V-REP & ROS Testbed for Design, Test, and Tuning of a Quadrotor Vision Based Fuzzy Control System for Autonomous Landing

Miguel A. Olivares-Mendez*, Somasundar Kannan, and Holger Voos
Automation Research Group
Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg

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 how to use the developed packages in ROS for the pose estimation based on vision and for the design and use of a fuzzy logic control system. It is also shown in this paper a novel vision based fuzzy control approach for the autonomous landing task on a static and on a moving platform. The control system is based on four fuzzy logic controllers (FLC) working in parallel on an external control loop based on the visual information. All the controllers were designed and tuned to command the vertical, longitudinal, lateral, and heading velocities of the UAV.

1 INTRODUCTION

Simulation environments are always an important tool in research, even more in the specific field of robotics. For this research topic there are many of them, and each one has its one limitations The Webots [1] is one of the most used, but there is no free version for academia and the cost is not that cheap. The toolbox of robotics for Matlab done by Peter Corke [2] 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 [3]. 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 framework comes with a 3D simulation environment called Gazebo 3D [4]. The main advantage point of this software is that all the packages (algorithms, control, etc) used in the virtual world of Gazebo, can be used with minor changes in the real version of the simulated robot. This fact implies an enormous reduction 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) [5]. 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 [6], 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 autonomous landing on a moving target 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 [7], [8],[9]. Visual terrain following (TF) methods have been developed for a Vertical Take Of and Landing (VTOL) UAV [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 cooper-

*Corresponding author. E-mail address: miguel.olivaresmendez@uni.lu; Tel.: +352 46 66 44 5478

ative 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 autonomous landing approach presented in this work is based on the control of the lateral, longitudinal,, vertical, and heading velocities of the quadrotor to modify its position to land on a predefined platform.

The outline of this paper is structured in the following way. Section 2 presents the configuration of the V-REP experimental platform. Section 3 shows the specific vision algorithm used for the autonomous landing task. The fuzzy control approach and the developed ROS package for this purpose are explained in section 4. Section 5 shows the experiments done with the developed testbed for tune the controls system approach and to test it for the autonomous landing on a static, and on a moving platform. Section 6 shows the conclusions and the future work.

2 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 7. 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 [5].

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

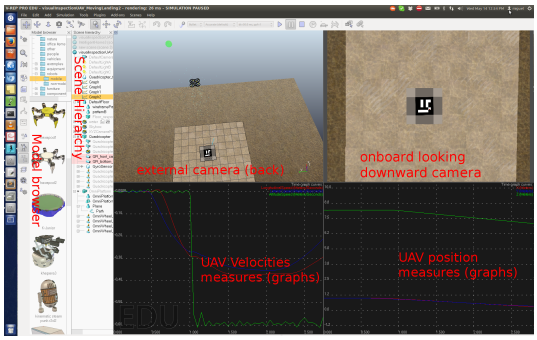

Figure 1: Capture frame of the V-REP environment.

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, heading 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 [5].

3 Vision algorithm

The vision algorithm had to obtain the position of the landing platform. With this information, the control system approach has to be able to command the UAV to center landing platform in the image, orientate it, and approximate it to the landing platform. 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 a moving landing platform with a code printed on it. 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 it is needed to include the camera calibration and the specific topic's name in the location where the camera published 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.

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

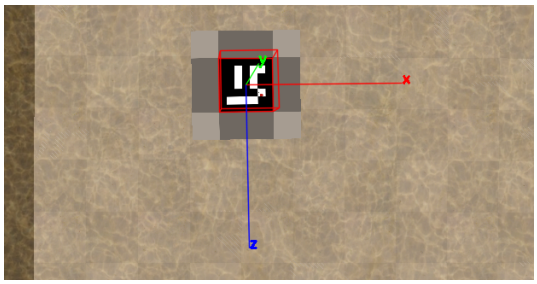

Figure 2: Virtual image captured by the virtual camera on the UAV and processed using the ArUco ROS package.

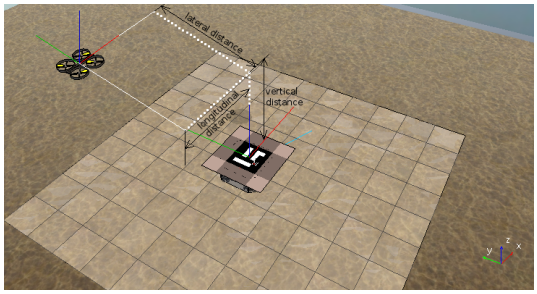

Figure 3: Explanation of the image processing algorithm.

4 Fuzzy Control System

The autonomous landing task defined in this work is based on the capability of the quadrotor to change its position to land over the moving landing platform. The UAV must center the landing platform, and once it is centered start to descend. To solve this task a control system approach was designed using four fuzzy controllers working in parallel. The longitudinal and lateral speed controllers keep the UAV positioning to have the moving landing platform in the center of the image. The vertical speed controller approximates the UAV to the landing platform. The heading controller modify the heading of the UAV to have the lading platform well oriented. The longitudinal, lateral, and heading velocity controllers have been designed as a fuzzy PID-like controller, with three inputs and one output. The vertical speed controller is just a simple fuzzy PD-like controller. The longitudinal, lateral, and vertical speed controllers have as outputs velocity commands in meters per seconds, and for the heading velocity controller is degrees per seconds. Figure 4 shows the control loop of the control system approach implemented for this work.

All the 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 for the longitudinal, lateral and heading velocity controllers and 9 rules for the vertical speed controller. The defuzzification method used is the height weight and the inference motor is the product. The

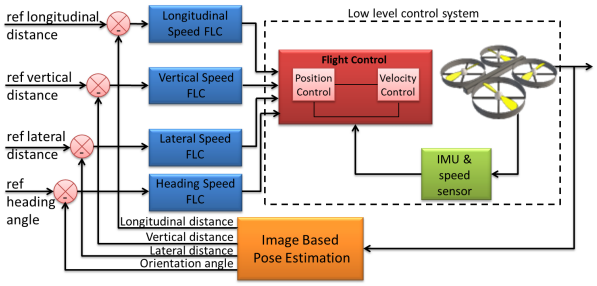

Figure 4: Control loop of the presented approach.

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. In the longitudinal and lateral speed controllers were taken into account the current pitch and roll angles of the UAV in the information obtained by the vision algorithm for the estimation of the translation in x and y axis respected to the moving landing platform.

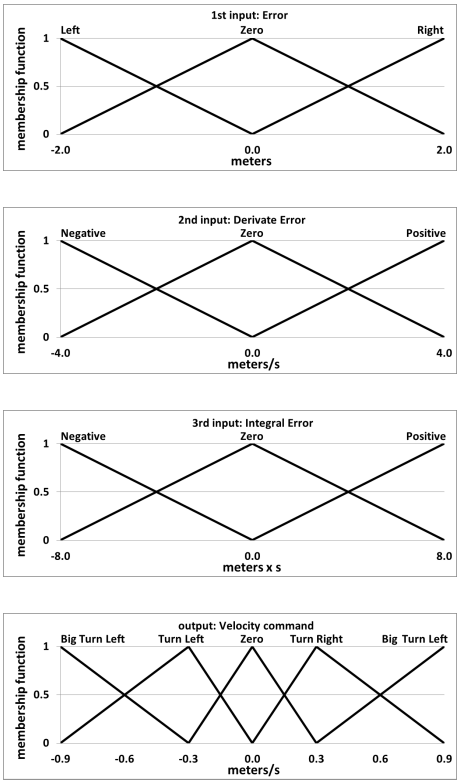

Figure 5: Initial design of the inputs and output variables for the fuzzy logic controller.

The Tables 1, 2, 3 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.

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

Table 1: 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 2: 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 3: Base of rules with value for the third input (integral of the error) equal to **positive**, before the manual tunning process

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 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

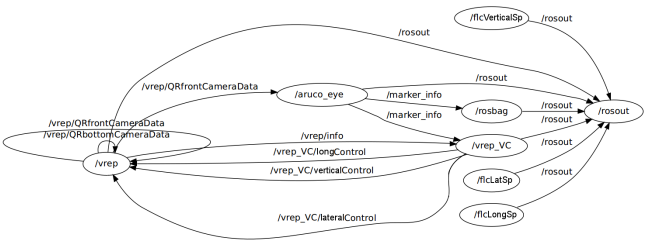

Figure 6: Interaction between all the actives processes from V-REP and ROS during the simulator tests.

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.

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

5 Experiments

The experimental part is divided in two phases. In the first one, the controllers have been tested and tuned using a step signal to keep one code in the center of the image in the V-REP environment and its connection with ROS. After that the tuned controllers were tested all together for the autonomous landing task.

5.1 Tuning Process

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 7. The location of the quadrotor is set to have a two meters step signal when it starts to work.

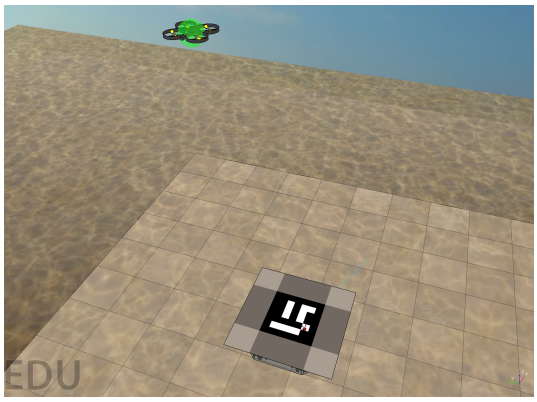

Figure 7: V-REP environment design with a quadrotor and the ArUco code.

A complete schema of all the processes involved in the simulation environment is shown in Figure 7. Where *vrep* is the V-REP simulator environment, explained in section 2, *aruco_eye* is the process of the vision algorithm explained in section 3, the *flcLatSp*, *flcLongSp*, and *flcVerticalSp* are the fuzzy controllers for the lateral, longitudinal and vertical speed respectively, and *flcOrientSp* (explained in section 4). 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 4). 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 lateral, longitudinal, and heading 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 4, 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 the three controllers is shown in Figure 9.

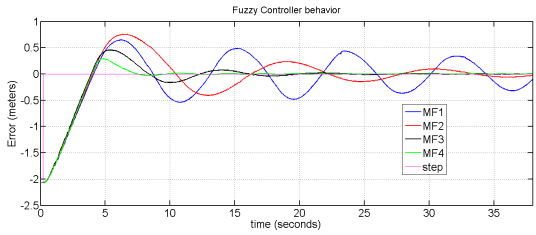

Figure 8: Response of the different lateral speed controllers during the tunning phase of the inputs' range adjustment.

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 4, 5, 6 show the final state of the rule base after the manual tuning process. Fromm the initial base of rules presented in section 4, 10 output's rules were changed.

Once the lateral speed controller is tuned, this information

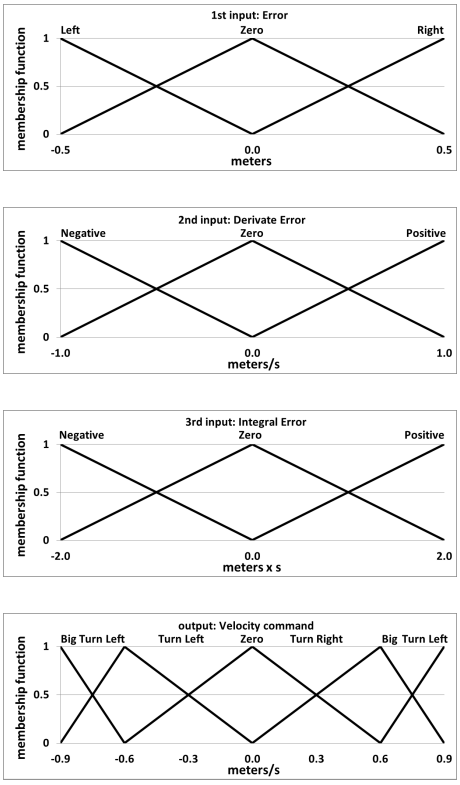

Figure 9: Final design of the variables of the fuzzy controller after the manual tuning process.

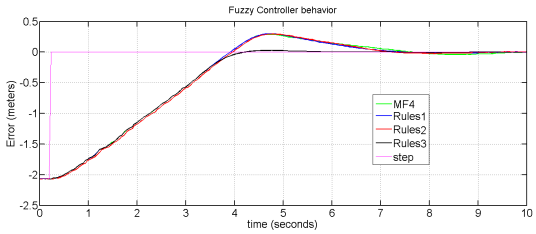

Figure 10: Response of the different lateral speed controllers during the tunning phase of the rules' base adaptation.

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

Table 4: 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 5: 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 6: Base of rules with value for the third input (integral of the error) equal to **positive**, after the manual tunning process

is used to set the other controllers, the longitudinal, vertical, and heading velocities. An adaptation to degrees (inputs), and degrees per seconds (output) was done for the heading controller.

5.2 Autonomous Landing Tests

Once the four controllers are obtained, they can be used all together to control the virtual quadrotor in the specific task of autonomous landing. Two different test are defined. In the first one, the landing platform is static, and in the second one it change its position in the plane x, y and turn over the z axis. In both cases the code is located over a ground robot, to be easy to configure the movements of the landing platform. The initial position of the quadrotor is the same in both cases, 8.0 meters of altitude, and a displacement of 1.6 and 0.9 meters respect to the landing platform for the x, y axis of the quadrotor respectively. The vertical velocity controller has the constraint of not to act until the error in the x and y axis of the QR are reduced under 0.5 meters. This is an strategy to reduce the potential disturbances created by the action of the descend of the Quadrotor.

The next Figures shows the behavior of the control system and the quadrotor for the autonomous landing on a static platform. The Figure 11 shows the evolution of the error for the lateral and longitudinal velocities controllers. This Figure shows how the error was reduced to zero in less than 8 seconds. The Figure 12 shows the evolution of the error for the heading controller.

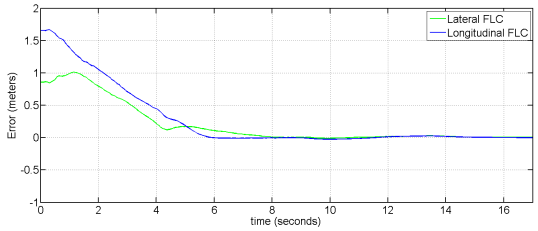

Figure 11: Evolution of the error for the longitudinal and lateral velocities controllers for the autonomous landing test on a static platform.

The Figure 13 shows the evolution of the position of the quadrotor in the three axis x, y, z during this test. This Figure also shows how the vertical velocity controller keep without any action until the error in x and y axis are reduced. Finally, the Figure 14 shows the evolution of the absolute heading of

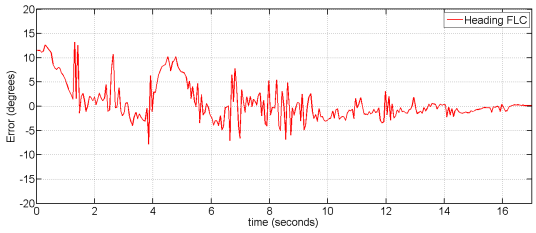

Figure 12: Evolution of the error for the heading velocity controller for the autonomous landing test on a static platform.

the quadrotor, and how the heading is affected by the action of the other controllers.

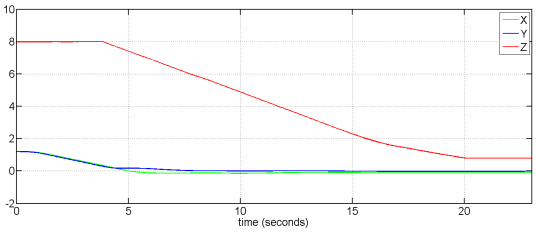

Figure 13: Evolution of the quadrotor position in x, y, z axis in the autonomous landing test on a static platform.

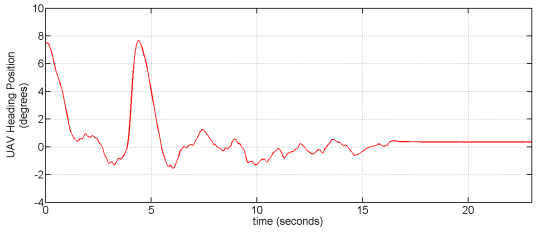

Figure 14: Evolution of the absolute heading of the quadrotor in the autonomous landing test on a static platform.

After the test of the autonomous landing on a static platform, a test on a moving platform was defined. Because the code was set on an omni-directional ground robot (available in the V-REP environment), it is possible to configure different movements and rotations for the landing platform. A set of movements in all direction was set for this test, and also a rotation in z. The evolution of the error for the lateral and longitudinal velocities controllers are shown in Figure 15. The Figure 16 shows the evolution of the error for the heading controller.

The evolution of the quadrotor position is shown in Figure 17. The Figure 18 shows the evolution of the absolute heading of the quadrotor.

The behavior of the control system approach was evaluated using the root mean square error (RMSE) during both tests. The Table 7 shows the RMSE values for the longitudinal, lateral and heading velocities controllers.

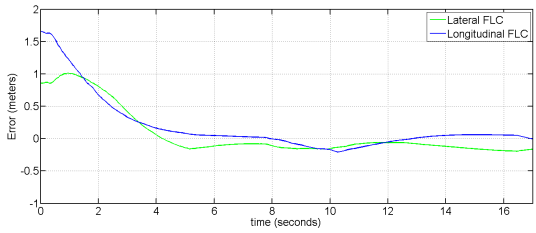

Figure 15: Evolution of the error for the longitudinal and lateral velocities controllers for the autonomous landing test on a moving platform.

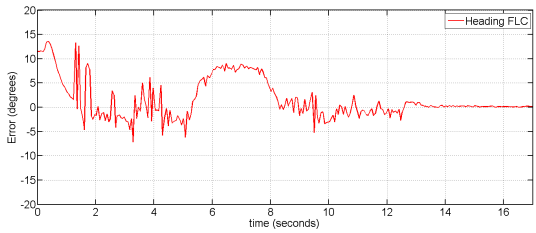

Figure 16: Evolution of the error for the heading velocity controller for the autonomous landing test on a moving platform.

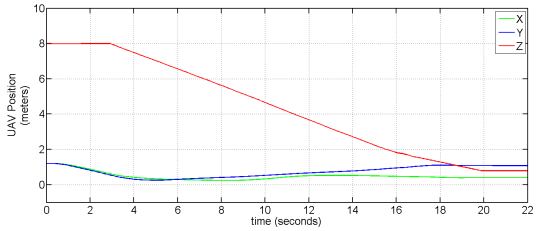

Figure 17: Evolution of the quadrotor position in x, y, z axis in the autonomous landing test on a moving platform.

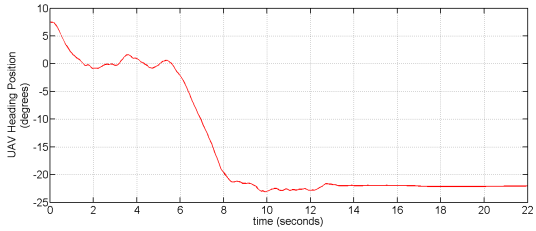

Figure 18: Evolution of the absolute heading of the quadrotor in the autonomous landing test on a moving platform.

Controller type	Static Platform (RMSE value)	Moving Platform (RMSE value)	units
Longitudinal	0.5346	0.4615	meters
Lateral	0.3661	0.3777	meters
Heading	3.9029	4.3728	degrees

Table 7: Root mean square error for the Longitudinal, Lateral and Heading velocities controllers in the two tests presented.

The videos related to the tests presented in this section are available in [20].

6 Conclusions and Future Work

In this paper is presented the process a testbed developed using the V-REP & ROS frameworks to design, test and tune a vision based control system to command an aircraft. A ROS package was developed to design, and use fuzzy controllers in ROs and specifically with a quadcopter. A C++ library for the detection of augmented reality codes was adapted to be used in ROS, and to estimate the relative position of the quadrotor respect to the code. Another ROS package to communicate the visual algorithm and the fuzzy control approach was also developed. The principal idea of this package is to make easy the uses of other visual algorithms and control approaches. It is also presented in this work a novel fuzzy control approach to command a quadrotor for autonomous landing tasks using the testbed, and all the developed ROS packages. The control system approach was tune in the simulator using a step signal, to be tested later in a more complex tasks as the autonomous landing on a static and on a moving platform. The good results obtained were evaluated by the RMSE with values below 40 cm for the lateral velocity controller, below 55 cm for the longitudinal velocity controller, and below 4.5 degrees for the heading controller. The author are focus now in test the control system approach with a real quadrotor for the autonomous landing task.

References

[1] Webots official site. http://www.cyberbotics.com, 2014.

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

[3] Robot operating system (ros). http://ros.org, 2014.

[4] Gazebo 3d simulator. http://ros.org/wiki/gazebo, 2014.

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

[6] Wiki site of the ros bridge with v-rep. http://wiki.ros.org/vrep_ros_bridge, 2014.

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

[8] 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*, pages 681–685, 2006.

[9] Celine Teuliere, Laurent Eck, and Eric Marchand. Chasing a moving target from a flying uav. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS2011*, pages 4929–4934, 2011.

[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*, pages 2339–2346, 2004.

[11] D. Lee, T. Ryan, and H.J. 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*, pages 971–976, 2012.

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

[13] Joseph Egbert and Randal W. Beard. Low-altitude road following using strap-down cameras on miniature air vehicles. *Mechatronics*, pages 831–843, 2011.

[14] G. 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*, pages 1090–1111, 2012.

[15] G. Antonelli, F. Arrichiello, S. Chiaverini, and P.R. 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*, pages 1337–1342, July 2013.

[16] Miguel angel olivares-mendez snt homepage, 2014.

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

[18] M.A. 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*, pages 2879–2884, Oct.

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

[20] Youtube channel of the automation research group at snt-university of luxembourg: Automation research group snt.uni.lu. https://www.youtube.com/channel/UCBkpapz06ViwK_, 2014.