2014 International Conference on Unmanned Aircraft Systems (ICUAS)
May 27-30, 2014. Orlando, FL, USA

Setting Up a Testbed for UAV Vision Based Control Using V-REP & ROS: A Case Study on Aerial Visual Inspection

Miguel A. Olivares-Mendez* Somasundar Kannan* Holger Voos*

Abstract—**This paper focuses on the use of the Virtual Robotics Experimental Platform (V-REP) and the Robotics Operative System (ROS) working in parallel for design, test, and tuning of a vision based control system to command an Unmanned Aerial Vehicle (UAV).Here, is presented how to configure the V-REP and ROS to work in parallel, and the developed software in ROS for the pose estimation based on vision and for the design and use of a fuzzy logic control system. It is also explained how to interact with a virtual and a real quadrotor (QR) to command it for the specific task of aerial visual inspection task. The control system approach presented in this work is based on three fuzzy logic controllers (FLC) working in parallel on an external control loop based on the visual information. The three controllers were designed and tuned to command the vertical, longitudinal and lateral velocities of the UAV. The task to accomplish by the control system is to modify the position of the UAV in real time for the visual inspection of an object or specific parts of a structure. The virtual environment of the V-REP was used to tune manually the control system. Finally, the behavior of the tuned controllers was validated by a set of tests in a real environment with a quadrotor.**

I. Introduction

The control tuning task is one of the biggest discussion topics on the robotics research field. With or without model dependence the design of a control system have to be adapted to each robot or plant to control. Traditionally, specific softwares have been used to simulate the model of the plant to control and the environment, like Matlab simulink. In this well-known software environment it is possible to have graph plotting about what is happening during the simulation, but the environment is closed and there is no possibilities to interactive in real time with the environment itself. In the specific case of robotics, there is an assertive approach done by Peter Corke in Matlab [1], which includes an extended section for quadrotor simulate environment. This is the result of years of investigation to implement all the details of a quadrotor environment. This toolbox is recommendable to understand many robotics control problems and its vision based on outer control loop. The disadvantage of this approach is that is difficult to make any modification to adapt the environment to each user necessities because of its innumerable lines of code. Another disadvantage of this approach is that there is not direct way to use what is implemented in the toolbox with a real robot. This is something that was solved with ROS [2]. The Robotic Operative System provides an open source framework to develop packages to interactive with real sensors, actuators, robots, etc, using a publisher/subscriber system for the communication between them. The main advantage of this pseudo-operative system, is that is easy to use, to develop new packages and to communicate with existing packages. The big number of users and developers in the ROS community make this software more interesting indeed. This software framework will be a milestone in robotics research field in the next years. The Gazebo 3D simulation environment [3] could be installed together with ROS, and allows to test the control approaches, vision algorithms, etc in a 3D virtual environment with simulated robots, sensors and actuators. The main advantage point of this software is that all the packages (algorithms, control, etc) used in the virtual world of Gazebo, could be used with minor changes in the real version of the simulated robot. This fact implies an enormous reduction

*Automation Research Group, Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg.
Corresponding author. Tel.: +352 46 66 44 5478; fax: +352 46 66 44 5356; E-mail: miguel.olivaresmendez@uni.lu (M. A. Olivares-Mendez).

of time in the software implementation part of any research. The disadvantage of the Gazebo simulator is the high requirements of CPU power and graphic card. The next step in 3D environment simulators is the software developed by Coppelia Robotics, the Virtual Robotics Experimentation Platform (V-REP) [4]. In comparison with the Gazebo software, this software can be installed and run without a powerful graphic card and does not required a powerful CPU. The V-REP comes with a large number of robots, sensors and actuators models, and several structures to create a virtual world just dragging and dropping. This simulator allows to interact with the virtual environment during the simulation running time. It is very easy to check how the control system response against disturbances, position changes of the target location or the addition of more robots, objects, structures or sensors in the scene. Another advantage of this software is the bridge with ROS, allowing to use everything developed in the previously mentioned framework. All these characteristics make the V-REP and the connection with ROS the ideal platforms for learn, teach, research and developed with robots.

This work focuses on the vision based control system of a quadrotor in the simulated environment to be used later with a real aircraft. A specific task of visual inspection with a quadrotor has been defined to test the V-REP and ROS connection, how the V-REP experimental platform works with quadrotors, and how the developed ROS packages works. There are many visual servoing applications present on the literature. Different vision-based algorithms have been used to follow a car from a UAV [5], [6],[7],[8]. Visual terrain following (TF) methods have been developed for a Vertical Take Of and Landing (VTOL) UAV [9], [10]. In [11] a description of a vision-based algorithm to follow and land on a moving platform and other related tasks are proposed. A cooperative strategy has been presented in [12] for multiple UAVs to pursuit a moving target in an adversarial environment. The low-altitude road following problem for UAV using computer vision technology was addressed in [13]. People following method with Parallel Tracking and Mapping (PTAM) algorithm has been developed in [14]. The visual inspection approach presented in this work is based on the control of the lateral, longitudinal and the vertical velocities of the quadrotor to modify its position to keep the specific object or the specific part of a structure in the center of the image from a safe predefined distance.

The outline of this paper is structured in the following way. Section II presents the configuration of the V-REP experimental platform. Section III shows the specific vision algorithm used for the visual inspection task. The fuzzy control approach and the developed ROS package for this purpose are explained in section IV. Section V shows the connection between V-REP, ROS and the developed packages to control in a first phase the virtual quadrotor and then a real aircraft. Section VI shows the conclusions and the future work.

II. V-REP Simulation Environment Configuration

In this section is presented the mayor details of the simulation environment of V-REP, as well as the modifications done, and the connects with ROS.

The V-REP presents an easy and intuitive environment to create your own virtual world and to include any of the robots that are provided, as well as objects, structures, actuators and sensors. Also, it allows to create your own robot by adding actuators, joints, sensors and basic forms. An example of a V-REP environment is shown in Figure 6. On the left side of the environment there is a list of robots, sensors, actuators, structures, etc that could be easily include in the simulation scene by drag and drop (model browser). In the next column, there is the scene hierarchy, where all the robots, sensors, graphs and structures of the current scene are represented. A script based on *LUA* script could be associated to each sensor and robot to interact with them, inside the V-REP environment of from the outside (e.g. C++ code or from a ROS package). The central window could be divided in one or more views, we divided it in four views, two external cameras (top and back), and the representation of the velocities and the position of the UAV. More detailed information about this robotics experimental platform is found in [4].

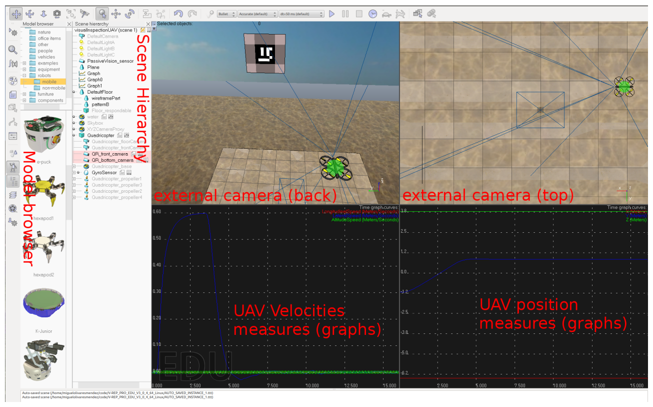

Fig. 1. Capture frame of the V-REP environment.

The V-REP comes with a quadrotor model that is configured to follow position commands. When a vision sensor is used to control a robot the most common way is to use velocity commands. In this work is presented a vision based control system, also known as Image Based Visual Servoing (IBVS), in which the control commands have to be velocity commands. The existing QR model presents some stability problems when the velocity commands are sent, therefore, some modifications needed to be done on the model based on the assumption that the symmetric inertia matrix is a diagonal matrix such as diag(Ixx, Iyy, Izz), as is presented in [15]. Basically a very low value of Ixx introduces poor damping in the roll attitude so purely a velocity control loop does not suppress the faster modes. So the quadrotor is drifting sideways. This is very common in real experimental quadrotors. In such cases it is always needed an outer position loop to hold the position of the quadrotor.

Once, the model was modified the inner control loop has to be adapted to the new model. Four classical PD controllers are defined for the hovering controller of the heading, roll, pitch and height. In this case it is taken into account the current measures of aircraft's angles (roll, pitch and heading), the altitude estimation, and the aircraft's velocities (longitudinal, lateral, orientation and altitude). The inner controllers developed in this phase for the virtual quadrotor were tested in a non exhaustive way. It is checked the correct behavior of the quadrotor hovering and against soft disturbances. The authors really recommend this tool and this process for teaching purpose, because of the high sensibility of the simulator environment and the new quadrotor model against minors changes in the controllers parameters.

The next step is to have the onboard camera feedback available to be processed in a ROS environment. The virtual quadrotor comes with two cameras onboard, but their image feedback can not be sent or published to be used by ROS. For this purpose it is necessary to add one (or more, depends on your needs) new vision sensor and attach it to the quadrotor. In the added vision sensor is possible to modify the child script to share the current image capture by the vision sensor to be processed by the corresponding ROS package. Following the ROS policy to exchange information, the image is published in a ROS' topic.

The specific visual algorithm developed as a ROS package has to get the current frame published, process it and send the extracted information to the control system. All this process is done by different developed ROS packages that will be explained into details in the next sections. There is also another process to be accomplish in the V-REP. The virtual quadrotor must receive the control commands and apply them. It is also done in a child script, but in this case, in the one that is associated with the virtual quadrotor. As well as in the image sharing process, in this case we have to define a ROS' type subscriber to get the control feedback as a new velocity reference for the inner control system.

All the implemented code explained in this section and the model adaptation of the quadrotor to receive velocity commands, as well as a scene example to use them are available on [16]. Most of them are based on the information extracted from the forum web site of Coppelia software [4].

III. Vision Algorithm

The vision algorithm had to obtain the location of the different points or parts from the object, structure or wall to inspect. With this information, the control system approach has to be able to command the UAV from its current position to the place to inspect the centering of the onboard camera on it. The vision algorithm is not the principal purpose of this work, for this reason we use a visual algorithm based on the detection, recognition, and processing of augmented reality (AR) markers or codes. The idea is to have several markers allowing to change the target code to process and changing the position of the UAV based on the selected code. In that way we emulate the selection of different target parts of the structure to inspect. Based on a markers database, the camera calibration parameters, and the size of the code, the algorithm is able to estimate the pose of the camera (quadrotor) respect to the marker, that is the orientation of the camera (the quadrotor) in the three axis, the distance between the camera (the quadrotor) and the code, as well as the lateral and vertical displacement versus the center of the marker. The developed ROS package for the visual algorithm is the adaptation to ROS of the ArUco software [17], a developed C++ library of augmented reality that uses OpenCV. It is an improved Hamming Code based algorithm with an error detection.

The ArUco-ROS package, called *aruco_eye* is subscribed (that is the ROS method to get information shared by other packages) to the specific image streaming publisher from the camera (real or virtual). The current frame is processed and the extracted information is sent by a publisher defined for this purpose.

To use this package with the virtual camera and a real one we just have to include the camera calibration information and the specific topic's name in where the camera publish the images. This could be done by the command line or configuring a roslaunch file. To do the tests in the virtual environment a panel with one of the ArUco codes added as a texture is included, as it is shown in the Figure 2. In the real world the codes where printed and paste into a wall.

The image processing algorithm consists on the estimation of the distance between the ArUco target code and the UAV in the three axis (longitudinal, lateral and vertical distances), as it is shown in the Figure 3.

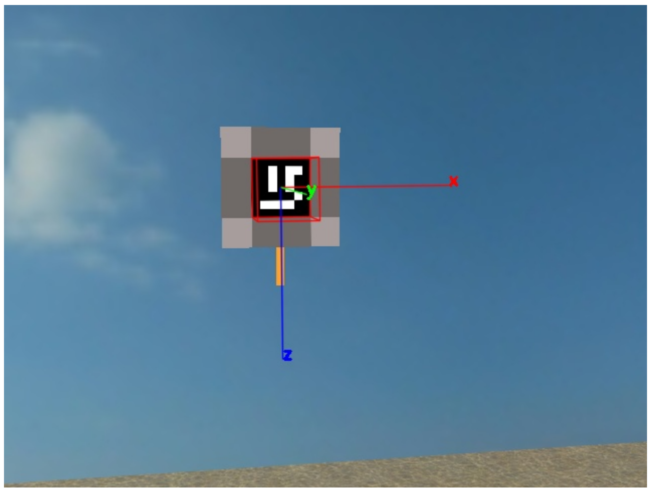

Fig. 2. Virtual image captured by the virtual camera on the UAV and processed using the ROS package of the ArUco library.

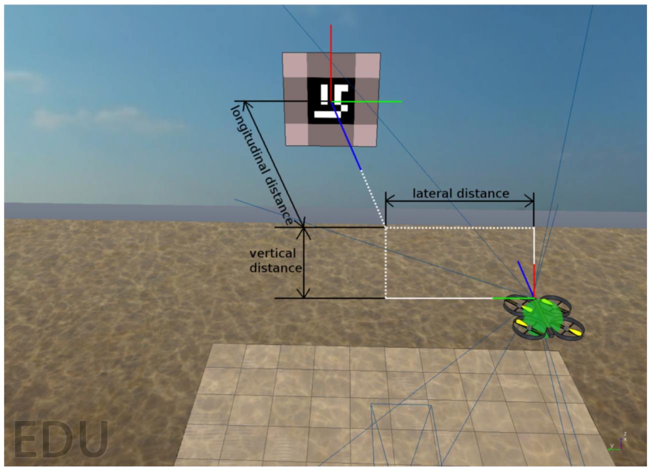

Fig. 3. Explanation of the image processing algorithm.

The developed ROS package presented in this section and some examples of how to use it, are available on [16].

IV. Fuzzy Control System

The visual inspection task defined in this work is based on the capability of the quadrotor to change its position from one specific part of the structure to inspect to another. The UAV must stay hovering in from of each desire location from a safety distance that allows to get high quality images and be aware of any potential collision against the structure produced by winds disturbance or the potential movement of the structure to inspect.

To solve this task a control system approach was designed using three fuzzy controllers working in parallel. One controller is designed to manage the longitudinal speed, to keep the UAV in a specific distance from the object or structure to inspect. The second controller manages the lateral speed of the UAV to keep it, horizontal in front of the center of the object or structure to inspect. And, finally the third controller manages the vertical speed of the aircraft to keep it, vertical in front of what it is wanted to inspect. The three controllers have been designed as a fuzzy PID-like controller, with three inputs and one output. All the controllers' outputs are velocity commands in meters per seconds in the three axis (x, y, z). Figure 4 shows the control loop of the control system approach implemented for this work.

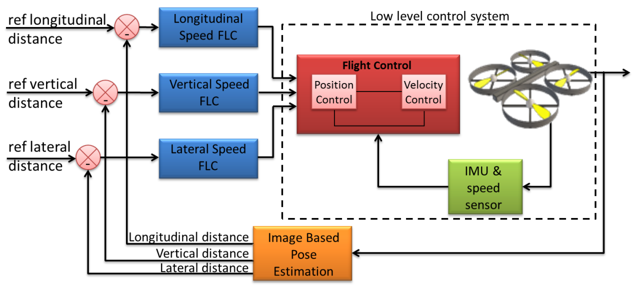

Fig. 4. Control loop of the presented control system approach.

The three controllers were defined in a simply way, with just three sets per each input, and five sets for the output, defined using triangular functions, that means that the rules' base is composed just with 27 rules. The defuzzification method used is the height weight and the inference motor is the product. The rule base was defined based on the heuristic information of the relation of the three inputs. The Figure 5 shows the basic design of the controllers before any modifications.

The Tables I, II, III show the initial definition of the rule base.

To implement the fuzzy controllers in the ROS environment a new ROS package was developed called MOFS-ROS. This ROS package is the adaptation to ROS of the own developed C++ library MOFS (Miguel Olivares' Fuzzy Software) [18]. As well as the C++ library, this new ROS

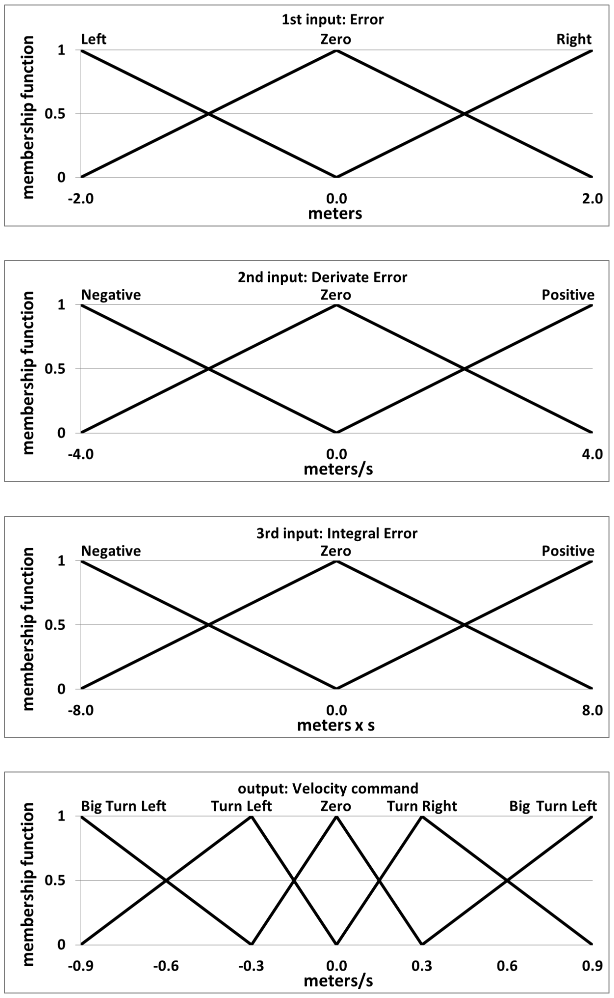

Fig. 5. Initial design of the inputs and output variables for the fuzzy logic controller.

Dot/error	Left	Zero	Right
Negative	Left	Zero	Right
Zero	Zero	Right	Right
Positive	Right	Right	Big Right

TABLE I. BASE OF RULES WITH VALUE FOR THE THIRD INPUT (INTEGRAL OF THE ERROR) EQUAL TO **NEGATIVE**, BEFORE THE MANUAL TUNNING PROCESS

Dot/error	Left	Zero	Right
Negative	Left	Zero	Zero
Zero	Left	Zero	Right
Positive	Zero	Zero	Right

TABLE II. BASE OF RULES WITH VALUE FOR THE THIRD INPUT (INTEGRAL OF THE ERROR) EQUAL TO **ZERO**, BEFORE THE MANUAL TUNNING PROCESS

Dot/error	Left	Zero	Right
Negative	Big Left	Left	Left
Zero	Left	Left	Zero
Positive	Left	Zero	Right

TABLE III. BASE OF RULES WITH VALUE FOR THE THIRD INPUT (INTEGRAL OF THE ERROR) EQUAL TO **POSITIVE**, BEFORE THE MANUAL TUNNING PROCESS

package (MOFS-ROS) allows to implement fuzzy controllers in an easy way, loading the specific characteristics of the controller and the rule base from two different txt files. This package interacts with the roscore and other packages by a service, that provides a new output each time that new inputs were received by the common subscriber/publisher ROS' communication policy. The specification of the subscriber and the publisher is done by a parameter in the *rosrun* command line or by a *roslaunch* file.

This ROS package allows to create new fuzzy controllers using triangular or trapezoidal membership functions, the product or the maximum inference motor and the height weight defuzzification method. Extended possibilities will be included in the next versions of this ROS package. More detailed information of this software is found in [19] in where is explained the C++ library version.

An extra ROS package was developed to put across the information from the visual algorithm, to the fuzzy control system. This package is also in charge of sending the controllers' output to the virtual or real quadrotor. This package receives two different names, when it is used with the virtual quadrotor is called *VREP_VC* (VREP visual control), and *(*visual_control_system). This package is the only one that has to be modified depending the UAV to be used, the number of controllers to use, and the error's signals to control. In this way the ArUco-ROS package and the MOFS-ROS package is kept without significant changes to be used with the virtual or the real environment, or either for future control approaches. This package also contains some extra process when a real AR.Drone quadrotor is in use. This process is called *emergency_ardrone_control*, and it allows to send basic commands to take off, land, and to do an emergency stop to the AR.Drone. Through this process the user can also select the specific ArUco code to set as a target.

Both of the developed ROS packages presented in this section and some examples of how to use it are available on [16].

V. Experiments

The experimental part is divided in two phases. In the first one, the controllers have been tested and tuned in the V-REP environment and its connection with ROS. After that the tuned controllers were tested in the real world in indoor tests with an AR.Drone parrot UAV.

A. Simulator tests

The simulation environment was set by a quadrotor with a looking forward camera, and a panel with one ArUCo code, as it is shown in Figure 6. The location of the quadrotor is set to have a two meters step signal for each controller, when it starts to work.

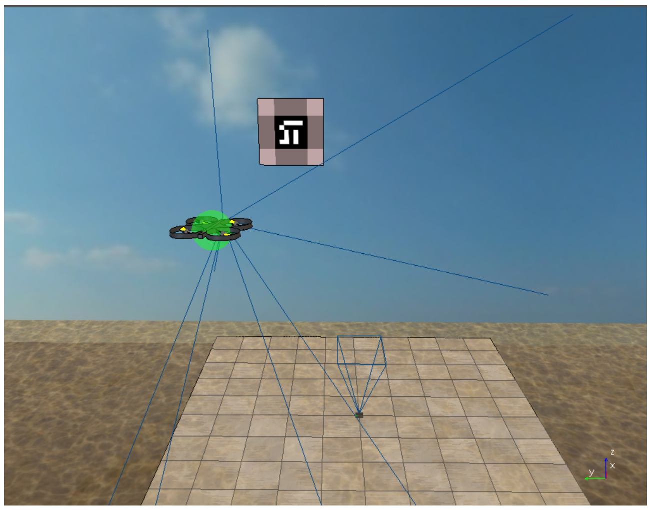

Fig. 7. V-REP environment design with a quadrotor and the ArUco code.

A complete schema of all the processes involved in the simulation environment is shown in Figure 6. Where *vrep* is the V-REP simulator environment, explained in section II, *aruco_eye* is the process of the vision algorithm explained in section III, the *flcLatSp*, *flcLongSp*, and *flcVerticalSp* are the fuzzy controllers for the lateral,

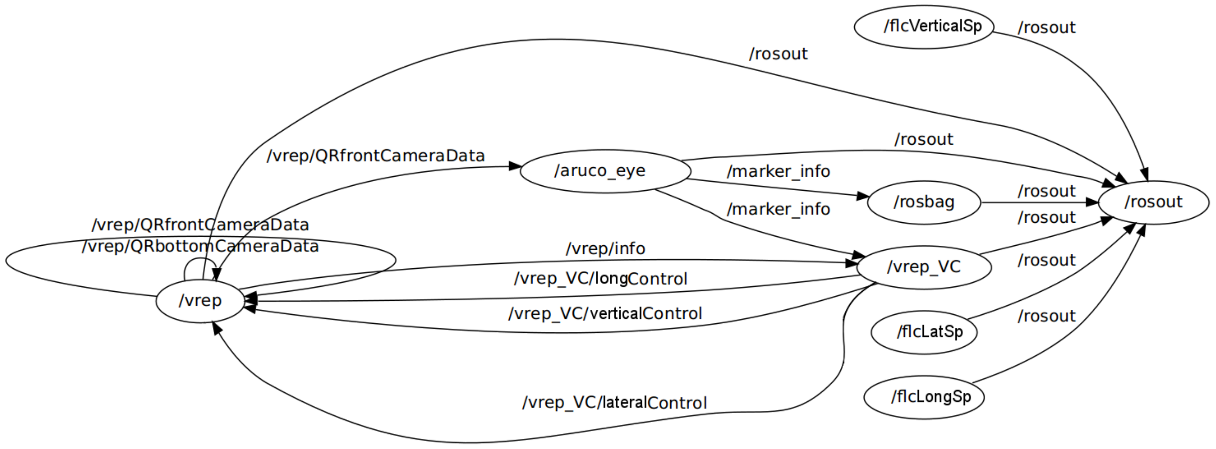

Fig. 6. Interaction between all the actives processes from V-REP and ROS during the simulator tests.

longitudinal and vertical speed respectively (explained in section IV). The *vrep_VC* is the process that shares the visual information to the control system and send the control commands to the virtual quadrotor in the V-REP (explained at the end of the section IV). The *rosbag* is an internal ROS package to store some data of each test. Finally, the *rosout* is the core of the ROS system.

During the manual tuning process, the range of the variables are first adapted and then the output of some specific rules. The V-REP simulation environment and in conjunction with ROS and the developed ROS' packages allow to test the controllers and to modify the different characteristics of the controllers easily. Because of the big similarities between the three controllers, the tuning process is applied to one of the three controllers, and then the results are used in the others. The tunning process is defined by the response of the lateral speed controller responding against a 2 meters step signal. First, the inputs' range have to be adjusted. The Figure 8 shows the response of some of the different configuration tested for the range of the inputs. It is started with the initial controller explained in section IV, represented as *MF1* with the blue line in Figure 8. Then several modifications are tested on the range of the inputs and the output, some of them are shown in the mentioned Figure as *MF2*, *MF3*, *MF4*, being the last one (represented by the green line) the one that gets a better response. The variables definition of for the three controllers is shown in Figure 9.

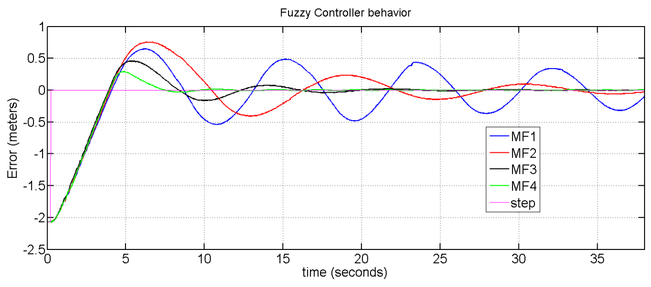

Fig. 8. Response of the different lateral speed controllers during the tunning phase of the inputs' range adjustment.

Based on this controller, tuning process it is continued with the adaptation of the rules' base. In this phase the output of few rules were modified to reduce the overshoot presented in the response of the best controller obtained in the previous phase (the *MF4*). Also for this phase the test is the same than in the previous phase, a step signal of 2 meters. The Figure 10 shows the behavior of some of the controllers tested. The behavior of the different controllers tested are quite similar unless one of them, that reduces completely the overshoot presented in the others (*Rules4*). The Tables IV, V, VI show the final state of the rule base after the manual tuning process. Fromm the initial base of rules presented in section IV, 10 output's rules were changed.

The table VIII shows the results of the behavior of the different controllers shown in Figures 8, 10. Three different measure methods are used for the comparison of the controllers. The first one is

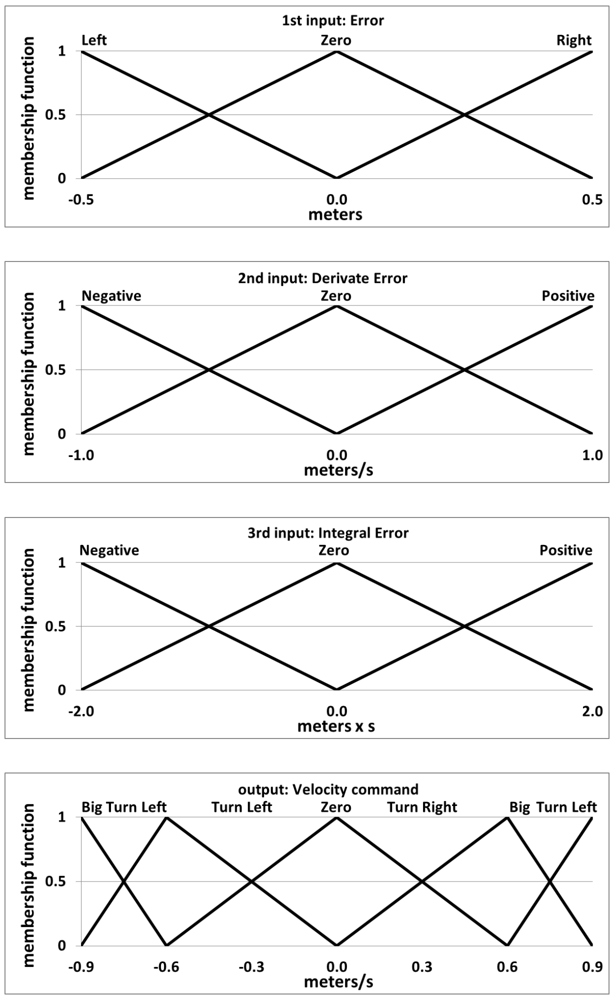

Fig. 9. Final design of the variables of the fuzzy controller after the manual tuning process.

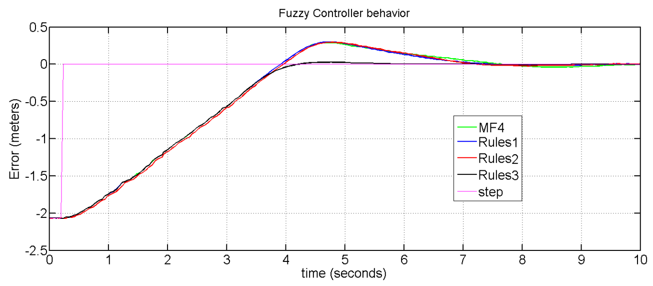

Fig. 10. Response of the different lateral speed controllers during the tunning phase of the rules' base adaptation.

the Root Mean Square Error (RMSE), the second one is the Integral of Time of the Square Error (ITSE), and the last one is the Integral of Time of the Absolute Error (ITAE). The last two are more realistic to compare the behavior between the controllers against a step signal because, both

Dot/error	Left	Zero	Right
Negative	Big Left	Left	Right
Zero	Left	Zero	Right
Positive	Right	Right	Big Right

TABLE IV. BASE OF RULES WITH VALUE FOR THE THIRD INPUT (INTEGRAL OF THE ERROR) EQUAL TO **NEGATIVE**, AFTER THE MANUAL TUNNING PROCESS

Dot/error	Left	Zero	Right
Negative	Left	Left	Zero
Zero	Left	Zero	Right
Positive	Zero	Right	Right

TABLE V. BASE OF RULES WITH VALUE FOR THE THIRD INPUT (INTEGRAL OF THE ERROR) EQUAL TO **ZERO**, AFTER THE MANUAL TUNNING PROCESS

Dot/error	Left	Zero	Right
Negative	Big Left	Left	Left
Zero	Left	Zero	Right
Positive	Left	Right	Big Right

TABLE VI. BASE OF RULES WITH VALUE FOR THE THIRD INPUT (INTEGRAL OF THE ERROR) EQUAL TO **POSITIVE**, AFTER THE MANUAL TUNNING PROCESS

of them penalize the error when the time elapsed is getting bigger.

Controller	RMSE	ITSE	ITAE
MF1	0.6693	12.3629	13.5439
MF2	0.6609	12.0570	12.5403
MF3	0.5806	9.3027	8.0576
MF4	0.5741	9.0961	6.5395
Rules1	0.5740	9.0940	6.5008
Rules2	0.5659	8.8387	6.4193
Rules3	0.5614	8.699	5.7758

TABLE VII. RESULTS OF THE SYSTEM RESPONSE FOR THE DIFFERENT CONTROLLERS SHOWN IN THE GRAPHS.

Once the lateral speed controller is tuned, this information is used to set the other controllers (for the longitudinal and vertical speed). The Figure 11 shows the excellent behavior of the three controllers working in parallel.

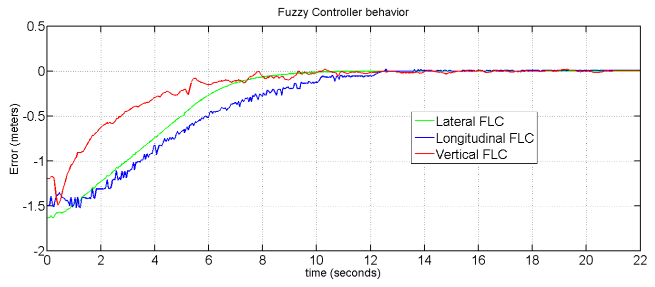

Fig. 11. Response of the three controllers (longitudinal, lateral and vertical velocities) working together in a step signal test.

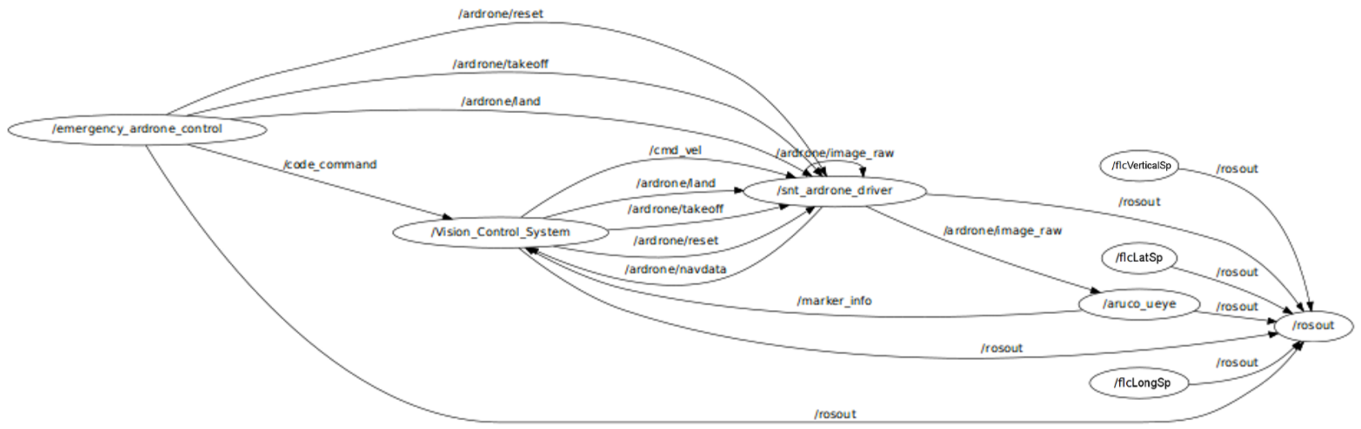

Fig. 12. Interaction between all the active processes of ROS during the real tests.

B. Real tests

After the experiments were done in the virtual environment, the tuned controllers must be tested on the real world with real quadrotor. The real tests of the tuned control system approach were done in the laboratory of the Automation Research Group at the University of Luxembourg. The testbed of the real experimental phase was set by a real quadrotor, the Ar.Drone parrot power edition [20], and five ArUco codes were printed and pasted on the a wall, as is shown in the Figure 13. We put more than one code to change the target code and check in this way the behavior of the control system.

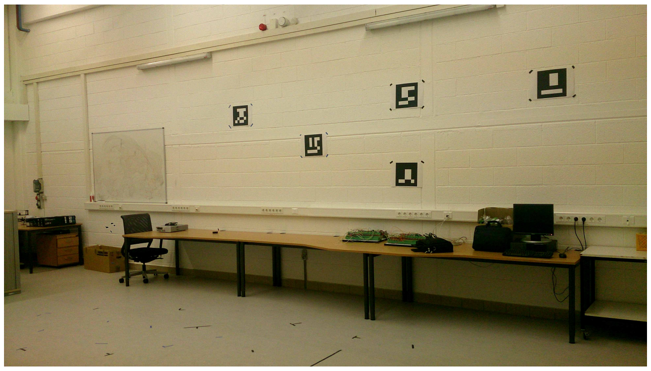

Fig. 13. Environment set for the real tests.

The Figure 14 show the image processing algorithm working with the real image captured by the onboard camera. In this Figure is shown the detection of the five codes at the same time that have been used in the real test.

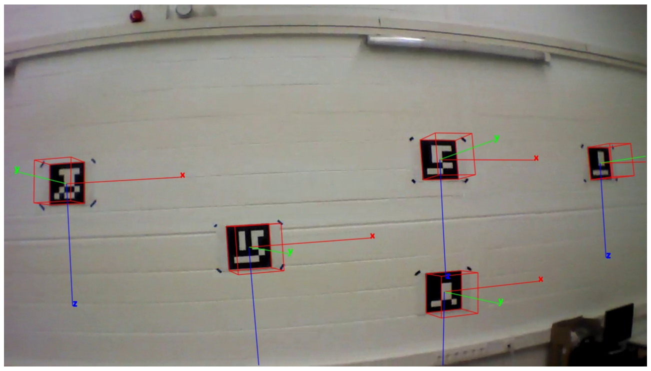

Fig. 14. Real image from the UAV camera processed using the ROS package of the ArUco library.

The used UAV has not an onboard computer, so a ground station is used to run the visual algorithms and the external control loop. As previously mentioned, all the software implementation for the real tests are done using ROS. The Figure 12 shows all the process involve in the real tests, all of them working under ROS. The *snt_ardorne_driver* is used to communicate the AR.Drone parrot with the onboard computer. This package is based on the ROS package *adrone_autonomy* [], in which some minors changes have been done in the adaptation of this package for our own necessities in these tests. The *(*Vision_control_system) process is the adaptation of the *(*vrep_VC) for the real test, that was explained at the end of the section IV. The *emergency_ardrone_control* is a process created specific for the real tests (also explained in section IV) to send basic commands to the AR.Drone and to select the target code. The *aruco_eye*, the *flcLatSp*, the *flcLongSp*, the *flcVerticalSp*, and the *rosout* are the same processes explained previously in the simulation experimental phase.

Different tests were defined to check the correct behavior of the control system approach for the visual inspection task. First, is tested how the control system responds when no changes are applied, that is hovering in front of one code. The Figure 15 shows the response of the three controller in a hovering test.

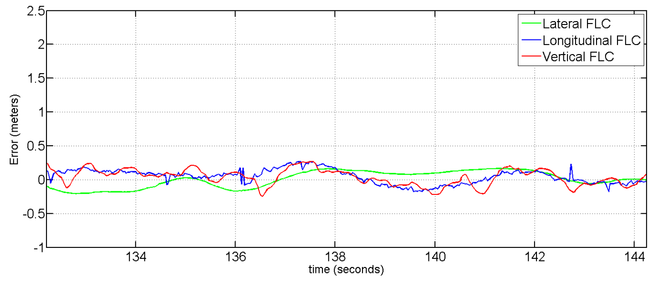

Fig. 15. Response of the three controllers in parallel in a hovering test.

Next, the respond of the control system against disturbances were tested. When the quadrotor was hovering in from of one code we push it by hand displacing its location. The Figure 16 shows two tests of this type, in where is possible to see the good response of the control system to compensate the disturbances.

Finally the control system was tested changing the target code. The Figure 17 shows two of these tests, where is possible to see that the control system responds against a maximum step of 2 meters reducing the error in less than 6 seconds.

The Table VIII shows the results of the tests presented in the graphs. In this Table is shown the size of the steps signals and how the different controllers responded by showing the settling time and the Root Mean Square error.

On of the problems of using a quadrotor without an onboard computer is the lack of communication and the unstable framerate of the images, that it affects strongly in the image processing and control tasks. Figure 18 shows a boxes graph of the frame rate during the tests. This graph shown that in 50% of the cases the framerate is between 12 and 16 frames per second (fps), but with a maximum value of 20 fps and a minimum value of 6.

The videos related to both experimental phases can be found at [21], [22].

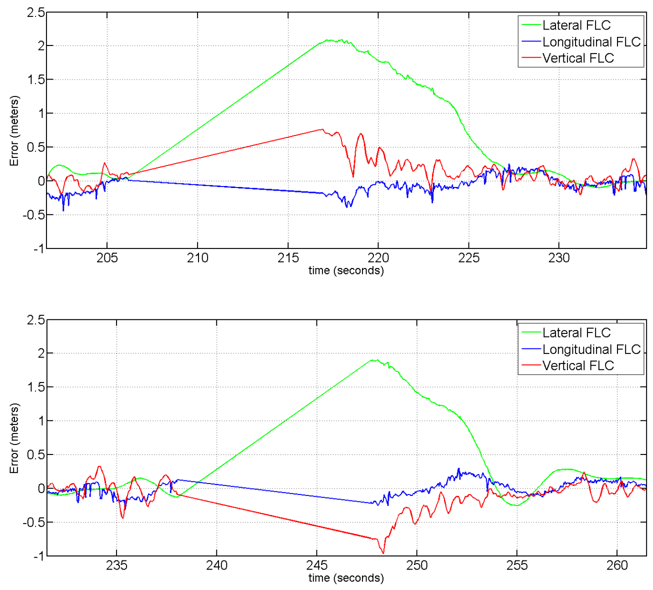

Fig. 16. Two tests of the response of the three controllers working in parallel against disturbances in the three axis.

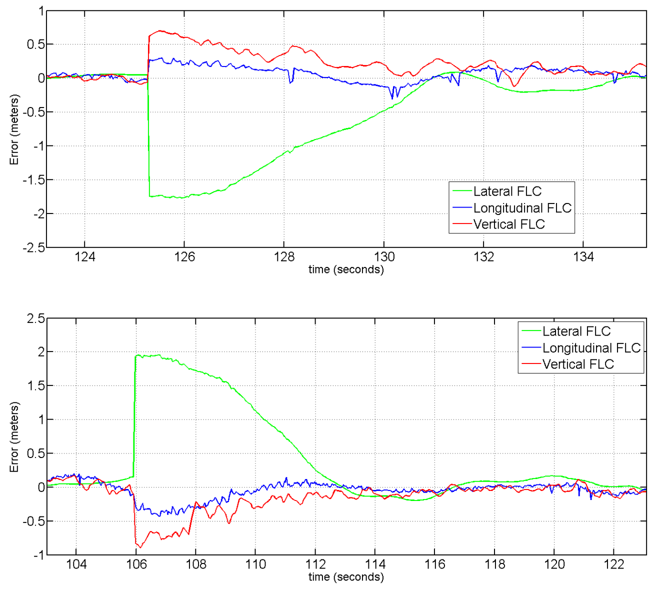

Fig. 17. Two tests of the response of the three controllers changing the code to inspect.

Type Test	Controllers	Step size (meters)	RMSE (meters)	Settling time (sec)
hovering	Lateral	—	0.1085	—
hovering	Longitudinal	—	0.1034	—
hovering	Vertical	—	0.1192	—
disturbance 1	Lateral	2.039	1.0544	8.0
disturbance 1	Longitudinal	-0.1813	0.1249	1.5
disturbance 1	Vertical	0.7540	0.2576	2.9
disturbance 2	Lateral	1.8916	0.8985	7.8
disturbance 2	Longitudinal	-0.2192	0.1194	5.1
disturbance 2	Vertical	-0.7371	0.2937	6.2
moving 1	Lateral	-1.772	0.8670	6.2
moving 1	Longitudinal	0.2922	0.1291	3.7
moving 1	Vertical	0.6976	0.2953	5.0
moving 2	Lateral	1.9292	0.8291	6.3
moving 2	Longitudinal	-0.2733	0.1352	3.5
moving 2	Vertical	-0.8320	0.2721	5.5

TABLE VIII. RESULTS OF THE SYSTEM RESPONSE FOR THE DIFFERENT CONTROLLERS SHOWN IN THE GRAPHS.

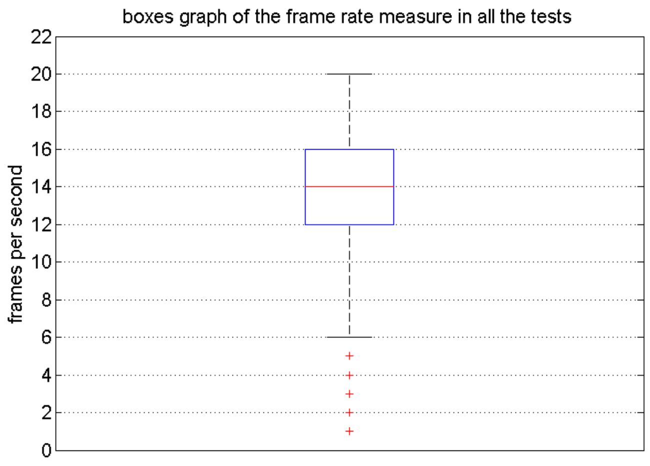

Fig. 18. Framerate measured during the tests.

VI. Conclusions and Future Work

In this paper is presented the process of setting up a testbed to design, test and tune in a control system to command an aircraft based on the image information extracted from the environment, and the developed control system approach for visual inspection. In it is explained how to use and configure the robotics tools, V-REP and ROS, working in parallel, to create a simulated environment to have the visual feedback, and to control a virtual UAV. Three different new ROS packages were developed and explained into details in this work. One for the control part, to design, create and use fuzzy logic controllers. A second one to process the image to extract the information from augmented reality codes in the virtual and real environment. A third one to be in charge of provide the extracted image information to the corresponding controller, and to communicate with the virtual and the real UAV. A control system approach was developed using the presented testbed to control the lateral, longitudinal, and vertical speed of a quadrotor for the visual inspection task. The control system was tuned using the V-REP simulator environment and the connection with ROS. The tuned control system was tested in real tests with excellent results.

The presented testbed for virtual and real environments is an excellent tool for learn, teach and research. All the developed software and the specific modifications done in the V-REP, and the ROS packages are completely available.

In the future, we shall extend this work to other types of control approaches and more complex tasks as tracking moving objects, or inspect a large structure. The authors are working on the developed of a control approach without using the augmented reality codes information, replacing them by using other vision algorithms or other sensors.

References

[1] "Peter corke's robotics toolbox for matlab," 2014. [Online]. Available: http://petercorke.com/Robotics_Toolbox.html

[2] "Robot operating system (ros)," 2012. [Online]. Available: http://ros.org

[3] "Gazebo 3d simulator," 2012. [Online]. Available: http://ros.org/wiki/gazebo

[4] "Coppelia robotics. virtual robotics experimentation platform (v-rep)," 2014. [Online]. Available: http://www.vrep.org

[5] P. Campoy, J. Correa, I. Mondragón, C. Martínez, M. Olivares, L. Mejías, and J. Artieda, "Computer vision onboard uavs for civilian tasks," *Journal of Intelligent and Robotic Systems*, pp. 105–135, 2009.

[6] W. Ding, Z. Gong, S. Xie, and H. Zou, "Real-time vision-based object tracking from a moving platform in the air," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006, pp. 681–685.

[7] J. Gomez-Balderas, G. Flores, L. García Carrillo, and R. Lozano, "Tracking a ground moving target with a quadrotor using switching control," *Journal of Intelligent and Robotic Systems*, pp. 1–14, 2012.

[8] C. Teuliere, L. Eck, and E. Marchand, "Chasing a moving target from a flying uav," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS2011*, 2011, pp. 4929–4934.

[9] B. Hérissé, T. Hamel, R. Mahony, and F.-X. Russotto, “A terrain-following control approach for a vtol unmanned aerial vehicle using average optical flow,” *Autonomous Robots*, pp. 381–399, 2012.

[10] F. Ruffier and N. Franceschini, “Visually guided micro-aerial vehicle: automatic take off, terrain following, landing and wind reaction,” in *Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004 IEEE International Conference on*, 2004, pp. 2339–2346.

[11] D. Lee, T. Ryan, and H. Kim, “Autonomous landing of a vtol uav on a moving platform using image-based visual servoing,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 971–976.

[12] U. Zengin and A. Dogan, “Cooperative target pursuit by multiple uavs in an adversarial environment,” *Robotics and Autonomous Systems*, pp. 1049–1059, 2011.

[13] J. Egbert and R. W. Beard, “Low-altitude road following using strap-down cameras on miniature air vehicles,” *Mechatronics*, pp. 831–843, 2011.

[14] G. R. Rodríguez-Canosa, S. Thomas, J. del Cerro, A. Barrientos, and B. MacDonald, “A real-time method to detect and track moving objects (datmo) from unmanned aerial vehicles (uavs) using a single camera,” *Remote Sensing*, pp. 1090–1111, 2012.

[15] G. Antonelli, F. Arrichiello, S. Chiaverini, and P. Giordano, “Adaptive trajectory tracking for quadrotor mavs in presence of parameter uncertainties and external disturbances,” in *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*, July 2013, pp. 1337–1342.

[16] “Olivares-mendez snt homepage,” 2014. [Online]. Available: http://wwwen.uni.lu/snt/people/miguel_angel_olivares_mendez

[17] “Aruco: a minimal library for augmented reality applications based on opencv: [online]. available:http://www.uco.es/investiga/grupos/ava/node/26,” 2013.

[18] M. Olivares-Mendez, P. Campoy, C. Martinez, and I. Mondragon, “A pan-tilt camera fuzzy vision controller on an unmanned aerial vehicle,” in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, Oct., pp. 2879–2884.

[19] I. Mondragón, M. Olivares-Méndez, P. Campoy, C. Martínez, and L. Mejias, “Unmanned aerial vehicles uavs attitude, height, motion estimation and control using visual systems,” *Autonomous Robots*, vol. 29, pp. 17–34, 2010, 10.1007/s10514-010-9183-2. [Online]. Available: http://dx.doi.org/10.1007/s10514-010-9183-2

[20] “Ar.drone parrot,” http://ardrone.parrot.com, 2013. [Online]. Available: http://ardrone.parrot.com

[21] “Youtube channel of the automation research group at snt - university of luxembourg,” 2014. [Online]. Available: https://www.youtube.com/channel/UCBkpapz06ViwK_cztjwqCAQ

[22] “Isruav project homepage of the automation research group at snt - university of luxembourg,” 2014. [Online]. Available: http://wwwen.uni.lu/snt/research/automation_research_group/projects/isruav