Development of a Sense and Avoid System for Small Fixed-Wing UAV

Bruno Manuel Bento Pedro
bruno.pedro@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

November 2024

Abstract

Given the rising number of applications of Unmanned Aerial Vehicles (UAVs) and consequent expansion of that market, enhanced flight safety systems need to be developed. The main objective of this work is to develop a Sense and Avoid (S&A) system for small fixed-wing UAVs. To achieve this, firstly, a literature review of the sensors and systems used to detect obstacles, cooperatively and non-cooperatively, was made, followed by a review of the main local and global path planning methods for collision avoidance. Then, a hardware implementation was proposed, consisting of two ultrasonic sensors, two laser rangefinders, and one LiDAR, integrated with a flight controller, a companion computer, and other components essential to the UAV operation. A complete software implementation was also proposed, ranging from the study and adaptation of the flight control software (PX4), with emphasis on the handle and communication of sensor data, to the development of a software prototype, based on the Vector Field Histogram (VFH) method, to be executed in the companion computer, to receive sensor data, obtain obstacle positions, and return setpoints of a collision avoidance trajectory. The validation tests have shown that the system is capable of, based on sensor data, compute obstacle positions, transform them to polar histogram format, and generate trajectory setpoints representing avoidance maneuvers of small deviations, with an update rate of 10Hz, thus real-time capable.
Keywords: obstacle detection, collision avoidance, Vector Field Histogram, flight controller, companion computer, ultrasonic sensor, laser rangefinder, LiDAR.

1. Introduction

Unmanned Aerial Vehicles (UAVs) have evolved from primarily military applications to a wide range of civil and commercial uses, such as surveillance, agriculture, logistics and media [1]. These applications often require UAVs to operate at low altitudes, where obstacles like buildings, trees and power lines pose significant collision risks. Consequently, the rapid expansion of the UAV market [2] emphasizes the need for reliable safety systems.

This work addresses the safety enhancement of small fixed-wing UAVs (with maximum takeoff weight under 25kg, range under 10km, endurance under 2h and flight altitude under 120m), with focus on the development of a Sense and Avoid (S&A) system, aimed at detecting obstacles and avoiding collisions autonomously during flight.

Building upon previous thesis [3–6] that modeled sensors, optimized sensing configurations, and simulated avoidance path-planning algorithms, for fixed-wing UAVs, the contributions of this work lie, firstly, on the design and proposal of a hardware implementation of a S&A system, incorporating a multi-sensor configuration, a flight controller, and a companion computer; and, secondly, on the proposal of a software implementation, based on the adaptation of the flight control software, PX4, to receive and handle data from the obstacle detection sensors, and the development of a software prototype to be executed on the companion computer, to receive sensor data from the flight controller, process it, and generate avoidance trajectory setpoints in real-time. Lastly, the complete S&A system is validated through bench testing in a rover robot.

Obstacle sensing systems in UAVs are generally divided into cooperative detection, when there is information exchange between the aircraft and the obstacle, and non-cooperative detection, otherwise. The latter requires proper sensor hardware, such as laser rangefinders, applied in [7] for a quadrotor UAV, Light Detection and Ranging (LIDAR), considered in [8] for fixed-wing UAVs, and ultrasonic sensors, applied in [9] also for a quadrotor. In turn, path planning methods for collision avoidance in UAVs can be global, when the obstacles are known, and local, when the obstacles are not expected and

the path is updated in real-time. Of the latter, it is worth noting the Vector Field Histogram (VFH) method [10], applied in [11] for fixed-wing UAV.

2. Hardware Implementation

The hardware implementation of a S&A system requires some decisions regarding the physical components to comprise it, as well as how they are configured and connected.

2.1. Sensor Hardware

Based on the sensing configuration presented in [6], the hardware chosen to support the obstacle detection is composed of three different types of non-cooperative active sensors: two ultrasonic sensors (Fig. 1a), two laser rangefinders (Fig. 1b), and one LiDAR (Fig. 1c). Their main specifications are summarized in Tab. 1.

(a) Ultrasonic sensor MB1242 [12].

(b) Laser rangefinfer LW20/C [13].

(c) LIDAR SF45/B [14].

Figure 1: Sensor hardware.

Table 1: Sensor hardware specifications.

	Ultrasonic sensor [12]	Laser rangefinder [13]	LiDAR [14]
Range (m)	0.20-7.65	0.20-100	0.20-50
Scan angle (°)	-	-	20-320
Resolution (cm)	1	1	1
Angular resolution (°)	-	-	<0.2
Update rate (Hz)	10	40-388	50-5000
Accuracy (cm)	±10	±10	±10
Dimensions (mm)	22x19x15	30x20x43	51x48x44
Weight (g)	5.9	20	59

Two different models of ultrasonic sensors from the Maxbotix are used, namely the MB1202 and MB1242. They share specifications, such as detection range, resolution, accuracy, and maximum update rate, which is constrained by the duration of a ranging cycle. The major difference between them is the type of beam pattern, wider for MB1202 (more noise clutter) and narrower for MB1242 (less noise clutter). To operate both sensors simultaneously on the same bus of the Inter-Integrated Circuit (I2C) communication interface and differentiate their range measurements, they must have different addresses. Thus, the I2C address of MB1202 was changed 0x68, and MB1242 was kept with the 0x70.

Two identical laser rangefinders, Lightware LW20/c, are considered. Their detection range goes up to 100m, much higher than ultrasonic sensor's. Relying on the speed of light, instead of the speed of sound, allows the update rate to be higher too. Moreover, it is tolerant to changes in background lighting conditions, wind and noise, and the accuracy is not generally affected by the color or texture of the target surface, nor the angle of incidence of the beam [13]. Once again, the I2C address of one of the two sensors had to be changed, therefore, using the Lightware Studio software provided by the manufacturer, one of the sensors was changed to 0x67 and the other kept the original address of 0x66.

To scan a wider area ahead the UAV, the Lightware SF45/B LiDAR sensor is used. With a detection range up to 50m, the major features of this LiDAR are the scanning angle, which can go from 20° to 320°, and the update rate, configurable from 50Hz to 5000Hz. The speed of rotation is dependent on the scan angle and can go up to 5 sweeps per second. Just like the laser rangefinder, it is also tolerant to changes in background lighting conditions, wind and noise [14]. The scanning angle was configured to range from -45° to 45°, given the turning rate limitations of a fixed-wing UAV. Regarding the communication interface, it was chosen not to use I2C, but serial communication through one of the telemetry (TELEM) ports of the flight controller.

2.2. Flight Controller and Companion Computer

It is to the flight controller that the obstacle detection sensors are connected, since it is capable of collecting their measurements and processing them in a first instance. However, due to its limited computational power, the processing necessary to the application of a collision avoidance method is left to a more powerful companion computer that directly communicates with the flight controller.

Figure 2: Holybro Pixhawk RPi CM4 baseboard parts (from left to right): case with fan, baseboard, Raspberry PI CM4, and Pixhawk 6X [15].

The flight controller chosen for this application is the Pixhawk 6X from Holybro, which, together with the Raspberry Pi Computer Module 4 (CM4) as companion computer, are integrated in the Holybro Pixhawk RPi CM4 baseboard. These components are presented in Fig. 2.

2.3. Electrical Layout

Together with some auxiliary hardware components, essential for the flight operation of a fixed-wing UAV, such as a power module, a battery, a DC motor, four servos, an Electronic Speed Controller (ESC), a GPS module, a radio receiver, and a telemetry module, an overall electrical layout of the hardware connections is designed, as shown in Fig. 3.

Even though the companion computer and the flight controller are internally connected in the baseboard through the serial TELEM2 port, a connection over Ethernet was used instead, due to higher bandwidth.

3. Software Implementation

The software implementation of the S&A system addressed in this work can be seen as an application with additional developments of existing open-source solutions. Figure 4 presents a diagram of the main components of the software (flight control software, ground control software, and companion computer software) and high-level interaction between them.

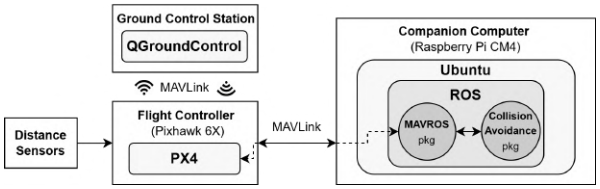


Figure 4: Diagram of S&A system software implementation.

3.1. Flight Control Software

The flight control software adopted in this work is the PX4 open-source project [15], due to its reliability, modular architecture, allowing for extension of functionalities, good documentation, and increasing presence in the industry, with a growing community of users and developers. It supports different types of vehicles, such as multicopters, fixed-wing UAVs, and rovers.

3.1.1. PX4 Internal Communication: uORB

The communication between internal modules of PX4 is done using the micro Object Request Broker (uORB) protocol. It is based on a mechanism to publish/subscribe messages in topics, allowing multiple independent instances of the same topic. Each uORB topic must have a prior definition of the fields that make up its message context.

The data from the obstacle detection sensors is published in the `distance_sensor` uORB Topic, whose fields are described in Tab. 2. The most important fields are the `device_id`, a unique ID of the sensor, the `current_distance`, the sensor range measurement, and the `current_yaw`, which is the only non-standard field, added to include the direction, in degrees (from $-45°$ to $45°$), in the horizontal plane (yaw) of the LiDAR.

Table 2: Fields of uORB topic `distance_sensor` [15].

Name	Units	Description
`timestamp`	ms	Timestamp
`device_id`	-	Sensor ID
`min_distance`	m	Minimum range
`max_distance`	cm	Maximum range
`current_distance`	cm	Current range
`current_yaw`	**deg**	**Current yaw**
`variance`	m^2	Variance
`signal_quality`	%	Signal quality
`type`	-	Sensor type
`h_fov`	rad	Horizontal FOV
`v_fov`	rad	Vertical FOV
`q`	-	Orientation quaternion
`orientation`	-	Sensor orientation

The idea is to have a single `distance_sensor` uORB topic with one instance for each sensor. However, the two ultrasonic sensors will share one instance due to driver limitations. The sensor data that is internally organized in the `distance_sensor` uORB topic is, then, streamed over MAVLink, both to the Ground Control Station (GCS) and the companion computer.

Other uORB topics are also used in the S&A system. For example, the `vehicle_local_position` is used to communicate the UAV local position, velocity and acceleration estimates, in a NED (North-East-Down) frame, the `trajectory_setpoint` is used to internally communicate position, velocity and acceleration setpoints in a local NED frame, and the `vehicle_local_position_setpoint` can be used to monitor the setpoints used by the position controller of PX4.

3.1.2. Drivers of Distance Sensors

The interface between the obstacle detection sensors and the PX4 is done by drivers, which are responsible for sensor initialization, acquisition of data measurements, primary data processing, and communication with the uORB messaging bus. They can be controlled over MAVLink Console commands.

The ultrasonic sensors are controlled by the built-in `mb12xx` PX4 driver. It was build in such a way that a single instance of the driver is capable of controlling multiple ultrasonic sensors connected to the same I2C bus, as long as they have different I2C addresses. In this case, the sensor update rate is defined by the driver, as well as time interval between reads of consecutive ultrasonic sensors. So, the former was set to 10Hz and the latter to 50ms, to match the the maximum time of a ranging cycle.

The laser rangefinders are controlled by the built-in `lightware_laser_i2c` PX4 driver. Contrary to what happens with the ultrasonic sensors, this driver is unable to control, in a single instance,

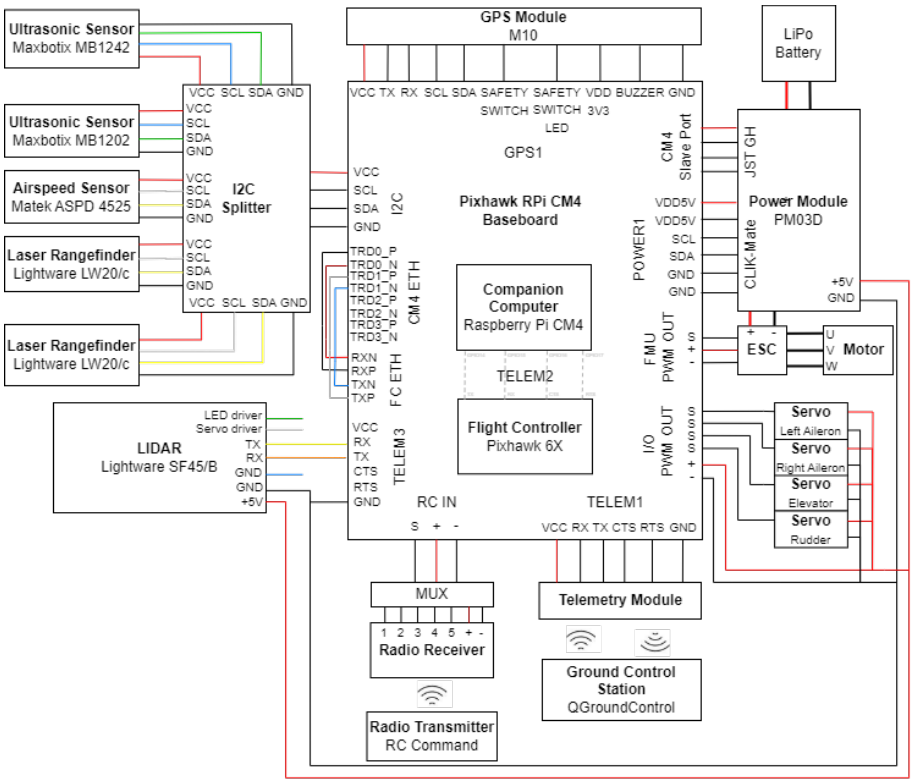


Figure 3: Hardware electrical diagram.

multiple sensors with different I2C addresses in the same I2C bus. For this reason, the solution found to have two lasers connected at the same time was by starting two independent instances of the driver in the startup shell script of PX4.

The LiDAR is controlled by the built-in `lightware_sf45_serial` PX4 driver. Unlike the previous drivers, it is not included in the firmware by default, so it needs to be manually enabled in the PX4 firmware configuration.

Even though the LiDAR sensor measures the scanning angle at each instant, the standard version of its driver does not publish these measurements, which are necessary to fulfill the custom `current_yaw` field added to the `distance_sensor` uORB topic. So, the `s_update()` function developed in [6] was included for that purpose.

3.1.3. MAVLink Communcation

The external communication between the flight controller and other devices, such as the GCS and the companion computer is done through MAVLink, standing for Micro Air Vehicle Link. The MAVLink messages are characterized by a name, an id, and fields containing the data to be transmitted.

PX4 includes MAVLink as a module and, generally, MAVLink messages stream data of an already existing uORB message with similar fields. Furthermore, it can have independent instances to communicate with different peripheral devices simultaneously.

The MAVLink message `DISTANCE_SENSOR` (`ID=132`) is the standard message used to communicate data from the obstacle detection sensors. Although most of the fields are similar to the ones of the homonym uORB topic, it had to be slightly modified to suit the S&A system, with the custom addition of the `current_yaw` and `device_id` fields.

Other MAVLink messages are important for the S&A system, such as `LOCAL_POSITION_NED`, a standard message used to communicate the local position of the UAV, `SET_POSITION_TARGET_LOCAL_NED`, used to communicate position, velocity or acceleration setpoints defined in the companion computer, `POSITION_TARGET_LOCAL_NED`, which retrieves data from the `vehicle_local_position_setpoint` uORB topic to monitor the setpoints that are actually being sent to the position controller of PX4, and `VFR_HUD`, used to communicate head-up display (HUD) information, such as airspeed, groundspeed, heading, throttle, altitude MSL, and climb rate

3.2. Ground Control Software

A GCS is a ground based system that allows a human operator to monitor, control and manage the systems of an UAV in real-time. In this work, the open-source QGroundControl was used as GCS software in a computer running Windows 10 OS, to communicate with PX4 over USB or telemetry.

3.3. Communication Between PX4 and Companion Computer

To communicate the obstacle detection sensors data from the flight controller to the companion computer, firstly, an Ethernet connection between them is set. Then, the MAVLink interface to use in the companion computer was chosen among MAVSDK, pymavlink and MAVROS.

MAVSDK [16] is a cross-platform high-level API to interface with MAVLink, that is easy to use, but has limited low-level access and control over the messages. On the opposite, pymavlink [17] is a low-level Python library that provides fine-grained control of the MAVLink messages, but presents a steeper learning curve. Lastly, MAVROS [18] is a Robot Operating System (ROS) package that acts as a bridge between ROS and MAVLink by translating the MAVLink messages into ROS messages, organized in ROS topics, and vice-versa. Since it includes well-tested PX4 support and allows the integration of the S&A system as a ROS package, MAVROS 1.19.1 with ROS1 Noectic was the option selected to interface with MAVLink, despite being more resource-intensive.

To communicate distinguishable data from the five obstacle detection sensors, the `distance_sensor` plugin of MAVROS had do be modified to map the sensors from the `device_id` field. Moreover, a custom ROS message was created to include the `device_id` and `current_yaw` fields.

3.4. Obstacle Detection and Collision Avoidance Software

Given that ROS provides a flexible framework for writing robotic software with MAVROS as MAVLink interface, the remaining steps of obstacle detection and collision avoidance can be developed as a software prototype within a ROS package. This approach is not a novelty, since there is already an open-source package, PX4-Avoidance [19], developed by the PX4 community, to enable obstacle detection with stereo-vision camera hardware and collision avoidance for multicopters.

Regarding the programming language, Python was chosen in this phase of development, since it is better for rapid prototyping, although C++ allows better performance. The `rospy` Python library provides an interface with ROS for creation of nodes, publish/subscription of topics, and interaction with services and parameters. A multithreading approach was considered, with the `threading` Python module, to allow multiple tasks to run concurrently within a single process.

The software prototype was divided in two main parts: **obstacle detection**, responsible for processing the data from the distance sensors and transform it into two-dimensional positions; and **collision avoidance**, responsible for generating, in real-time, an avoidance trajectory for the UAV. The approach followed here was based on the VFH method.

3.4.1. Software Architecture

The architecture of the software prototype is illustrated in Fig. 5, including the files, classes, methods, and the data flow between methods.

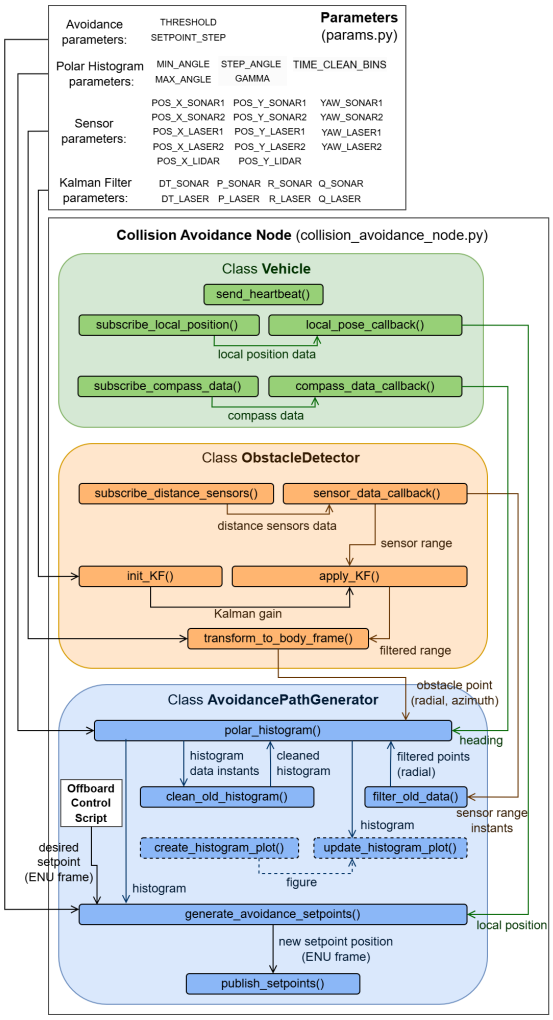


Figure 5: Software architecture of the obstacle detection and collision avoidance implementation.

The software is organized in two main files: Parameters (`params.py`), where the main parameters of the system, related to the distance sensors, Kalman filter, polar histogram, and avoidance process are configured/tuned; and the Collision Avoidance Node (`collision_avoidance_node.py`), where all the code developments are included.

3.4.2. Implementation of Obstacle Detection

The obstacle detection part of the software is implemented within the class **ObstacleDetector**. It starts with the subscription of five ROS topics, one for each sensor, where data are being published by

MAVROS. This way, everytime new sensor data is published on the corresponding topic, a callback function is called to save it in sensor-specific variables and process it. A one-dimensional Kalman filter, from the `filterpy` Python package, is applied to the range measurements of the ultrasonic sensors and laser rangefinders to smooth noisy sensor data and provide a better estimate of the true distance to the obstacles.

Then, the filtered range measurements of the sensors, $\hat{d}$, are transformed to polar coordinates of the UAV body reference frame, so that they represent two-dimensional obstacle positions. For this, the position in Cartesian coordinates (x_{sens}, y_{sens}) and orientation, β_{sens}, of each sensor, in the body frame, specified as parameters, are used to compute the radial and azimuthal components of the obstacle position, (r_{obs}, φ_{obs}), from

$$r_{obs} = \sqrt{(\hat{d}\cos(\beta_{sens}) + x_{sens})^2 + (\hat{d}\sin(\beta_{sens}) + y_{sens})^2} \quad (1)$$

and

$$\varphi_{obs} = \arctan\left(\frac{\hat{d}\sin(\beta_{sens}) + y_{sens}}{\hat{d}\cos(\beta_{sens}) + x_{sens}}\right). \quad (2)$$

3.4.3. Implementation of Collision Avoidance

The collision avoidance part of the software is implemented mainly within the class **AvoidancePathGenerator**, using some methods of the class **Vehicle**.

Having the position of the obstacles detected by a sensor, in polar coordinates of the body frame, and aiming to apply the VFH method, a polar histogram is generated to represent the obstacle density in space, using the algorithm flowchart of Fig. 6. It starts by the creation of the histogram using the parameters `MIN_ANGLE`, `MAX_ANGLE` and `STEP_ANGLE`, followed by a loop with frequency of 20Hz to update its bins using the most recent obstacle position detected by each sensor, in East-North-Up (ENU) frame, $(r_{obs}^{ENU}, \varphi_{obs}^{ENU})$, by finding the bin in which the obstacle is inserted and computing its obstacle density, h_k, with the arbitrary function: $h_k = \frac{50 - r_{obs}}{50}$.. For safety reasons, the obstacle density of a bin is spread to its neighbor bins, using a function controlled by the parameter γ: $h_{k\pm a} = h_k$ $(a = 1, ..., \gamma)$. That update process includes methods to erase old sensor data, as well as old histogram data.

Next, concurrently to the update of the polar histogram, new trajectory setpoints for the UAV are generated with frequency of approximately 10Hz from the algorithm flowchart of Fig. 7. This process starts from desired setpoint positions, in ENU frame, given by an external offboard control script, which are used to determine the desired direction.

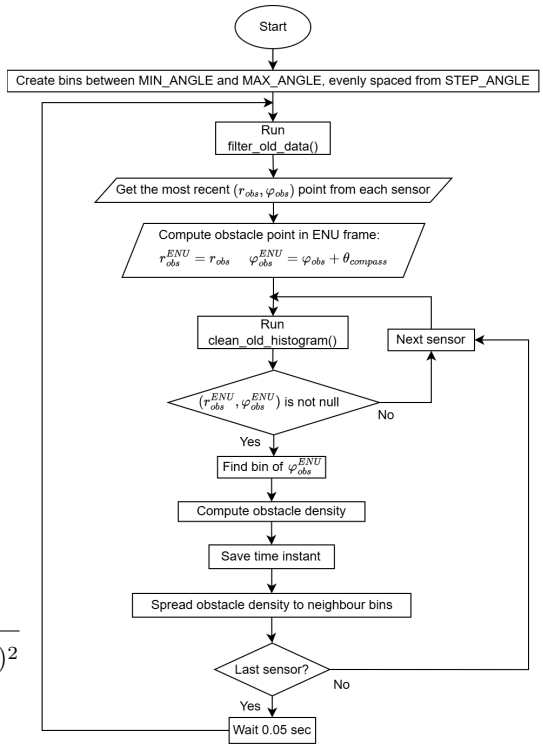


Figure 6: Flowchart of the algorithm for creation and continuous update of the polar histogram.

Then, the bins of the polar histogram with an obstacle density below a `THRESHOLD` (available bins) are selected and, from these, the one corresponding to a direction closer to the average between the desired direction and the direction followed in the last iteration is chosen. From that direction, a new setpoint velocity in Cartesian coordinates is generated using the `SETPOINT_STEP` parameter, as well as a new setpoint position in the ENU frame using local position data. Finally, it can be chosen to publish the new setpoint position or velocity.

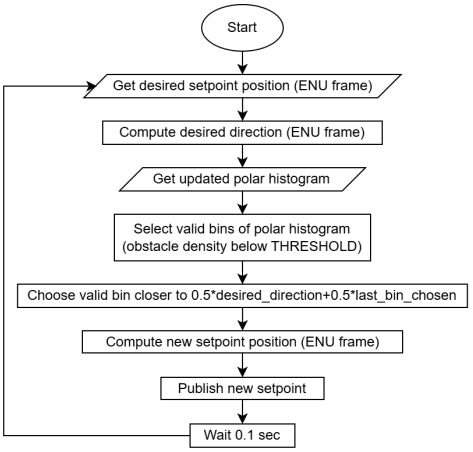


Figure 7: Flowchart of the algorithm to generate avoidance setpoints.

4. Validation Tests

To validate the previous hardware and software implementations of the S&A system, a few real-world bench tests were performed in a rover robot, given the risks associated with testing new developments in flight.

4.1. Rover System Setup

The hardware for testing in a rover was adapted from the electrical layout of Fig. 3, resulting in the setup of Fig. 8. For flight control software, the `rover_pos_control` module of PX4 1.14.3 was used.

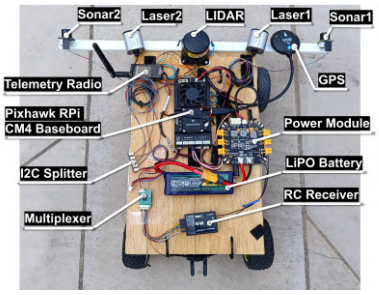
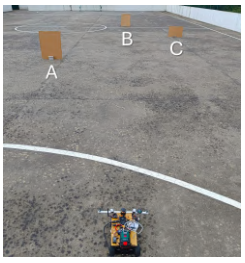


Figure 8: Top view of rover setup (labeled hardware).

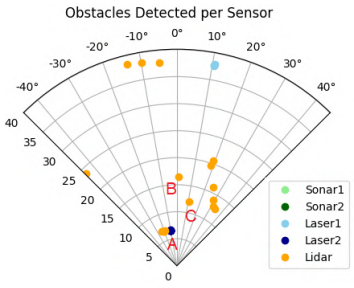
4.2. Static Vehicle and Static Obstacles

The first test was conducted with a static rover in front of three static obstacles A ($0.55m^2$), B ($0.55m^2$) and C ($0.35m^2$), arranged as shown in Fig. 9a. The objective was to validate the capabilities of the system to detect obstacles, estimate its relative positions and translate them to the polar histogram of the VFH method.

The system was run for around 10 seconds, using the parameters of Tab. 3.



(a) Arrangement of static obstacles A, B and C.

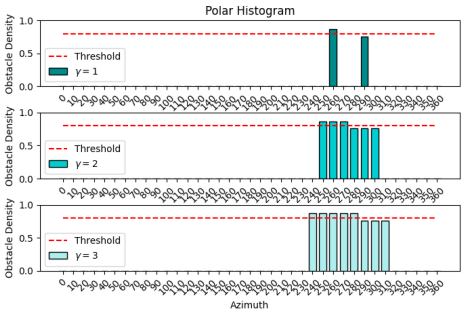


(b) Obstacles detected by sensors during the test.

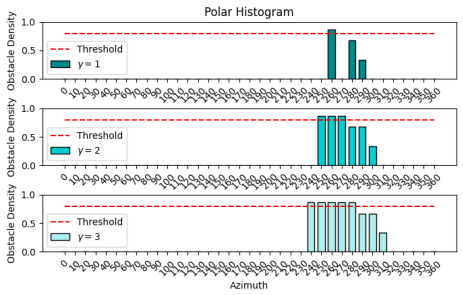
Figure 9: Static vehicle and static obstacles.

The data of the obstacles positions in polar coordinates of the rover body frame, (r_{obs}, φ_{obs}), were plotted in Fig. 9b, separated by sensors. It can be observed that the ultrasonic sensors did not detect obstacles, since no obstacles were in their range. The laser rangefinders detected obstacles along the directions they were pointed. The LiDAR presented detections in different directions, as expected.

It is possible to identify obstacle A, successfully detected by both the Laser2 and the LiDAR, as well as obstacles B and C, only detected by the LiDAR. The cluster of LiDAR points in the right-side, the single point in the left-side, and the points in the top of the plot, including the one from Laser1, correspond to the detection of the field walls. This way, it can be concluded that the system was able to detect the target obstacles successfully.



(a) $t_1 \approx 1.8s$.



(b) $t_2 \approx 5.4s$

Figure 10: Polar histograms for $\gamma = 1$, $\gamma = 2$ and $\gamma = 3$.

The translation of the obstacles positions to polar histogram, with bins from 0° to 360°, and step angle of 10°, were plotted in Figs. 10a and 10b, from two time instants, t_1 and t_2, when the pairs of obstacles A,B and A,C were detected, respectively.

At $t \approx 1.8s$, there are three main bins, one for 260° from the Laser2 detection of obstacle A, another for 280° from the LiDAR detection of obstacle B, and a smaller bin for 290° from the Laser1 detection of the wall. At $t \approx 5.4s$, there is the same main bin for 260°, and another for 290° from the LiDAR detection of obstacle C. The second and third plots of the previous figures, present the effect of spreading the obstacle density to neighbor bins, which are controlled by γ. Having $\gamma = 1$ leads to 0 neighbors, $\gamma = 2$ leads to 2 neighbors for each main bin, and $\gamma = 3$ leads to 3 neighbors for each main bin. An example of a threshold line of 0.8 is also presented - the bins with obstacles densities above 0.8 are con-

Table 3: Obstacle detection and collision avoidance software parameters.

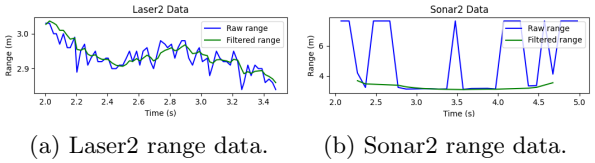
Parameter	Value	Parameter	Value	Parameter	Value
POS_X_SONAR1 (m)	0.2	POS_Y_LASER2 (m)	-0.1	R_LASER	1
POS_Y_SONAR1 (m)	0.2	YAW_LASER2 (°)	-10	Q_LASER	$\begin{bmatrix} 10^{-1} & 0 \\ 0 & 10^{-1} \end{bmatrix}$
YAW_SONAR1 (°)	0	POS_X_LIDAR (m)	0.2	MIN_ANGLE (°)	0
POS_X_SONAR2 (m)	0.2	POS_Y_LIDAR (m)	0	MAX_ANGLE (°)	360
POS_Y_SONAR2 (m)	-0.2	DT_SONAR	0.1	STEP_ANGLE (°)	10
YAW_SONAR1 (°)	0	P_SONAR	7	GAMMA	2
POS_X_LASER1 (m)	0.2	R_SONAR	1	TIME_CLEAN_BINS (s)	0.1
POS_Y_LASER1 (m)	0.1	Q_SONAR	$\begin{bmatrix} 10^{-1} & 0 \\ 0 & 10^{-1} \end{bmatrix}$	THRESHOLD	0.9
YAW_LASER1 (°)	10	DT_LASER	0.05	SETPOINT_STEP	3
POS_X_LASER2 (m)	0.2	P_LASER	50		

sidered unavailable and the others are considered available.

4.3. Static Vehicle and Moving Obstacle

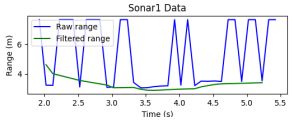
The next test was performed with a static rover and an obstacle ($0.55m^2$) moving in front of it, from the left to right, at a speed of about $1m/s$ and a distance of around 3m. The objective of this test was to validate the detection of dynamic obstacles and its reflection in variations of the polar histogram. The same parameters of Tab. 3 were used, with exception to THRESHOLD, which was changed to 0.90, to make the system react only for obstacles below 5m of radial distance.

To assess the characteristics of the sensor data that feed the system, the raw and filtered range measurements of the sensors were saved, truncated to the time intervals where the obstacle is detected, and plotted in Figs. 11a, 11b, 11c, 11d, and 11e, respectively from the first to the last to detect the obstacle. Given the positions of the sensors in the vehicle, the sequence of detection by the lasers and ultrasonic sensors is as expected for an obstacle moving from left to right.

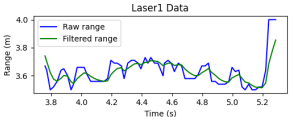


(a) Laser2 range data.

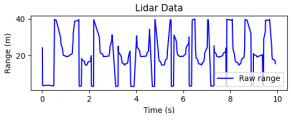
(b) Sonar2 range data.



(c) Sonar1 range data..



(d) Laser1 range data..



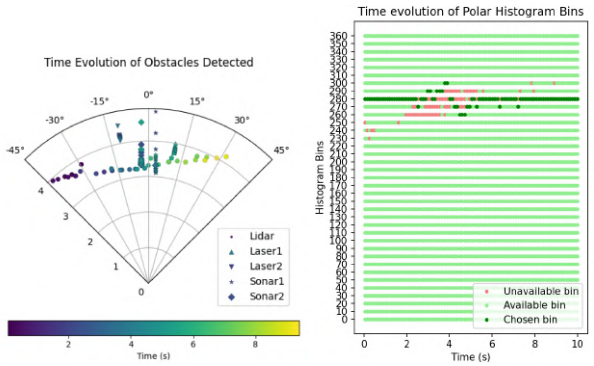
(e) LiDAR range data.

Figure 11: Raw and filtered obstacle detection sensors data

For all cases, the obstacle is detected through range measurements between 2.8m and 4m, which is within the tolerated distances. Moreover, the Kalman filter performed reasonably for the lasers and ultrasonic sensors, reducing the noise and dampening the effect of outlier measurements that could lead the system to unnecessary reactions. Measurements of the ultrasonic sensors above 7m were not considered for filtering, since these sensors measure the maximum range (7.65m) when no obstacles are detected. Finally, the LiDAR data was not subject to any filtering process, but presented good results by detecting the obstacle at each scan.

The transformation of the data from all sensors to obstacle positions in polar coordinates, in the vehicle body frame, over the execution of the test, resulted in Fig. 12a. The points are labeled by the sensors that originated them, such that the cluster of points distributed approximately along the $-11°$ azimuth came from Laser2, the points along the $-3°$ azimuth came from Sonar2, the points along the $3°$ azimuth came from Sonar1, the points along the $11°$ azimuth came from Laser1 and the other scattered points came from LiDAR. Once again, the evolution of the point positions in time is in accordance with the trajectory of the obstacle from left to right.

That classification of available/unavailable histogram bins was plotted over time in Fig. 12b, together with the bins chosen each time. The desired setpoints arbitrarily imputed to the system were such that the desired direction was of $280°$. As soon as the obstacle covered that direction, the corresponding bin became unavailable and the system was forced to chose another bin direction. Throughout the time, as the obstacle moved to the right-side, the bins affected were dynamically blocked and released, from lower to higher angles, while the system was dynamically choosing the available bin closer to the desired direction. This way, it can be concluded that the system successfully detects a dynamic obstacle and presents an intended solution to avoid a collision.



(a) Position of obstacles detected in polar coordinates of body frame, over time.

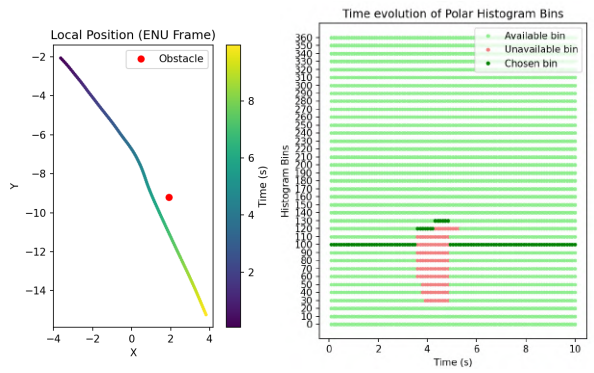
(b) Available, unavailable, and chosen bins of the polar histogram, over time.

Figure 12: S&A system outputs as a function of time for static vehicle test.

5. Moving Vehicle and Static Obstacles

Finally, a test was performed with the rover moving towards one static obstacle ($0.55m^2$). The objective was to validate the capabilities of the system to, based on the detection of obstacles, perform a real-world collision avoidance maneuver. The software was tuned with the parameters of Tab. 3, hanging to $\gamma = 4$ and `TIME_CLEAN_BINS=1`, to enhance safety. Regarding the avoidance part of the software, it was decided to only publish setpoint positions.

The rover performed the test at an average speed of $2m/s$. The local position (ENU frame) over time was plotted in the Fig. 13a, together the position of the obstacle. It shows that the rover was following a linear trajectory towards the obstacle and, around 3m before colliding to it, a small deviation to the right on the trajectory was made, allowing the successful avoidance of a collision with the obstacle.



(a) Local position of Vehicle in earth-fixed ENU frame over time.

(b) Available, unavailable, and chosen bins of the polar histogram, over time.

Figure 13: S&A system outputs as a function of time for moving vehicle test.

The classification of histogram bins was plotted in Fig. 13b. Initially, the rover was physically aligned to the desired direction of 100° and, at the time instant $t = 3.6$s, the detection of the obstacle led to the blocking of bins from 60° to 110°, forcing the system to choose the direction of 120° and start the avoidance trajectory. In the following seconds, the blocked bins eventually evolved to the range from 30° to 120°, forcing the vehicle to follow the direction of 130°, until a moment when the polar histogram data is cleaned and, since no further obstacles were detected, the system returned to the desired direction of 100°, finishing the avoidance trajectory.

It is important to note that, as shown in Fig. 13a, the desired direction followed before starting the avoidance maneuver and the desired direction followed after passing the obstacle was not the same, even though the system published setpoint positions in direction of 100° in both cases. This can be due to a faulty calibration of the compass, which led to inaccurate heading measures during the test.

This way, it can be concluded that the system was able to perform obstacle detection and collision avoidance in the presence of a single obstacle, at a speed around $2m/s$, by generating a trajectory of avoidance setpoint position at a frequency of 10Hz.

6. Conclusions

This work presented a solution of a S&A system to enhance the flight safety of small fixed-wing UAVs. A hardware implementation was proposed for the system using two ultrasonic sensors, two laser rangefinders and one LiDAR as detection sensors, a Pixhawk 6X as flight controller, a Raspberry Pi CM4 as companion computer, and some other components to complete the setup. A software implementation was also proposed, regarding the flight control software, PX4, which was adapted and dissected into the most important parts (uORB message bus, drivers of distance sensors and MAVLink communicaiton); the ground control software; and the companion computer software, centered on the development, in the ROS environment, of a version of the VFH method as a software prototype to receive sensor data and perform obstacle detection and collision avoidance.

Finally, the system was validated through real-world tests in a rover. The first test, with static vehicle and static obstacles, showed that the system was able to determine obstacle positions and translate them to a polar histogram. The second, with static vehicle and one moving obstacle, showed the capabilities of the system to detect dynamic obstacles and choose, in real-time, the direction to follow, accordingly. The last test, with the rover moving towards one static obstacle, showed the overall S&A

system capability of performing obstacle detection and collision avoidance through a small deviation maneuver from trajectory setpoint positions published at a rate of 10Hz.

This work leaves an open path to some future improvements. Regarding obstacle detection, based on the work done to handle multiple sensors in PX4 and receive their data separately in the companion computer, it would be possible to implement sensor fusion techniques, aiming to features such as obstacle tracking. Regarding collision avoidance, more advanced methods, such as the variants of VFH (e.g VFH+ and VFH*) could be implemented to provide a more reliable and optimized solution.

References

[1] P. G. Fahlstrom, T. J. Gleason, M. H. Sadraey, P. Belobaba, J. Cooper, and A. Seabridge, editors. *Introduction to UAV Systems*. Aerospace Series. Wiley, 5th edition, 2022. ISBN: 9781119802617.

[2] L. Kapustina, N. Izakova, E. Makovkina, and M. Khmelkov. The Global Drone Market: Main Development Trends. *SHS Web Conf.*, 129:11004, 2021. doi:10.1051/shsconf/202112911004.

[3] Juliana Alves. Path Planning and Collision Avoidance Algorithms for Small RPAS. MSc Thesis in Aerospace Engineering, Instituto Superior Técnico, Lisboa, Portugal, June 2017.

[4] Nuno Alturas. Modeling and Optimization of an Obstacle Detection System for Small UAVs. MSc Thesis in Aerospace Engineering, Instituto Superior Técnico, Lisboa, Portugal, January 2021.

[5] Pedro Serrano. Optimization of Obstacle Detection for Small UAVs. MSc Thesis in Aerospace Engineering, Instituto Superior Técnico, Lisboa, Portugal, June 2022.

[6] Marta Portugal. Optimal Multi-Sensor Collision Avoidance System for Small Fixed-Wing UAV. MSc Thesis in Aerospace Engineering, Instituto Superior Técnico, Lisboa, Portugal, November 2023.

[7] Stanislav Shougaev and Moshe Idan. Laser Range-Finder Based Obstacle Avoidance for Quadcopters. In *AIAA Scitech 2019 Forum*, San Diego, CA, USA, January 2019. doi:10.2514/6.2019-1408.

[8] Subramanian Ramasamy, Roberto Sabatini, Alessandro Gardi, and Jing Liu. LIDAR obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. *Aerospace Science and Technology*, 55:344–358, 2016. doi:10.1016/j.ast.2016.05.020.

[9] Nils Gageik, Paul Benz, and Sergio Montenegro. Obstacle Detection and Collision Avoidance for a UAV With Complementary Low-Cost Sensors. *IEEE Access*, 3:599–609, 2015. doi:10.1109/ACCESS.2015.2432455.

[10] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991. doi:10.1109/70.88137.

[11] Shulong Zhao, Xiangke Wang, Hao Chen, and Yajing Wang. Cooperative Path Following Control of Fixed-wing Unmanned Aerial Vehicles with Collision Avoidance. *Journal of Intelligent & Robotic Systems*, 100(3):1569–1581, Dec 2020. doi:10.1007/s10846-020-01210-3.

[12] Maxbotix. I2CXL-MaxSonar - EZ Series, 2023. URL `https://maxbotix.com/pages/i2cxl-maxsonar-ez-datasheet`. (accessed on 25 May 2024).

[13] Lightware. LW20/c Manual, 2020. URL `https://www.documents.lightware.co.za/LW20%20-%20LiDAR%20Manual%20-%20Rev%2012.pdf`. (accessed on 25 May 2024).

[14] Lightware. SF45/B Guide, 2021. URL `https://support.lightware.co.za/sf45b/`. (accessed on 02 Jun 2024).

[15] PX4. PX4 Autopilot User Guide v1.14, 2023. URL `https://docs.px4.io/v1.14/en/`. (accessed on 1 Apr 2024).

[16] MAVSDK Guide. Introduction, 2024. URL `https://mavsdk.mavlink.io/main/en/index.html`. (accessed on 11 Sep 2024).

[17] MAVLink Developer Guide. Using Pymavlink Libraries (mavgen), 2024. URL `https://mavlink.io/en/mavgen_python/#using-pymavlink-libraries-mavgen`. (accessed on 11 Sep 2024).

[18] ROS Wiki. MAVROS, 2024. URL `https://wiki.ros.org/mavros`. (accessed on 11 Sep 2024).

[19] PX4. PX4-Avoidance, 2022. URL `https://github.com/PX4/PX4-Avoidance`. (accessed on 30 Mar 2024).