Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online
CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 48-9 (2015) 073–078

Autonomous path tracking control design for a comercial quadcopter [★]

Raúl M. Criado [*] Francisco R. Rubio [*]

[*] *Escuela Superior de Ingeniería, Universidad de Sevilla Avd. Camino de los Descubrimientos, s/n. 41092 Sevilla, Spain (e-mail: r.martinezcriado@gmail.com; rubio@us.es)*

Abstract: This paper describes the methodology followed to design a functional autopilot for the quadcopter AR.Drone developed by Parrot. The main goal is to design a control strategy for autonomous path tracking in the XY plane, comparing two different control techniques. Three phases are carried out to achieve this objective: model identification, control design and evaluation, and implementation. At the identification phase two Hammerstein models are obtained, they are characterized by having a static non-linearity preceding a linear transfer function. The control design phase is based on the use of a cascade control to regulate the position with an inner speed control loop. Furthermore, two different techniques to reduce the path tracking error are compared.

Keywords: cascade control, path tracking, quadcopter control, multi-rotor, UAV.

1. INTRODUCTION

From many years now, Unmanned Aerial Vehicles (UAVs) have been developed for several military purposes, such as offensive weapons or surveillance systems. Although the civil area was almost alien to this technology, the scientific research related to UAVs non-military applications have grown significantly along the past 15 years, due to the reduction in the production costs and the large increase of the computing power related to the size of the technology involved.

The quadcopter is a VTOL (Vertical Take-Off and Landing) platform, that belongs to the helicopter-type within the UAVs family. It is provided with four brushless rotors that generate the lift force and allow the quadcopter to increase its payload capability. While having one single rotor compels to include complex mechanisms on the rotor and the blades for steering and manoeuvring, quadcopters are controlled by modifying the angular velocity of each single rotor to accomplish any desired motion. This feature increments the agility of the helicopter and turns it into a feasible option when rapid and precise paths are required.

This type of aircraft requires an uninterrupted real-time control action to stabilize itself, on account of the fact that it is an unstable system. Thus, this case of study is a challenge within the Control Engineering field.

The AR.Drone from Parrot was initially developed as a toy, yet some features present on the device, such as on-board stabilization and Wi-Fi connectivity, make it a complete platform to shift the research focus from basic control of the model towards complex applications such as mixing control techniques with computer vision.

In order to communicate with the AR.Drone, the software used is TrackDrone Lite, García-Nieto Rodríguez et al. (2012). Developed by the *Universitat Politècnica de València*, TrackDrone Lite includes tool to implement different control laws using *C#* code. Furthermore it enables the communication with the drone and streams some of the flight data in real-time.

2. MODELLING AND IDENTIFICATION

The analysis of the physics behind a quadcopter is complex. To simplify it, we can assume that it behaves as a rigid solid wherein one lift force and torque per rotor are applied in addition to the weight.

As mentioned before, the AR.Drone is a self-stabilized platform and the goal of this research is to substitute an hypothetical human pilot by an autopilot. The movement over the XY plane is obtained by the modification of the pitch and roll angles. When an angle set-point is sent to the drone, it tries to follow the reference using its pre-installed inner control loops.

These control loops are implemented on the operating system running on-board and will not be modified. Thanks to them, the response to set-point changes of pitch and roll angles are decoupled and the AR.Drone no longer has a highly non-linear behaviour.

2.1 Methodology

The system is identified as a black-box model, choosing the angles pitch and roll as inputs of the system and the coordinates X, Y, and the components of the speed v_x, v_y as the outputs. The speed evolution was registered at the

[★] This work was partially supported by the Spanish Ministry of Education (MECD) under national research projects DPI2012-37580-C02 and DPI2013-44135-R. Corresponding authors are with the Department of Systems Engineering and Automation. University of Seville, Spain. (e-mail: rubio@us.es)

2405-8963 © 2015, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2015.08.062

open-loop experiments as the output, sending the angles pitch and roll as input step signals. Three main aspects must be considered to obtain the maximum information from the real system during the open-loop experiments:

- Step-time input signals with a duration of five seconds are necessary to register both, the transient-state and the steady-state.
- The inputs are dimensionless and can take values within the range [-1,1]. To track the non-linearities, it is important to set steps in the whole range.
- Registering enough data is necessary to carry out a validation after the identification.

On Fig. 1 and Fig. 2 only a part of the data registered during the open-loop experiments is represented, within them was covered the whole range [-1, 1] for both angles pitch and roll.

After the open-loop experiments, the data package was processed using the System Identification Toolbox (SIT) of MATLAB©. This application enables the selection of different options and identification models. The models used were identified with the half of the data measured, making use of the second half for validation. As recom-

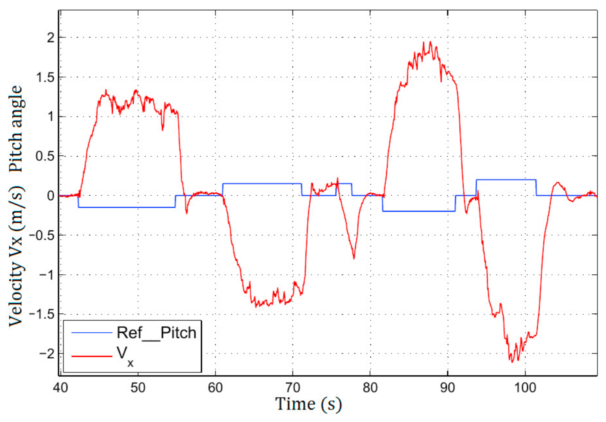

Fig. 1. Input steps [-0.15, +0.15, -0.25, +0,25], $Roll - V_y$.

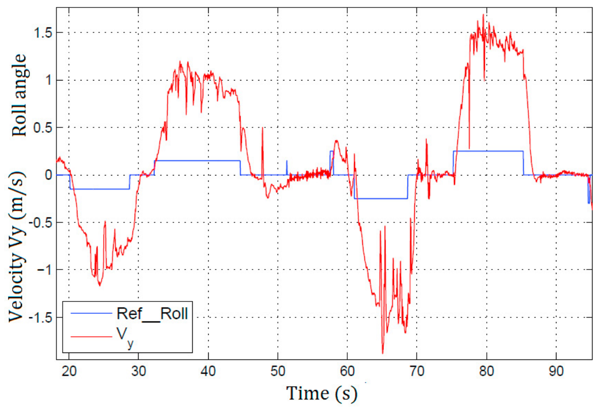

Fig. 2. Input steps [-0.15, +0.15, -0.25, +0,25], $Roll - V_y$.

mended in Åstrom and Hägglund (2006) for non-linear system identification when linear controllers are used, and already proven for Ar.Drone before on the study presented in Hernández-Hernández et al. (2013), two Hammerstein model were chosen to relate the pitch angle with v_x and the roll angle with v_y, respectively. The non-linearity block is a static gain map modelled as piecewise function (one per model) and the linear transfer functions identified are two SOTD (Second Order Time Delay) models on discrete-time domain.

2.2 Results

On Fig. 3 the identified model of the AR.Drone is shown. It is composed of two independent subsystems (*X Subsystem, Y Subsystem*), within which the positions X and Y are obtained by two integrators at the output signals (velocities v_x, v_y). There is a saturation in both angle signals

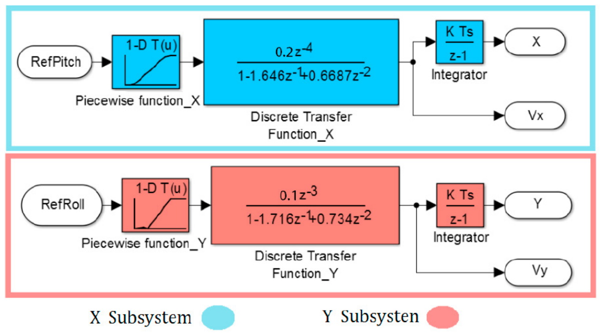

Fig. 3. Complete identified model.

around +0.35 and -0.35 respectively. This saturation is imposed by the on-board stabilization loops to guarantee that the critical angles, where the system is uncontrollable, are not reached. These saturations correspond to a value of maximum angles between 20°-24°.

The static non-linearities before the linear blocks are characterized by the next vectors. The saturations of the angles pitch and roll were fixed at ±0.35 when designing the linear controllers:

$Input_x = [-0.5, -0.4, -0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5]$

$OutputU_x = [0.325, 0.325, 0.29, 0.19, 0.0855, 0, -0.089, -0.19, -0.29, -0.325, -0.325]$

$InputU_y = [-0.8, -0.48, -0.37, -0.28, -0.08, 0, 0.15, 0.35, 0.71, 0.8]$

$OutputU_y = [-0.367, -0.330, -0.384, -0.277, -0.085, 0\ 0.165, 0.357, 0.3407, 0.3670]$

The comparison between the real system and the identified model is plotted on Fig. 4, these data was used to validate and choose the model using the SIT and to readjust the static non-linearity on both Hammerstein models (obtained directly from the SIT), reducing the mean and maximum values of the difference between the velocity components of the real system and the model:

- $Mean\ e_{v_x} = 0.18\ m/s,\ \ Max\ e_{v_x} = 1.11\ m/s$
- $Mean\ e_{v_y} = 0.13\ m/s,\ \ Max\ e_{v_y} = 1.28\ m/s$

3. CONTROL DESIGN

The goal is to develop a control strategy for path tracking within XY plane. The track is loaded through a *.txt* file

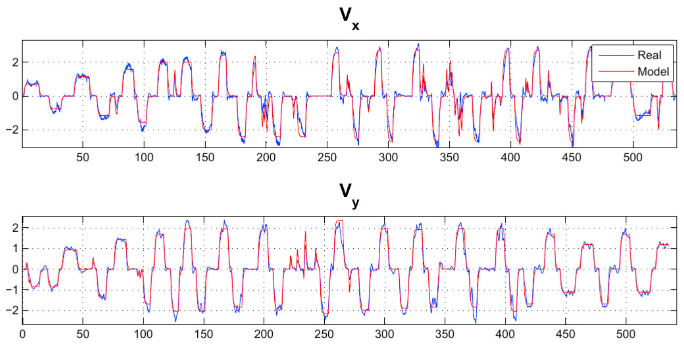

Fig. 4. Validation data.

containing a vector with the coordinates of a sequence of waypoints to be reached by the drone. A control loop to follow yaw angle inputs is already implemented by Parrot. Hence, an orientation control is not discussed in this paper and the initial orientation is maintained along the path.

3.1 Cascade control

The main architecture of the control technique developed in this work is the so named *cascade control*. This strategy is built by nesting two control loops as shown in Fig. 5. The inner loop is called *the secondary loop* and contains the *slave* controller; the outer is called *the primary loop* and corresponds to the *master* controller.

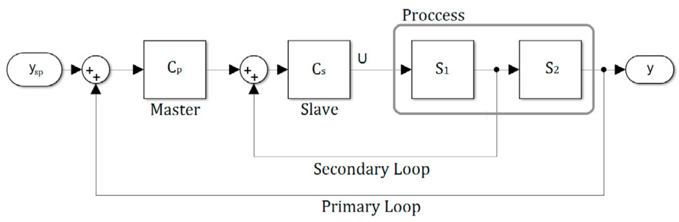

Fig. 5. Cascade control structure.

Cascade control is useful in systems where the access to several inner variables is possible. In the current project, two cascade control strategies were applied, one per subsystem. The main advantage provided by cascade control is to use the inner variable measurements to a quicker disturbance rejection before it influences the primary variable. Hence, it is important that the essential disturbances act in the inner loop. Finally, the secondary loop must have a dynamic at least four times faster than the primary loop.

As showed on Fig. 6, two cascade structures are used to regulate the position coordinates X and Y where the speed components v_x, v_y are chosen as the inner variables. This is consistent with the fact that essential perturbations on a flight are mainly air currents, which are detected by speed measurements as air is the fluid in which the quadcopter flies.

3.2 PID controller

In first place, the well-known Proportional Integral Derivative (PID) controller is employed for both, the primary and secondary loops. To control a discrete transfer function at the simulation phase, the *Discrete PID Controller* block is used inside MATLAB Simulinlk. As antiwindup method, the conditional integration was chosen, as exposed on Packard (2005):

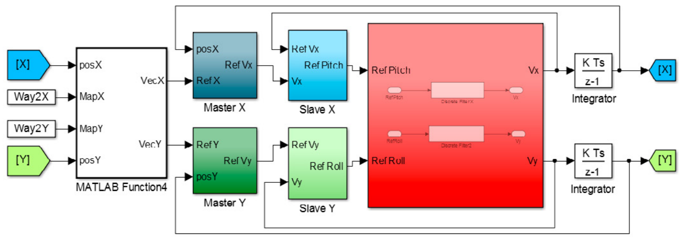

Fig. 6. Two parallel cascade control architectures on our system.

- If $u \geq U_{max} > 0$, and $r - y < 0$, continue integrating.
- If $u \geq U_{max} > 0$, and $r - y > 0$, stop integrating.
- If $u \leq U_{min} < 0$, and $r - y > 0$, continue integrating.
- If $u \leq U_{min} < 0$, and $r - y < 0$, stop integrating.

With u as control signal, U_{max} and U_{min} as saturation points, r as the set-point and y as the process variable.

As shown in (1), the forward approximation has been used, where P, I, and D are the parameters corresponding to the proportional, integral and derivative actions respectively. N is the derivative high frequency filter implemented to noise reduction and T_s is the sampling time, equal to 0.06 seconds.

$$Y = P + IT_s\frac{1}{z-1} + D\frac{N}{1 + NT_s\frac{1}{z-1}} \qquad (1)$$

The PID controller was used as the controller on primary loops. On the secondary loops, a comparison between the performance of PID controllers against GPC (Generalized Predictive Control) controllers was made.

3.3 GPC controller

This paper does not explain the complete theory about Generalized Predictive Control, it focuses only on how it was implemented. To begin with, the model used to obtain the output prediction is the transfer function of each subsystem:

$$(1 - 1.646z^{-1} + 0.6687z^{-2})v_x(t) = 0.2z^{-4}u_x(t-1) + \frac{e(t)}{\Delta}$$

$$(1 - 1.716z^{-1} + 0.73z^{-2})v_y(t) = 0.1z^{-3}u_y(t-1) + \frac{e(t)}{\Delta}$$

Where $e(t)$ is a white noise and $\Delta = 1 - z^{-1}$.

The applied control sequence minimizes a multi-stage cost function of the form described in (2), where the deviation from desired future behaviour and the increments of control actions are penalized.

$$J(N_1, N_2, N_u) = \sum_{j=N_1}^{N_2} \delta(j)\left[\hat{y}(t+j|t) - \omega(t+j)\right]^2 + \sum_{j=1}^{N_u} \lambda(j)\left[\Delta u(t+j-1)\right]^2 \qquad (2)$$

With N_1 and N_2 as the minimum and maximum costing horizons, N_u as the control horizon, $\delta(j)$ and $\lambda(j)$ as weighting sequences and $\omega(t+j)$ as the future reference trajectory, which is usually a smooth approximation from the current value of the output $y(t)$ towards the known reference.

As the future references on velocity are not known, it is assumed that they are all equal to the actual reference that comes out of the master controller each sample time, missing the anticipation on set-point changes.

Even though some benefits of the Predictive Control are lost, the GPC is employed here as a method to obtain an optimally tuned controller equivalent to a classic controller. One secondary goal of this paper is to settle the bases for future applications of the GPC using the AR.Drone platform, replacing the cascade structures by two GPCs preceded by a trajectory generator (for position and velocity trajectories).

We set $\delta(j) = 1$ and $\lambda(j) = k, \ \forall\ j = 1..N_u$ with k as the tuning parameter. As recommended in Camacho and Bordons A (2004), the horizons were set as follows:

$$N_1 = d+1 \ ; \ N_2 = d+N \ ; \ N_u = N \ , \ with\ N = 50 \quad (3)$$

N is the number of sampling times that is necessary to cover the main transient state until the steady state is reached, which is in this case 3 seconds and the sampling time $T_s = 0.06\,s$. The parameter d is the dead time on each subsystem, which are $d_x = 4$ and $d_y = 3$ sampling times.

As the model identified relates the angles pitch and roll with de components of the speed v_x, v_y, the GPC controllers were used on the inner loops instead of the outer. Thus, as the GPC depends heavily on the accuracy of the model, we assume that the speed values are more reliable than the position ones (which accumulate the error due to the action of the integrator).

The GPC controllers were implemented *MATLAB Function* blocks the same way of the examples exposed on Camacho and Bordons A (2004). To implement this controller on the real system, in order to avoid the inversion of a 50x50 matrix each sample time, the control law was pre-calculated fixing the parameter λ for different values and then implemented on $C\#$.

3.4 Improvements to path tracking

After some simulations with MATLAB Simulink, it was observed that the quadcopter behaviour, obtained employing the control architecture explained before, did not reach the performance requirements (explained on 4.3).

Before applying these improvements, the trajectory generation consisted on changing the set-point of master controllers into the next waypoint as soon as the quadcopter reached the last waypoint (it is considered radius of 10 cm, 14 cm or 20 cm to reach them and is another parameter to tune).

When the trajectory is composed of displacements on the two coordinates, as the two cascade control strategies are independent and decoupled of each other, the behaviour of *X subsystem* is, in general, different from the behaviour of the *Y subsystem*. Hence, a deviation from the straight path connecting the last and the next waypoint appears, Fig. 7.

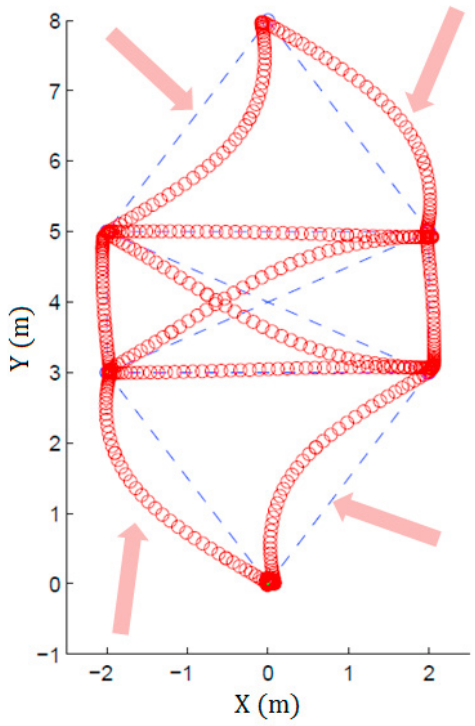

Fig. 7. Deviation from the path.

To partially avoid this behaviour, two different strategies were compared:

Intermediate points generator

A more complex trajectory generation was elaborated by sending some intermediate points between each pair of waypoints as reference. This strategy synchronizes the two primary loops at intermediate positions. Thus, there are three new parameters of tuning: the number of segments in which the trajectory between two waypoint is divided, the length of them and the approximation distance selected to change the reference from an intermediate point to the next one. Hence, the highest possible deviation from the path is indirectly set with the approximation distance while time speed performance is scarified.

Velocity limiter

Another strategy to avoid path deviation is the one proposed on Hernández-Hernández et al., 2013. It consists on rectifying the reference vector of velocity coming out of the master controllers V_o. The components of the velocity reference are decoupled (as seen before) and this technique allows the synchronization of secondary loops on *X Subsystem* and *Y Subsystem* respectively.

The strategy is based on making the reference velocity vector to point at any time to the next waypoint. To achieve this, the orthogonal projection V_r of the velocity reference vector V_o onto the vector connecting the actual position with the objective waypoint is calculated.

After that, the projection V_r is the new reference vector and needs to be limited by a saturation imposed by the physical maximum velocity or by the user (as less value has the saturation, less deviation from the path has the quadcopter). The result is V_{lim} (Fig. 9).

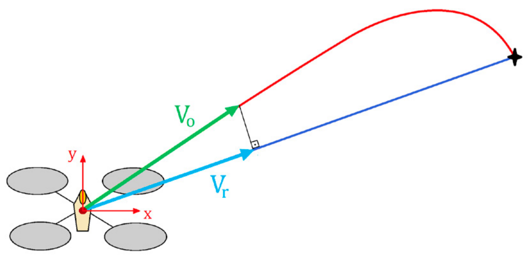

Fig. 8. Correction of velocity reference, by the orthogonal projection over the the displacement vector

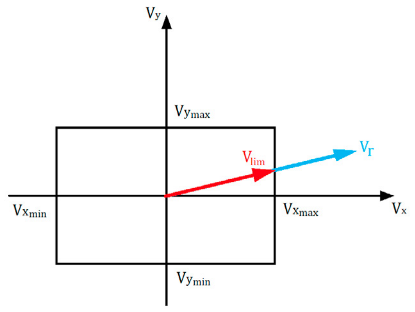

Fig. 9. Reference velocity saturation.

The *Intermediate points generator* is implemented on a *MATLAB Function* block to be tested before its implementation with C# language. The block receives the output signal of the master controllers and provides the set-point signal of the slave controllers. Both strategies are compatible with each other but here they are compared to obtain different architectures of control.

4. SIMULATION RESULTS

4.1 PID tuning

The ITAE minimization criterion was chosen as PID tuning method, due to the good performance indicated on the literature. It is about setting the parameters P, I, D according to the minimum value of the ITAE value along a simulation.

$$ITAE = \int_0^\infty t|e(t)|dt \tag{4}$$

The implementation is simple when the MATLAB Optimization Toolbox is used together with Simulink, as explained in Martins (2005). Thus, a function that finds a local minimum of the ITAE value is called, starting from a initial tuning of the PID controller. Each iteration means a simulation of the Simulink file with our controller with a step function as set-point.

Depending on the step input, established inside the Simulink file, the minimum ITAE criterion provides a more aggressive tuning. The master controllers obtained with this criteria are tuned as PD instead of PID. Having PIDs as master controllers would manage worse the overshoot of the subsystems (X, Y) and there is no need of the integration part to be stable because both subsystems already have one integrator each one. The parameter N is set to 10 in all PID controller and is not a variable to adjust.

4.2 GPC tuning

The ITAE minimization criterion does not have the same efficacy with GPC tuning because the control signal fluctuates abruptly. Starting from the tuning gave by the ITAE criterion, we incremented λ until the control signal was soft enough, losing response time.

4.3 Performance

Three different adjustments were tested, two PIDs–PIDs strategies and one PIDs–GPCs. Each adjustment was tested in three configurations: with the *Velocity limiter* block, with the *Intermediate point generator* and without them.

To evaluate the performance of each controller along the simulations, we used the evaluation criteria of the competition on Control Engineering 2012, *Autonomous trajectory control of a quadcopter vehicle*, proposed on Blasco et al. (2012).

It consists on a multi-objective evaluation, allowing us to compare the time required to complete the circuit, the maximum deviation and the average deviation with a performance index.

The best performance according to this criteria was the PIDs-GPCs strategy with the *Velocity limiter* block:

- PID–GPC X: $\|\mathbf{P}: 0.951\|\mathbf{I}: 0\|\mathbf{D}: 0.293\|\lambda = 250\|$
- PID–GPC Y: $\|\mathbf{P}: 0.913\|\mathbf{I}: 0\|\mathbf{D}: 0.277\|\lambda = 150\|$

5. EXPERIMENTAL RESULTS

The best tuned PIDs–PIDs and the PIDs–GPCs strategies were implemented on the real system. It is observed that the identified model has a slightly different static gain than the real system and we had to adjust the controllers.

The GPCs controllers had an unstable behaviour and it was necessary to tune them with a higher λ on both slave controllers until $\lambda = 1500$, and to multiply their output signal by a factor (0.8 on v_x controller and 0.9 on v_y controller). Furthermore, the GPC controllers still had an unacceptable reaction to the noise on velocity measurements and a low-pass filter on the measured signal was implemented.

With all this modifications, the performance of the GPC strategy got worse. Nevertheless, the PIDs-PIDs strategy, which is more robust to differences between identified model and real system, worked well with the AR.Drone after a few modifications on the parameters. No low-pass filter was needed because it already had one on the derivative action.

The PID-PID *speed limiter* strategy had the best performance with the real system followed by the PID-PID with *intermediate points generator*. Despite the fact that the last one had less average and maximum deviation from path, the speed limiter block provides better time

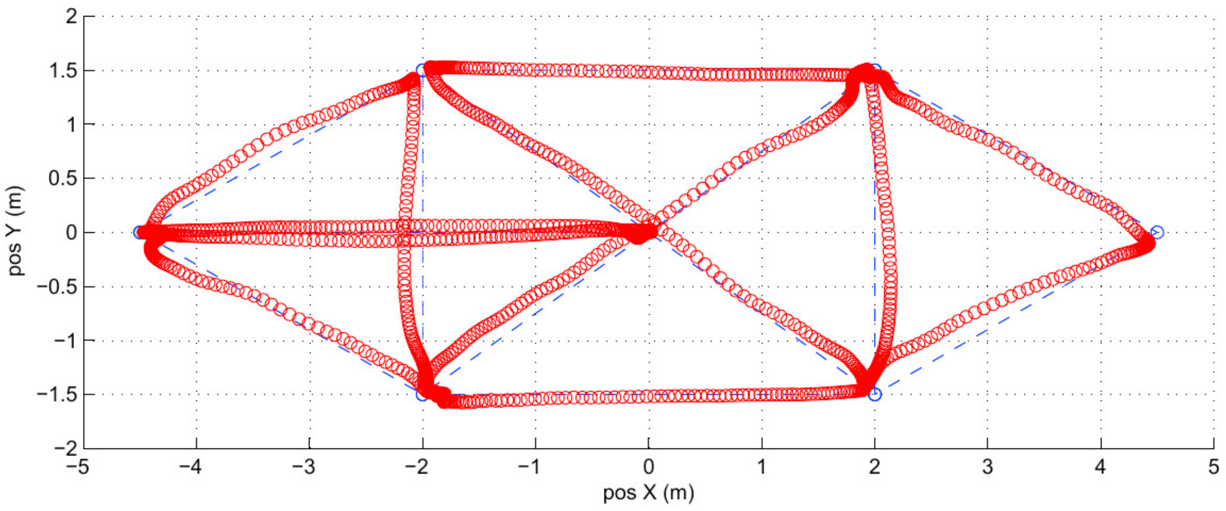

Fig. 10. Path tracking with real system. PID-PID speed limiter strategy.

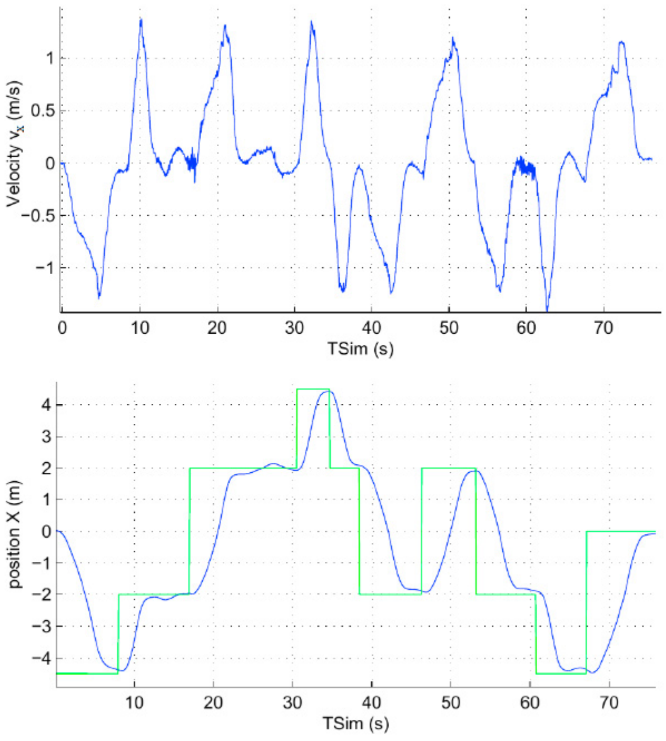

Fig. 11. Velocity v_x and X position on real system.

performance with an acceptable path deviation, as can be observed in Fig. 10.

- PID master (X): $\|\mathbf{P}:1\|\mathbf{I}:0\|\mathbf{D}:0.4\|$
- PID slave (v_x): $\|\mathbf{P}:0.23\|\mathbf{I}:0.28\|\mathbf{D}:0.027\|$
- PID master (Y): $\|\mathbf{P}:0.9\|\mathbf{I}:0\|\mathbf{D}:0.01\|$
- PID slave (v_y): $\|\mathbf{P}:0.6\|\mathbf{I}:0.05\|\mathbf{D}:0.07\|$

6. CONCLUSION

According to the results obtained within this research, it can be concluded that the PIDs-PIDs strategy is robuster against differences between the identified model and the real system, because the GPC controllers depend strongly on the model.

Each strategy added to solve path deviation shows a good performance depending on the requirements to be fulfilled. The *speed limiter* strategy does not correct the path as well as the *point generator*, but provides the quadcopter with better time performance.

Introducing computer vision based strategies using the frontal camera could make it possible to add more capabilities to the autopilot. For example following objects in motion or the navigation with external references.

ACKNOWLEDGEMENTS

Assistance provided by Prof. Manuel G. Ortega was greatly appreciated.

REFERENCES

Åstrom, K.J. and Hägglund, T. (2006). *Advanced PID Control*. ISA, Research Triangle Park, NC 27709.

Blasco, X., García-Nieto, S., and Reynoso-Meza, G. (2012). Control autónomo del seguimiento de trayectorias de un vehículo cuatrirrotor. Simulación y evaluación de propuestas. *Revista Iberoamericana de Automática e Informática Industrial, Elsevier*, 9, 194–199.

Camacho, E.F. and Bordons A, C. (2004). *Model Predictive Control*. Springer, Sevilla.

García-Nieto Rodríguez, S., Blasco Ferragud, F.X., Sanchís Saez, J., Herrero Durá, J., Reynoso Meza, G., and Martínez Iranzo, M.A. (2012). *TrackDrone Lite 2.0*. URL `http://hdl.handle.net/10251/16427`.

Hernández-Hernández, L., Pestana, J., Casares-Palomeque, D., Campoy, P., and Sánchez-López, J.L. (2013). Identificación y control en cascada mediante inversión de no linealidades del cuatrirrotor para el Concurso de Ingeniería de Control CEA IFAC 2012. *Revista Iberoamericana de Automática e Informática Industrial, Elsevier*, 10.

Martins, F.G. (2005). Tuning PID Controllers using the ITAE Criterion. *International Journal of Engineering Education*, 21, 867–873.

Packard, A. (2005). *Feedback and Dynamic Systems, course notes. Saturation and antiwindup strategies.* Beckerley, University of California.