of the flight stack. The outstanding modules in the solution presented in this work are the flight controller and sensor data processing [14]. Here, we propose an obstacle avoidance function implemented as a parallel application running together with flight stack. As shown in Fig. 2, the specific data processing working with real data will be working in a module embedded in PX4 flight stack. This obstacle avoidance module in real time receives the range sensor data to apply the obstacle detection logic. This logic, when triggered, computes an appropriate evasive maneuver to be inserted in the intended trajectory to guarantee the safety conditions of a vehicle and keeping low impact in the user-predefined intended mission. In our practical approach, this last step is done by suspending the operation of autopilot while the evasive maneuver is performed, to return the control when the detected hazardous condition has finished.

In the proposed solution, the navigation and mission management algorithms running in the available PX4 software will interact with the module developed for conflict detection and avoidance. The flight control module is the core where the navigation solution is computed based on the extended Kalman filter (EKF) sensor fusion algorithms. Specifically, it contains three sensor data processing blocks:

- An AHRS (attitude and heading reference system) to estimate the vehicle attitude and creates a direction vector used as reference for vehicle displacement. It takes as input the triaxial accelerometers and gyroscopes data to obtain the attitude, usually represented as direction cosine

[1] NuttX Real-Time Operating System, http://nuttx.org/

matrix (DCM) or the orientation parameters of the vehicle (roll, pitch, yaw angles)

- An inertial navigation system (INS) to compute the trajectories and corrections when vehicle moves between single points using the DCM data and accelerometers input integrated to compute the vehicle position and kinematics.
- Several EKF algorithms are available to fuse sensor data in a function that exploits the specific noise and accuracy characterization of each sensor to estimate the vehicle navigation parameters. There are three different modes of EKF operation.

- EKF1: Only use the DCM for attitude control and the inertial navigation for position prediction (dead reckoning)
- EKF2: Use the GPS for 3D velocity and position. The GPS altitude can be used if barometer data is very noisy.
- EKF3: If there is no GPS, it can use optical flow to estimate 3D velocity and position.

In any case, an EKF estimates the position, velocity, and angular orientation of the vehicle; its output is compared with the intended trajectory provided by the navigator based on the missions sent from ground station. Then, a control loop based on proportional-integral-derivative (PID) modules is applied and finally mixed with the control inputs from radio control (RC) to generate the individual inputs sent to each electronic speed controller (ESC) regulating the power sent to each engine in the multirotor.

The dynamics of multicopter platforms have been recently analyzed by aeronautical and control communities **Error! Reference source not found.Error! Reference source not found.**. The vehicle control is a six degree-of-freedom system (position and orientation, pose), while the main actuation points (control output) are total upward thrust force and the moments on roll, pitch, and yaw axes. The performance and adjustment of sensor fusion algorithms was addressed by the authors in a previous work [13], with appropriate analysis based on real data and impact of the EKF parameters used in this module on the final navigation accuracy. As mentioned, this work explores the possibility of extending data processing including a LIDAR sensor in order to implement obstacle detection and avoidance function, working in parallel with the flight management unit in the same PixHawk board, sending appropriate commands to modify the flight, and evading the detected hazardous situation.

The process developed will be running in parallel with the flight controller, allocated in PX4 flight stack. As mentioned, flight stack contains the applications embedded in hardware for drone control, while PX4 middleware is the interface that allows the flow of data from sensors to applications or messages from companion computers. The internal communications among PX4 modules are managed through a publish/subscribe system called uORB. uORB allows to publish the data coming from the sensors and make them available to the applications of the flight stack, obtaining a reactive system and totally parallelized. Besides, uORB allows the interchange of messages among processes running in flight stack.

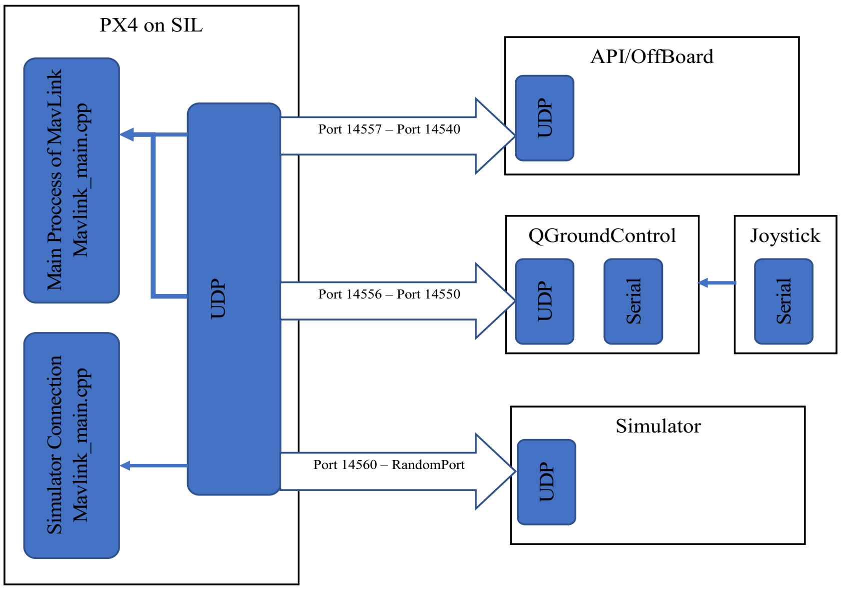
2.2 Ground control system (QGround)

A fundamental aspect to consider in the design of the whole system is the ground controller station. Despite a variety of UAV, ground controllers can be listed, the compatibility to PixHawk limits the options mainly to Mission Planner or QGroundControl. QGroundControl is a powerful flight control system with a complete interface integrated with the firmware of the PX4. A basic functionality provided by the ground station is the configuration and calibration functionality for different drone elements, as well as the radio control systems that conduct the mission manually. Another important functionality of this controller is the creation of different flight missions using waypoints. QGroundControl also facilitates the interaction between different flight modes that the MAVLink communication protocol provides, since QGroundControl is a detailed graphical interface to interact with the flight controller through the MAVLink communication protocol. The main flight modes are

- Position mode. Control mode through the RC transmitter that automatically stabilizes the drone in a fixed position compensating the wind and other forces.
- Manual/Stabilized mode. In this mode the aircraft is stabilized when the RC control sticks are centered. Manual flight is operated, moving the sticks in the desired direction.
- Return to launch mode. The RTL flight mode returns the drone to its initial position.
- Hold mode. The hold flight mode (a.k.a. “Loiter”) causes the vehicle to stop and maintain its current GPS position and altitude.
- Mission mode. In mission mode, the drone will follow a programed flight mission, stored in the flight controller, which is composed of navigation commands or waypoints.
- Follow-Me mode. Follow-me mode allows a multicopter to follow autonomously a user with an Android phone or tablet with the QGroundControl application installed and configured.
- Offboard mode. The vehicle follows the commands provided by MAVLink communication protocol using an external computer connected to QGroundControl.

Finally, QGroundControl includes an API to the Gazebo simulator and allows the execution of missions through “flight routes creation” module in order to replicate them using the simulator.

Fig. 3 Simulator structure/PX4



2.3 Communications

As mentioned above, there are two main communication modes in PX4, MAVlink, and uORB. uORB oversees the internal communication of PX of processes running in the same execution space, while MAVlink has in charge external communications between PX4 and other systems such as QGroundControl.

uORB is an asynchronous communication system based on the publication and subscription of messages for communication between processes or threads. uORB allows the creation of new messages in a simple way, but in this work, only predefined messages will be used within PX4 [14]. Subscription to a set of messages will be necessary for the treatment of the information received by the sensor, and other parameters such as the mode of operation of the drone or the current position of the drone. Only the publication of commands defining the action that needs to perform to evade obstacles will be used.

MAVlink is a communication protocol specially developed for the exchange of information between ground control stations and flight controllers integrated in UAVs. It contains the necessary information to carry out the interaction between the different controllers that make up the system. For instance, the waypoints transmission protocol describes the way in which waypoints that define the mission are communicated from ground station to drone. These waypoints are the reference points that indicate to the flight controller the path to follow, the height, and the speed. To request the waypoints, the drone sends the message "MISSION_REQUEST (WP)" to the ground station, which must respond by sending the respective waypoints to complete the mission. Another protocol is related to system parameters that determine the situation of the aircraft, orientation, GPS positioning, speed, etc. Finally, MAVLink is the protocol that allows communication between external processes to PX4 to manage the mission.

2.4 Simulation facilities in PX4: SIL and HIL design

The hardware architecture has the possibility of connecting with simulators to analyze the final software embedded in UAVs. Simulation allows to test the software in "realistic" situations avoiding the inherent risk to real flights when experimental analysis is needed to validate a new feature. Among the available simulation tools integrated with PX4 software, Gazebo is highly recommended because it provides powerful 3D simulation ecosystem, versatility, and pre-existing available libraries of models. For instance, it supports a variety of models of vehicles such as quadrotors (Iris and Solo models), airplanes, Rover, etc. These two features and the easy interaction with elements such as ROS, MAVLink, and QGroundControl make it a competitive candidate to be used as the main simulation model in this work. Other existing alternatives are

- jMAVSim, an exclusive multirotor simulator characterized by its lightness, easy to configure, and allows to check if the aircraft takes off, turns, and lands properly.
- AirSim, a simulator that provides simulations of physics and a visualization very close to the real world, it consumes a large amount of resources intensively.

Gazebo has been used as simulation environment making it possible to debug navigation and object avoidance algorithms integrated in the PixHawk flight controller. In addition, Gazebo offers various models of real autonomous vehicles, saving modeling time, and focusing in the objective of this work; the model of range sensor was tested to implement obstacle avoidance. PX4 is prepared to run SIL/HIL simulations to reproduce the desired design conditions in realistic situations. This step allows to check new mission, control algorithms, or important changes in configuration before flying the quadrotor. This work uses SIL for PX4 and interact through Gazebo and QGroundControl. Software in the loop is a system simulation where systems are modeled and run under software without any hardware. To develop a control software, usually, the system will have to pass simulation and verification tests before the software is implemented into a real system. In this work, software in the Loop (STIL) tests will be used. Figure 3 shows the architecture integrating SITL in PX4 system [14].

In Fig. 3, the communication between the simulator and PX4 is performed through MAVlink, specifically through port 14,560 with user datagram protocol (UDP). The simulator sends the information from sensors to a dedicated PX4 module, and this module responds with the information of actuators in order to make simulator execute the corresponding movements of the vehicle in the simulation domain accordingly to the implemented control solution. Different airframes can be selected to run the simulation, i.e., multirotors (with/without optical flow sensors) and planes. In the case of quadrotors, the model taken in this work to test the developed architecture is the Iris model developed by 3D Robotics [14].

Alternatively, in the case of PixHawk, a hardware in the loop (HITL) configuration allows executing directly the code written in the flight controller before being connecting to sensors in the real environment. That fact allows a first contact in the study of the effect of parameters of navigation and obstacle avoidance algorithms. In this case, the JMAVSim or Gazebo (running on a development computer) is connected to the flight controller hardware via USB/UART. The simulator acts as a gateway to share MAVLink data between PX4 and QGroundControl. The simulator is connected to QGroundControl via UDP and bridges its MAVLink messages to PX4.

Therefore, SITL runs on a development computer in a simulated environment and uses firmware specifically generated for that environment with simulation drivers providing synthesized environmental data to analyze the system reaction. Alternatively, HITL runs the final PX4 firmware in "HITL mode," on normal hardware in final physical PixHawk board. The simulation data enters the system at a different point than for SITL, with appropriate physical IO interfaces with the board. In summary, HITL runs PX4 on the current hardware using standard firmware, but SITL actually executes more of the standard system code.

2.5 Gazebo environment and integration with PixHawk

Gazebo is a powerful 3D simulation environment for autonomous robots that is particularly suitable for testing object avoidance strategies. Some of the available models include quadrotors (Iris and Solo), hexacopters (Typhoon H480), generic plane, Rover, and submarine, etc. In this work we have selected the Iris quad model because it has the minimum needed set of sensors for navigation and performing missions (GPS, INS, magnetometer) as will be commented later.

The simulation environments in Gazebo are composed of several files programed in the SDF (simulation description format) language. This format is based on a complete description of all the elements that form the simulation from the simulated world to the robot or vehicle itself and the sensors. In addition, it is structured in XML format for easy adding and removing of elements. These files are the following:

- Files with .world extension. This file is called when you start the simulation with a specific model. In this work the drone model 3RD Iris and the LIDAR sensor are defined in this file.
- Files with .sdf extension. This file contains the logic programing for models that will be executed in the simulation using the .world file.

Besides, Gazebo uses plugins. A plugin is a fragment of code that is compiled as a shared library and inserted into the simulation. Plugins are the means in this environment used to add sensors to the model and simulate them. These sensors can be of several types, i.e., odometry, force, ultrasound, and lasers sensors. These behave similar to SDF files, but are compiled statically in Gazebo, giving additional flexibility to users to choose the desired functionality to include in simulations. Regarding the types of plugin, there are six types, namely, world, model, sensor, system, visual, and GUI. Each type of plugin is used for functionality, for example, a plugin of the world type is used to control the properties of the world, such as the physical engine and ambient light. A plugin of model type is used to control the connections and the state of a model, and the sensor type is used to acquire the information from the environment and control the behavior of the sensor.

From the models of aircraft that simulator provides, in this work, the 3DR Iris model is used because, as mentioned above, the sensors are integrated into the model. Besides, this model is part of the existing aircraft in the model repository including in the PX4 firmware, so it is compatible with the software of the flight controller. Drone 3RD Iris is provided by Gazebo in the library with the corresponding .sdf file for this model as illustrated in Annex 1. With the available basic definition of vehicle, the simulated range sensor is built to be collocated with vehicle, and its reference position and orientation. In next section, the development of full model is commented to simulate the detections, considering as a reference commercial model.

3 Proposal of LIDAR simulation in Gazebo environment

As indicated in the previous section, the role of simulation with UAVs is important for a fast and effective verification of the developed system including communications and integration of sensors in the PX4, paying attention to the integrity of vehicle and sensors before testing the system in real conditions. An essential part of this work lies in the ability to interact with the environment that surrounds the drone using sensors and communication established via MAVLink. The design is focused in enabling the flight controller to make decisions and manage the drone's trajectory during the execution of the mission to avoid obstacles or return to a safe area taking control of flight controller. For this goal, a LIDAR sensor is selected as the main sensor of the simulation, since this type of sensors is common in the real world for the good trade-off between detection capability and payload, which facilitates the integration with PX4 PixHawk flight controller, in addition to allow drone to perform a quite accurate and especially fast obstacle detection.

The drone model selected for the simulation has integrated an RPLIDAR sensor, which is a low-cost LIDAR version that provides a 360° swept field.

3.1 Data model of LIDAR in Gazebo

A LIDAR sensor has been simulated in Gazebo using a data model based on a sensor in the market, a model with a good compromise between reliability and payload. The chosen LIDAR model is a light one, the Hokuyo URG-04LX, which generates a vector of radial distances with a 240° vision interval. The characteristics are

Laser class 1. $\lambda = 785$ nm
Range, 20 to 10,000 mm
Accuracy, from 60 to 1000 mm; ± 30 mm
from 1000 to 10,000 mm; ± 3%
Scan time, 100 ms
Power, 5VDC + − 5%
Weight, 160 g

The rotation frequency was modified to be equal to LIDAR used in real vehicles, and the noise was adapted to be similar as possible to the real sensors.

The system developed in this work integrates the 3RD Iris model of the real UAV previously use to execute missions (Fig. 4). From this point, the proposal develops a 2D LIDAR sensor simulation that is mounted on the top to evaluate its capacity for detecting obstacles.

The model of sensor, moving on the top of quadrotor, is indicated in Annex 1. In this work, the focus is in the area of subscription to the data flow (specifically in the section of the laser) to obtain the data that the LIDAR sensor coupled to the 3RD Iris model will be collecting during the flight. This sensor is aimed at detecting obstacles when a mission is running. LIDAR will be the basis for the development of applications that, using the data obtained by the sensor when an obstacle is detected, allow to establish a connection with the flight controller by means of a series of MAVLink messages in order to modify the trajectory to evade this obstacle.

An important aspect in the model of sensor, in order to develop the control system, is the indication of the attitude of sensor for each returned datum (range and angle). The LIDAR is moving with the drone, taking into account the position of the upper part where the laser sensor is located. The system obtains the information of vehicle position expressed in quaternions, which are an extension of real numbers usually employed for representation of 3D orientation of solid bodies in aerospace

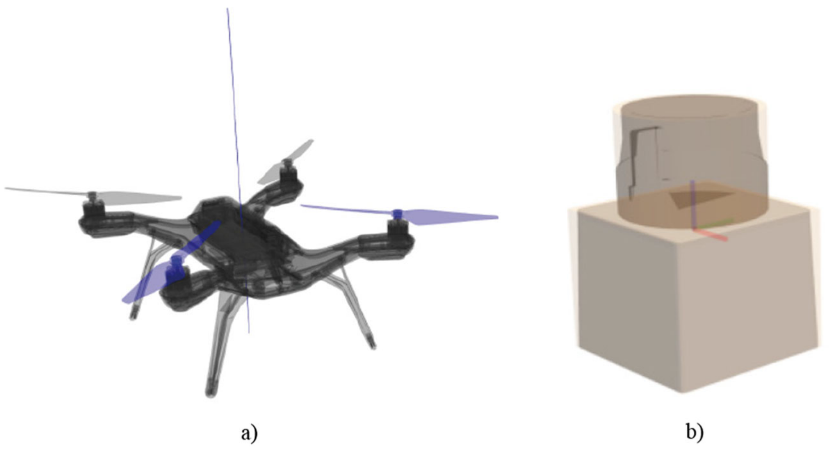


Fig. 4 Models implemented in Gazebo. A) a Iris quadrotor. b)b lidar LIDAR 2D scanner

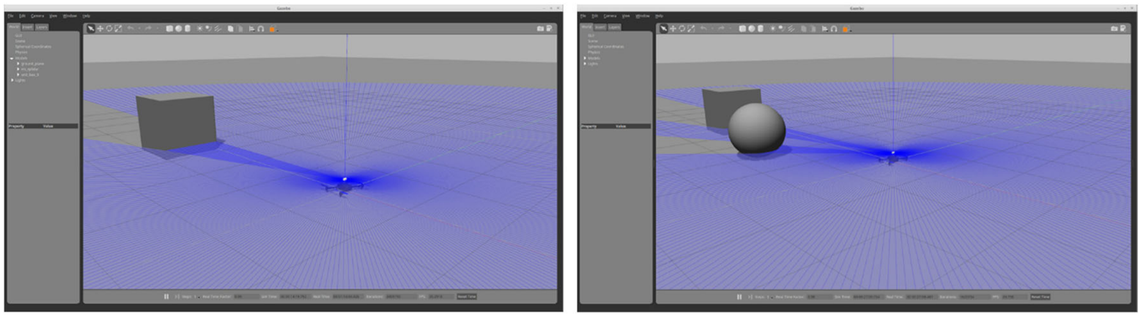


Fig. 5 2D LIDAR simulation on Gazebo with one and two obstacles

applications, in which both the position and attitude are required [15, 16].

This representation is based on a point, which will normally be the base of the robot. In this case, the reference of the sensor is taken with respect to the base of the drone. From the quaternion, the angles of Euler can be obtained, which is composed of three angles that define the rotation of a triangle with respect to a reference point [17]. The quaternions are represented as follows: q = [q_0, q_1, q_2, q_3]. In order to obtain the Euler angles, the following equation is used:

$$\begin{pmatrix} \text{roll} \\ \text{pitch} \\ \text{yaw} \end{pmatrix} = \begin{pmatrix} arctan \dfrac{2(q_0q_1 + q_2q_3)}{1{-}2\left(q_1^2 + q_2^2\right)} \\ arcsin2(q_0q_2{-}q_1q_3) \\ arctan \dfrac{2(q_0q_3 + q_1q_2)}{1{-}2\left(q_2^2 + q_3^2\right)} \end{pmatrix} \quad (1)$$

After obtaining the angle and distance to which the obstacle is located, this information is encapsulated in the predefined MAVlink message and it is sent to the PX4. After analyzing the information received in PX4, MAVlink frequency is not enough to send the distance sensor information correctly.

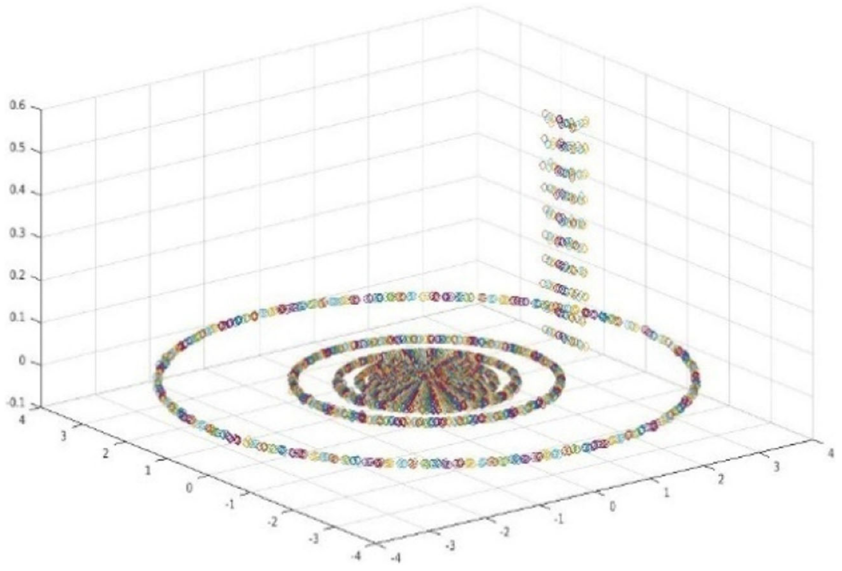
In order to validate the 2D LIDAR model, some static trials have been performed with the vehicle in a fixed position on ground and one or two obstacles as displayed in Fig. 5.

3.2 Proposal of LIDAR simulation

Before final simulation, data received should be processed in order to analyze data corresponding to the ground that were not of interest to the system, regardless of the height of the sensor. For this reason, the ground was characterized with the objective of filtering it, prior to the application of the algorithms. A simulation without objects was carried out to characterize the ground, obtaining the average and the standard deviation of the measurement for 32 rays.

After analyzing this data, the data seemed to follow a normal distribution where the values closest to the average in all the iterations corresponded to ground data. To ensure a filtering of 99.7% of the data referring to the ground, the data included in the interval [$\mu - 3\sigma$, $\mu + 3\sigma$] were eliminated for

Fig. 6 Removal of ground with LIDAR simulation on Gazebo



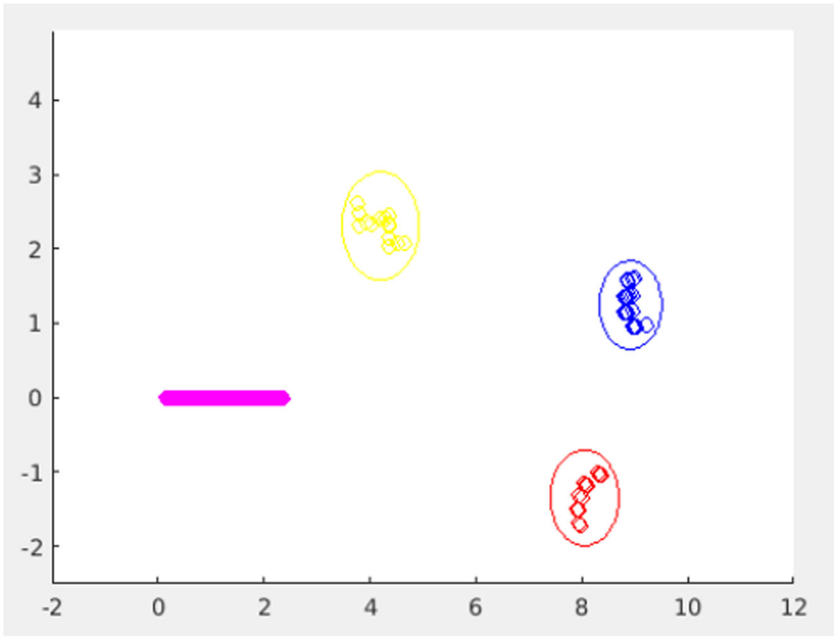


Fig. 7 Illustration of clustering algorithm on LIDAR simulated data

each ray. This method allows to filter quickly and effectively the data that did not provide information on the scenario.

After filtering the ground (Fig. 6), the next step is to generate clusters from the data collected in Gazebo to detect and differentiate objects in the different scenarios. The chosen environment to perform this task is MATLAB (where a previously generated module for fusion of data could be reused). First, a script is generated that divides all the data stored in iterations. An iteration is set as the data collected by all the light rays of the LIDAR in 360 degrees. For each iteration, a cluster algorithm is applied that detects the objects of this iteration and sends them to the fusion module mentioned above that will be responsible for associating the new clusters with those of previous iterations or creating a new object, if not association is possible (Fig. 7). Finally, to verify the efficiency of the implemented clustering algorithm, a module is generated that allows all the iterations to be visualized dynamically, making a tracking of the vehicle using the ground truth that provides the Gazebo data.

This process was carried out in “off-line” mode, processing the simulated data and logged in a data file. The MATLAB environment can be connected to the simulation and provide the results dynamically as Gazebo generates the data. To do that, the Robotics System Toolbox can be used to directly receive the data generated by Gazebo simulator, an example can be viewed in [18], in which the Gazebo simulator and Simulink time are synchronized so that the Gazebo sensor data is visualized using MATLAB function block and MATLAB plotting functionalities. Even more, the results of a process running in MATLAB can be directly sent back to the PX4 processor with the available functionalities to support communications with MAVlink protocol [19]. The details of co-simulation are beyond current work, but it is useful to show possibilities that can be developed in a future work to analyze solutions implemented with co-processors running on-board the UAV and connected using MAVlink.

The selected clustering algorithm is determined by the need of a method that calculates the number of clusters to be detected. Several works have been developed [20, 21], and the two major algorithms used for problems of this type are density-based spatial clustering of applications with noise (DBSCAN) and Gaussian mixtures. The DBSCAN algorithm is a density-based cluster method that allows dense regions to be detected in space of the problem and separated when it detects areas with no or little density data. These sparse areas are detected as noise by the algorithm, which is a novelty within cluster methods that generally classifies all data within a group.

4 Proposal of a general method for implementation of new sensors on board

This section presents the synthesis of the developed elements in order to reach the final goal: to create a system that, through the data obtained by the sensor, is able to detect the presence of obstacles interposed in the flight mission and react before they are too close by sending a series of MAVLink commands to interact with the flight controller and manage the trajectory to carry out evasive maneuvers. Once the drone encounters an unknown element in the execution of the mission, the obstacle detection application will be activated. Simulators that currently have support under PixHawk allow the flight controller’s flight code to manage a computer-modeled vehicle in a simulated “world.” Interaction with this vehicle is similar to

Table 1 Messages for flight mode selection and mission parameters

MAV_CMD_DO_SET_MODE	Set system mode
Mission parameter #1	Mode, as defined by ENUM MAV_MODE
Mission parameter #2	Custom mode—system-specific, depending on particular autopilot specifications
Mission parameter #3	Custom sub mode—system-specific, depending on particular autopilot specifications
Mission parameter #4	Empty
Mission parameter #5	Empty
Mission parameter #6	Empty
Mission parameter #7	Empty

interaction with a real vehicle, using the ground controller which communicates with the flight controller using the MAVLink API integrated in the simulator. This API defines a set of MAVLink messages that supply data from simulated world sensors to the flight controller and return the values of the motor and the different actuators that will be applied to the control of the simulated vehicle.

4.1 Architecture

In the following diagram, the different components are shown in a typical SITL simulation environment for any of the simulators compatible with the air traffic controller. The different parts of the system are connected through UDP and can be run on the same computer or another computer on the same network.

Figure 8 shows how the sensor module added in the simulator interacts with the system. A part of the developed module is on the vehicle and, in turn, the vehicle is part of a world in which we have all the modules of the simulation scenario and other modules such as the communication through MAVlink. As the communication module and the sensor module are in the same world, they can communicate with each other to send the information to PX4 via MAVlink with the sensor message already predefined. The PX4 controller, integrated with the obstacle avoidance functionality developed, computes the control outputs sent to vehicle model in the simulation.

4.2 Communications with PX4: MAVLink messages from Gazebo sensor and and uORB between modules

As indicated, in PX4 there are two main communication modes, MAVlink and uORB. uORB is responsible for the internal communication of PX4, within the same execution space, while MAVlink is responsible for external communications between PX4 and other systems such as the ground control station or other companion computers, if installed. In this paper, the execution of the obstacle detection process is a secondary task in PX4 flight stack, using uORB within the system itself to communicate processors. In the case of having needed more resources for the execution of the module, it could have been executed in an external hardware and send the commands through MAVlink. Thus, an alternative would be to process the data in another microprocessor, for example, a Raspberry Pi and communicate with PX4 through MAVlink.

Therefore, the objective is establishing a communication with the flight controller to modify the mission accordingly to the conditions through a dedicated process running in

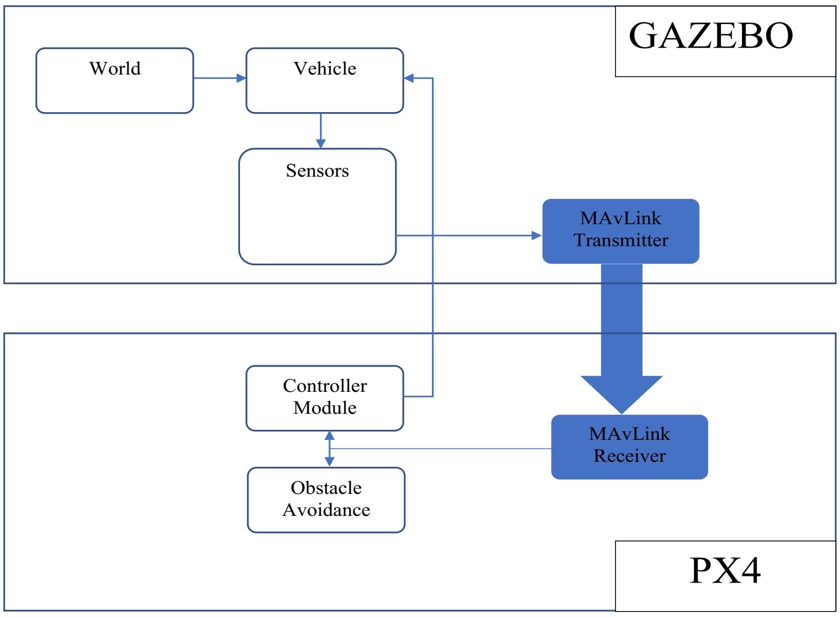


Fig. 8 Integration of Gazebo simulator Gazebo with PX4 and obstacle avoidance

parallel to the navigation. The command messages allow to greatly simplify the programing of the application. The general process is to select the flight mode and the specific parameters for that mode, as indicated in the following tables (Tables 1, 2) [22].

This capacity to select flight modes allows us to use this communication together with the data obtained from the sensors, to make a system that modifies the flight path when a mission is running in autonomous mode, providing commands that insert new waypoints into the mission to evade obstacles. This is implemented using the pause command, which makes the vehicle hold the current position or continue, depending on first parameter is 0 (pause current position) or 1 (continue). Besides, if this command contains coordinates and height as additional parameters, the drone will go to these coordinates and height, and stop. The height will be the same as the drone because the sensor can only detect objects in the same height.

4.3 Obstacle avoidance

Although it is not the main objective of this project, a functional detection system and obstacle avoidance module [23, 24] has been developed to check in detail the integration of simulated 2D LIDAR sensor data in the system, consisting of three main phases:

- No danger. In this phase, the measured distance is checked and compared with the safety distance. If the distance is smaller, then the system will enter into danger phase.
- Danger. This phase is created to avoid false readings or when reading is the distance to the ground. In the last case, if the projected distance is similar to the estimated height, with a margin of error, the obstacle is discarded, using basic geometry to estimate height as $h = \cos \alpha \times$, where h is the height and α is the pitch angle.
- Avoidance. To avoid possible obstacles, the command "insertion of a new waypoint" will be used, taking as parameter coordinates and height to insert in the mission.

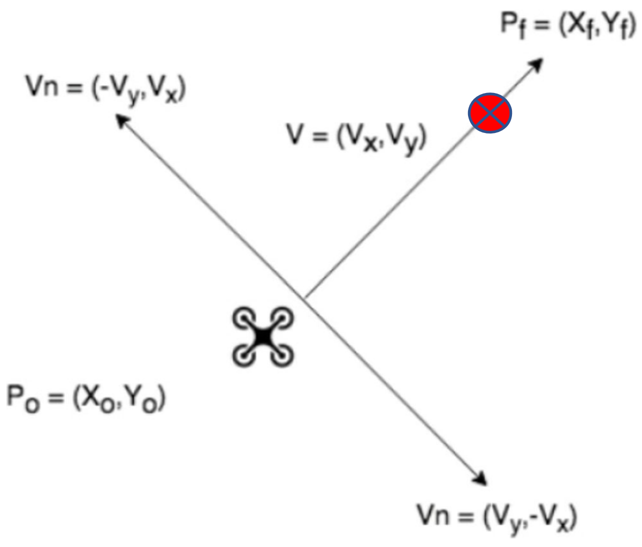


Fig. 9 Geometry of obstacle avoidance solution

Two new coordinates are computed in the line perpendicular to the next waypoint of the mission. Once the drone reaches the new position for collision avoidance, the drone will continue the original mission, returning the safety system to the non-danger state. The new coordinates are computed with direct geometric calculations. A normal vector to velocity is used to define the inserted waypoint (Fig. 9).

Table 2 Available modes with PX4

Value	Field name	Description
0	MAV_MODE_PREFLIGHT	System not ready to fly, booting, calibration, etc.
80	MAV_MODE_STABILIZE_DISARMED	System allowed to be active, under assisted RC control, disarmed
208	MAV_MODE_STABILIZE_ARMED	System allowed to be active, under assisted RC control, armed
64	MAV_MODE_MANUAL_DISARMED	System allowed to be active, under manual (RC) control, no stabilization, disarmed
192	MAV_MODE_MANUAL_ARMED	System allowed to be active, under manual (RC) control, no stabilization, armed
88	MAV_MODE_GUIDED_DISARMED	System allowed to be active, under autonomous control, manual setpoint, disarmed
216	MAV_MODE_GUIDED_ARMED	System allowed to be active, under autonomous control, manual setpoint, armed
92	MAV_MODE_AUTO_DISARMED	System allowed to be active, under autonomous control and navigation, disarmed
220	MAV_MODE_AUTO_ARMED	System allowed to be active, under autonomous control and navigation, armed

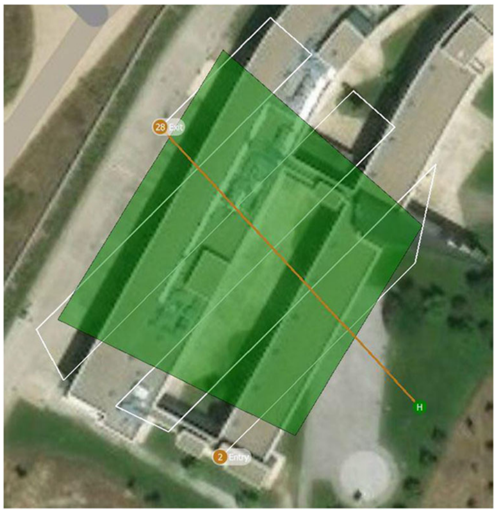


Fig. 10 Prepared mission with surveillance pattern 1

So, the computed waypoint is

$$\overline{P_n} = \overline{P_0} \pm \text{SD} \times \overline{v}_n \tag{2}$$

being SD (safety distance), a configuration parameter mentioned above. To decide the sign of the avoidance maneuver (whether turn to the left or right), left and right sectors of sensor readings are analyzed. If there is no obstacle at the left, the drone will turn to this side; conversely, if there is an obstacle, the drone will look at possible dangers on the right side and, if there is no danger, turn on the right, but if there is danger turn 180° to the back part of the drone and repeat the process. If for some reason the drone is in danger in all directions, it will automatically execute the landing command.

This is a simple and direct conflict avoidance logic implemented in the PX4 to show the capability to execute a safety process in parallel with the FMU logic. A more advanced collision avoidance strategy will be studied in future works, considering heuristics or convex optimization techniques as shown in [25].

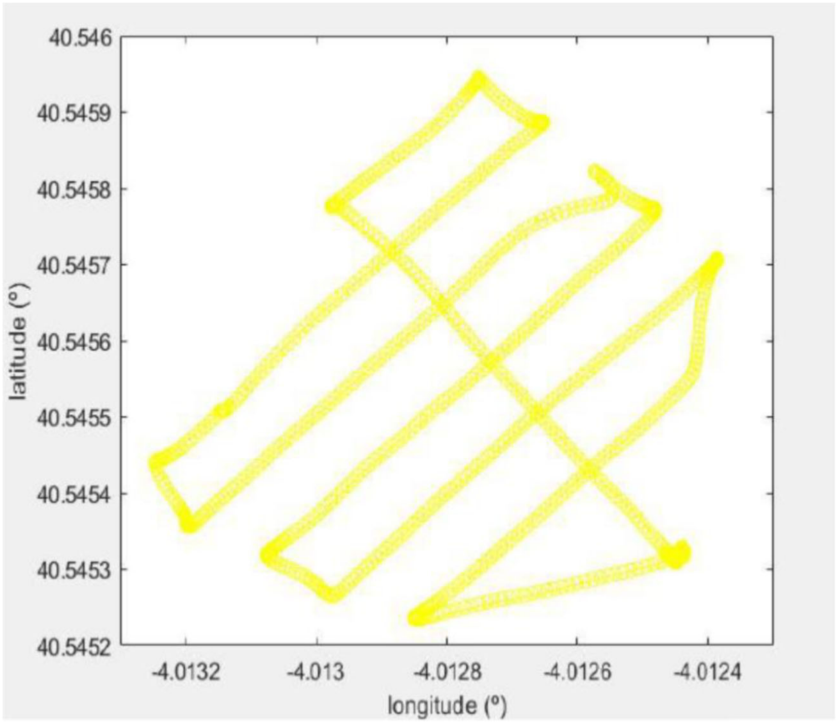
5 Evaluation in realistic conditions

The methodology proposed in [13] was designed to assess the navigation system performance under realistic illustrative conditions. The first step of this methodology is the definition of the UAV platform, the type, cinematic characteristics, set of sensors, and the navigation algorithms with associated parameters. This section illustrates the whole system working with the implemented functions (navigation and obstacle avoidance). In the first place, the navigation function is illustrated together with data logging from an altimeter sensor with a real flight to build height maps in a certain area. Secondly, the obstacle avoidance integrated with navigation is shown in a simulated environment with the sensor and vehicle models running in Gazebo.

5.1 GPS and INS data fusion for navigation and height data collection

The data fusion process of PixHawk (EKF2 filter) provides the orientation of the vehicle (attitude) integrating data from

Fig. 11 Recorded position of drone (computed by navigation function)



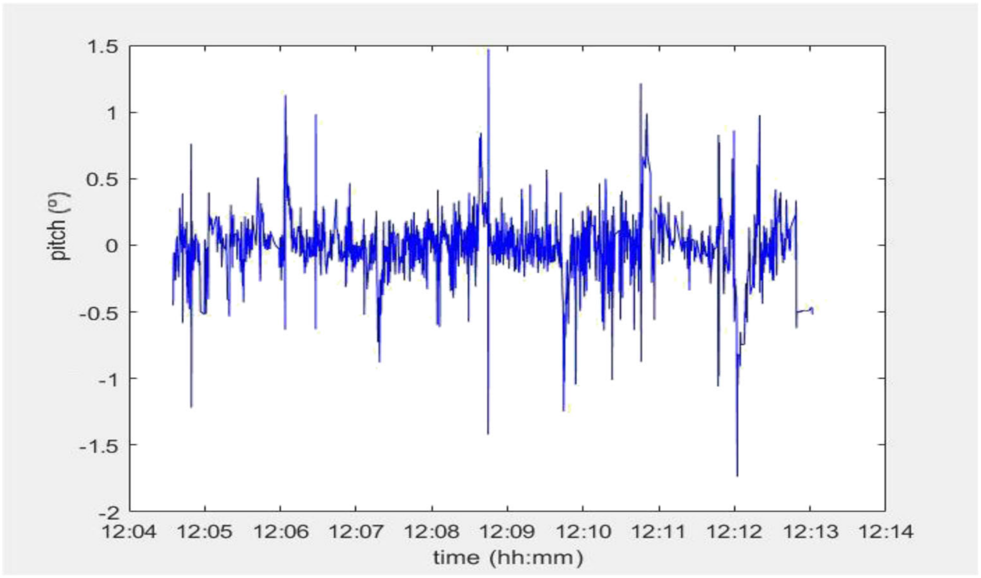


Fig. 12 Recorded pitch angle of drone (computed by navigation function)

magnetometer, gyroscope, and accelerometer sensors, and then fuses the result with accelerometers and GPS sensors to estimate position and velocity. So, it is a loosely coupled fusion architecture which uses GPS position and velocity measurements based on INS inputs [26, 27]. The estimated state vector resulting from this multi-sensor fusion contains the attitude vector (represented with a quaternion), 3D position and velocity, and the bias terms for gyro and accelerometer data. The EKF fuses the available sensor inputs to carry out the predictions between GPS observations. It considers the noise affecting to all measurements in the estimation of attitude and cinematic parameters to increase the accuracy and consistency of the output, even when the vehicle losses the GPS signal in certain time intervals. The PX4 system counts with several EKF algorithms to process all sensor data with a logic which depends on the specific noise, availability, and accuracy characterization of each sensor, switching among different EKF solutions running in parallel:

- EKF1. Only use the DCM for attitude control and the inertial navigation for AHRS reckoning for position control.
- EKF2. Use the GPS for 3D velocity and position. The GPS altitude could be used if barometer data is very noisy.
- EKF3. If there is no GPS, it can use optical flow to estimate 3D velocity and position.

Therefore, the navigation in PX4 is able to select the best EKF mode for each situation and execute both EKF1 and EKF2 in parallel if it is necessary. Some illustrative results of this function and logged data were explained in [13]. The experimental flights were carried out with auto-pilot mode supervised by human controller to change the mode of flight if

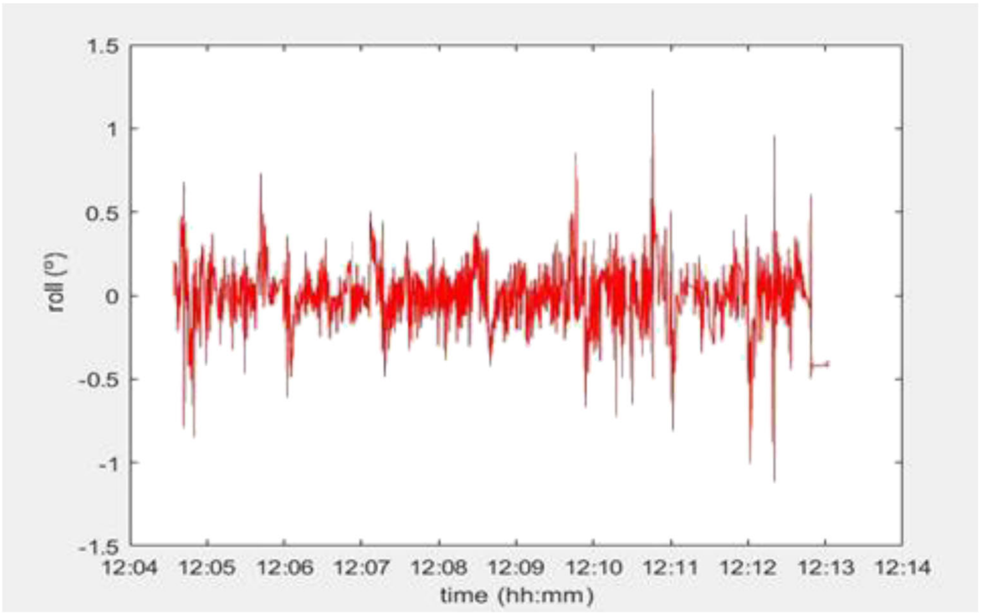


Fig. 13 Recorded roll angle of drone (computed by navigation function)

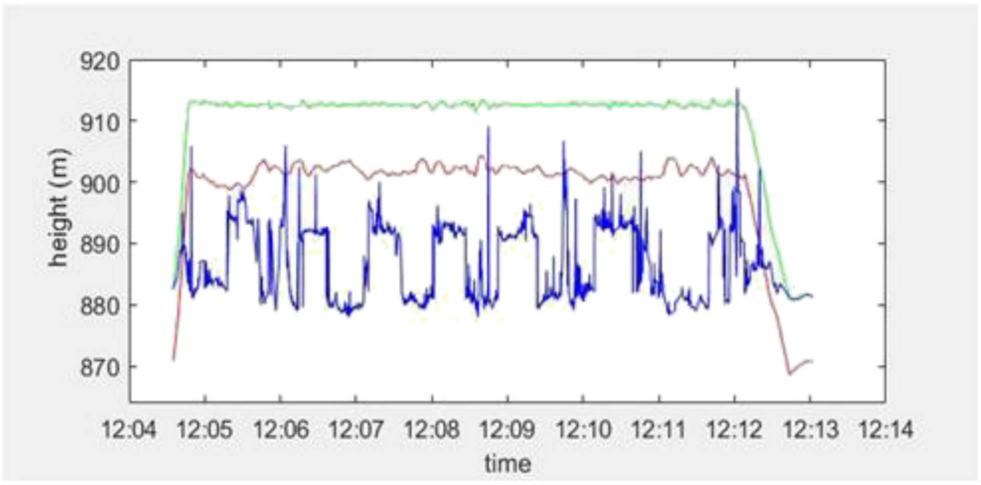


Fig. 14 Recorded heights provided by barometer, GPSS, and altimeter

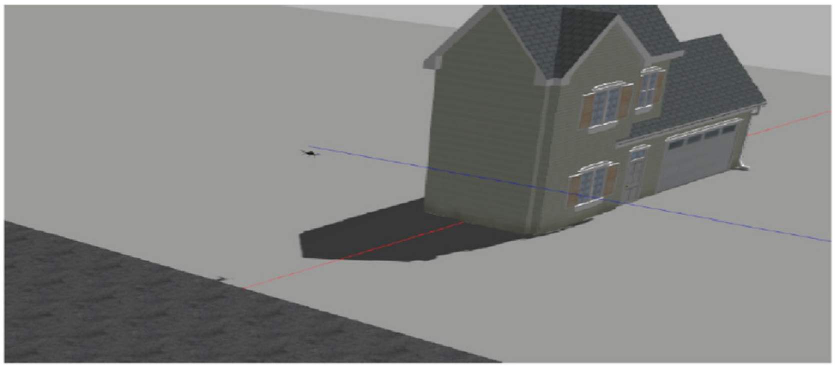


Fig. 15 Simulation of a house as obstacle in Gazebo

needed. The predefined trajectory of a mission contains a sequence of waypoints with the positions defining the path together with the associated velocities at each position or hold time if the speed is predefined to zero. The flight controller takes waypoints as references to control the vehicle and follow the desired path, using the set of points in the trajectory as "fly-by" waypoints, i.e., intersections of straight lines where the vehicle is expected to perform transitions with turns that "flies by" the waypoints.

In the performed mission, a feature of the ground controller was used to execute "programmed" missions, the "pattern" tool available in QGround. This feature is useful for routinary tasks (surveillance, data collection, training, etc.) and was applied to log the integrated rangefinder sensor to build maps of terrain and buildings in the area of interest. As shown in Fig. 10, the pattern feature creates a pattern over the specified polygonal area so that the flight executes settings appropriate for creating the dataset with height data. We illustrated this function with a pattern going above a building of the university to test the altimeter functionality.

The results can be seen in the following figures. First, the 2D locations provided by sensor fusion are shown in Fig. 11, while the attitude (pitch, roll) can be seen in Figs. 12 and 13. These values were recovered from logged output of navigation function after fusion of GPS and INS data.

The mission integrated the LIDAR-Lite sensor, connected to PixHawk with I2C and QGroundControl configured to keep it active and log the data (the parameter SENS_EN_LL40LS activates the reading from port associated to connector I2C).[2] The sensor was mounted pointing to ground, with a maximum range of 40 m. It provides the distance measured with respect to the ground, which must be corrected to compute the height considering the attitude of vehicle to correct the deviations with respect to the vertical line:

$$corrected\ height = distance\ \ \cos\alpha \cos\beta \tag{3}$$

Being α, β, the roll and pitch angles estimated from INS by navigation function. As can be seen in Fig. 14, the computed height is consistent with the other available sources integrated in the navigation function. Barometric height is depicted in green while GPS is in red, the difference between then is due to barometric calibration, corrected with GPS reference. The distance to ground is the blue line, which reflects the height of the building (around 10 m) with jumps appearing each time the UAV flew over the building in the predefined pattern.

[2] http://docs.px4.io/en/sensor/LIDAR_lite.html

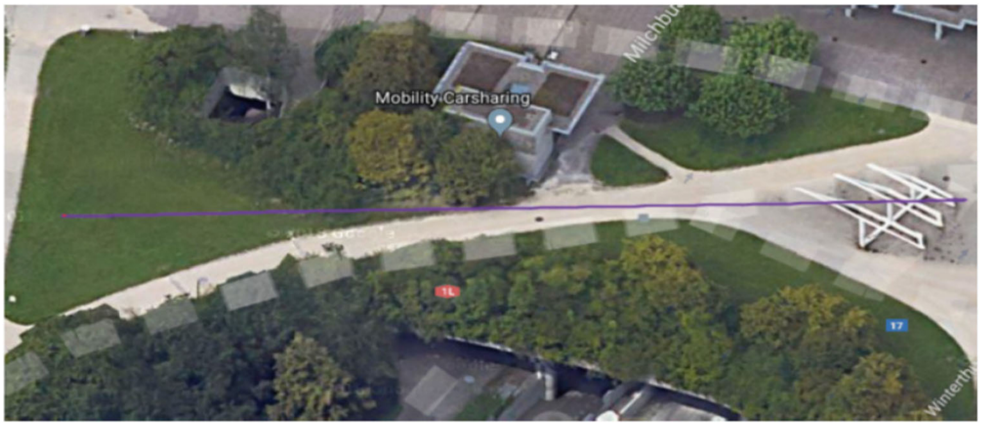


Fig. 16 Free mission

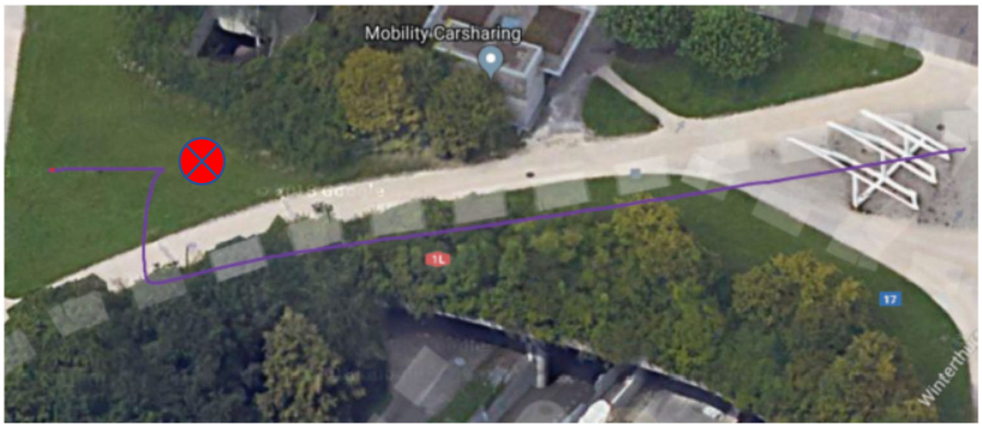


Fig. 17 Mission with the simulated obstacle in trajectory

5.2 Integration of obstacle avoidance and validation

Finally, the simulation framework was used in order to obtain results with all components integrated in the developed system. Besides illustrating collision avoidance, an additional objective of this section is the verification of fully integrated modules by execution of missions specified with the ground station. It shows the appropriate

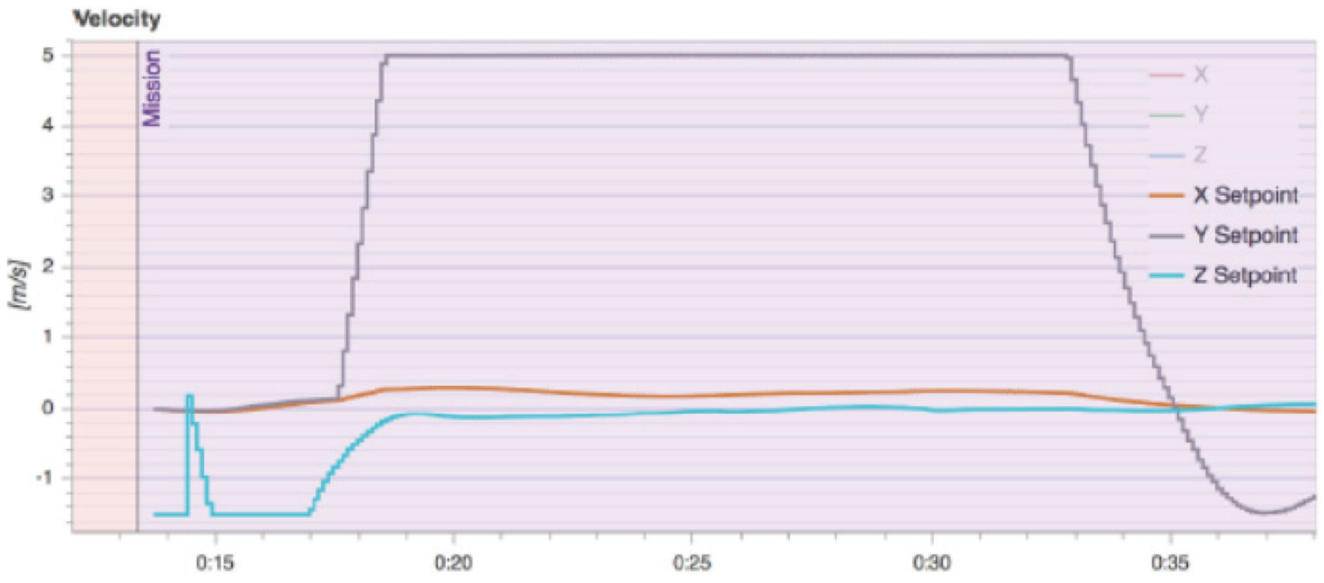


Fig. 18 Velocity without obstacles

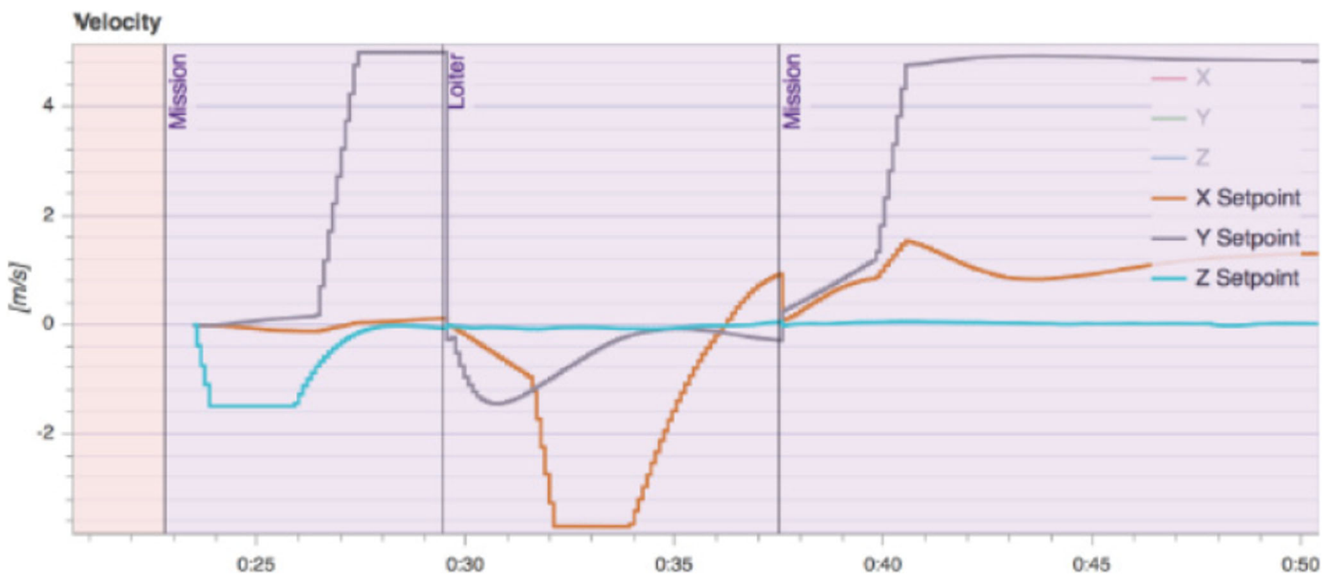


Fig. 19 Velocity with obstacles

reaction of the module operating with the data provided by simulation, with the sensor model running together with the virtual vehicle.

In the first place, the installation of Gazebo and development libraries was done with Linux Mint 18.3, a distribution of Linux based on Ubuntu 16.04. Then, the simulations were run by starting PX4 SITL and Gazebo with the airframe configuration to load (Iris quadrotor model). The easiest way to do this is to open a terminal in the root directory of the PX4 Firmware repository and call make for the targets selected:

```
$ cd src/Firmware
$ no_sim=1 make posix_sitl_default gazebo
```

After this, PX4 keeps waiting reply from simulator and Gazebo can be started in a separate console:

```
$ cd src/Firmware
$ source Tools/setup_gazebo.bash $(pwd) $(pwd)/build/posix_sitl_default
```

Then, Gazebo can be started with the 3RD Iris drone integrating the LIDAR model. As indicated above, the PX4 running in the other terminal will automatically receive the simulated data from Gazebo through port UDP 14560. After this, the system is ready to carry out the simulation tests. The trajectory can be generated in QGroundControl station and uploaded in the flight controller to start the mission.

The results obtained with simulation show the obstacle avoidance system integrated in PX4 and processing LIDAR data in parallel with navigation while the drone executes the programed mission. A basic tests was defined as a mission without obstacle and then the same mission with obstacles, putting the drone in the situation where it must turn in one

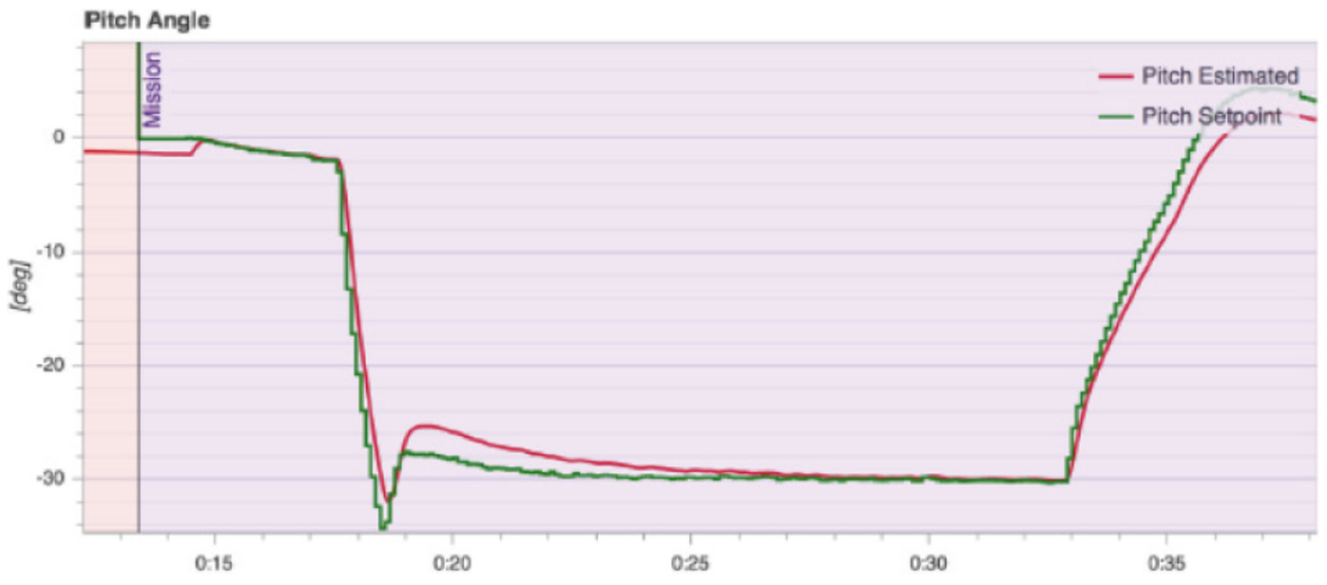


Fig. 20 Pitch without obstacle

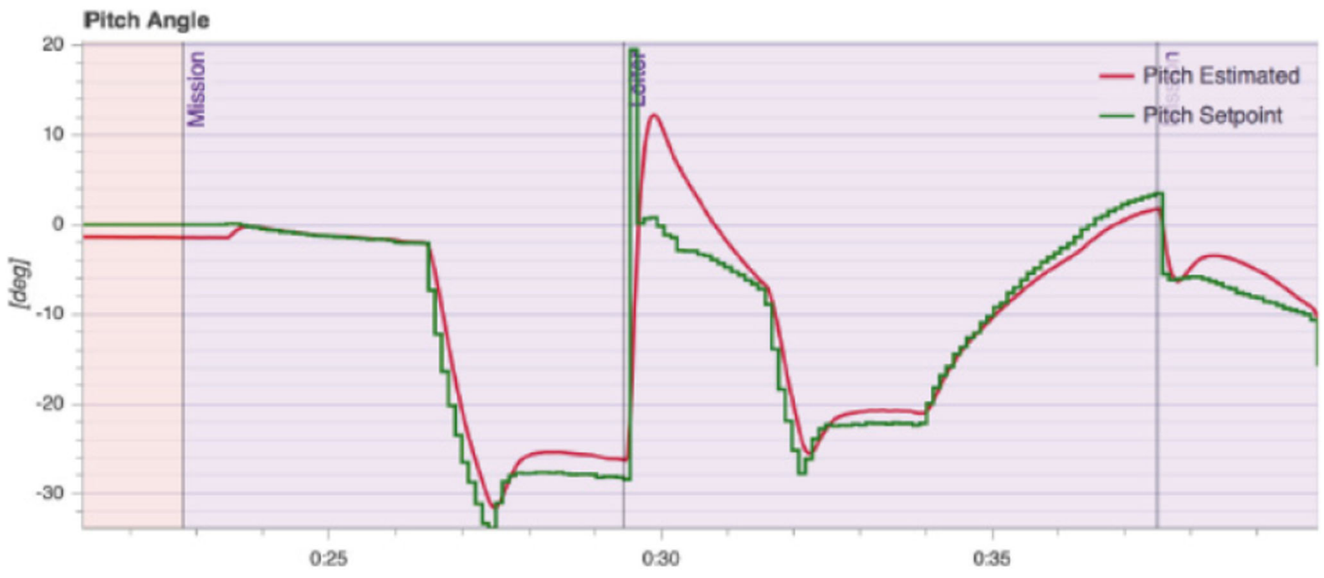


Fig. 21 Pitch with obstacle

or more objects, in this case a house as shown in Fig. 15. The simplest mission is a simple single-point mission; the drone will have to take off and go to point 1 without stopping and waiting for new orders there. In Fig. 16, we can see the mission to be carried out without obstacle and Fig. 17 with the obstacle in the way.

Figures 18 and 19 show velocities of flight in Cartesian coordinates. In the case with no obstacle, the velocity is constant since the mission starts with take off until the end. The vertical component of velocity changes at the beginning of the mission while the drone is ascending, and the rest of the mission remains constant. As shown in Fig. 18, when the drone detects the object, both components of velocity go down to 0 or even are negative to counteract the braking, and after that velocity takes the indicated value to carry out the modified mission shown in Fig. 17.

Regarding the attitude of drone, as shown in Figs. 20 and 21, mission starts in both figures, and there are few seconds where the pitch is constant while the drone is taking off. After that, as can be seen in Fig. 21, it leans forward to move, and when the evasion maneuver begins, the pitch increases to positive values to stop the drone and stabilizes at 0, then increases again to avoid the object; once the object is avoided, returns to 0, to return to the mission target and lean forward again to advance and continue the mission until the end. Conversely, Fig. 20 shows a pitch with no changes practically until mission ends.

Regarding heading, Figs. 22 and 23 show the yaw of drone, it performs a horizontal turn of 90° to avoid the obstacle as

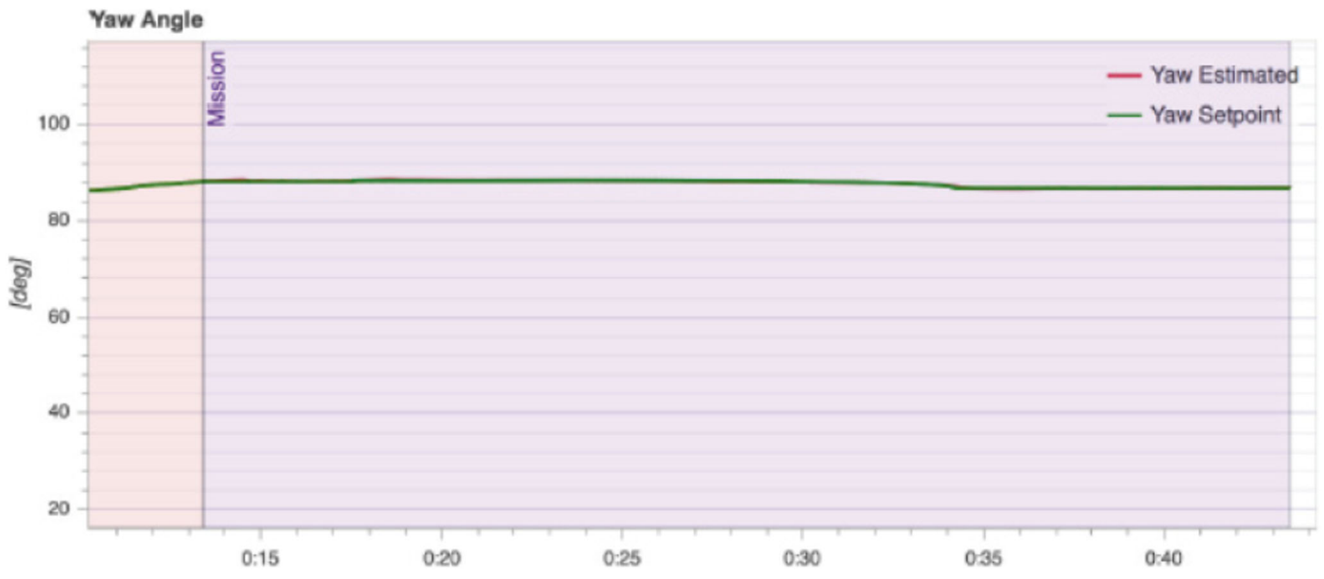


Fig. 22 Yaw without obstacle

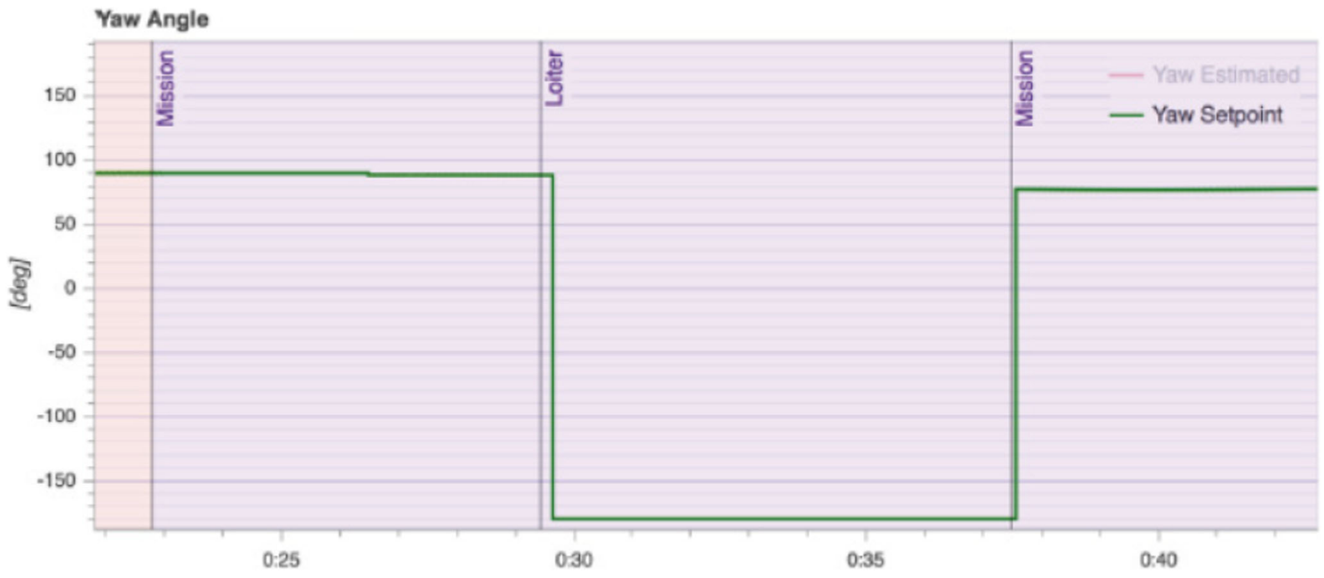


Fig. 23 Yaw with obstacle

shown in Fig. 23. The trajectory starts with heading of 90°, it turns to increase it in 90° until 180°, although this is indicated in the graph as − 180°. As reference, Fig. 22 shows the trajectory without obstacle; in this case, the yaw remains stable throughout the trajectory.

6 Conclusions

This paper presented the platform (PixHawk/PX4) and methodology to experiment with realistic simulated conditions for UAV autonomous navigation by integrating an obstacle avoidance module embedded in PX4 flight stack and simulated in full detail. The module processes range sensor data and interacts with the flight control process to carry out evasive maneuvers. Based on data analysis and characterization, the algorithms can take advantage of available sources. In addition, a modeling and software simulation methodology has been added using Gazebo to model 2D LIDAR sensors. Results shown successful integration of the sensor and a dedicated data processing task running in the platform. This tool is especially interesting to test the obstacle avoidance function, as previous analysis before operation in real scenarios.

So, the work developed to improve the current autopilot system implemented in PX4 has addressed the integration of a LIDAR sensor to implement an illustrative obstacle avoidance functionality. The possible solutions to interact with the PX4 system were analyzed, including the communication between processes with uORB and how to create a secondary task that runs in the background and performs the task in real time.

As future work, it is necessary to add a driver to obtain the actual laser distance sensor data and include it in the communication message in order to test the system with real data. The same methodology could be used to integrate another type of distance sensor such as SONAR or RADAR. A more precise evasion system could be developed and evaluate the performance compared with other alternatives in the state of the art, considering also the use of convex optimization techniques for trajectory re-planning. Finally, this system may need a computing capacity greater than that provided by PixHawk, so it could be implemented in a parallel computer that communicates with the main system through MAVlink but with the same structural scheme proposed in this work, comparing the computational load of both alternatives.

Acknowledgments Authors are grateful to Enrique Martínez Martínez and Ivan Serrano Ruiz, Graduate Engineers at UC3M in 2018 for their helpful collaboration in this work.

Funding information This work was funded by public research projects of Spanish Ministry of Economy and Competitiveness (MINECO), references TEC2017-88048-C2-2-R, RTC-2016-5595-2, RTC-2016-5191-8 and RTC-2016-5059-8.

LIDAR Simulation in Gazebo Environment

This section illustrates the use of Gazebo environment to generate the sensor input for navigation and obstacle avoidance function. As mentioned, the Iris quadrotor model developed by 3D Robotics is available for direct use from QGround platform and PX4 flight control, containing models of inertial sensors and GPS receiver as output data, and receiving input signals from flight controller to drive the four engines and simulate its dynamics. As an example of configuration for the simulation in Gazebo, the available model includes simulation of GPS noise, with a behavior similar to that typically found in real systems, a feature useful for testing applications that might be impacted by noise in positioning.

So, the GPS noise is enabled if the target vehicle's SDF file contains a value for the gpsNoise element (i.e., it has the line: <gpsNoise> true </ gpsNoise>). It is enabled by default in many vehicle SDF files, including the quadrotors solo.sdf and iris.sdf. To enable/disable GPS noise the following steps should be carried out:

```
1. Build any gazebo target in order to generate SDF files (for all vehicles). For example:
      make posix_sitl_default gazebo_iris
2. Open the SDF file corresponding to the target vehicle
   (e.g. ./Tools/sitl_gazebo/models/iris/iris.sdf).
3. Search for the gpsNoise element:
      <plugin name = 'gps_plugin' filename = 'libgazebo_gps_plugin.so'>
      <robotNamespace />
      <gpsNoise> true </ gpsNoise>
      </ plugin>
4. If it is present, GPS is enabled. It can be disabled just by deleting the line: <gpsNoise> true </
   gpsNoise>
```

Analogously, the model of the LIDAR sensor has been developed with ".sdf" file which reflects the logic behind the sensor's rays and the information produced. After the definition, the available Iris drone model was extended with a LIDAR sensor coupled in its upper part. The basic specification of Gazebo file is as follows:

```
<? xml version = "1.0"?>
<sdf version = '1.5'>
<model name = 'iris_rplidar'>
<! - Load the model 3RDIris ->
<include>
<uri> model: // iris </ uri>
</ include>
<! - Load the model of the Lidar and position it on the drone ->
<include>
<uri> model: // rplidar </ uri>
<pose> 0 0 0.1 0 0 0 </ pose>
</ include>
<! - Establishes the union between the drone and the lidar ->
<joint name = "rplidar_joint" type = "revolute">
<child> rplidar :: link </ child>
<parent> iris :: base_link </ parent>
<axis>
<xyz> 0 0 1 </ xyz>
<limit>
<upper> 0 </ upper>
<lower> 0 </ lower>
</ limit>
</ axis>
</ joint>
</ model>
</ sdf>
```

To understand in a more detailed way the operation and the configuration for this sensor, the corresponding .sdf file for LIDAR sensor containing the information referring to the sensor is shown below, including the physical parameters of sensor (location, mass, geometry, etc.), configuration of sensor range and angular coverage, and a facility to visualize the data:

```
<model name="rplidar">
<link name="link">
<!—Physical configuration of sensor including location, mass, geometry, moment of inertia, etc. -->
<inertial>
<pose>0 0 0 0 0 0</pose>
<mass>0.19</mass>
<inertia>
<ixx>4.15e-6</ixx>
<ixy>0</ixy>
<ixz>0</ixz>
<iyy>2.407e-6</iyy>
<iyz>0</iyz>
<izz>2.407e-6</izz>
</inertia>
</inertial>
<visual name="visual">
<geometry>
<box>
<size>0.02 0.05 0.05</size>
</box>
</geometry>
</visual>


<!-- Configuration of the sensor rays, which will have a measurement range between 0.3 meters and 10
meters and with a spectrum of 360° samples corresponding to one ray for each degree-->
<sensor name="laser" type="ray">
<ray>
<scan>
<horizontal>
<samples>360</samples>
<resolution>1.0</resolution>
<min_angle>-3.14</min_angle>
<max_angle>3.14</max_angle>
</horizontal>
</scan>
<range>
<min>0.3</min>
<max>15</max>
<resolution>1.0</resolution>
</range>
</ray>


<!-- Plugin that allows to visualize the information acquired by sensor rays while transmitting this
information. In addition, this type of plugin will create a topic in Gazebo with a function to
visualize the information that the sensor is collecting. So if you access the topics menu in Gazebo
you can find it with the name / laser / scan, as indicated in the line of code topicName -->

<plugin name="ros_ray_sensor_controller" filename="libgazebo_ros_laser.so">
<topicName>laser/scan</topicName>
<frameName>rplidar_link</frameName>
</plugin>
<always_on>true</always_on>
<update_rate>1.0</update_rate>
<visualize>true</visualize>
</sensor>
</link>
</model>
</sdf>
<!-- vim: set et fenc= ff=unix sts=0 sw=2 ts=2 : -->
```

References

1. Garcia J. Molina J.M., J. Trincado. Analysis of real data with sensors and estimation outputs in configurable UAV platforms. Sensor Data Fusion: trends, solutions and applications 2017. IEEE Bonn, Germany, 10–12 Oct. 2017
2. Groves PD (2015) Navigation using inertial sensors. IEEE AES Magazine 30(2):42–69
3. Britting KR (2010) Inertial navigation systems analysis. Artech House
4. Farrel JA (2008) Aided navigation: GPS with high rate sensors. McGraw-Hill, New York
5. Choi H, Kim Y (2014) UAV guidance using a monocular-vision sensor for aerial target tracking. Control Eng Pract 22:10–19
6. Torres-González A, Martínez-de Dios JR, Ollero A (2017) Robot-beacon distributed range-only SLAM for resource-constrained operation. Sensors 17(4):903
7. Ferrick A., Fish J., Venator E. and Lee G.S. UAV obstacle avoidance using image processing techniques Technologies for Practical Robot Applications (TePRA), 2012 IEEE international conference on 23–24 2012
8. Fasano G, Accado D, Moccia A, Moroney D (2016) Sense and avoid for unmanned aircraft systems. IEEE Aerosp Electron Syst Mag 31(11):82–110
9. Rankin G., Tirkel A., Leukhin A. Millimeter wave array for UAV imaging. MIMO Radar Radar Symposium (IRS), 2015 16th International 24–26 June 2015
10. Gualda D, Ureña J, García JC, García E, Alcalá J (2019) Simultaneous calibration and navigation (SCAN) of multiple ultrasonic local positioning systems. Information Fusion 45:53–65
11. Fasano G, Accardo D, Tirri AE, Moccia A (2016) Experimental analysis of onboard non-cooperative sense and avoid solutions based on radar, optical sensors, and data fusion. IEEE Aerosp Electron Syst Mag 31(7):6–14
12. Liu Z, Foina AG (2016) An autonomous quadrotor avoiding a helicopter in low-altitude flights. IEEE Aerosp Electron Syst Mag 31(7):30–39
13. J. García, J.M. Molina. Analysis of sensor fusion solutions for UAVs, Conference of the Spanish Association for Artificial Intelligence (CAEPIA). 2018. Granada, Spain. 23-26 2018
14. Meier L., PX4 Development Guide (online). **Available:** https://dev.px4.io/en/
15. Trawny, Nikolas, and Stergios I. Roumeliotis. Indirect Kalman filter for 3D attitude estimation. University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep 2 (2005)
16. Sola, Joan. Quaternion kinematics for the error-state Kalman filter. arXiv preprint arXiv:1711.02508 (2017)
17. G. Cai., B. M. Chen, Unmanned rotorcraft systems, Chapter 2 - Coordinate System and Transformation, 2011
18. https://es.mathworks.com/help/robotics/examples/perform-co-simulation-between-simulink-and-gazebo.html(accessed Oct 19)
19. https://es.mathworks.com/help/supportpkg/px4/ug/enable-mavlink-px4.html(accessed Oct 19)
20. T. R. Tuinstra, Range and velocity disambiguation in medium PRF radar with the DBSCAN clustering algorithm, , de : aerospace and electronics conference (NAECON) and Ohio innovation summit (OIS) Dayton, OH, USA, 2016
21. Y. Zhao Y. Su, Vehicles detection in complex urban scenes using Gaussian mixture model with FMCW radar, IEEE Sensors J , vol. 17, n° 18, pp. 5948–5953, 2017
22. MAVLINK common message set, https://mavlink.io/en/messages/common.html
23. Fasano G, Accado D, Moccia A (2016) Sense and avoid for unmanned aircraft systems. IEEE Aerospace and Electronic Systems Magazine:82–110
24. Kotegawa T (2016) Proof-of-concept airborne sense and avoid system with ACAS-XU flight test. IEEE Aerosp Electron Syst Mag 31(7):53–62
25. Schulman J et al (2013) Finding locally optimal, collision-free trajectories with sequential convex optimization. Robotics: science and systems 9(1)
26. Martí E, García J, Escalera A, Molina JM, Armingol JM (2012) Context-aided sensor fusion for enhanced urban navigation. Sensors 2012:16802–16837
27. Layh T, Demoz GE (September 2017) design for graceful degradation and recovery from GNSS interruptions. IEEE AESS Magazine Vol 32(N9):4–17