Evaluation of Non-Cooperative Sensors for Sense and Avoid in UAV Systems

Pablo Arroyo Izquierdo
pabariz@etsid.upv.es

Instituto Superior Técnico, Universidade de Lisboa, Portugal

September 2019

Abstract

The number of civil and military applications for Unmanned Aircraft Vehicles (UAV) is increasing in the last years, such as firefighters, search-and-rescue missions or package delivery, among others. Due to their ability of performing a large variety of important tasks with higher manoeuvrability, longer endurance and less risk to human lives, small UAV are particularly suitable. But to carry out these tasks, it is mandatory to guarantee a safe performance and a correct integration into non-segregated airspace. Integrating unmanned aircraft into civil airspace requires the development and certification of systems for sensing and avoiding (SAA) other aircraft. In particular, non-cooperative Collision Detection and Resolution (CD&R) for UAV is considered as one of the major challenges to be addressed.

The new project Enhanced Guidance, Navigation and Control for Autonomous Air Systems based on Deep Learning and Artificial Intelligence started by Boeing at the Center for Aerospace Research (CfAR) requires from definition of the SAA system and a rigorous analysis of it before the system can be developed and eventually certified for operational use.

This paper will be focused on evaluating the capabilities of the non-cooperative sensors for a SAA system, reviewing the different sensors available and the data fusion techniques to merge the information provided by the different sources. Finally, algorithms for visual cameras image processing using machine and deep learning techniques will be developed and compared, with the aim to provide an effective obstacle detection capability.

Keywords: Sense and Avoid ; Non-cooperative sensors, Data Fusion; Machine Learning, Deep Learning

1. Introduction

Unmanned Aircraft System (UAS) is currently a field that utilizes the leading technologies advances and challenges some of the current regulatory and operational practices of major aviation authorities around the world. For that reason, the interest in the UAS has grown over the last years and lots of applications have appeared in the scene, not only military but also civil or commercial. The fact that the human pilot is not in the aircraft introduces the first problem: they can't see the surrounding environment to understand what is happening around the aircraft. That is one of the reasons to introduce a Sense and Avoid (SAA) system.

2. Sense and Avoid Architecture

The core of SAA systems consists of mainly three steps:

1. **Sense**. Methods for surveilling the environment around the aircraft.
2. **Detect**. To analyze the information provided by the sensing methods to determine whether there is any obstacle around the aircraft and it is a threat.
3. **Avoid**. To choose a suitable action for the aircraft to avoid the threat.

Sensors can be cooperative or non-cooperative. Detection can range from alerting the pilot to estimating the motion and trajectory of the obstacle. The avoidance strategy can range from a basic action (e.g., descend) to some complex actions that consider environmental factors. Both, basic and complex actions, can be remotely operated or achieved by autonomous methods.

According to the agreements reached at the *Sense and Avoid Workshops* where US FAA and Defense Agency experts discussed a number of fundamental issues [1], an effective SAA system needs to provide two common services. They are a self-separation service that would act before a collision avoidance maneuver is needed, i.e., ensuring that the aircraft remain with a safe separation from each other, and a collision avoidance ser-

vice to protect a small collision zone and usually achieved by an aggressive maneuver.

To achieve these services, the following list of sub-functions is required as described in [2]:

1. **Detect** any type of hazards, such as traffic, terrain or weather. At this step, it is merely an indication that something is there.

2. **Track** the motion of the detected object. This requires gaining sufficient confidence that the detection is valid and making a determination of its position and trajectory.

3. **Evaluate** each tracked object, to decide if its track may be predicted with sufficient confidence and to test the track against some criteria that would determine whether a SAA maneuver is needed or not. The confidence test would consider the uncertainty of the position and trajectory. The uncertainty could be great when a track has started, and again whenever a new maneuver has first detected. A series of measurements may be required to narrow the uncertainty about the new or changed trajectory. Also, when a turn is perceived, there is uncertainty about how great a heading change will result.

4. **Prioritize** the tracked objects based on their track parameters and the tests performed during the evaluation step. In some implementations, this may help to deal with limited SAA system capacity, while in others prioritization might be combined with the evaluation or declaration steps. Prioritization can consider some criteria for the declaration decision that may vary with the type of hazard or the context of the encounter (e.g., within a controlled traffic pattern).

5. **Declare** that the paths of the own aircraft and the tracked object and the available avoidance time have reached a decision point that does indeed require maneuvering to begin. Separate declarations would be needed for self-separation and collision avoidance maneuvers.

6. **Determine** the specific maneuver, based on the particular geometry of the encounter, the maneuver capabilities and preferences for the own aircraft, and all relevant constraints (e.g., airspace rules or the other aircraft's maneuver).

7. **Command** the own aircraft to perform the chosen maneuver. Depending upon the implementation of the SAA, this might require communicating the commanded maneuver to the aircraft, or if the maneuver determination was performed onboard, merely internal communication among the aircraft's sub-systems.

8. **Execute** the commanded maneuver.

The architecture presented in Figure 1 is designed to meet all these aspects.

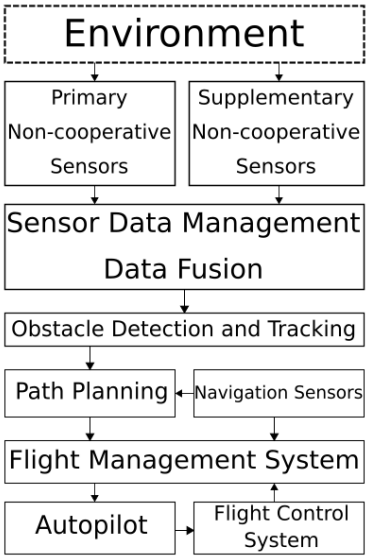


Figure 1: System Architecture

3. Non-cooperative sensors

Any application for SAA needs non-cooperative sensors since the intruder will not always be a well-equipped aircraft and in this case, a cooperative sensor will not be able to detect it. The application studied in this paper would be performed by a small UAV in lower altitudes where non-cooperative traffic is allowed and where ground obstacles could also be important to take into account and, for that reason, to introduce non-cooperative sensors in the architecture takes more importance than ever.

The main advantages and drawbacks of the sensors are summarized as follows. More information can be found at [3].

3.1. Radar

Radars are a good solution for non-cooperative sensing and some of their advantages are: (1) They can estimate all terms of the track state, (2) The reliability of detection and initial detection distance can be regulated by selecting the level of transmitter power, (3) They can deal with bad weather conditions since the degradation is well modeled and it can be well compensated increasing the emitted power or with the proper carrier frequency and (4) apart from distance information, speed information could be obtained if applying Doppler processing.

Nevertheless, they are quite demanding in terms of onboard resources, cost, size, weight, required electric power; thus could be not suitable for small UAV. Also, they have a low resolution.

3.2. LIDAR

LIDARs (Light Detection and Ranging or Laser Imaging Detection and Ranging) operate in a similar way to radars but using the light emitted by a laser instead of using microwave energy emitted by an antenna, however, their resolution is higher.

A LIDAR as a primary sensor can be compact enough to be suitable for small UAV operations but only provides a feasible solution for SAA in short range operations, such as the ones executed by small fixed wing UAV or multirotor, carried out in airspaces where the velocity is very limited. Another solution would be to use them as secondary sensors in conjunction with EO cameras in order to perform the determination of range and range rate for close objects. The information that can be obtained from a LIDAR is angular and distance information.

3.3. Acoustic sensor

Acoustic sensors are usually microphones, that measure the level of acoustic pressure emitted by an object, such as the engine of an intruder aircraft. However, acoustic sensors are strongly dependent on the type of engine and environmental conditions. There are a lot of noise sources that are not a threat, which can disturb the transmission of the intruder's sound and they are very difficult to predict. Even the speed of sound is dependent on the thermodynamic properties of the material that transmits the wave, which leads to echoes, distortions and nonlinearities, and, in turn, to false alarms or missed detections. The recognition of the threat is then performed by estimating the spectral signature of the sound received in order to perform a correlation analysis with the spectral signature of an aircraft engine, but problems in the transmitted sound can persist anyway.

For that reason, they are not a good option as a primary source of reference for a SAA system. However, they can be a low-cost option as an auxiliary source.

3.4. EO camera

Electro-optical (EO) systems are systems that use a combination of electronics and optics to generate, detect, and/or measure radiation in the optical spectrum. They can operate in the visible and IR wavelengths. EO cameras are a good primary source of information for small UAV flying into class G airspace with the proper weather conditions and can be combined as a secondary source with other sources such as radars or LIDARs to increase the overall angular resolution and the data rate. One of the major drawbacks is that they can suffer from lots of noise, which can be compensated while processing the image, but they provide a high resolution and long range solution.

4. Data Fusion

Data fusion is the process of combining information from several different sources to provide a robust and complete description of an environment or process of interest. Data fusion becomes a key element when dealing with different sensors, different sources of information with different characteristics. For a configuration with a radar, LIDAR and visual EO camera, a schema such as the one in Figure 2 is proposed.

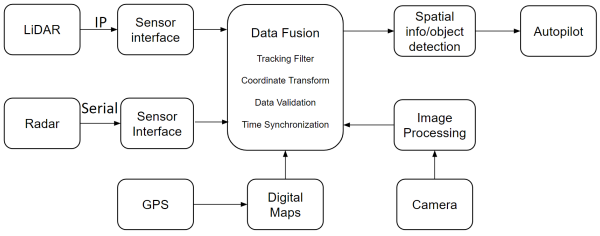


Figure 2: Data fusion schema

4.1. Classification

Data fusion architectures can be classified as distributed or sensor-level and centralized or central-level.

- **Distributed.** Measurements of each source node are processed independently before the information is sent to the fusion node. Therefore, each node provides an estimate of the object's state based only on its local views, and this information is the input to the fusion process, which provides a fused global view. Sensor-level fusion reduces the exchange of data between nodes, paying the cost of a higher computational load.

- **Centralized.** The fusion node resides in the central processor that receives information from all input sources. Therefore, all fusion processes run on a central processor that uses the raw measurements provided from the sources, which are combined to obtain a single set of tracks. Central-level fusion major drawback is that if a sensor measurement is degraded, it affects the entire estimation process. Globally, the computational load is lower than in sensor-level architecture, however, generally more data have to be exchanged between network nodes.

4.2. Data association

It is the first step within the data fusion process. The data association techniques aim to determine from a set of measurements which ones correspond to each target. The following algorithms give a good performance for the data association task:

- K-means
- Probabilistic Data Association
- Joint Probabilistic Data Association
- Multiple Hypothesis Test
- Distributed Joint Probabilistic Data Association
- Distributed Multiple Hypothesis Test
- Graphical Models

4.3. State estimation

From a set of redundant observations for one target, the goal is to find the set of parameters that provides the best fit to the observed data. It is also the task of these algorithms to determine the state of the target (such as position) normally under movement for the SAA applications, given the different observations of each sensor. Thus, these techniques can fall under tracking techniques. The next methods are used for state estimation:

- Maximum Likelihood and Maximum Posterior
- Kalman Filter, Extended Kalman Filter and Unscented Kalman Filter
- Particle Filter
- Distributed Kalman Filter
- Distributed Particle Filter
- Covariance Consistency Methods (Covariance Intersection and Covariance Union)

Nevertheless, state estimation methods could be divided into two groups: linear dynamics and non-linear dynamics, which is the case of the UAS SAA application, then, an algorithm an algorithm capable of cope with these non-linearities must be chosen, such as the Extended Kalman Filter, used in many literature such as [4].

4.4. Fusion decision

These techniques aim to make a high-level inference about the events and activities that are produced from the detected targets. These techniques use the knowledge of the perceived situation, provided by many sources and in different ways depending on the data fusion application. It is the fusion itself. The fusion process requires to reason while taking into account the uncertainties and constraints of the system. Algorithms used in this field to perform the decision-making are:

- Bayesian Methods
- Dempster-Shafer Inference
- Abductive Reasoning and Fuzzy Logics
- Semantic Methods

Fuzzy logic algorithms have been used in sensor fusion applications, e.g. for LIDAR and camera fusion in [5].

4.5. Other solutions

Other way to use the information of two or more sensors to enhance the performance of a single one, is presented in the schema in Figure 3.

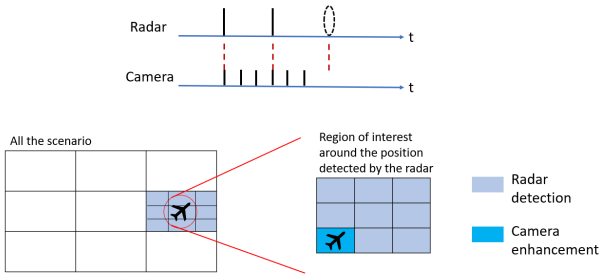


Figure 3: Schema of a camara-based enhancement of the radar measurement

It is proposed a radar as a primary source of information. The radar has as an advantage among LIDARs a longer detection range, nevertheless, it has a low resolution, i.e. in a given scenario the output *pixel* is very large. As a secondary source, a camera. A camera is also a long range solution with high resolution but it is more likely to produce a false detection due to a cluttered environment. Combining both sensors like in Figure 3 helps to overcome the limitations of each sensor.

To avoid getting a false detection from the camera, its information is only employed when a measurement from the radar is received. However, this radar measurement would not be as accurate as desirable, then the camera helps to enhance it. Instead of running the algorithm for the camera obstacle detection over all the scenario, it is run only in a region of interest created around the position measured by the radar, reducing considerably the chances of a false alarm. Therefore, an accurate position of the target will be obtained by the camera.

5. Machine and Deep Learning for obstacle detection using visual electro-optical sensors

The major part of the SAA architectures have a camera in their configurations, thus, it results interesting to find out a way to process their information. The use of artificial intelligence (AI) is a very interesting approach to solve this problem. In particular, implementations using Support Vector Machine (SVM), a machine learning technique, and Convolutional Neural Networks (CNN), a deep learning technique, are compared.

5.1. Faster Regions with Convolutional Neural Networks for aircraft detection

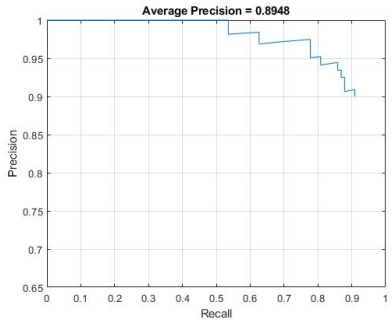
Among the algorithms that use CNN for image classification, the faster Regions with CNN (R-CNN) is, as it is indicated by its name, not only the faster one but also the most accurate and thus the most likely to produce good results in an onboard system. For those reasons, it is the one used. The goal with the implementation of this technique is to detect aircraft in the sensor image. For that, the algorithm has to be trained for the object class that it has to detect. Thus, in the end, it would be able to detect that class in the image. The training dataset is extracted from [6]. In total, 960 images with the class *aeroplane* have been found, reserving 99 of them for testing. The training has been conducted in order to optimize both accuracy and computational time during the classification and detection. The parameters summarized in Table 1 have proved to obtain the best results.

Results for the detection using those parameters in terms of accuracy and time are shown in Figure 4. The characteristics of the computer where the results are obtained from are AMD Ryzen 5 2600X Six-Core (3.60 GHz) CPU, 16 GB RAM and NVIDIA Quadpro P2000 GPU.

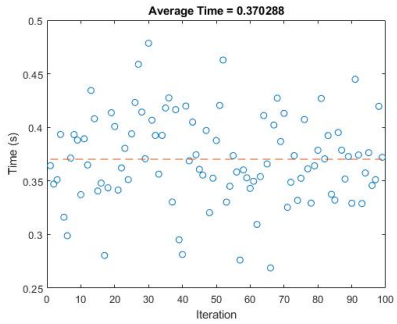
Finally, the implementation of this method will return an easily interpretable array of size $numberObstacles + 4 \cdot numberObstacles$, being the first element the number of obstacles, the next four the X coordinate and Y coordinate of the top-left corner of the bounding box, the height and the width of that bounding box, respectively, and so on until reaching the total number of obstacles.

5.2. SVM and edge detection for obstalce detection above the horizon

An edge detection algorithm is a very fast way to detect objects, nevertheless, it can suffer a lot from ground clutter and thus produce a lot of false detections. There is where the SVM comes in. The machine learning technique will be used for classifying both ground and sky, then the line that best separates them, i.e, the horizon line, will be found and finally the edge detection algorithm will work only in the sky part of the image.



(a)



(b)

Figure 4: Accuracy and prediction time with the options in Table 1

To perform the training it was created a dataset of 202 images of each class. Nevertheless, with such an amount of images, the file created after the training was too heavy which made the classification computationally expensive. Thus, the results that will be shown are built with a dataset of 29 images of each class, that provides also a good performance in the major part of the tested images and reduces the computational time one order of magnitude.

In the case of machine learning is the programmer the one who establishes the features that will lead to the classification, then, to select them is the first step of the training algorithm. The feature selected are:

- Hue: value from 0 to 1 that corresponds to the color's position on a color wheel. As hue increases from 0 to 1, the color transitions from red to orange, yellow, green, cyan, blue, magenta, and finally back to red.
- Saturation: amount of hue or departure from neutral. 0 indicates a neutral shade, whereas 1 indicates maximum saturation.
- Maximum value among the red, green, and blue components of a specific color.
- Range value (maximum value to minimum value) of the 3 by 3 neighborhood around the corresponding pixel in the input image.

Parameter	Value
Network	ResNet-18
Solver	Stochastic Gradient Descent with momentum optimizer
Initial Learning Rate	10^{-3}
Maximum number of epochs	30
Mini-batch size	1
Detection threshold	0,2

Table 1: Selected parameters

- Standard deviation of the 3 by 3 neighborhood around the corresponding input pixel.
- Entropy value of the 9 by 9 neighborhood around the corresponding pixel in the input image.

Once the sky segmentation is performed using these data and the horizon line is found, the obstacle detection is performed by detecting the edges of them, which results in a binary image with the edge pixels of the object candidates to be an obstacle with the value of 1 and the rest 0. The objects are filled and then, if there is ground in the image, it is removed. Therefore, the part of the image below the horizon line is not considered for the detection. The output is collected in an easily interpretable array of size $numberObstacles + 3 \cdot numberObstacles$, being the first element the number of obstacles, the next three the X coordinate, Y coordinate and radius of the obstacle, respectively, and so on until reaching the total number of obstacles.

5.3. Results

Some results of both implementations are shown in Figure 5.

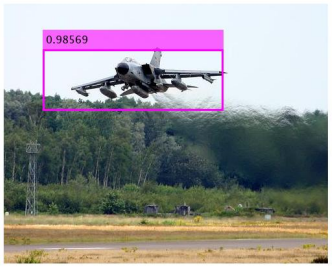
In a scenario with the obstacle is too close, results are shown in Figure 6. It can be noted that the implementation of the SVM + edge detection is most likely to get struggled and thus, data association will be needed to determine which detections correspond to each target.

On the contrary, if they are too far, shown in Figure 7, the faster R-CNN does not get the same amount of confidence when detecting and even it can not distinguish the properties of an aircraft in some cases and thus the detection fails.

The implementation of the SVM + edge detection is not only meant to detect aircraft but also any obstacle above the horizon line. Thus, Figure 8 shows different results in those cases when something else is out there, such as a tree or a building. In order to detect those obstacles with the faster R-CNN algorithm, it would need to be trained for those classes (tree or building, for instance), therefore, it could be extended.

Table 2 summarizes the main characteristics of each implementation.

Therefore, the next conclusion can be reached: to use the implementation of the faster R-CNN as



(a) Faster R-CNN

(b) SVM + edge detection

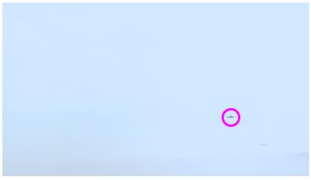
Figure 5: Results of both implementations in a generic scenario. The average time to perform the detection in (a) is 0,4680 s while in (b) it rises up to 7,0793 s

(a) Faster R-CNN

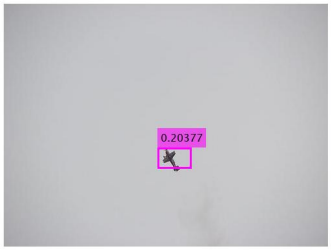
(b) SVM + edge detection

Figure 6: Results of both implementations in a scenario where the obstacle is closer. The average time to perform the detection in (a) is 0,3726 s while in (b) it rises up to 7,1355 s

(a) Faster R-CNN



(b) SVM + edge detection



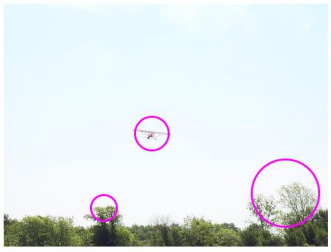
(c) Faster R-CNN

(d) SVM + edge detection

Figure 7: Results of both implementations in a scenario where the obstacle is far away. The average time to perform the detection in (a) is 0,2123 s while in (b) it rises up to 7,4500 s. In (c) it is 0,3824 s and in (d) 7,2569 s

	Faster R-CNN	**SVM + edge**
Average time	0,3626 s	7,1441 s
Range	Short - medium	Medium - long
Other obstacles	No (needs to be trained for more classes)	Yes (only above the horizon line)

Table 2: Comparison of both implementations



(a) Tree detections

(b) Building detections

Figure 8: Other obstacle detection with the SVM + edge implementation

the main resource for aircraft detection, since it is faster and provides better results at short ranges, and to use the implementation of the SVM + edge detection to be run time to time and warn the system of distant threats. Nevertheless, other configurations can be implemented:

- A faster R-CNN trained for different types of obstacles will overcome the limitation of only detecting aircraft, but it will still perform detections in short-medium range if the time is not compromised.
- An edge detection algorithm (without the SVM capability to perform sky segmentation) if the camera is always pointing to the clear sky, that will overcome the time limitation, performing detections in a range of 0,1 to 0,2 seconds.

6. Simulation Environment

Robot Operating System (ROS) is an open source robotic software system that can be used without licensing fees. By being an open source software, there is access to the source code and one can modify it according to their needs, being possible to contribute to the software by adding new modules. One of the main purposes of ROS is to provide communication between the user, the computer's operating system and the external hardware (sensors, robots, etc). One of the benefits of ROS is the hardware abstraction and its ability to control a robot without the user having to know all of the details of the robot. For instance, to move a robot

around, a ROS command can be issued, or custom scripts in Python or C^{++} can cause the robot to respond as commanded. The scripts can, in turn, call various control programs that cause the actual motion of the robot respecting its kinematics and dynamics limitations.

The chosen simulator for this project was Gazebo, which is a 3D physics simulator based on rigid-body dynamics, allowing for a good computational performance. Gazebo is capable of real-time simulation with relatively simple worlds and robots and, with some care, can produce physically plausible behavior.

ROS integrates closely with Gazebo through the *gazebo_ros* package, which provides a Gazebo plugin module that allows bidirectional communication between Gazebo and ROS. Simulated sensor and physics data can stream from Gazebo to ROS and actuator commands can stream from ROS back to Gazebo. In fact, by choosing consistent names and data types for the data streams, it is possible for Gazebo to exactly match the ROS API of a robot. When this is achieved, all of the robot software above the device-driver level can be run identically both on the real robot, and (after parameter tuning) in the simulator.

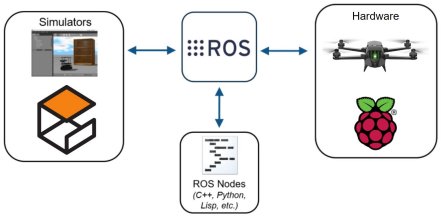


Figure 9: Prototyping

For the current application, Gazebo and the availability of ROS packages allow the integration of different sensors into a quadrotor model and an easy access to their outputs. These sensors can be a LIDAR or a camera, for example. In order to test the capability of the UAS to acquire pictures of the environment and classify them regarding its obstacles, a simulation in Gazebo was conducted.

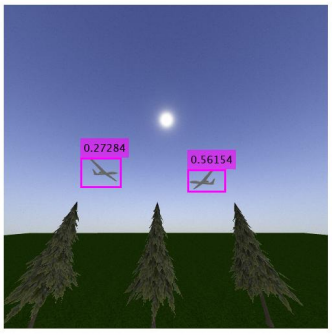
A simple environment was set for simulation purposes, mainly due to the fact of the computational resources needed to run Gazebo. A simple scenario with blue sky and grass ground was firstly defined, to which some obstacles were added. Said obstacles encompassed a set of pine trees already modeled within the Gazebo Graphical User Interface (GUI). Regarding the airborne obstacles, an online model was used [7]. This model was selected based on its resemblance with a commercial aircraft while having smaller dimensions, making the simulation run smoother.

To simulate our UAV, a simple CAD model was developed and converted to a file type that Gazebo can interpret as a solid body subjected to the laws of physics. In order to collect pictures of the environment, a camera model was also implemented with the characteristics listed in Table 3. The camera takes 30 pictures per second composed of squared pixels. Noise is modeled as a Gaussian process with zero mean and standard deviation of 0,007.

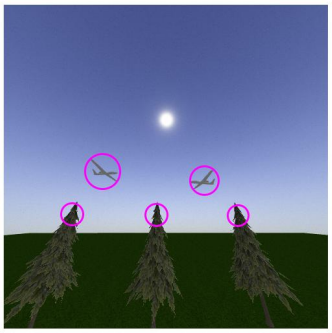
Parameter	Value
Update rate [fps]	30,0
Resolution [pixels $\times$ pixels]	500 $\times$ 500
Pixel format	R8G8B8
Field of view [degrees]	114,6
Range [m]	200

Table 3: Camera characteristics

Some results of the implementations developed in Section 5 running over the images collected from this simulation environment are shown in Figure 10.



(a) Faster R-CNN



(b) SVM + edge detection

Figure 10: Algorithms run over images collected from the simulation environment

It can be seen that although the intruder UAV model is not the most similar to a commercial air-

craft (which could be improved), they are detected in both cases, as well as the tops of the trees overlooking the horizon in the case of the SVM and edge detection implementation, thus proving Gazebo's ability to be the platform where to test the developed algorithms.

7. Conclusion and Future Work

The definition and direction of the project are now clear.

Among the different non-cooperative sensors, three are particularly suitable for the project: radars, LIDARs and visual EO cameras. Some guidance is given in when to use each one. Combinations between radar and camera for long range operations or LIDAR and camera for short range operations are desirable since they can complement each other's limitations. An interesting approach to the camera and radar combination was presented in Section 4.5.

Section 4 dedicated to the data fusion is a complete guide for the future researcher, addressing the steps to follow and the aspects to take into account for the application within the frame of this project. A good choice of algorithms could be k-means for data association, EKF for state estimation and fuzzy logic to make the decision.

Related to the experimental work carried out in Section 5, on one hand, it has been proved that using machine learning is a good technique to remove the ground clutter in the image, nevertheless the algorithm stills being computationally heavy. On the other hand, deep learning techniques show a good performance to detect aircraft plus it can be trained for other obstacle categories. In terms of the results, the implementation of the faster R-CNN is a robust solution for short range operations while the implementation of the SVM and edge detection algorithm obtains better results in long range although it takes more time to compute, thus, a time to time scan with this implementation could be an option while using the faster R-CNN as a primary source of information.

Finally, a simulation environment is being prepared. ROS and Gazebo are a good frame for robot operations and simulations in the context of the project, thanks to the possibility of extending the algorithms tested there to a real robot and the easy integration of different sensing devices, such as a camera or a LIDAR.

With all these, the most immediate activity recommended as future work is:

1. Extending the Matlab algorithms to be used with ROS and Gazebo in real time. Using sockets or compiling the code in Python or C^{++} can serve to achieve this purpose.
2. Extending the faster R-CNN implementation to other classes of obstacles.
3. Trying to achieve a faster performance for the SVM implementation.

The next step on the project would be to test other sensors such as a LIDAR or a radar to figure out how to process their information and to perform data fusion with the information given by the camera processing carried out in this paper. Eventually, the information of the final state of the obstacles obtained after the data fusion will be sent to the online path planning algorithm, where a trajectory to avoid the obstacles is generated and optimized. Integrating the navigation sensors' information within this framework would be desirable as well.

References

[1] sUAS Aviation Rulemaking Committee. *Comprehensive set of recommendations for sUAS regulatory development*. April 2009.

[2] Plamen Angelov. *Sense and Avoid in UAS*. Wiley, 1^{st} edition, 2012. ISBN:978-0-470-97975-4.

[3] Giancarmine Fasano, Domenico Accardo, Antonio Moccia, and David Moroney. Sense and avoid for unmanned aircraft systems. *IEEE Aerospace and Electronic Systems Magazine*, 31:82–110, 11 2016.

[4] Giancarmine Fasano, Domenico Accardo, Anna Elena Tirri, Antonio Moccia, and Ettore De Lellis. Radar/electro-optical data fusion for non-cooperative uas sense and avoid. *Aerospace Science and Technology*, 46:436 – 450, 2015.

[5] Gangqiang Zhao, Xuhong Xiao, Junsong Yuan, and Gee Wah Ng. Fusion of 3d-lidar and camera data for scene parsing. *Journal of Visual Communication and Image Representation*, 25(1):165 – 183, 2014. Visual Understanding and Applications with RGB-D Cameras.

[6] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[7] Elias Gonzalez. Uav/uas concept. https://grabcad.com/library/uav-uas-concept-1, June 2019.