Towards an autonomous airborne robotic agent

Daniel Soto-Guerrero[1], José Gabriel Ramírez-Torres[2] and Jean-Pierre Gazeau[3]

[1,2] *Cinvestav Tamaulipas, México e-mail: dsoto@tamps.cinvestav.mx, e-mail: grtorres@tamps.cinvestav.mx*
[3] *Institut Pprime, CNRS Université de Poitiers e-mail: jean.pierre.gazeau@univ-poitiers.fr*

Abstract.

Commercialy available Unmanned Aerial Vehicles (UAVs) rely on the Global Positioning System (GPS) to define their flight plan, while assuming an obstacle-free environment. The work presented on this article aims to set the foundation towards an autonomous airborne agent, capable of locating itself by means of computer vision, modeling its environment, planning and executing a three dimensional trajectory. On the first stage of development we solved the localization problem using artificial markers and tested a PID controller to make the vehicle follows a given trajectory (a lemniscate); as results, we show flight data captured during real flights. This development would facilitate the integration of far more complex flight behaviors than GPS only guided flight plans.

1 Introduction

In order to make an autonomous agent out of a UAV, it is required to locate the vehicle with respect to a fixed reference frame, to know its environment and to command it to navigate autonomously. So, given a fully functional UAV with an onboard camera, we aimed to: *a*) guarantee a safe operation of the UAV *b*) locate it with respect to a fixed reference frame on the ground, using visual feedback and *c*) control the vehicle so it can execute flight patterns.

The following sections describe the accomplished intellectual developments, the architecture of the control application, its capabilities and further possible developments.

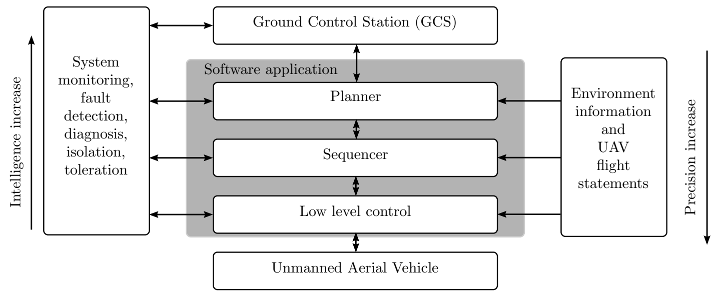

Ground Control Station (GCS)
Software application
Planner
Sequencer
Low level control
Unmanned Aerial Vehicle
System monitoring, fault detection, diagnosis, isolation, toleration
Environment information and UAV flight statements
Intelligence increase
Precision increase

Fig. 1: The three-layer software architecture for an autonomous robot.

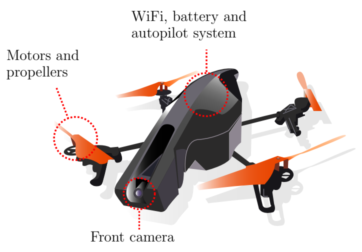
2 Related work

A robot consists of a series of highly heterogenous systems that are complex in nature and require an orchestrated integration to function properly, some of those features to name a few are: multi-robot coordination, collision avoidance, human interaction and planning. Therefore, control architectures are proposed to organize by hierarchies the modules providing different functionalities; among one of the most important, the posibility to have a digital representation of the environment on the computer controlling the robot.

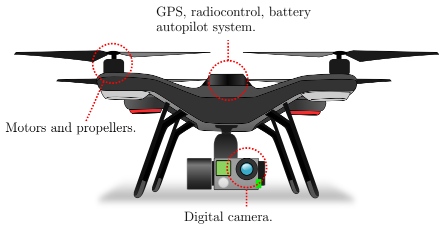
The hierarchical architecture of the control application is based on designs already tested in mobile robotics, such as the one proposed by Chen *et al.* [1] (see Fig.1). The architecture consists of three layers: *a*) the Low level control layer allows to directly manage and access all hardware peripherals in real-time. *b*) the planner is the process that gives the current status of the robot and its environment, creates a plan for the robot to achieve a certain goal, *c*) the sequencer is the intermediate layer between the low level control and the planner that implements a set of *well-tested* [7] behaviors that can be used in sequence to execute the plan created by the planner.

Similar architectures to the one shown on Fig. 1 have been used for service robotics, industrial robots interacting with humans [2] and a group of identical robots [10, 5]; their description may differ, but all of their corresponding software architectures can be shaped to a three layer architecture so the objective would remain the same: to provide sensing, planning, supervision and execution capabilities to fulfill a task. As an example, the ability layer on a service robot mentioned by Luna-Gallegos *et al.* [7] can be grouped into the sequencer layer mentioned on this paper as a set of *well-tested* behaviors.

On the field of UAVs, control architectures have been tested following a reactive approach, *i.e.* they act proportionally to an error metric, usually defined by finding and tracking an object with computer vision [12, 11]. On this article, we describe how we plan to develop a three layer architecture for the control of UAVs and the first steps we have taken.



(a) The AR-Drone.



(b) Front side of the Solo.

Fig. 2: The two drones tested.

3 Hardware description

This work was successfully tested with two different vehicles, for which we had to use two different versions of the Low level control layer. The first UAV we tested was the Solo from 3D Robotics, which is compatible with the MAVLink protocol [8] and the second vehicle we tested was the AR-Drone v2. Both platforms are ready-to-fly UAVs and feature an onboard monoscopic camera and a WiFi link.

To connect to the AR-Drone, we used the package created to control it with ROS. The software development was based on the Linux operating system and the Robotic Operating System (ROS) [9]. For the Solo (see Fig. 2b) we used Gstreamer[1] to receive the video feed and Dronekit (the python library created to interface with UAVs compatible with MAVlink) to gain access to the vehicle. For each vehicle we have a hardware remote control, the pilot has the option to intervene or not in basic maneuvers such as take-off and landing. We gave a bigger priority to pilot commands over autonomous control; in case of unforeseen situations, the pilot can bypass the autonomous control immediately by operating the hardware controller.

The approach we tested was implemented into a three layer architecture on ROS, this simplifyed its development and looking into the future it will make possible two things: the creation of a swarm of UAVs and its migration onboard the UAV. Increasing the independency of the UAV from the Ground Control Station (GCS).

4 Proposed approach and methodology

The overall disposal of all components, according to the three layer architecture structructure is shown in Fig. 3. The hardware interface to the AR-Drone

[1] Webpage: http://gstreamer.freedesktop.org/

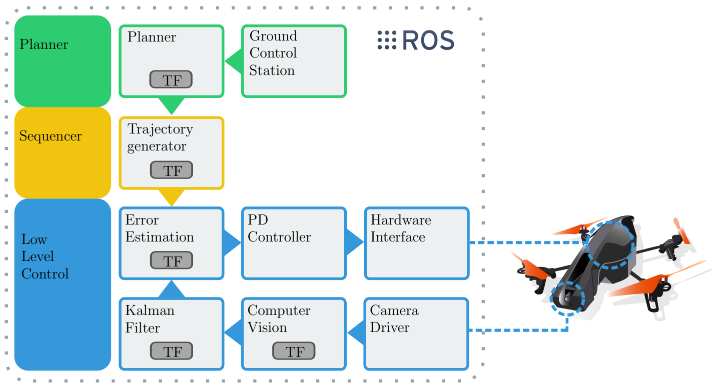


Fig. 3: The three-layer architecture for the UAV

was the only component fully functional and running on ROS when this development started. From top to bottom we show the GCS and the planner node, the planner remains as future work. The trajectory planner generates the lemniscate trajectory and defines the desired position for the drone $\mathbf{r}_d(t)$. The low level control consists on several nodes, the first one being a hardware interface to the flight controller of the drone and the camera. The current state estimation is accomplished with the computer vision and kalman filter nodes, the current state is then used to define a proper control command in the error estimation and PD controller nodes.

As mentioned before, we aimed to locate the UAV using visual feedback to command it and describe a certain trajectory. The proposed scenario is shown on Fig. 4, the UAV overflies artificial markers fixed on the ground, pointing its camera downwards, the video feed and navigation data are sent to the GCS for processing. Figure 4 also shows the reference frames attached to the monoscopic camera C, the world reference frame W, the center of gravity of the vehicle B and the NED frame (X: North, Y: East, Z: Down).

Dealing with spatial relationships between reference frames is a very common task in robotics, expressed as homogeneous transformations ${}^{W}_{C}\mathbf{T}$ the rigid body transformation from reference frame C to W is denoted by:

$$ {}^{W}_{C}\mathbf{T} = \begin{pmatrix} {}^{W}_{C}\mathbf{R} & {}^{W}_{C}\mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} $$

where ${}^{W}_{C}\mathbf{R} \in SO(3)$ and ${}^{W}_{C}\mathbf{t}$ are the rotation and translation components, respectively. We used the work from *Foote* [3] to manage all rigid body transformations. Note that by solving ${}^{W}_{C}\mathbf{T}$, we can locate B with respect to W. Because the camera is rigidly mounted on the UAV ${}^{B}_{C}\mathbf{T}$ is known beforehand and the location of B with respect to W can be computed with ${}^{B}_{W}\mathbf{T} = {}^{B}_{C}\mathbf{T}\,{}^{W}_{C}\mathbf{T}^{-1}$. To compute ${}^{W}_{C}\mathbf{T}$ we used the technique developed by Garrido *et al.* [4]; which

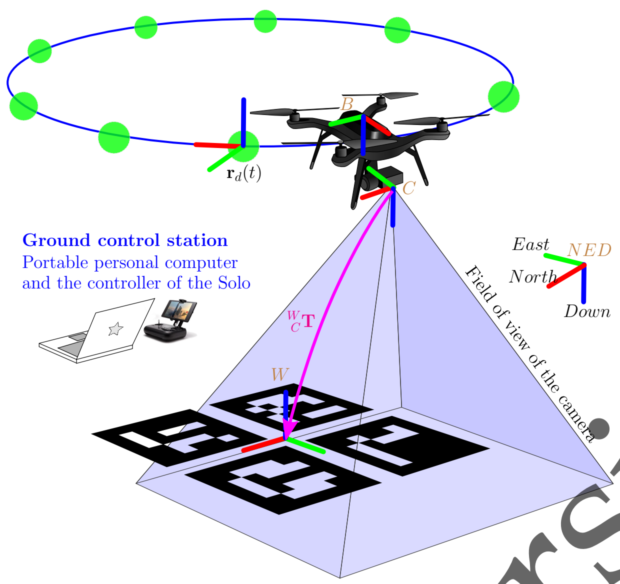

B
$\mathbf{r}_d(t)$
C
Ground control station
Portable personal computer
and the controller of the Solo
East
NED
North
Down
Field of view of the camera
${}^W_C\mathbf{T}$
W

Fig. 4: The use case scenario for the 3DR Solo. For every reference frame, the color convention is: X axis red, Y axis green and Z axis blue. Notice the orientation of B is similar to the NED reference frame.

consists on segmenting from the images taken by the camera the artificial markers located on the ground, beacause the size of every marker is known, the pose of the camera is estimated from all the detected corners.

We added a Kalman filter [6] over ${}^B_W\mathbf{T}$ to improve the resilience to errors caused by inaccuracies in inertial measurements, camera parameters, corner detection, image rectification and pose estimation. The state vector for the Kalman filter was defined as $\mathbf{x} = [x, y, z, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\psi}]$, it defines the position and velocities of the XYZ coordinates and the yaw around the z-axis angle with respect to W. From the inertial sensors onboard the UAV, we receive the horizontal velocity components with respect to the B reference frame, current flight's height and orientation $\mathbf{z}_I = [v_x, v_y, h, \Psi]_{k,B}$ at 200Hz; the *a priori* estimate of the Kalman filter was updated using the inertial measurements with respect to W by rotating Ψ radians around the yaw ψ axis. The state transition model is:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{k+1} = \begin{bmatrix} x \\ y \end{bmatrix}_k + \Delta_t \mathbf{R}_z(\psi) \begin{bmatrix} v_x \\ v_y \end{bmatrix}_k \qquad \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}_{k+1} = \mathbf{R}_z(\psi) \begin{bmatrix} v_x \\ v_y \end{bmatrix}_k$$

$$\dot{z}_{k+1} = \frac{h_k - z_k}{\Delta_t} \qquad \dot{\psi}_{k+1} = \frac{\Psi_k - \psi_k}{\Delta_t}$$

$$z_{k+1} = h_k \qquad \psi_{k+1} = \Psi_k$$

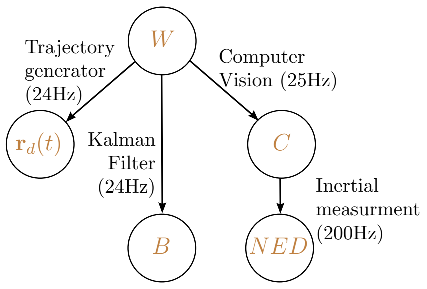


Fig. 5: The graph representing the rigid body transformation between frames.

The *a posteriori* step runs at 24 Hz, a slower rate than the *a priori*, using as measurement the pose of the camera $\mathbf{z}_C = [x, y, z, \psi]_k$, estimated by computer vision [4]. After the update process in the Kalman filter, state vector $\mathbf{x}$ defines the latest estimation for the pose of the UAV with respect to W, *i.e.* ${}^{B}_{W}\mathbf{T}$.

For now, the trajectory to be described by the vehicle is a lemniscate, defined as a parametric function $\mathbf{r}_d(t)$ that defines the desired position and pose (Euler angles: roll θ_d, pitch ϕ_d and yaw ψ_d, see Fig. 6b):

$$\mathbf{r}_d(t) = \begin{bmatrix} x_d(t) \\ y_d(t) \\ z_d(t) \\ \psi_d(t) \end{bmatrix} = \begin{bmatrix} a\sin(\frac{t}{\epsilon}) \\ b\sin(\frac{2t}{\epsilon}) \\ c\sin(\frac{3t}{\epsilon}) \\ 0 \end{bmatrix} \tag{1}$$

Figure 5 shows the resultant directed graph using nodes as reference frames and labels on edges as the processes that update the spatial relationship between two reference frames. The direction of every edge represents the origin and target frames of the homogeneous transform. Then, the error measurement is given by:

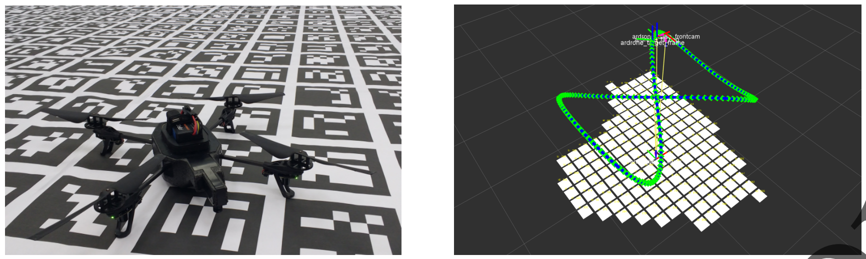
$${}^{r_d}_{B}\mathbf{T} = \begin{bmatrix} \mathbf{R}_e & \mathbf{t}_e \\ \mathbf{0} & 1 \end{bmatrix} = {}^{r_d}_{W}\mathbf{T}\,{}^{B}_{W}\mathbf{T}^{-1}$$

After decomposing $(\theta, \phi, \psi)_e = \mathbf{R}_e$ on its three Euler angles we can compute a control command using a Proportional-Derivative controller:

$$\mathbf{u} = K_p \begin{bmatrix} \mathbf{r}_d(t) - \mathbf{x} \\ \psi_e \end{bmatrix} + K_d(\dot{\mathbf{r}}_d - \dot{\mathbf{x}})$$

where $\mathbf{x} = [x, y, z, \psi]$ and $\dot{\mathbf{x}} = [\dot{x}, \dot{y}, \dot{z}, \dot{\psi}]$ are estimated by the Kalman filter described before.

5 Results



(a) Artifical markers and the AR-Drone. (b) The virtual scneario on Rviz.

Fig. 6: The software architecture working, creating a virtual representation of the real world and locating the drone with resect to the center of the ArUco board.

The proposed approach was tested with the AR-Drone 2.0 and the 3DR Solo. The AR-Drone was modified, so the front camera pointed downwards and we could get a higher quality image from above the ground level. The Solo had a gimbal installed, as a result, we dynamically had to compute ${}^{B}_{C}\mathbf{A}$ using the navigational data we received from the UAV. The camera settings for the GoPro are very versatile, for this exercise, we used a *narrow* field of view with a resolution of 1028×720 pixels. The AR-drone was flown indoors at a maximum altitude of 1.4 meters, the Solo flew outdoors and gained altitude to 5 meters above ground level.

The computer vision algorithm was set to track a board of artificial markers with different sizes; for the Solo the board measured $1.4 \times 2.4m$ and 2×5 artificial markers, for the AR-Drone we used a board $4 \times 4m$ and 20×21 markers (see Fig. 6a). The application here described creates a virtual representation of the world on Rviz, an standard tool on ROS; what is shown on Fig. 6b is an screenshot of Rviz displaying: the location of the vehicle, the trajectory to follow and the detected board.

On Fig. 7, we display the results as measured by the computer vision system while executing the lemniscate maneuver in x and y coordinates with respect to W. The $\mathbf{r}_d$ plot is the desired trajectory, corresponding to the lemniscate, for completeness, we also display the error plot. The maximum measured error was 30 cm. The paramters for the lemniscate trajectory with the AR-Drone were: $a = 1.0$, $b = 0.8$, $c = 0.2$, $\epsilon = 30.0$, with a height offset of $z = 1.2$.

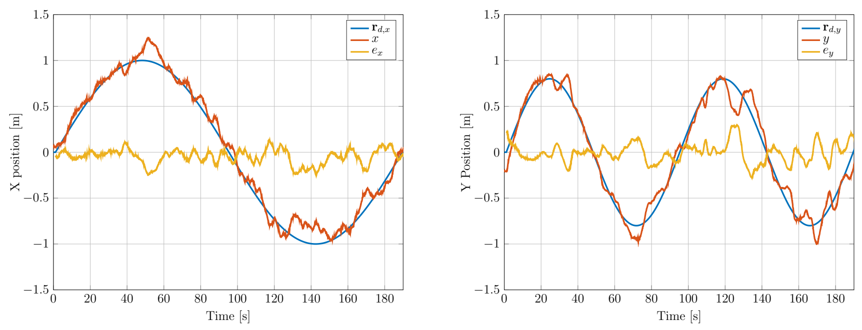


Fig. 7: Navigation data of an actual flight of the vehicle describing the lemniscate trajectory. On both graphs we display the desired trajecty, the estimated position and the error metric.

6 Conclusions and future work

We have discussed a three layer architecture intended for the control of UAVs, that successfully guided the vehicle to describe the spline trajectory. Because the framework we used for this development runs on multiple platforms, including ARM on embedded computers, it is plausible to execute it onboard the UAV. Further development on the Sequencer and Planner layers would make the UAV and autonomous agent and leads the way towards a swarm of UAVs. Additionaly, the planner that defined the waypoints will be extended with a path-finding algorithm. This architecture will make possible to integrate far more complex flight plans and do not only rely on GPS for positioning.

This document shows the results from the first step on our development and implementation roadmap. The next step is to execute it onboard the UAV. We are currently looking forward to extending the computer vision system with an visual odometry approach.

References

1. H. Chen, X. m. Wang, and Y. Li. A survey of autonomous control for uav. In *Artificial Intelligence and Computational Intelligence, 2009. AICI '09. International Conference on*, volume 2, pages 267–271, Nov 2009.
2. G. Dumonteil, G. Manfredi, M. Devy, A. Confetti, and D. Sidobre. Reactive planning on a collaborative robot for industrial applications. In *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, volume 02, pages 450–457, July 2015.

3. Tully Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pages 1–6, April 2013.
4. S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014.
5. J. Goryca and R. C. Hill. Formal synthesis of supervisory control software for multiple robot systems. In *2013 American Control Conference*, pages 125–131, June 2013.
6. Phil Kim. *Kalman Filter for Beginners*. A-JIN publishing company, 2011.
7. K. L. Luna-Gallegos, E. R. Palacios-Hernandez, S. Hernandez-Mendez, and A. Marin-Hernandez. A proposed software architecture for controlling a service robot. In *2015 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, pages 1–6, Nov 2015.
8. L. Meier, D. Honegger, and M. Pollefeys. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, may 2015.
9. Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
10. M. Schöpfer, F. Schmidt, M. Pardowitz, and H. Ritter. Open source real-time control software for the kuka light weight robot. In *2010 8th World Congress on Intelligent Control and Automation*, pages 444–449, July 2010.
11. F. Vanegas and F. Gonzalez. Uncertainty based online planning for uav target finding in cluttered and gps-denied environments. In *2016 IEEE Aerospace Conference*, pages 1–9, March 2016.
12. L. Yang, B. Xiao, Y. Zhou, Y. He, H. Zhang, and J. Han. A robust real-time vision based gps-denied navigation system of uav. In *2016 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 321–326, June 2016.