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

A Trajectory Tracking and 3D Positioning Controller for the AR.Drone Quadrotor

Lucas Vago Santana[1], Alexandre Santos Brandão[2], Mário Sarcinelli-Filho[3] and Ricardo Carelli[4]

***Abstract*— In this paper a complete framework is proposed, to deal with trajectory tracking and positioning with an AR.Drone Parrot quadrotor flying in indoor environments. The system runs in a centralized way, in a computer installed in a ground station, and is based on two main structures,namely a Kalman Filter (KF) to track the 3D position of the vehicle and a nonlinear controller to guide it in the accomplishment of its flight missions. The KF is designed aiming at estimating the states of the vehicle, fusing inertial and visual data. The nonlinear controller is designed with basis on a dynamic model of the AR.Drone, with the closed-loop stability proven using the theory of Lyapunov. Finally, experimental results are presented, which demonstrate the effectiveness of the proposed framework.**

I. Introduction

The study of control techniques applied to unmanned aerial vehicles (UAV) has been extensively explored in academic research, with remarkable results already published [1], [2], [3]. Most of the published works have the use of rotary-wing aircrafts as experimental platform as a common factor, much probably because of its versatility to perform in-flight maneuvers in any direction and its capability to hover as well. Due to these characteristics, such aircrafts are ideal platforms to perform tasks like environment exploration, surveillance, search and rescue, aerial photography and others.

However, some of these results are limited, in real-world applications, mostly because they are based on an external computer vision system, like the Vicon Motion Capture System, used to determine the UAV localization. This kind of approach is very popular in works whose maneuvers are quickly and accurately explored, such as in [1], where quadrotors perform ball juggling, and [2], where they precisely fly through a circular hoop.

On the other hand, some works deal with UAV navigation using only the sensors installed onboard the vehicle. This is the case in [4], where a quadrotor with an assembled onboard computer is used for navigation between floors in indoor environments. The work presents a technique for simultaneous localization and mapping (SLAM) through the fusion of information coming from a laser scanner, an inertial measurement unit (IMU) and a video camera.

Using the AR.Drone quadrotor as experimental platform, some interesting works can be cited. In [5], a set of experiments based on servo-visual cooperative position control is presented. There is not an explicit fusion filter, but the inertial data are still used together with visual data to guide the UAV in formation with a ground robot. In [6] the Parallel Tracking and Mapping (PTAM) technique is adopted as the tool to extract the visual data and a Extended Kalman Filter (EKF) is adopted to fuse it with inertial measurements. There, the system is evaluated through flying missions consisting of following some test figures, with the state estimation provided by the EKF. More recently, in [7], a framework has been presented to perform an onboard path-following control, demonstrating the feasibility of using a small single-board computer attached to the AR.Drone as the central processing unit to guide the aircraft.

In such context, this work proposes a method to control the AR.Drone quadrotor in 3D trajectory tracking and 3D positioning tasks in indoor environments. The main goal is to provide to the reader information on how to develop a computational system capable of automatically control a quadrotor, which can be easily adapted for other applications in robotics.

It is noteworthy that several published works have addressed solutions to the problem discussed here, so it is important to highlight the differences of this manuscript, considering similar works. The first relevant difference is the focus on the AR.Drone as a low-cost experimental platform, while most of another published results involving the problem of trajectory tracking are based on more expensive platforms and vision systems [2], [8], [9]. Another important difference is in terms of the methodology applied for sensor fusion. While most works perform the state estimation through an EKF [4], [6], we show how to implement a simple KF to fuse the available sensorial data to track the AR.Drone states. Finally, the last important difference is the proposed nonlinear controller, which is based on a simplified mathematic model to represent the AR.Drone dynamics. Usually, other works are based on complex mathematic models to represent the quadrotor dynamics [10], [11], but our method shows that it is possible to get good results, in terms of control, with the AR.Drone in a simpler way.

To discuss such topics, the paper is hereinafter organized as follows: Section II briefly presents the Parrot AR.Drone quadrotor, its main characteristics, the main reasons for its

*This work was supported by CNPq and FAPES.

[1]L. V. Santana is with the Department of Industrial Automation, Federal Institute of Espírito Santo, Linhares - ES, Brazil lucas@ifes.edu.br

[2]A. S. Brandão is with the Department of Electrical Engineering, Federal University of Viçosa, Viçosa - MG, Brazil alexandre.brandao@ufv.br

[3]M. Sarcinelli-Filho is with the Graduate Program on Electrical Engineering, Federal University of Espírito Santo, Vitória - ES, Brazil mario.sarcinelli@ufes.br

[4]R. Carelli is with the Institute of Automatics, National University of San Juan, San Juan, Argentine rcarelli@inaut.unsj.edu.ar

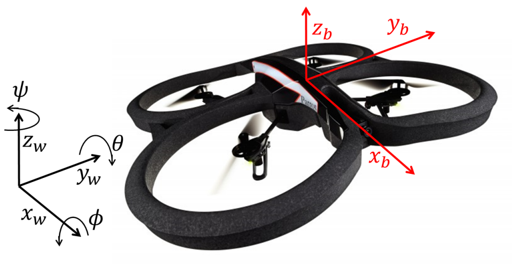

Fig. 1. The AR.Drone 2.0 quadrotor and the coordinate systems adopted ($\{w\}$ and $\{b\}$ are the global and the body coordinate systems, respectively).

choice as experimental platform and a mathematic model describing its dynamics. In the sequel, Section III presents the state estimation technique adopted to use the sensor data available in a KF implementation. After, Section IV discusses the control method adopted, Section V presents an overview of the system architecture, and Section VI shows and discusses some experimental results. Finally, some important observations are pointed out and some future work are outlined in Section VII.

II. The AR.Drone Quadrotor Platform

The experimental platform chosen in this work is the AR.Drone quadrotor, from Parrot Inc, in its version 2.0, which is shown in Figure 1, with the coordinate systems adopted.

It is an autonomous aerial vehicle (a rotorcraft one) commercialized as a hi-tech toy, originally designed to be controlled through smartphones or tablets, via Wi-Fi network, with specific communication protocols. The AR.Drone is easily purchased in the market (at a reduced cost[1]) and it is quite easy to buy spare parts to keep it operative, as well as for maintenance. In addition, Parrot provides a set of software tools, which makes easier to develop communication and control algorithms for the AR.Drone platform, encouraging and supporting users to create applications for it (additional details can be found in [12]).

These are the main reasons for its selection as the experimental platform in this work. We also have know other famous and robust similar platforms, such as the AscTec Hummingbird III, for instance. However, its high price[2] and the difficulty of access make it an unfeasible platform for our demands.

A. AR.Drone Sensors Data

The AR.Drone 2.0 is equipped with two embedded boards. The first one, called the sensor board, contains a set of sensors, such as accelerometers, gyroscopes, magnetometers, an ultrasonic sensor and a barometric sensor. The second one, labeled the main board, is a processing unit based on an ARM Cortex-A8 processor, with 1GHz of clock frequency, running an embedded Linux operating system. This board manages the data coming from the sensor board, the video streamings from a frontal and a bottom cameras, and the wireless communication network of the UAV.

The firmware installed on the main board is capable of performing automatically the procedures of take-off, landing and flight stabilization, besides responding to external motion commands. The AR.Drone also delivers the set of variables

$$\mathbf{q} = \begin{bmatrix} z & v_x & v_y & \phi & \theta & \psi \end{bmatrix},$$

where

- ϕ, θ and ψ represent the orientation angles of the AR.Drone, referred to the global coordinate system;
- z represents the UAV altitude relative to the ground underneath it. The ground is considered a flat surface and the firmware already treats inclinations on ϕ and θ;
- v_x and v_y represent the linear velocities developed by the vehicle, relative to the reference axes x_b and y_b.

These information, as well as how to get access to them, can be found in [12]. The algorithms internally applied to generate them are discussed in [13]. Therefore, it is recommended to the interested reader to check these references to get more knowledge about the technology involved in the firmware of the AR.Drone Parrot quadrotor.

Despite these characteristics, the AR.Drone does not have the capability of keep hovering or navigating for a long time in a completely autonomous way. The take-off and landing maneuvers happen autonomously, but once in the air the UAV slowly starts sliding away from its initial position. This effect is known as drifting in the literature, and results mostly from measurement errors integrated along time. Thus, to ensure a better performance during hovering, positioning or trajectory-tracking maneuvers, it is necessary to continuously estimate the UAV posture from external references and to correct its position error through a closed-loop control system.

B. AR.Drone Motion Control

The motion commands for the AR.Drone are encoded under a specific protocol, over its Wi-Fi network. In this protocol, the command signals are normalized, and arranged as elements of the control signal vector

$$\mathbf{u} = \begin{bmatrix} u_{\dot{z}} \; u_{\dot{\psi}} \; u_{\phi} \; u_{\theta} \end{bmatrix} \in \begin{bmatrix} -1.0, +1.0 \end{bmatrix},$$

where

- $u_{\dot{z}}$ represents a linear velocity command, which causes displacements over the z_w axis;
- $u_{\dot{\psi}}$ represents an angular velocity command, which causes rotations around the z_w axis;
- u_{ϕ} represents an inclination command related to the x_w axis, which indirectly represents a command of linear velocity related to the y_b axis [5], [14], [15], [16]. For this reason it will be hereinafter called u_{v_y};
- u_{θ} represents an inclination command related to the y_w axis, which indirectly causes linear velocity related to the x_b axis [5], [14], [15], [16]. For this reason it will be hereinafter called u_{v_x}.

[1]As low as USD 300.00.
[2]As high as USD 4,000.00.

C. AR.Drone Mathematic Model

The dynamic model of a generic quadrotor is already known in the literature, and can be represented as [10]

$$\begin{cases} m\ddot{x} = (\cos\psi\sin\phi + \cos\psi\cos\phi\sin\theta)u_1 \\ m\ddot{y} = (-\cos\psi\sin\phi + \sin\psi\cos\phi\sin\theta)u_1 \\ m\ddot{z} = (\cos\phi\cos\theta)u_1 - mg \\ I_{xx}\ddot{\phi} = u_2 - (I_{zz} - I_{yy})\dot{\theta}\dot{\psi} \\ I_{yy}\ddot{\theta} = u_3 - (I_{xx} - I_{zz})\dot{\phi}\dot{\psi} \\ I_{zz}\ddot{\psi} = u_4 \end{cases} \tag{1}$$

Here, the UAV movements are described by rigid body transformations and the point of interest, the one whose coordinates should be controlled, is its center of mass. In such model, m represents the mass of the vehicle, g is the gravity acceleration, and I_{xx}, I_{yy} and I_{zz} are the moments of inertia. The signals $u_1, \cdots, u_4$ are control parameters correspondent to forces and torques generated by the quadrotor propellers, also modeled in [10].

As any model, (1) does not completely represent what happens to the aircraft in real flights. For instance, such model does not take into account the drag forces caused by the vehicle friction with the wind. It is known it is very difficult to obtain an universal mathematic model to represent flying systems, thus the aim should be to get a model precise enough for the intended application.

According to [13], in the AR.Drone firmware a model similar to (1) is considered in addition with other aerodynamic effects to achieve flight stabilization. However, the firmware algorithm and the model parameters are restricted to developers. Regardless, we still can take advantage of this AR.Drone internal processing results by modelling its dynamics in function of its control actions $\mathbf{u}$.

In this procedure we assume the AR.Drone behavior when answering a given input command near to the one of a linear system. This observation was used before in several related works [5], [6], [14], [15], [16] and in this way it is not necessary to deal with the complex dynamics of a quadrotor (1). Thus, assuming this context, we propose the model to represent the AR.Drone dynamics in its own coordinate system as

$$\begin{cases} \dot{v}_x = K_1 u_{v_x} - K_2 v_x \\ \dot{v}_y = K_3 u_{v_y} - K_4 v_y \\ \ddot{z} = K_5 u_{\dot{z}} - K_6 \dot{z} \\ \ddot{\psi} = K_7 u_{\dot{\psi}} - K_8 \dot{\psi} \end{cases}, \tag{2}$$

where $\dot{v}_x$ and $\dot{v}_y$ represent linear accelerations with respect to the axes x_b and y_b, $\ddot{z}$ represents the linear acceleration with respect to the axis z_w, and $\ddot{\psi}$ represents the angular acceleration with respect to the axis z_w. The parameters $K_1, \cdots, K_8$ are proportionality constants to be experimentally identified.

Adopting such model, one assumes four degrees of freedom of interest ($v_x, v_y, \dot{z}$ and $\dot{\psi}$), each one modeled as an independent linear system in function of the AR.Drone control signals ($u_{v_x}, u_{v_y}, u_{\dot{z}}$ and $u_{\dot{\psi}}$), having the aircraft center of mass as the point of interest to the controller (the point whose position is being controlled).

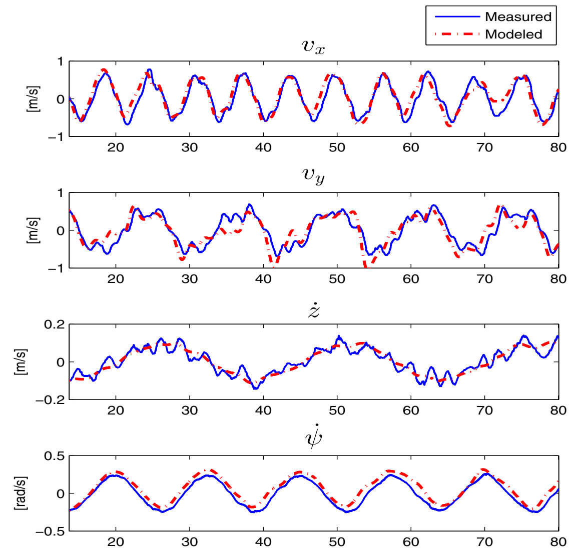

Fig. 2. Comparison between measured data flight and the simulation of (2) for the same input signals.

This model cannot be adopted as a complete representation of the UAV dynamics, but it is precise enough to our control purposes, as detailed ahead, in the Section VI. For now, as one can check in Figure 2, (2) provides a fair approximation of the measures of the AR.Drone sensors along a flight mission. Flight tests also have shown that $\ddot{z}$ and $\ddot{\psi}$ can be represented directly in the global coordinate system, once the AR.Drone firmware already compensates for the influence of the θ and ϕ orientation angles.

Thus, the model in (2) referenced to the global frame becomes

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} K_1\cos\psi & -K_3\sin\psi & 0 & 0 \\ K_1\sin\psi & K_3\cos\psi & 0 & 0 \\ 0 & 0 & K_5 & 0 \\ 0 & 0 & 0 & K_7 \end{bmatrix} \begin{bmatrix} u_{v_x} \\ u_{v_y} \\ u_{\dot{z}} \\ u_{\dot{\psi}} \end{bmatrix} - \begin{bmatrix} K_2\cos\psi & -K_4\sin\psi & 0 & 0 \\ K_2\sin\psi & K_4\cos\psi & 0 & 0 \\ 0 & 0 & K_6 & 0 \\ 0 & 0 & 0 & K_8 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \dot{z} \\ \dot{\psi} \end{bmatrix}, \tag{3}$$

which will be used in Section IV to propose the autonomous flight controller.

III. State Estimation

This Section presents the method implemented to estimate the state vector of the vehicle to feedback the autonomous controller. The method used is the Kalman Filter (KF), an usual state estimation technique for robot control [17].

Regarding UAV state estimation, the most commonly adopted approach is the Extended Kalman Filter (EKF), which is a generalization of the KF for nonlinear systems, as described in [3], [6] and in our previous work [18]. However, from the experience with the AR.Drone, we found that it is possible to obtain a good state estimation for control purposes using just a simple kinematic model, instead of a more complex nonlinear model for the filter prediction step.

The kinematic model is inspired in examples of tracking systems discussed in [19] and we discuss how to adapt it to the AR.Drone in the sequel.

A. The Kalman Filter Implementation

The Kalman Filter is a recursive algorithm that uses noisy input data to produce an optimal state estimation for linear systems, represented as [17], [19]

$$\begin{aligned} \mathbf{x}_k &= \mathbf{F}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \\ \mathbf{z}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \end{aligned} \tag{4}$$

where $\mathbf{x}$ represents the system state vector, $\mathbf{F}$ is the state transition model matrix, $\mathbf{B}$ is the control input matrix, $\mathbf{z}$ is the possible states measurement vector, $\mathbf{H}$ is the sensor observation model matrix, and, finally, $\mathbf{w}$ and $\mathbf{v}$ represent Gaussian noises with zero average. The subscript k refers to a discrete time instant.

In terms of implementation, Algorithm 1 illustrates the steps of the KF procedure, where $\hat{\mathbf{x}}$ is the filter state estimation output, $\hat{\mathbf{P}}$ is the error covariance matrix, $\mathbf{K}$ is the Kalman gain, $\mathbf{Q}$ is the covariance matrix of the process noise, and $\mathbf{R}$ is the covariance matrix of the measurement noise.

Algorithm 1: The Kalman Filter procedure.

```
1: $\overline{\mathbf{x}}_k = \mathbf{F}\hat{\mathbf{x}}_{k-1} + \mathbf{B}\mathbf{u}_k$
2: $\overline{\mathbf{P}}_k = \mathbf{F}\hat{\mathbf{P}}_{k-1}\mathbf{F}^T + \mathbf{Q}$
3: $\mathbf{K}_k = \overline{\mathbf{P}}_k\mathbf{H}^T\left(\mathbf{H}\overline{\mathbf{P}}_k\mathbf{H}^T + \mathbf{R}\right)^{-1}$
4: $\hat{\mathbf{x}}_k = \overline{\mathbf{x}}_k + \mathbf{K}_k\left(\mathbf{z}_k - \mathbf{H}\overline{\mathbf{x}}_k\right)$
5: $\hat{\mathbf{P}}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}\right)\overline{\mathbf{P}}_k$
```

In our application, the KF should track the point of interest (the center of mass of the AR.Drone), whose states are defined by

$$\mathbf{x}_k = \begin{bmatrix} x & y & z & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\psi} \end{bmatrix}^T, \tag{5}$$

where x, y, z, $\dot{x}$, $\dot{y}$ and $\dot{z}$ are the position, in (m), and the linear velocities of such point, in (m/s), at the global reference system. The angular position and velocity around the z axis are ψ, in (rad), and $\dot{\psi}$, in (rad/s), respectively.

The process model is defined as

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \\ \psi_{k+1} \\ \dot{x}_{k+1} \\ \dot{y}_{k+1} \\ \dot{z}_{k+1} \\ \dot{\psi}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \delta t \dot{x}_k \\ y_k + \delta t \dot{y}_k \\ z_k + \delta t \dot{z}_k \\ \psi_k + \delta t \dot{\psi}_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{z}_k \\ \dot{\psi}_k \end{bmatrix}, \tag{6}$$

where δt is the sampling time. Notice that (6) is a simple representation of the matrix $\mathbf{F} \in \mathbb{R}^{8\times 8}$.

This model is a discrete implementation of a kinematic constant velocity model as described in [19], which assumes velocity changes in the interval $[t_k \rightarrow t_{k+1}]$ as a white noise. To apply (6), δt should be small and the variance chosen for the velocities in the Kalman Filter should reflect its possible changes from one time step to next one. To accomplish this statement, the implementation follows what is described in the sequel.

For every system loop, the main program runs the Algorithm 1 ensuring that the AR.Drone firmware delivers $\mathbf{q}$ with a $\delta t \approx 16ms$, which is used to define the measurement vector

$$\mathbf{z_1} = \begin{bmatrix} z \\ \psi \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} z \\ \psi \\ \cos\psi\ v_x - \sin\psi\ v_y \\ \sin\psi\ v_x + \cos\psi\ v_y \end{bmatrix} \tag{7}$$

and the observation model

$$\mathbf{H_1} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{8}$$

At this point, the system is already able to run the KF algorithm and estimate the position of the AR.Drone by odometry, integrating and rotating the velocity data available in $\mathbf{q}$. In such case, the position estimation is still subject to drifting problems along time. To solve the drifting problem, the KF needs to be fed with some other sensorial position reading. Our solution (better explained in Section III-B), is to design a visual system to provide additional position readings for the KF algorithm. The visual algorithm is capable of estimating the AR.Drone position referenced to the global coordinate system, processing images of a landmark captured by the AR.Drone frontal camera. Thus, when this new data, here represented by x_w and y_w, is available it can be used to augment the measure vector of the KF to

$$\mathbf{z_2} = \begin{bmatrix} x \\ y \\ z \\ \psi \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} x_w \\ y_w \\ z \\ \psi \\ \cos\psi\ v_x - \sin\psi\ v_y \\ \sin\psi\ v_x + \cos\psi\ v_y \end{bmatrix} \tag{9}$$

and the observation model to

$$\mathbf{H_2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{10}$$

Therefore, the state estimation method represented in Algorithm 1 is executed every system loop, since it is guaranteed a small δt ($\approx 16ms$). Furthermore, at each loop the system analyze how to execute the Algorithm 1: using $\mathbf{z_1}$ and $\mathbf{H_1}$, if the vector $\mathbf{q}$ is the only data available, or using $\mathbf{z_2}$ and $\mathbf{H_2}$ if there is visual measurements available, besides the vector $\mathbf{q}$. Notice that if there is visual data available, the covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ also needs to be augmented to take into account the variances of x_w and y_w.

Using such approach, the KF will be able to estimate the AR.Drone position by odometry, integrating and rotating the velocities over time and, at the same time, to correct this estimation whenever the visual data is available. By doing that, it is possible to increase the robustness of the KF under a temporary loss of visual data, since that is not the only source of information about the state of the aircraft.

To reinforce this idea, Figure 3 illustrates a fragment of the experiment 1 presented in Section VI. There, three lines are shown, all of them representing the x variable referred to the global frame. The first representation is estimated by odometry (the dot-dashed line). To generate this data, the variables v_x, v_y and ψ were stored and then directly integrated to produce this estimation. The next representation is estimated by the Kalman Filter algorithm presented in this Section (the solid line). The last representation is the result of the computer vision algorithm implemented (the dashed line), stored during the experiment.

In this experiment, the AR.Drone is pushed away twice from its initial position ($0\ m$), and automatically recovers it. During the maneuvers, the system deals with some lacks of visual measurements, such that the visual system is not able to compute the AR.Drone position. In such cases, the graphic shows the visual estimation of x as a constant value (the previous value is kept).

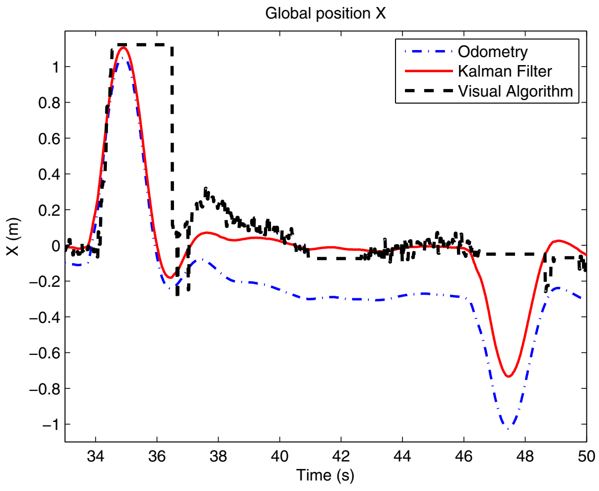

Fig. 3. Comparison of the global x position estimation using pure odometry, Kalman Filter and the visual algorithm.

Notice that the KF estimation of x is an improved combination of the other two sources. As it can be seen, the odometry estimation drifts away from the KF estimation along time, due to the cumulative integration errors involved in the odometry process (the drifting problem). When there is a lack of visual measurement ($t \approx 34s \rightarrow t \approx 37s$, $t \approx 40s \rightarrow t \approx 42s$ and $t \approx 47s \rightarrow t \approx 49s$), the visual estimation of x temporarily stops, but even in such cases the KF continues estimating the position from the velocity measures, correcting it as soon as a new visual data arrives.

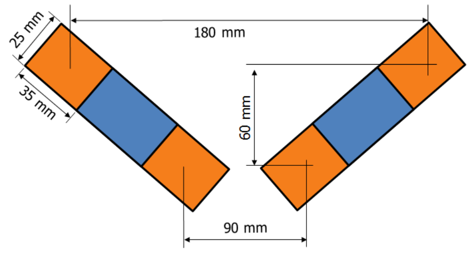

Fig. 4. The dimensions of the visual mark.

Finally, it is important to highlight that to finish the implementation of the KF, it is necessary to ensure its performance by adjusting the process covariance matrix ($\mathbf{Q}$) and the measurement covariance matrix ($\mathbf{R}$). In this work, an off-line calibration was used to tune the filter, after storing data from experimental non-autonomous flights.

B. Visual Data Extraction

The computer vision application here adopted aims at localizing the AR.Drone quadrotor along its flight, using the images provided by its frontal camera. This procedure is needed to allow the KF algorithm to correct its state estimation with the absolute readings of x_w and y_w.

Our algorithm implementation is based on the OpenCV library, an open source computer vision library which makes faster and easier to implement computer vision algorithms and to compute linear algebra operations. Most of the code used is inspired in the examples in [20] and [21].

An overview of the visual algorithm implemented is in the sequel, but its general idea is to search in the camera image for 2D coordinates of a colored mark and geometrically associate them to the 3D coordinates of the same mark, previously known. We have tested some other camera pose algorithms, based on homograph and structure from motion (SfM), but these methods dependency on image features detection turns out to be slow, compared to our proposal.

The first step is to define a mark for visual detection. It is made using a couple of colored stickers, arranged as shown in Figure 4. The stickers are the same provided by Parrot with the AR.Drone, and the mark shape was chosen after some practical tests with the system, where this format have shown to be more effective than other coplanar ones.

Then, for every system loop a raw image is captured by the AR.Drone frontal camera and a segmentation process is performed over it, looking for the orange color. As the result of this process a filtered image is generated, as illustrated in Figure 5.

Next, in the filtered image a search is conducted looking for the white blobs. Once found a blob, its area is compared to a predefined threshold. If the area is bigger, the algorithm calculates the coordinates of the blob center. This procedure is repeated until the whole filtered image has been analyzed.

After this step, the algorithm classifies the result as a good visual observation or a bad visual observation of the mark. A good visual observation occurs when exactly four blob centers are detected in the whole filtered image. A bad

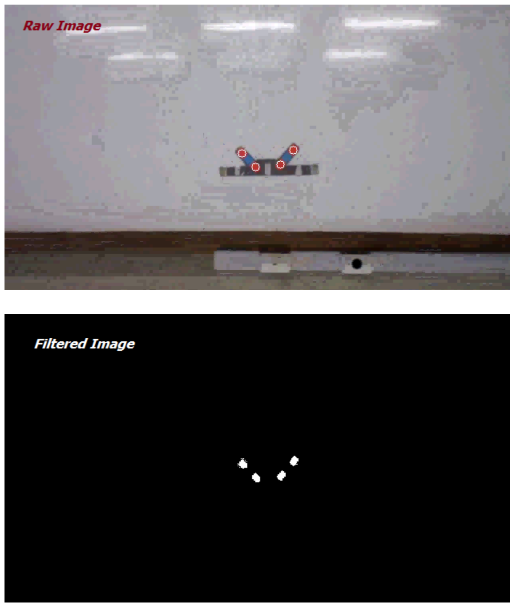

Fig. 5. Raw and filtered images of the visual marker.

visual observation occurs when more or less than four blob centers are identified. This step is primordial to increase the robustness of the system, preventing a bad visual information caused by a noisy image. When a good visual observation is detected, the system holds four 2D image coordinates relative to the centers of each blob detected, which corresponds to the centers of the marks in the colored stickers. With a previous knowledge of the 3D coordinates of the mark it is possible to use the SolvePnP algorithm to calculate the camera pose relative to the mark (more details about the SolvePnP algorithm can be found in [22], while in [20] there are examples on how to implement such an algorithm using the OpenCV library).

The OpenCV SolvePnP implementation returns a vector $t_{vec} = \begin{bmatrix} x_c & y_c & z_c \end{bmatrix}$, which represents the 3D mark position referred to the camera coordinate system. Thus it is necessary

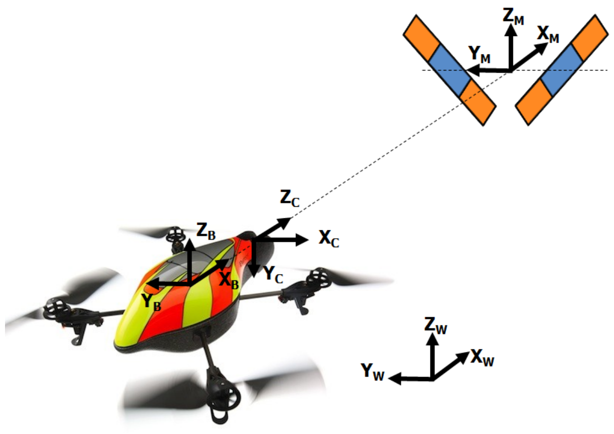

Fig. 6. Illustration of all the coordinate systems. $\{W\}$ is the global one, $\{B\}$ is the one attached to the AR.Drone, $\{C\}$ is the one associated to the camera and $\{M\}$ is the one associated to the visual mark.

to represent t_{vec} in global coordinates in order to find x_w and y_w, to be inputted to the KF. It is worthy mentioning that these measures are given in meters, the same unit used in the program to describe the 3D coordinates of the mark on its own coordinate system $\{M\}$ and that the z_c is discarded in this application (Figure 6 shows all the mentioned coordinate systems).

In order to convert t_{vec} from the camera coordinate system to the global coordinate system, a sequence of rigid body transformations is now applied. For this, first a fixed rigid body transformation converts the t_{vec} reading from the camera coordinate system $\{C\}$ to the AR.Drone coordinate system (located at its center of mass) $\{B\}$. Then, the orientation angles ϕ, θ and ψ are used in a rotation compensation from the AR.Drone coordinate system $\{B\}$ to the global coordinate system $\{W\}$. Finally, the relative distances between the origin of the global coordinate system $\{W\}$ and the mark coordinate system $\{M\}$ are compensated through a translation. This way, the variables x_w and y_w can be computed and used in the KF implementation.

In experimental tests, under a well-controlled illumination and using the mark dimensions of Figure 4, the proposed visual system can detect the target within a distance in the range of $0.5m$ until $3.5m$, which gives a fair flight area in indoor environments.

The visual algorithm here presented runs every system loop, trying to estimate the AR.Drone position from the frontal camera image. If during a loop, a good visual observation is detected, the resulting values of x_w and y_w are fed into the KF to correct its state estimation. Else, if a bad visual observation is detected caused by bad illumination, obstacles, loss of sight of the mark or any other sources of error, the image processing is discarded until a new image arrives in a next loop. Thus during system loops with a bad visual observation, the KF estimation is based only on odometry.

IV. The Proposed Autonomous Controller

This Section presents the nonlinear controller proposed to guide the AR.Drone in positioning and trajectory tracking tasks. Knowing that (3) represents the UAV model at the global frame, the objective is to propose a controller to guide the AR.Drone from its current position and orientation $\mathbf{X} = \begin{bmatrix} x & y & z & \psi \end{bmatrix}^T$ to their desired values $\mathbf{X}_d = \begin{bmatrix} x_d & y_d & z_d & \psi_d \end{bmatrix}^T$. Notice that $\mathbf{X}_d$ can be a function of time, thus characterizing trajectory tracking, or a constant, thus characterizing positioning.

Rewriting (3) as

$$\ddot{\mathbf{X}} = \mathbf{f_1}\mathbf{U} - \mathbf{f_2}\dot{\mathbf{X}} \tag{11}$$

where

$$\ddot{\mathbf{X}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\psi} \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} u_{v_x} \\ u_{v_y} \\ u_{\dot{z}} \\ u_{\dot{\psi}} \end{bmatrix}, \quad \dot{\mathbf{X}} = \begin{bmatrix} v_x \\ v_y \\ \dot{z} \\ \dot{\psi} \end{bmatrix},$$

with $\mathbf{f_1}$ and $\mathbf{f_2}$ being the two 4×4 matrices of (3), an inverse dynamic controller can be proposed, in a way quite similar to the one proposed in [23]. To do that, we adopt the control law

$$\mathbf{U} = \mathbf{f_1}^{-1}(\boldsymbol{\nu} + \mathbf{f_2}\dot{\mathbf{X}}), \quad (12)$$

with

$$\boldsymbol{\nu} = \ddot{\mathbf{X}}_d + \boldsymbol{\kappa}_p\tilde{\mathbf{X}} + \boldsymbol{\kappa}_d\dot{\tilde{\mathbf{X}}},$$

$\tilde{\mathbf{X}} = \mathbf{X}_d - \mathbf{X} = \begin{bmatrix} x_d - x & y_d - y & z_d - z & \psi_d - \psi \end{bmatrix}^T$ (the tracking error),

$$\boldsymbol{\kappa}_p = \begin{bmatrix} K_{p_x} & 0 & 0 & 0 \\ 0 & K_{p_y} & 0 & 0 \\ 0 & 0 & K_{p_z} & 0 \\ 0 & 0 & 0 & K_{p_\psi} \end{bmatrix}$$

and

$$\boldsymbol{\kappa}_d = \begin{bmatrix} K_{d_x} & 0 & 0 & 0 \\ 0 & K_{d_y} & 0 & 0 \\ 0 & 0 & K_{d_z} & 0 \\ 0 & 0 & 0 & K_{d_\psi} \end{bmatrix}$$

(diagonal positive defined gain matrices).

Substituting (12) in (11) the closed-loop equation governing the dynamics of the position and orientation errors becomes

$$\ddot{\tilde{\mathbf{X}}} + \boldsymbol{\kappa}_p\tilde{\mathbf{X}} + \boldsymbol{\kappa}_d\dot{\tilde{\mathbf{X}}} = 0, \quad (13)$$

for which we should demonstrate that such errors converge to zero when $t \to \infty$ (asymptotic stability of the closed-loop control system).

In order to analyze the stability of the equilibrium of (13), the radially unbounded Lyapunov candidate function

$$\mathbf{V}(\tilde{\mathbf{X}}, \dot{\tilde{\mathbf{X}}}) = \frac{1}{2}\tilde{\mathbf{X}}^T\boldsymbol{\kappa}_p\tilde{\mathbf{X}} + \frac{1}{2}\dot{\tilde{\mathbf{X}}}^T\dot{\tilde{\mathbf{X}}} \geq 0 \quad (14)$$

is chosen. After taking its first time derivative and replacing (13), one gets

$$\begin{aligned} \dot{\mathbf{V}}(\tilde{\mathbf{X}}, \dot{\tilde{\mathbf{X}}}) &= \tilde{\mathbf{X}}^T\boldsymbol{\kappa}_p\dot{\tilde{\mathbf{X}}} + \dot{\tilde{\mathbf{X}}}^T\ddot{\tilde{\mathbf{X}}} \\ &= \tilde{\mathbf{X}}^T\boldsymbol{\kappa}_p\dot{\tilde{\mathbf{X}}} + \dot{\tilde{\mathbf{X}}}^T\left(-\boldsymbol{\kappa}_p\tilde{\mathbf{X}} - \boldsymbol{\kappa}_d\dot{\tilde{\mathbf{X}}}\right) \\ &= -\dot{\tilde{\mathbf{X}}}^T\boldsymbol{\kappa}_d\dot{\tilde{\mathbf{X}}} \leq 0. \quad (15) \end{aligned}$$

Based on the theory of Lyapunov for nonlinear systems, one can conclude from (15) that $\tilde{\mathbf{X}} \to 0$ with $t \to \infty$, meaning that the closed-loop control system is asymptotically stable.

V. System Architecture

This Section presents a brief overview of the proposed software architecture. Notice that it is necessary to establish a communication channel between the AR.Drone quadrotor and an external computer, and the OpenCV library should be running as well, to compute the vision algorithm. In this work, we adopted the solution proposed in [24], which suitably integrates the AR.Drone SDK and the OpenCV library, as necessary.

For a better comprehension of the implementation, Figure 7 illustrates how the algorithms interact. There, the centralized computer station runs the code under an infinity loop. Every cycle, the sensor and video data are required through the communication channel, established with the AR.Drone over a Wi-Fi network. With the data available in the centralized computer, the next step is to get the observations x_w and y_w from the visual algorithm of Section III-B. In the sequence, with or without x_w and y_w, the KF is executed as described in Section III-A. Finally, with the results of the KF state estimation the last step is to calculate the control actions, with the algorithm of the Section IV and send them to the AR.Drone, again through the Wi-Fi network.

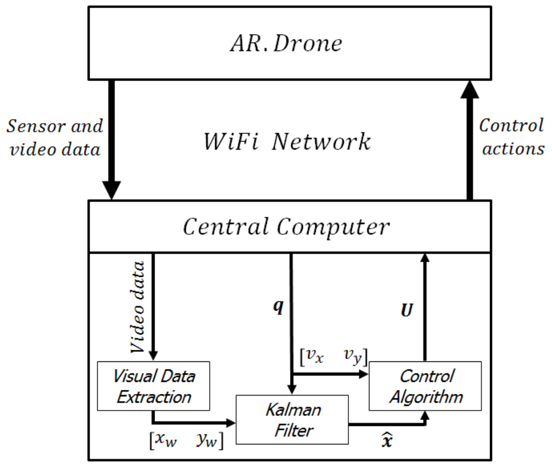

Fig. 7. Software architecture adopted. x_w and y_w are the global position coordinates provided by the visual system. $\mathbf{q}$ is the set of variables provided by the AR.Drone sensor board. $\hat{\mathbf{x}}$ is the KF state estimation output and $\mathbf{U}$ is the set of control actions of the AR.Drone.

VI. Experimental Results

In this Section, four different experiments related to 3D positioning and trajectory tracking tasks are presented, using the AR.Drone aircraft. All the experiments are presented in the same way: a brief explanation about them and some graphics of the variables of interest $\mathbf{X}_d = \begin{bmatrix} x_d & y_d & z_d & \psi_d \end{bmatrix}^T$. In addition, for the trajectory tracking experiments 3 and 4, tables containing the root mean square error analysis are also presented, which are not presented in the other experiments.

An observation about the experiments is the execution of the procedures of take-off and landing, which are manually performed. As soon as the AR.Drone is in the air, the automatic controller is enabled. Therefore, due to initial numerical errors, these data are deleted from the plotted graphics, focusing only in the vehicle guidance through the autonomous controller.

Another important characteristic to emphasize is that all the experiments run under the same controller configuration, i.e., the gain matrices $\boldsymbol{\kappa}_p$ and $\boldsymbol{\kappa}_d$ remain the same, to show the robustness of the proposed framework for different trajectories.

A. Experiment 1: Hovering

The experiment 1, characterized in Figure 8, has the simple goal of keeping the rotorcraft anchored on the reference position $\mathbf{X}_d = \begin{bmatrix}0.0 & 0.0 & 1.2 & 0.0\end{bmatrix}^T$. Some disturbances occur during flight, moving the AR.Drone away from its reference. The objective is to demonstrate the capability of the system to hold/recover a 3D position, even under strong disturbances and temporary loss of the visual mark. The video available in the link `http://youtu.be/dO5tlmWcyWY` presents it.

B. Experiment 2: Positioning at the Corners of a House-shaped Polygon

The experiment 2, illustrated in Figure 9, aims at positioning the AR.Drone in a set of desired positions (waypoints), whose values change from one to another in the set after an interval of $5s$. The objective here is to analyze how quick is the AR.Drone response, when an input step is given. The video correspondent to such an experiment can be found in `http://youtu.be/RKkoMjk58oA`.

It is important to highlight that there is no lines to be followed from a desired position to the following one. The solid line that appears in Figure 9 is just a graphical representation of the shortest distance between two successive desired positions.

C. Experiment 3: Tracking a Circular Trajectory

In the experiment 3 the UAV should track a trajectory defined by a circumference with 1 m of radius, which can be parameterized as $\mathbf{X}_d = \begin{bmatrix}\sin(0.8t) & \cos(0.8t) & 1.2 & 0.0\end{bmatrix}^T$. Figure 10 illustrates the results of the flight mission. During the experiment, the AR.Drone constantly looses the visual mark due to the relationship between the dimension of the trajectory and the angle of view of the onboard camera (which cannot observe the whole scene). Moreover, the obstacle that appears in the middle of the circle also causes temporary losses of the visual mark. The video available in `http://youtu.be/amarVZp7GtY` illustrates it.

Table I presents the root mean square and the maximum errors correspondent to each degree of freedom of interest. Notice that the largest error is in the x direction, and it is about $148mm$. Observing the experiment data, one can observe that the larger errors occur when the trajectory changes its direction. Such effect can be attenuated by tuning the controller gains, what it is out of the scope of this work.

TABLE I
ERROR ANALYSIS FOR EXPERIMENT 3: CIRCULAR TRAJECTORY TRACKING.

	x (meters)	y (meters)	z (meters)	ψ (radians)
RMSE	0.0645	0.0655	0.0374	0.0162
Max. error	0.1481	0.1404	0.0944	0.0296

D. Experiment 4: Tracking an Eight-shaped Trajectory

In the experiment 4, illustrated in Figure 11, all the four degrees of freedom are controlled at the same time. In such experiment a sloped eight-shape trajectory defined by

$$\mathbf{X}_d = \begin{bmatrix} 0.5\sin(0.8t) \\ \sin(0.4t) \\ 1.2 + 0.5\sin(0.4t) \\ -\frac{\pi}{6}\sin(0.4t) \end{bmatrix}$$

should be tracked. The objective is to demonstrate that the control system proposed can guide the four degrees of freedom in an independent way. The video available at `http://youtu.be/aE9r6szkQzo` highlights it.

As one can see in Table II, the RMSE for the x and y coordinates decreases, while the value correspondent to the z coordinate and to the orientation ψ increases, in comparison with the previous experiment. In these circumstances, this analysis confirms the independence assumptions between the degrees of freedom of Section II-C, in the case of the AR.Drone quadrotor.

It is very important to highlight that the velocities along the trajectories influence the RMSE results. In addition, as aforementioned, the results could be improved by tuning the parameters of the $\boldsymbol{\kappa}_p$ and $\boldsymbol{\kappa}_d$ gain matrices. Thus, one can conclude the analysis stating that under some practical conditions it is possible to guarantee the performance of the system presented in this paper.

To be more specific, in a series of other non documented experiments with the same controller gains, we have found that all tested trajectories that require speeds $\dot{x}_d$ and $\dot{y}_d$ below 1 m/s, $\dot{z}_d$ below 0.5 m/s and $\dot{\psi}_d$ below $\frac{\pi}{4}$ rad/s can be performed with approximately the same results. In trajectories that require higher speeds the results starts to deteriorate. The reader familiar with the AR.Drone knows that its limit values comes in part from its regulation for manual flight (performed by the manufacturer), which also limits the velocities in autonomous flights.

TABLE II
ERROR ANALYSIS FOR EXPERIMENT 4: EIGHT-SHAPED TRAJECTORY TRACKING

	x (meters)	y (meters)	z (meters)	ψ (radians)
RMSE	0.0570	0.0554	0.0827	0.0211
Max. error	0.1368	0.2139	0.1571	0.0722

E. State Estimation: Kalman Filter vs Odometry

Another important practical issue to be clarified is why not completely trust in the AR.Drone sensors to execute the position estimation. Considering the onboard estimation of v_x, v_y and ψ available, at a first look it seems to be just a matter of rotating and integrating them to obtain the position of the vehicle in the global coordinate frame by odometry.

Previously, in Section III-A, it was stated that this procedure is not recommended for a long time flight, due to the sensor drift. Here this statement is reinforced using the data of the experiment 4, as it can be seen in Figure (12). The experimental data correspondent to the x and y positions are collected and plotted one against the other, in a way similar to the example of Section III-A. The measurements provided

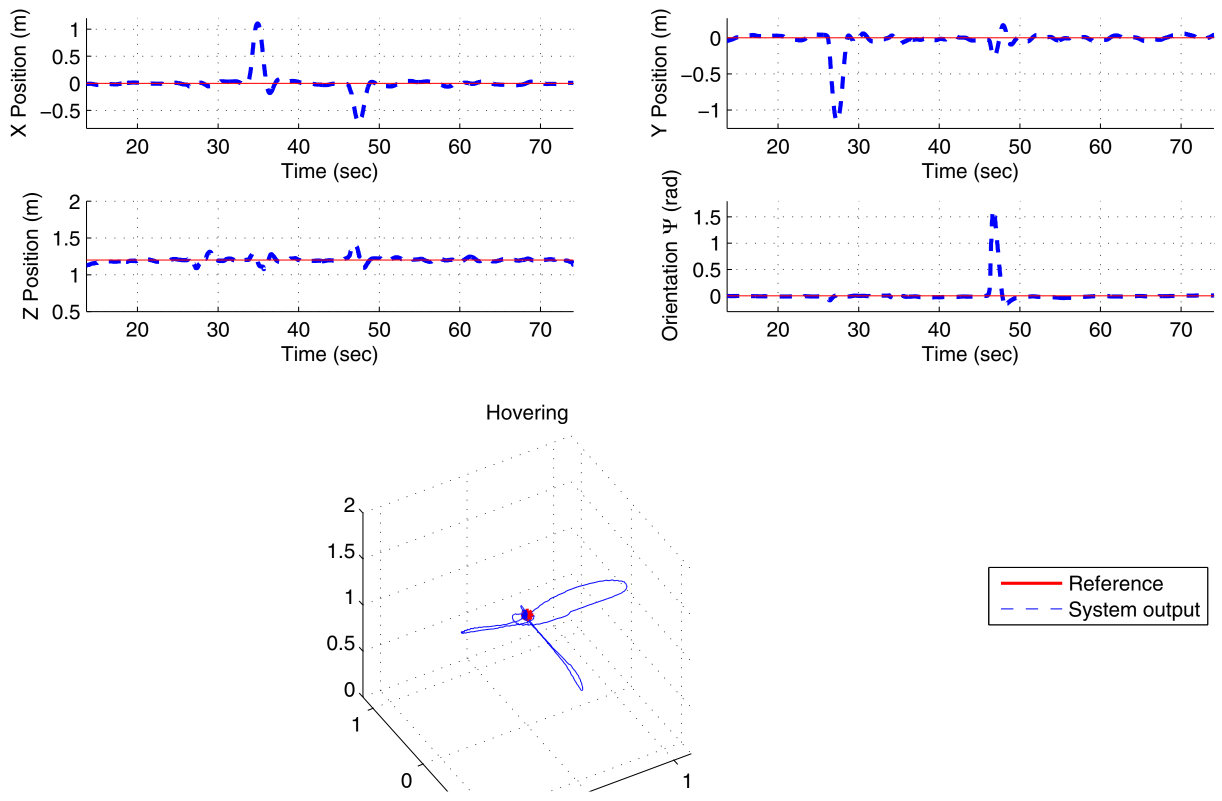

Fig. 8. Experiment 1: Hovering.

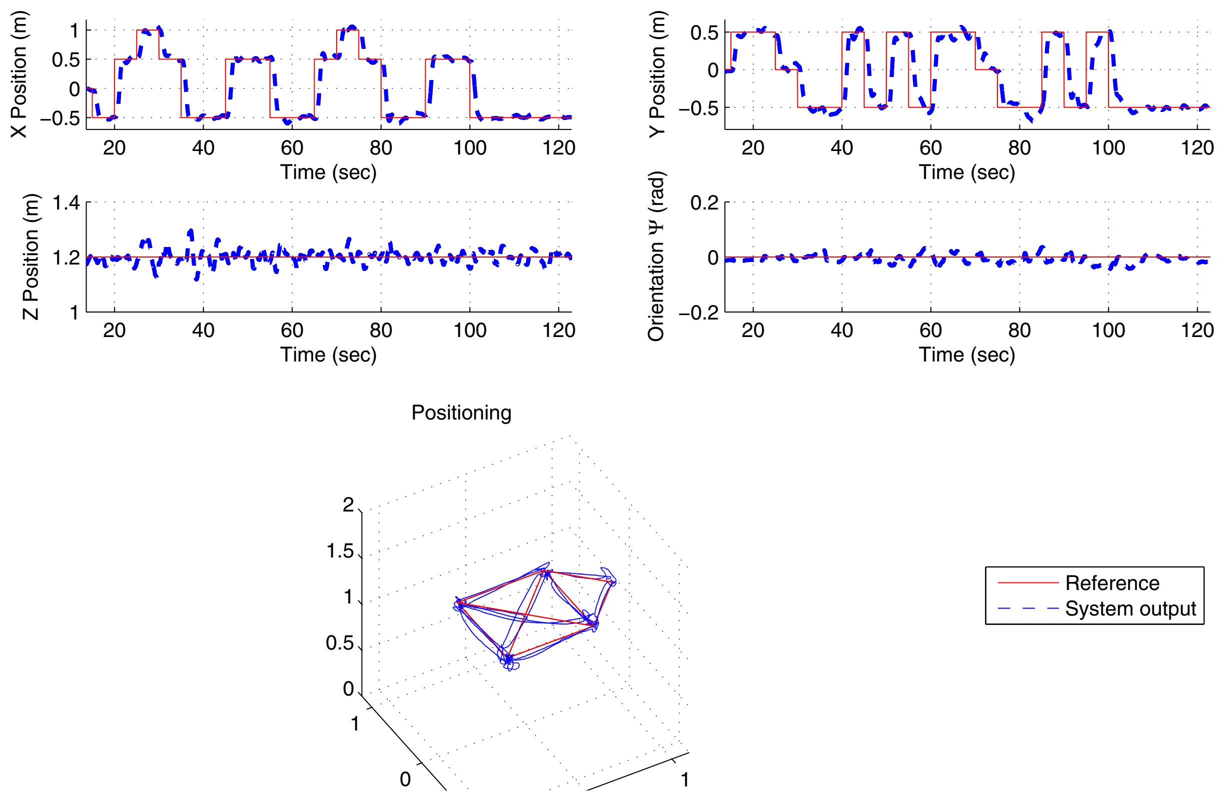

Fig. 9. Experiment 2: Positioning task (the successive desired positions are the corners of the polygon).

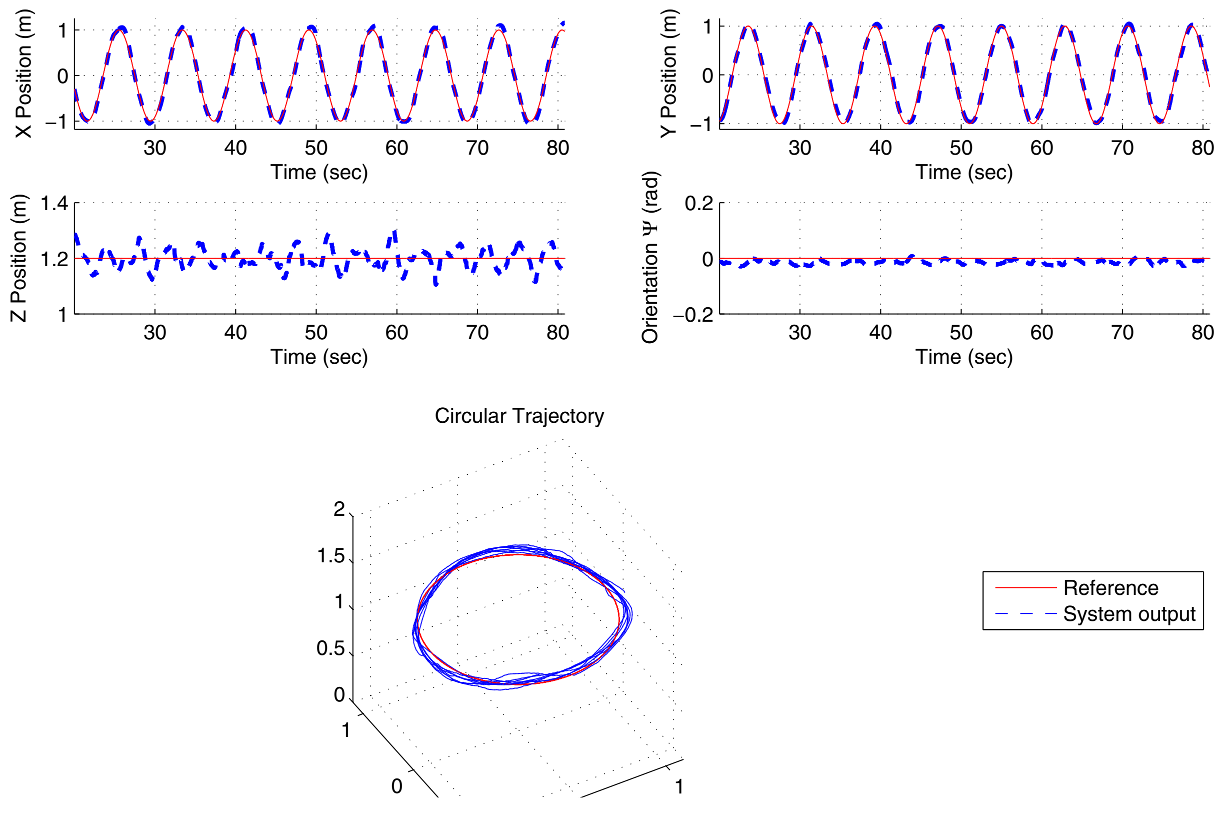

Fig. 10. Experiment 3: Circular trajectory.

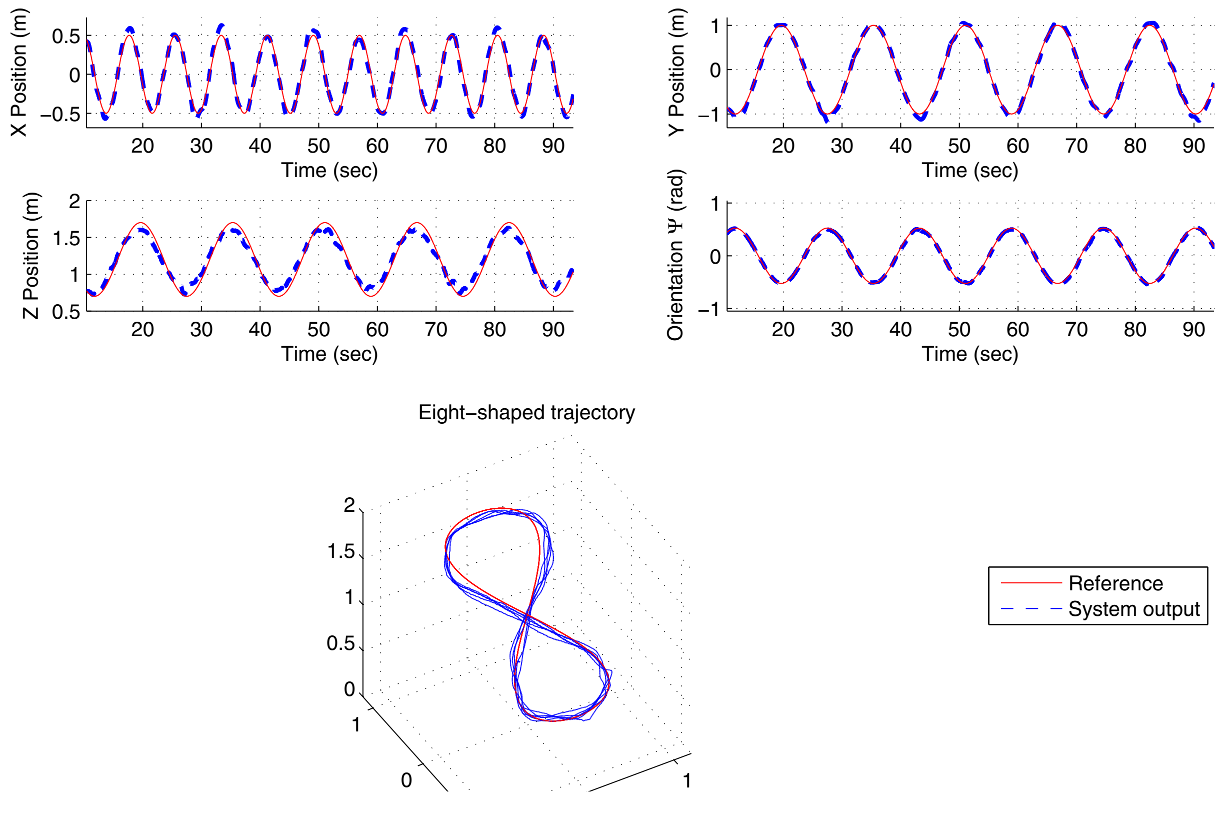

Fig. 11. Experiment 4: Eight-shaped trajectory.

by the odometry, the Kalman Filter and the visual algorithm here implemented are considered to generate three different plots.

It is interesting to notice that the drift occurs more in the y variable. Also, the sporadically noisy data provided by the visual system for x does not significantly interfere in the filtered estimation. For a better perception, just a top-view (XY-plane) of the trajectory accomplished by the aircraft is plotted, to allow comparing the three methods of estimation. As one can see from such a figure, the use of the Kalman Filter improves the state estimation, considering the sensorial data available.

VII. Conclusions

The method proposed in this work adds robustness to the problem of 3D trajectory tracking and 3D positioning with the AR.Drone quadrotor in indoor environments, for being based on visual and inertial information. The method adopts the fusion of both sensorial data, thus compensating for the problems associated to the inertial sensors and for the problems associated to computer vision as well.

An important contribution of this manuscript, discussed in Sections II and VI, is that for control purposes the AR.Drone dynamics can be approximated by four independent linear systems, modeled considering the control signals the aircraft accepts. This procedure makes easier to design a nonlinear controller to guide the aircraft when navigating, as shown in Section IV. Another important contribution is in Section III, where it is shown to be possible to implement a simpler tracking Kalman Filter to estimate the state of the vehicle and to feed it back to the control algorithm.

As seen in Section VI, the method can be used to guide the AR.Drone over different types of trajectories in the 3D space, even under orientations ψ_d different of zero and under disturbances. It is also shown that it is possible to guide the vehicle over different trajectory velocities without needing to constantly adjust the controller gains, respecting some practical limitations. Also, as the experiments here discussed confirm, the proposed procedure allows the rotorcraft to momentarily loose the target object (it goes out of the field of vision of the vehicle), retrieving that information later, without loosing control, thus confirming that the system copes suitably with sporadic disturbances.

Surely some practical limitations require attention and treatment. Amongst them, it is important to highlight to the reader that in practical applications the visual information is limited to indoor environments, due to the size of the visual mark and the algorithm of color segmentation, which implies the system dependency on controlled illumination conditions. Additionally, there are delays in communication, originated by the Wi-Fi network or inherent to the system, like the notorious differences in the update rate between the inertial and visual data, which requires some treatment during execution of the Kalman Filter. In our method this problem is solved adjusting the variance matrices after several experimental flights. In spite of that, however, the proposed method has proven to be effective in terms of accomplishment of the proposed tasks.

In the future, it is intended to apply a similar method to model and control the coordinated navigation of more than one UAV, in formation flights.

Acknowledgment

The authors thank CNPq – Conselho de Desenvolvimento Científico e Tecnológico, a Brazilian agency that supports scientific and technological development (grant 473185/2012-1) – for the financial support granted to this work. They also thank the Federal Institute of Espírito Santo and the Federal University of Espírito Santo for supporting the development of this research. Dr. Sarcinelli-Filho also thanks the additional financial support of FAPES - Fundação de Amparo à Pesquisa do Espírito Santo to the project. Dr. Brandão also thanks Federal University of Viçosa, Brazil and FAPEMIG - Fundação de Amparo à Pesquisa de Minas Gerais - for supporting his participation in this work.

References

[1] M. W. Müller, S. Lupashin, and R. D'Andrea, "Quadrocopter ball juggling," in *Proceedings of the 24th IEEE/RSJ International Conference on Intelligent Robot Systems*, San Francisco, USA, September 2011, pp. 5113–5120.

[2] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, May 2011, pp. 2520–2525.

[3] S. Weiss, M. W. Achtelik, M. Chli, and R. Siegwart, "Versatile distributed pose estimation and sensor self-calibration for an autonomous mav," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, St. Paul, MN, USA, May 2012, pp. 31–38.

[4] S. Shen, N. Michael, and V. Kumar, "Autonomous multi-floor indoor navigation with a computationally constrained mav," in *Proceedings of International Conference on Robotics and Automation*, Shanghai, China, 2011, pp. 20–25.

[5] T. Krajnik, V. Vonasek, D. Fiser, and J. Faigl, "Ar-drone as a platform for robotic research and education," in *Research and Education in Robotics - EUROBOT 2011*, ser. Communications in Computer and Information Science, D. Obdrzalek and A. Gottscheber, Eds. Springer, 2011, vol. 161, pp. 172–186.

[6] J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," in *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal, October 2012, pp. 2815–2821.

[7] J. J. Lugo and A. Zell, "Framework for autonomous on-board navigation with the ar.drone," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1-4, pp. 401–412, 2014.

[8] M. Turpin, N. Michael, and V. Kumar, "Trajectory design and control for aggressive formation flight with quadrotors," *Auton. Robots*, vol. 33, no. 1-2, pp. 143–156, Aug. 2012. [Online]. Available: http://dx.doi.org/10.1007/s10514-012-9279-y

[9] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 3480–3486.

[10] J. Kim, M.-S. Kang, and S. Park, "Accurate modeling and robust hovering control for a quadrotor vtol aircraft," *Journal of Intelligent & Robotic Systems*, vol. 57, no. 1-4, pp. 9–26, 2010.

[11] A. S. Brandão, M. Sarcinelli-Filho, and R. Carelli, "High-level underactuated nonlinear control for rotorcraft machines," in *Proceedings of the IEEE International Conference on Mechatronics*. Vicenza, Italy: IEEE, Febrary 27 – March 1 2013, pp. 279–285.

[12] S. Piskorski, N. Brulez, P. Eline, and F. DHaeyer, *AR.Drone Developer Guide*, Parrot, December 2012, sDK Version 2.0.

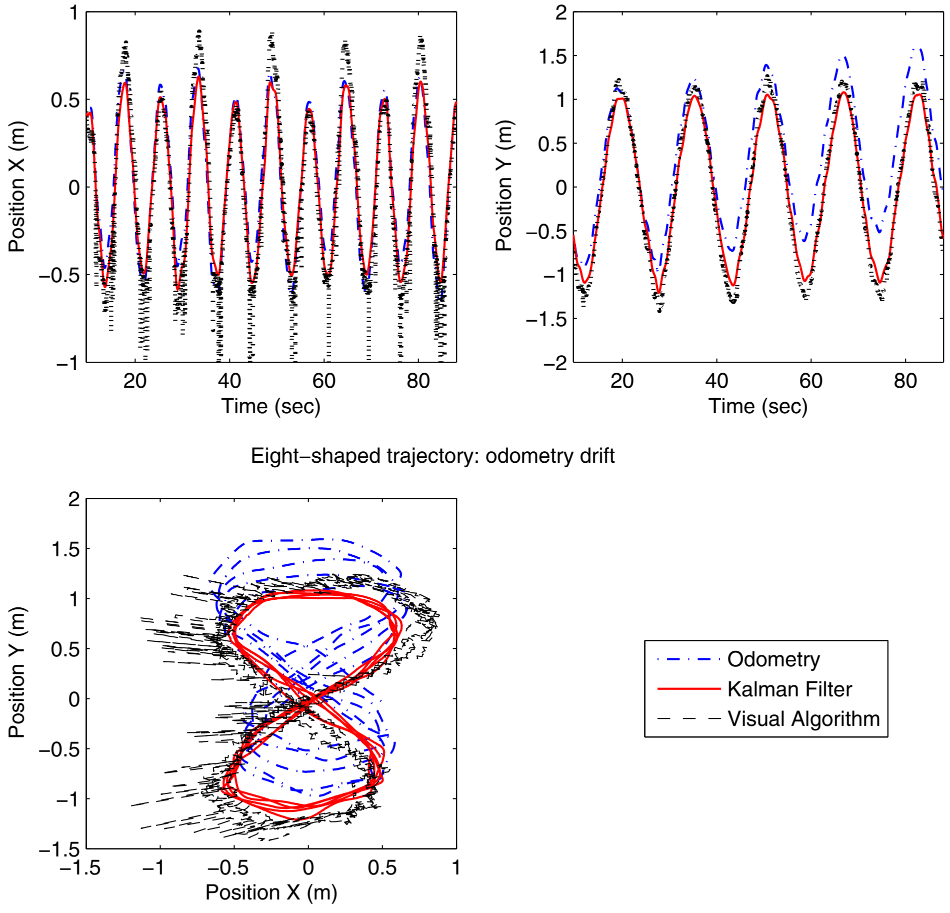

Fig. 12. Demonstration of the odometry drift in the case of the Experiment 4.

[13] P.-J. Bristeau, F. Callou, D. Vissiere, and N. Petit, "The navigation and control technology inside the ar.drone micro uav," in *Proceedings of the 18th IFAC World Congress*, vol. 18, Milan, Italy, August-September 2011, pp. 1477–1484.

[14] A. Hernandez, C. Copot, R. De Keyser, T. Vlas, and I. Nascu, "Identification and path following control of an ar.drone quadrotor," in *Proceedings of the 17th International Conference of System Theory, Control and Computing (ICSTCC'13)*, Sinaia, Romania, October 2013.

[15] P. Falcón, A. Barreiro, and M. D. Cacho, "Modeling of parrot ardrone and passivity-based reset control," in *Proceedings of the 9th Asian Control Conference (ASCC'13)*, Istanbul, Turkish, June 2013.

[16] E. Bilgin, D. E. Sanabria, P. J. Mosterman, and K. Zhang. (2013, october) Ar drone simulink development-kit v1. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/43719-ar-drone-simulink-development-kit-v1

[17] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[18] L. V. Santana, M. S. Filho, and R. Carelli, "Estimation and control of the 3d position of a quadrotor in indoor environments," in *Proceedings of the 16th International Conference on Advanced Robotics (ICAR'13)*, Montevideo, Uruguay, November 2013.

[19] Y. Bar-Shalom, T. Kirubarajan, and X.-R. Li, *Estimation with Applications to Tracking and Navigation*. New York, NY, USA: John Wiley & Sons, Inc., 2002.

[20] D. L. Baggio, S. Emami, D. M. Escriva, K. Ievgen, N. Mahmood, J. Saragih, and R. Shilkrot, *Mastering OpenCV with Practical Computer Vision Projects*. Packt Publishing, Limited, 2012.

[21] R. Laganière, *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Pub Limited, 2011.

[22] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *Int. J. Comput. Vision*, vol. 81, no. 2, Feb. 2009.

[23] J.-J. Slotine and W. Li, *Applied Nonlinear Control*. Prentice Hall, Oct. 1991.

[24] GitHub.com. (2013, january) Cv drone. [Online]. Available: https://github.com/puku0x/cvdrone