Fine 3D path following of a quadcopter

Manuel Zafra Villar[1] and Jose María Cañas Plaza[2]

Universidad Rey Juan Carlos, Madrid, Spain

Abstract. This paper addresses the design and implementation of a path following controlling system for a drone which relies on 3D localization by visual markers. It has been designed only for indoor flights. Special attention is paid to accuracy of the position estimation algorithm, robustness of the path following controller and real time operation. The path following system is composed of two components, one responsible of the image analysis and 3D pose estimation and another responsible of the drone navigation. It has been experimentally validated both in Gazebo simulator and in a real drone.

1 Introduction

In recent years, there has been an emerging interest in the use of *Unmaned Aerial Vehicles* (UAVs) with applications such as 3D mapping, military tasks, security, inspection [6] or agriculture. Maybe quadcopters are the most popular aerial vehicles now. Currently there are several lines of research and projects with UAVs, including prototypes from big companies like *Project Wing* from *Google* or *Prime Air* from *Amazon*. This growing interest in aerial robotics also is reflected in several international competitions like Int. Micro Air Vehicle Indoor Flight Competition IMAV [1], DronesForGood [2], Mohamed Bin Zayed Int. Robotics Challenge (MBZIRC) [3], which foster the research and development of UAVs technology.

In order to accomplish some of the above mentioned taks, the aerial vehicles must work autonomously without the constant supervision of human operators. Many capabilities are desired in this context, like path following, self-localization, precision landing, obstacle detection and avoidance among others. There are several frameworks for UAVs that help in developing such capabilities and provides support for different hardware platforms. For instance, AEROSTACK [15], PX4 Flight Stack [4], APM Flight Stack [5], Telekyb [20], Hector quadrotor framework [21], beyond other general robotics middlewares like ROS.

One important capability to achieve the UAV autonomy is following of predefined paths in 3D [9, 5, 19]. For this behavior the UAV should continuously

[1] http://www.imavs.org
[2] http://www.dronesforgood.ae
[3] http://www.mbzirc.com
[4] http://px4.io
[5] http://ardupilot.com

know its 3D position and control its movements to advance through the desired 3D path.

3D localization can be achieved in outdoor scenarios with the help of sensors such as GPS, Inertial Measurement Unit (IMU) or altimeter. In indoor environments higher accuracy is needed as there may be multiple obstacles within a few meters. There, solutions may come from external motion capture systems [8, 7, 22] or from onboard computer vision [1, 18], optionally using IMUs too. Typically motion capture systems monitor the position of the vehicles at a high frequency, like 100 Hz. The procedures may vary depending on the camera used (RGB, RGBD, ToF...) and whether the camera is on the scenario or onboard the drone. When using on board cameras, three main techniques can be distinguished. First, *Visual Odometry* in which the position is estimated by calculating the incremental movement between separate pictures extracting feature points of images. Even though this method can provide good short-term accuracy, the error rapidly increases with movement. Second, *Visual SLAM* family of algorithms allows the mapping of the area observed and simultaneously the localization of the camera. MonoSLAM, PTAM, DSO [4], LSD-SLAM and SVO [3] algorithms belong to this family. And third, *Localization based on Markers* (fiducial systems), which is based on the previous knowledge of the environment's map [2]. The map is a colletion of visual markers whose absolute position is known, so when the camera detects any of them the absolute camera position can be obtained estimatind the relative position camera-marker.

The control of the movements can be achieved with several approaches like PID controllers, fuzzy controllers, Internal Model Control [17], Model Predictive Control (MPC) [16], direct vision-based control [19], etc.. Direct vision-based control has been used in other capabilities like precision landing too.

The goal of this project is to develop the first prototype of a vision-based path following system for quadcopters in indoor environments. The navigation will be based on a path tracking method, ensuring a robust position control. In order to accomplish that, the self location will depend on computer vision algorithms relying on visual markers. Given the features of the environment, the system should function with minimal spatial errors in order to avoid obstacles in narrow scenarios.

2 Infrastructure

Several hardware and software pieces have been used in this work. The hardware platform used is *ArDrone2.0* from Parrot (Figure 2). This quadcopter was developed in 2012 as an enhancement over *ArDrone1.0* The onboard computer runs a *Linux* OS, and communicates with the pilot through a self-generated Wi-Fi spot. The onboard sensors include an ultrasonic altimeter enhanced with an air pressure sensor, as well as 3-axis gyroscope, accelerometer and magnetometer, which are used to provide stabilisation. It is also equipped with a 720p front camera and a ventral QVGA sensor.

2.1 AprilTags beacons

AprilTags is a visual fiducial system widely used for tasks including robotics, augmented reality and camera calibration. This system was developed in 2011 by Ed Olson [11]. It is similar to QR Codes, both are two-dimensional bar codes (Fig. 1), but AprilTags were designed to encode far smaller data payloads (between 4 and 12 bits) and introduces a new encoding system which addresses some specific problems with 2D bar codes. It enhances robust detection in more rotation angles and robustness against false positives, allowing them to be detected from longer ranges.

AprilTags library detects any of its markers in a given image, providing the unique ID of the tag as well as its location (height and width pixel) in the image. It also provides the 3D relative transformation between tag and camera, but this was not used here because we wanted to rely only in our existing 3D self-localization component [10], which has been also employed in other projects and includes several mechanisms to increase robustness and accuracy.

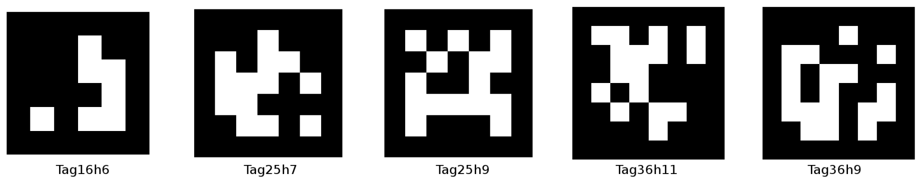


Fig. 1. Sample AprilTags markers

2.2 JdeRobot framework

The system has been developed using JdeRobot[6], an open source robotics and computer vision development framework. In this framework robot applications are built from several distributed nodes which communicates themselves trough ICE (*Internet Comunication Engine* from ZeroC) or ROS messages grouped on explicit *interfaces/topics*. It is fully ROS compatible, provides support to many sensors and actuators, several libraries and tools. Some of them are oriented to aerial vehicles. It simplifies hardware access from applications providing an abstraction layer over the manufacturer's software such as *Ar.Drone SDK* in *ArDrone*.

Progeo is a JdeRobot library for projective geometry which offers functions that relate 2D and 3D points. Having the camera extrinsic and intrinsic parameters, obtained by calibration, there is a *project* function that computes the 2D pixel on which a given 3D point in space projects in the image. There is also

[6] http://jderobot.org

a *backproject* function that computes the 3D ray containing all the 3D points which projects into the same pixel.

CameraCalibrator is a JdeRobot opencv-based tool which obtains the intrinsic parameters of a camera using an intuitive GUI and a calibration pattern. It provides a simple process and a configuration file where settings like calibration pattern, method, number of images taken or delay between images can be changed. It delivers the parameters on a *.yml* output text file. This tool was used to obtain the camera parameters for the real ArDrone camera (Figure 2, right) and the simulated one.

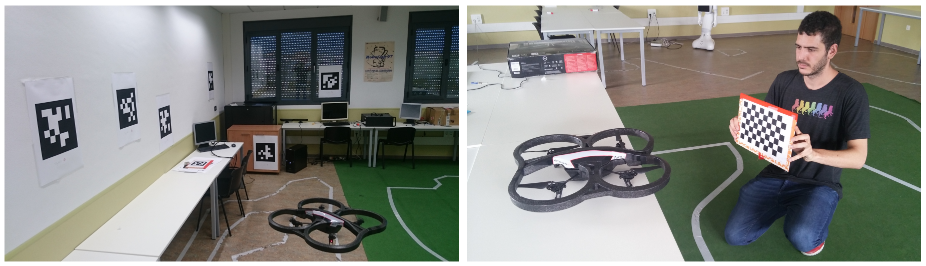


Fig. 2. Real ArDrone2 and calibration of its onboard camera

Ardrone_server is the JdeRobot driver that allows cammunication with the ArDrone quadcopter. It encloses the ArDrone SDK libraries provided by the manufacturer. Its most important interfaces for this work are: *CMDVel*, to send velocity commands; *Ardrone_extra*, that allows execution of complex maneuvers methods like taking off or landing; and *Camera*, for onboard camera images.

Uav_viewer is a JdeRobot tool that allows the user to manually control an ArDrone through a Graphical User Interface (GUI).

JdeRobot includes one plugin for a quadcopter similar to *ArDrone2.0* (Fig. 4) in Gazebo simulator. It has been extensively used in this project, in particular on the initial stages. Gazebo is a multi-robot realistic simulator for both outdoor and indoor environments. It is able to simulate several robots with their sensors in a 3D world, and to emulate physical interactions between the objects in the 3D world thanks to its integration with physics libraries, such as *Bullet* or *Ogre*. The simulator is under constant development, and is maintained by the *Open Source Robotics Foundation*.

3 3D path following system

The design of the developed application is composed of two components as shown in Figure 3. One of them is tasked with image analysis and 3D position estimation and the other one manages drone's movement and controls its position given the estimated location. Communication between both components, and them and

Ardrone_server driver is done through ICE interfaces. All the implementation code is open source and is publicly available [7].

First, *VisualLoc* component receives the images taken by the quadcopter's camera. It performs an analysis of the image looking for the presence of *AprilTag* markers. Once the (several) markers are located, if any, it applies projective geometry to estimate the relative 3D position of the camera to each of the detected markers, and so, each 3D absolute camera position observation. In order to get an unique absolute 3D camera position it applies spatial fusion by a weighted average that filters observations. These weights take into account the 3D distance between the markers and the camera, giving higher weight to those marker detections closer to the camera. Finally, a temporal fusion is carried out by a *Kalman Filter* to provide more robustness and the final estimation is sent to client components.

Second, *Navigator* component recieves the position estimations and generates a combination of velocities that control the movement of the quadcopter. It is a reactive controller. The position control is based on a predefined 3D path that the drone must follow. Given the prevalent horizontal feature of the indoor environments, the algorithm rests heavily on the rotation angle along the Z axis. The algorithm predicts the future position of the vehicle and adjusts the steering angle so the future position error in relation to the path is minimized.

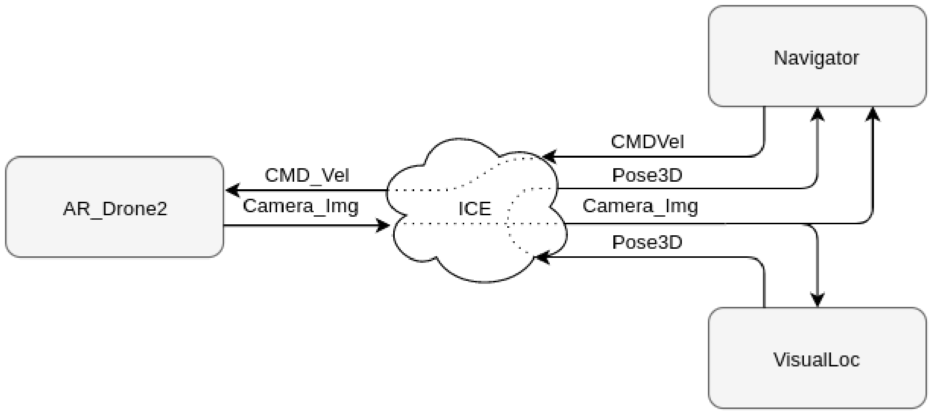


Fig. 3. Global system design

3.1 Beacon based visual 3D localization

The software architecture of the *VisualLoc* component consists of several modules. The *MainWindow* module implements a GUI where the *World* class renders

[7] https://github.com/RoboticsURJC-students/2016-pfc-Manuel_Zafra

a 3D world showing position estimation results. *CameraManager* module is in charge for the computer vision algorithms. The *Sensors* module manages ICE interfaces and shared memory with the other modules. Following JdeRobot's standards, each module runs on a different thread that periodically calls its `update()` method. Calling this method causes the modules to carry out their main tasks in an iterative way.

A *GeometryUtils* library has been developed in order to define a series of geometric calculus methods. Those methods include calculation of the intersection between planes and lines, generation of rotation matrices, or conversions between quaternions and euler angles. Pin-hole camera model has been used due to its simplicity and accuracy. The software structure that defines the camera is *TPinHoleCamera* class, defined in *progeo* library. It contains the camera parameters which must be obtained by previous calibration. Information regarding markers absolute position (map) and parameters of the onboard drone camera are loaded from two text files, *markers.txt* and (*camera.yml*).

The *ProcessImage* method, in *CameraManager* module, processes the 2D image captured by the camera looking for markers, as well as estimates the 3D position of the camera. The markers found are instanced by *MarkerInfo* class, which contains the id, size and position of each marker. Position is stored in two matrices, one with the position of the marker regarding the world and another one with the position of the world regarding the marker. The *AprilTags* detection method is applied to a greyscale converted image, it generates an array containing all detected markers. They are highlighted on the original image and shown on the GUI.

A series of geometric operations are applied on each marker, starting with the *OpenCV*'s function *SolvePnP*, which returns the translation and rotation vectors that determine the position of the marker in relation to the camera. In order to acquire the full RT matrix, *OpenCV*'s function *Rodrigues* is used. The matrix containing the estimated position of the camera referred to the world is obtained by multiplying the calculated matrix and the matrix of the position of the marker in relation to the world.

The position estimations for each marker are stored in an array, then, a *spatial fusion* is performed with a weighted filter. This filter uses a different weight for each estimation based on the relative distance to the marker. Closer markers get a higher weight. Particular weight values were set by experimental testing, analysing the distances where the system lost accuracy. The filter computes all the weights and then obtains a ratio for each estimation following equation (1). The final estimation is computed applying the weighted average of 3D coordinates of estimations, following the equation (2). The rotation angles of the estimations are weighted following the special mechanism of equation (3), as they cannot be directly summed.

$$ratio_i = \frac{weight_i}{weight_{total}} \tag{1}$$

$$[x, y, z]_{fusion} = \sum_{i=1}^{n}([x_i, y_i, z_i] \cdot ratio_i) \tag{2}$$

$$\alpha_{fusion} = \arctan\left(\frac{\sum(\sin\alpha_i \cdot ratio_i)}{\sum(\cos\alpha_i \cdot ratio_i)}\right) \tag{3}$$

Due to occlusions or missing detections some markers may be not detected. After the spatial fusion, a *temporal fusion* is applied using a Kalman filter to provide more robust 3D position estimations on such conditions. With it smoother results are obtained and spike errors are also eliminated. The variation on a single pixel may result in sudden changes in the raw 3D position estimation, the *Kalman Filter* mitigates those sudden variations that may occur. The values of the noise covariance matrices must be adjusted with experimental testing to achieve good results. The final 3D position estimation is sent to *Pilot* component and shown in the world window of the GUI, in conjuntion with the particular estimation from each marker.

3.2 3D position control

The *Navigator* component consists of three modules: *Interfaces*, *Gui* and *Pilot*. System inputs are ArDrone's camera images (*Camera_Img*) and position estimations given by *VisualLoc* (*Pose_3D*). The outputs are a combination of linear and angular velocities (*CMDVel*) and *ArDrone_extra* (for additional commands such as taking off or toggling cameras). Additionally, only in simulation, the absolute true position given by the simulator is available as input so the position estimation error can be calculated.

The *Interfaces* module creates and manages the ICE communication interfaces. The needed parameters, such as IP address and port, are included in a text file that is loaded by ICE. It also manages shared memory between the rest of the modules, managing critical sections with a mutex. The *Gui* module implements the graphical user interface using *PyQt* library. The GUI shows (a) the images captured by the quadcopter's camera, (b) a real-time graph with the error between the vehicle's position and the desired path, and (c) a 3D world window rendered with *OpenGL*. This world shows the quadcopter's position in relation to a coordinate axis and the path to follow, as well as the vehicle's trail. Additionally, several buttons allow the user to manipulate the quadcopter with actions such as pausing or resuming movement, taking off, landing and toggling cameras. In the *Pilot* module all the position information is processed and the velocity commands are iterativelly generated. After some experimental testing, we came to the conclusions that steering angle along vertical axis (yaw) is a key factor for an accurate navigation, and keeping constant lineal velocity does not limitate too much the navigation. Thus, developing a control system around the steering angle was a suitable option.

Our algorithm is based on position prediction, so that steering angle adapts to the predicted error minimizing it. Only horizontal components are considered in the error prediction given that the vertical components are minimal and don't affect steering control. It starts calculating the direction vector between the current position and the desired postion, that is, the corresponding path point following Eq.(4). Then unit vector is computed, Eq.(5) and decomposed. Vertical velocity is computed directly from the Z component of the unit vector and the predefined constant linear velocity v_k, Eq.(7). Horizontal velocity is obtained by calculating the modulus of the X and Y component of the unit vector, Eq.(6).

$$\mathbf{V} = \mathbf{P_{ath}} - \mathbf{P_{ose}} \tag{4}$$

$$\mathbf{u}_v = \frac{\mathbf{V}}{|\mathbf{V}|} \tag{5}$$

$$v_x = |\mathbf{u}_{vxy}| \cdot v_k \tag{6}$$

$$v_z = |\mathbf{u}_{vz}| \cdot v_k \tag{7}$$

The horizontal predicted distance that the vehicle will travel until the next controller iteration is calculated from the horizontal velocity previously obtained and the lapse of time between iterations, Eq.(8). The current steering angle must be calculated in order to compute the future position. Considering that the received position rotation angles are expressed in quaternions, a transformation to euler angles must be done, Eq.(9). Once those values are calculated, future position point is obtained following equations (10) and (11).

$$d_\tau = v_x \cdot \Delta_t \tag{8}$$

$$\theta_z = \arctan^2 \left(\frac{2 \cdot (q_0 \cdot q_3 + q_1 \cdot q_2)}{1 - 2 \cdot (q_2^2 + q_3^2)} \right) \tag{9}$$

$$X_f = d_\tau \cdot \cos\theta_z + x_{pose} \tag{10}$$

$$Y_f = d_\tau \cdot \sin\theta_z + y_{pose} \tag{11}$$

In order to compute the desired steering angle the future lateral error is obtained from the difference between the predicted future position and the desired path point, Eq.(12). Finally, the steering angle is composed of the steering angle needed, δe, plus a steering angle gain that depends on the predicted future error, Eq.(13). The needed steering angle is obtained by calculating the yaw angle needed by the vehicle to face the navigation point. The factor K_g is the gain rate of the steering adjustment and must be obtained experimentally. This adjustment ensures a minimization of the error by correcting vehicle's trajectory and smoothing its movement. The velocity commands sent to the quadcopter through the method `CMDVel()` are v_x, v_z and δ_θ.

$$L_{fe} = -\sin\theta_z \cdot (X_{path} - X_f) + \sin\theta_z \cdot (Y_{path} - Y_f) \tag{12}$$

$$\delta_\theta = \sin \delta_e + K_g \cdot (L_{fe}/v_x) \tag{13}$$

4 Experiments

Experimental tests have been performed on simulated and real drones. Simulations were run in *Gazebo* simulator, creating a custom 3D world representing a typical flat where several AprilTags markers were placed (Figure 4). The two system components were fine tunned separately on simulation. Then, once validated, the whole system was succesfully tested both on simulated and on a real scenarios.

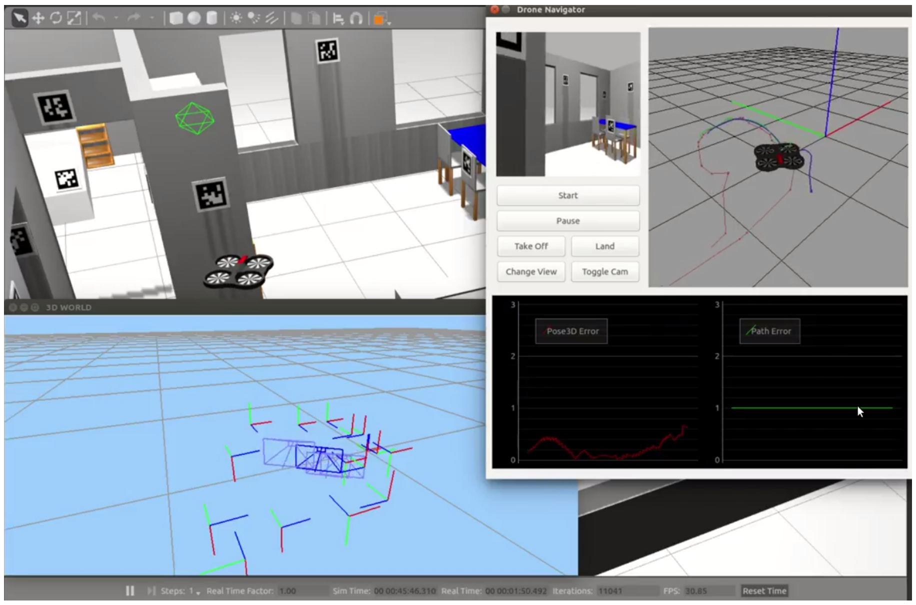
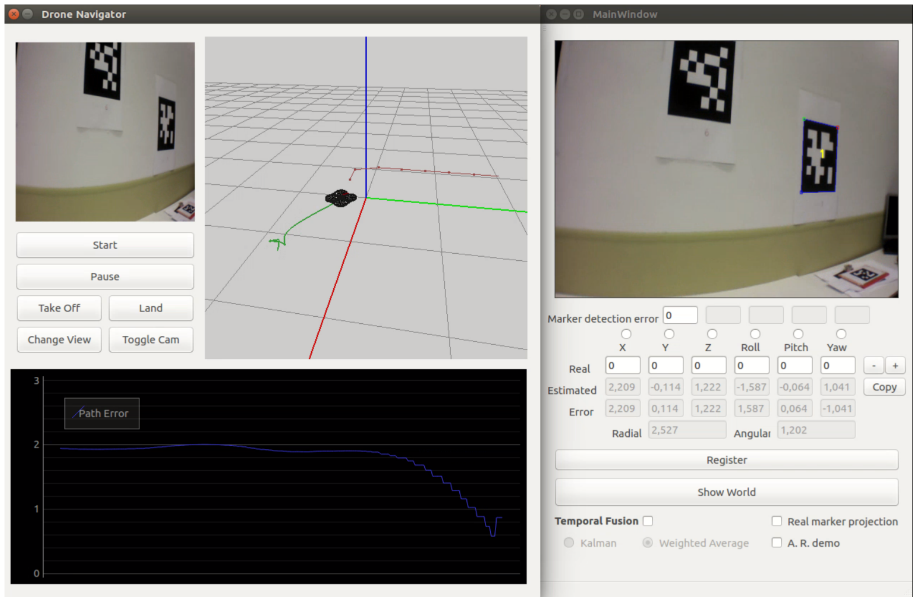


Fig. 4. Position based control of a simulated drone

Self-localization experiments in simulation showed high errors at the beginning. Through analysis, we detected that the algorithm was working properly but some used parameters were not correct. For instance, *Kalman Filter* noise covariance matrices needed to be adjusted to properly model the real noise. In addition, used intrinsic parameters of the camera were not accurate and this caused localization errors. This was solved by a more accurate camera calibration. One of the most important conclusions of these experiments was that the markers needed to be bigger due to an erratic operation of the marker detection process caused by the high mobility of the aerial vehicle. The final marker size chosen was 25cm. Furthermore, it was noted that the accuracy of the position estimation algorithms decays rapidly when the distance of the camera to the marker is larger than 4 meters, confirming the results on [10].

Control experiments in simulation showed good results from the beginning. We had to run several tests in order to find the proper parameters of the control algorithm. Vehicle's optimal speed is 0.1-0.4 m/s, and steering angle gain rate, K_g showed stable results within the range [0.1, 0.3]. The gain rate must be proportional to vehicle's speed in order to have a stable postion control. Lower gain rates are not enough to minimize the position error, while higher rates cause an erratic movement.

The first real scenario experiments showed an unstable system behavior. Several tests were needed to find the correct parameters. Noise matrixes in the self-localization algorithm had to be re-adjusted so they could reflect the new noise model. Several marker sizes were also tested (17cm., 23cm. and 33cm.), obtaining the best results with 33cm. markers. Also, quadcopter camera needed to be calibrated. The position control component also showed erratic behaviour due to the natural drift of the real vehicle and the magnitude of the velocity commands. Even though the drift caused by the slow movement of the vehicle could not be completeley corrected, speed parameters were adjusted so its impact was minimal. Furthermore, the movement of the vehicle caused a blurry effect on cameras, being angular speed a critical factor for this issue.



Marker detection error 0

	X	Y	Z	Roll	Pitch	Yaw
Real	0	0	0	0	0	0
Estimated	2,209	-0,114	1,222	-1,587	-0,064	1,041
Error	2,209	0,114	1,222	1,587	0,064	-1,041

Radial 2,527 Angular 1,202

Temporal Fusion ☐
Kalman [✓ Weighted Average]
☐ Real marker projection
☐ A. R. demo

Fig. 5. Position based control of the real drone

5 Conclusions

We have designed and developed an autonomous quadcopter 3D path following system which is based on a visual markers self-localization technique on indoor scenarios. We have integrated several known technologies, adapting them to our purposes. The system has been succesfully validated on simulated and real quadcopters.

Experimental results show that the system is limited in velocity due to the blurriness in the images when taken at high speed. Another key factor is the size of the markers. Taking into account such limitations and even though noisy data can affect the performance, it has proven to be stable and robust enough to follow simple 3D routes.

Future lines involve the integration of other visual auto-localization algorithms without any markers at all. Furthermore, more exhaustive position control methods could be explored in order to cope with the quadcopter's drift.

Acknowledgements

The research leading to these results has received funding from the RoboCity-2030-III-CM project (Robótica aplicada a la mejora de la calidad de vida de los ciudadanos. fase III; S2013/MIT-2748), funded by Programas de Actividades I+D en la Comunidad de Madrid and cofunded by Structural Funds of the EU. It has also received funding from the Banco de Santander and Universidad Rey Juan Carlos Funding Program for Excellence Research Groups (ref. Computer Vision and Image Processing (CVIP)).

References

1. Wu, A., Johnson, E.N., Kaess, M., Dellaert, F., Chowdhary, G.: Autonomous Flight in GPS-Denied Environments Using Monocular Vision and Inertial Sensors. J. Aerospace Inf. Sys.. 2013 Apr 1;10(4):172-86.
2. Apvrille, L., Dugelay, J.L., Ranft, B.: Indoor autonomous navigation of low-cost mavs using landmarks and 3d perception. Proc. Ocean and Coastal Observation, Sensors and Observing Systems. 2013
3. Forster, C., Pizzoli, M., Scaramuzza, D.: SVO: Fast semi-direct monocular visual odometry, in Robotics and Automation (ICRA), 2014 IEEE International Conference on. IEEE, 2014, pp. 15–22.
4. J., Engel, Koltun, V., Cremers, D.,: Direct sparse odometry, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017
5. Beul, M., Krombach, N., Zhong, Y., Droeschel, D., Nieuwenhuisen, M., Behnke, S.: A High-performance MAV for Autonomous Navigation in Complex 3D Environments, International Conference on Unmanned Aircraft Systems (ICUAS) June 2015 DOI: 10.1109/ICUAS.2015.7152417
6. Nikolic, J., Leutenegger, S., Burri, M., Huerzeler, C., Rehder, J., Siegwart, R.: A UAV System for Inspection of Industrial Facilities, IEEE Aerospace Conference, 2013 10.1109/AERO.2013.6496959

7. Lupashin, S., Hehn, M., Mueller, M., Schoellig, A., Sherback, M., D'Andrea, R.: A platform for aerial robotics research and demonstration: The Flying Machine Arena, Mechatronics 24 (2014) 41–54
8. Jiménez, J., Zell, A.: Framework for Autonomous On-board Navigation with the AR.Drone, Journal of Intelligent & Robotic Systems, January 2014, Volume 73, Issue 1–4, pp 401–412
9. Hehn, M., D'Andrea, R.: Real-Time Trajectory Generation for Quadrocopters, IEEE TRANSACTIONS ON ROBOTICS, 31(4), 2015
10. López-Cerón, A., Cañas, J.M.: Accuracy analysis of marker-based 3D visual localization XXXVII Jornadas de Automática, Madrid, 2016
11. Olson, E.: A robust and flexible visual fiducial system, IEEE International Conference on Robotics and Automation (ICRA), 3400-2407, 2011.
12. Pestana, J., Sanchez-Lopez, J.L., de la Puente, P., Carrio, A., Campoy, P.: A vision-based quadrotor multi-robot solution for the indoor autonomy challenge of the 2013 international micro air vehicle competition: Journal of Intelligent & Robotic Systems, pages 1–20, 2015.
13. J.L. Sanchez-Lopez, J. Pestana, P. de la Puente, R. Suarez-Fernandez,and P. Campoy: A system for the design and development of vision-based multi-robot quadrotor swarms. In Unmanned Aircraft Systems (ICUAS), 2014 International Conference on, pages 640–648, May 2014
14. J. L. Sanchez-Lopez, J. Pestana, P. Puente, and P. Campoy: A reliable open-source system architecture for the fast designing and prototyping of autonomous multi-uav systems: Simulation and experimentation. Journal of Intelligent & Robotic Systems, pages 1–19, 2015.
15. J.L. Sanchez-Lopez, R. A. Suárez Fernández, H. Bavle, C. Sampedro, M. Molina, J. Pestana, and P. Campoy: AEROSTACK: An Architecture and Open-Source Software Framework for Aerial Robotics 2016 International Conference on Unmanned Aircraft Systems (ICUAS) June 7-10, 2016. Arlington, VA USA
16. A. Hernandez, H. Murcia, C. Copot, R. De Keyser: Model Predictive Path-Following Control of an AR.Drone Quadrotor Memorias del XVI Congreso Latinoamericano de Control Automático, CLCA 2014, Octubre 14-17, 2014. Cancún, Quintana Roo, México
17. A. Hernandez, C. Copot and R. De Keyser Tudor Vlas and Ioan Nascu Identification and Path Following Control of an AR.Drone Quadrotor 2013
18. A. Rodriguez-Ramos, C. Sampedro, A. Carrio, H. Bavle, R. A. Suarez Fernandez, Z. Milosevic, P. Campoy: A Monocular Pose Estimation Strategy for UAV Autonomous Navigation in GNSS-denied Environments IMAV 2016
19. T. Nguyen, G. K. I. Mann and R. G. Gosine Vision-Based Qualitative Path-Following Control of Quadrotor Aerial Vehicle 2014 International Conference on Unmanned Aircraft Systems (ICUAS) May 27-30, 2014. Orlando, FL, USA
20. V. Grabe, M. Riedel, H.H. Bulthoff, P.R. Giordano, and A. Franchi: The telekyb framework for a modular and extendible ros-based quadrotor control. In Mobile Robots (ECMR), 2013 European Conference on, pages 19–25, Sept 2013
21. S. Kohlbrecher, J. Meyer, T. Graber, K Petersen, O. von Stryk, and U Klingauf. Robocuprescue 2014-robot league team hector darmstadt (Germany). RoboCupRescue 2014, 2014.
22. N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar: The GRASP Multiple Micro-UAV Testbed, IEEE Robotics Automation Magazine,vol. 17, pp. 56–65, September 2010.