Reengineering the Paparazzi autopilot navigation system

Antoine Drouin, Pascal Brisset, Yannick Jestin

REENGINEERING THE PAPARAZZI AUTOPILOT NAVIGATION SYSTEM [1]

Antoine Drouin [*,1] **Pascal Brisset** [*,1]
Yannick Jestin [*]

[*] *ENAC, Toulouse, France*

Abstract: Over the past three years, the Paparazzi project has created a free micro air vehicle system. Emerging uses of unmanned air vehicles has entailed new interactions to control the flight. The project has now reached a milestone and requires a reengineering of the navigation kernel interface.
After describing the current architecture of our system, we analyze the new needs arising from human machine interaction and flight management system. We then propose a layered design of the navigation functional core.

Keywords: Autopilot architecture, MAV, HMI, reengineering.

1. INTRODUCTION

The Paparazzi project has created a free [2] umanned air vehicle (UAV) system. It consists in airborne hardware and software (the autopilot), ground infrastructure (the ground station), communication protocols (the datalink) and a simulation environment. It started in 2003 as a hobby project and is now developed by ENAC and a community of researchers, hobbyists and industrials.

The autopilot is able to perform autonomous navigation on fixed wing vehicles. It is highly configurable and has been flown on a variety of vehicles ranging in mass from 250g (figure 1) to several kg.

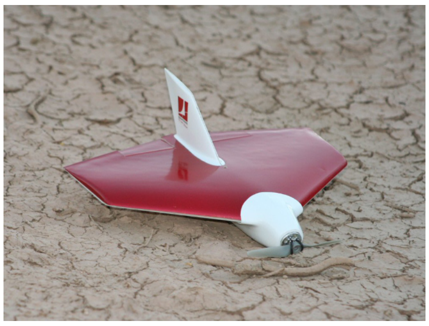
Fig. 1. The 30cm span, 300g Slayer from Miraterre Flight Systems.

Operating an autonomous micro air vehicle (MAV) involves a number of different tasks:

- The one shot configuration and tuning of the autopilot for a given vehicle;
- The flight plan preparation, possibly including simulation;
- The flight monitoring, possibly including real time modifications;
- The flight data analysis.

In this paper, we describe the current architecture of the Paparazzi system which allows to perform the above mentioned tasks. We then describe news needs arising from recent case studies that were incompatible with the current navigation kernel. We end up by presenting a new architecture for the navigation kernel.

[1] Partially supported by US Army contract R&D 1042-AM-01
[2] FLOSS: Free, Libre, Open-Source Software

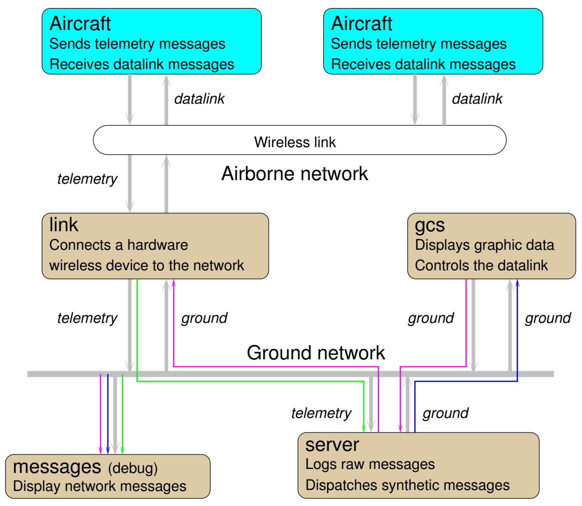

Fig. 2. The agents of the Paparazzi system. Aircraft, link and gcs agents may appear several times.

2. THE PAPARAZZI SYSTEM

The Paparazzi system is extensively described in (Brisset and Drouin, 2004; Brisset *et al.*, 2006) and cooperatively documented in a wiki[3].

This section focuses on the distributed architecture, the airborne subsystem, the navigation scripting language and the current available interactions with the navigation kernel.

2.1 *Distributed Architecture*

In order to offer maximum flexibility and openness, the system was designed from start as a distributed one (figure 2). Ground agents use the http protocol for data transfers and the Ivy software bus (Buisson *et al.*, 2002) for middleware. Ivy provides a basic message subscription mechanism and is available for most languages and architectures.

The system was designed to allow simultaneous operation of several vehicles and multiple ground stations.

The airborne vehicles may share a RF network or use a point to point connection to the ground station.

The "link" agent acts as a gateway between the RF side of the world and Ivy. It can speak different protocols and provides retry and acknowledgment services for hardware lacking those functionalities.

The "server" is a central agent that holds the configuration of the aircrafts (flight plan, settings etc) and other data (maps, topographic). It process the raw aircraft telemetry messages, log them for replay or analysis. It can redispatch synthetic

[3] www.recherche.enac.fr/paparazzi

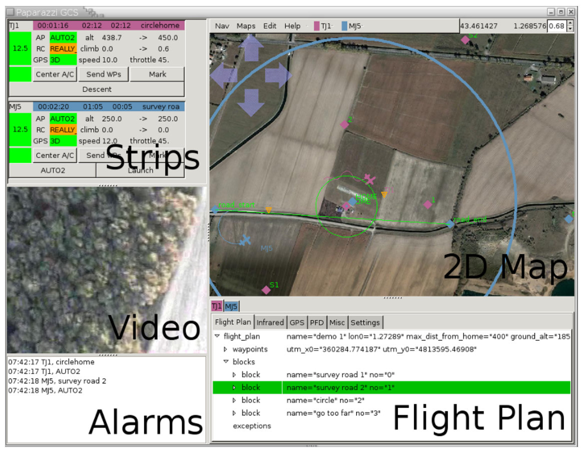

Fig. 3. The current ground control station

asynchronous message to agents like ground control stations.

The ground control station (GCS) allows for configuration, flight plan edition, simulation, flight control and log analysis. It typically runs in a laptop (figure 3).

"Simulators" are just regular ground agents and can be mixed with real vehicles or replay data.

The "Gaia" agent provides simulated data for environmental factors such as wind conditions and GPS coverage.

2.2 *Airborne subsystem*

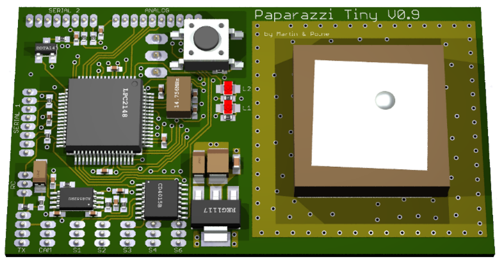

Fig. 4. The *tiny* controller board includes a GPS receiver and antenna

The figure 4 shows a "Tiny" controller board, which at 20g and 70mm*40mm, is the most integrated hardware supported by Paparazzi.

One of the unique features of the system is the use of infrared thermopiles for attitude sensing (figure 5 shows a typical airborne setup). For the airborne code, focus has been placed on security and reliability with the use of formal methods, code generation and critical code segregation(Albert, 2005).

2.3 *Navigation Scripting*

The behavior of the vehicle while in autonomous mode is described using a scripting language.

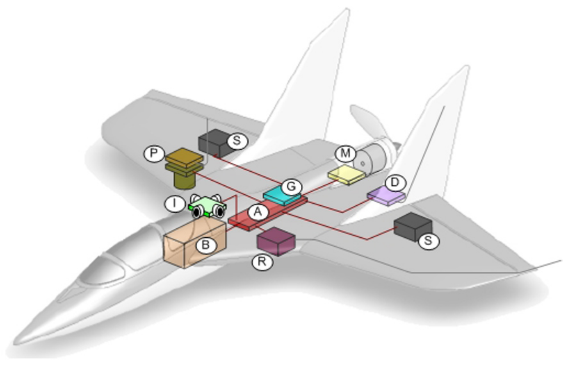

Fig. 5. The Paparazzi airborne system components: (A)utopilot Control Board, (B)attery, (D)atalink Radio-Modem & Antenna, (G)PS Receiver, (I)R Sensors Board, (M)otor & Controller, (R)C Receiver & Antenna, (S)ervos, (P)ayload = Camera & Video Transmitter

This language defines stages and defines blocks as groups of stages. It provides a complete flow control mechanism (jump, conditional, loops, etc.) as well as expressions evaluation and exceptions.

Parameters used in a flight plan can be computed expressions. In the next example, UAV is asked to perform 5 circles at progressively increasing altitudes for exactly one minute at each altitude:

```
<for var = "i" from = "1" to = "5">
  <circle wp = "HOME" radius="75"
          alt = "ground_alt+50*$i"
          until = "stage_time>60" />
</for>
```

We provide an API to access the hardware, so that the control of peripherals such as a pan/tilt camera or landing gears/lights is possible from the flight plan.

Complex functions such as landing procedures can be factorized in libraries and called from the flight plan.

Being a Turing-complete language, it guarantees that any problem can be addressed and offers a very high flexibility. The tradeoff is that its complexity makes it poorly suited to real time interactions as we'll see in the next part.

3. NEW REQUIREMENTS

We now present two case studies pinpointing the weaknesses of the previously described system. The first one comes from the evolution of the user interfaces and the second one is related to a new flight management system. Through these two examples, the need for a new navigation kernel is identified.

3.1 User Interaction with the system

3.1.1. History of the GCS The user interaction with the system has reached different stages, depending on the context of use: telemetry only, basic telecontrol with the radio controller, and telecontrol with a full-fledged ground control station.

In the first period of its development, the system did not provide bidirectional datalink as the most affordable tradeoff was to downlink telemetry through the audio channel of the video link for display on the GCS. The only uplink was the radio control security link. Interaction was limited to toggling the autopilot on or off.

As the system has become operational (2004), we have needed to be able to tune control loops during the flight and to interact with the flight plan script. The commands were performed with the radio control, uplink commands were sent using spare channels, and feedback was downlinked to the GCS.

This method was very error prone, because the outdoor pilot would need to maintain an accurate representation of the aircraft settings state and blindly activate the switch and potentiometer. The GCS operator gave him vocal feedback. By that time, it became clear that the lack of digital uplink was a major obstacle for further development.

3.1.2. Current GCS A digital datalink was added between the ground station and the vehicle (end 2005). The generic mechanism to directly adjust the value of a parameter in the autopilot was provided through dialog boxes on the GCS display. This mechanism was reused to implement more complex actions on navigation scripts and to allow random access to flight plan blocks. Richer interaction with waypoints was provided through direct manipulation by dragging their icons on the map. At this stage, the outdoor pilot role had been reduced to security only, and numerous flights have been performed with the RC transmitter switched off. The current GCS can be used for the flight preparation, the flight control, the settings, post-flight missions log analysis, and allows for simultaneous control of multiple MAV.

The Paparazzi system was mostly used and designed with MAV flight competitions in mind. The goals for the early flight missions were simple, with scoring rules fostering fully autonomous operations relying on waypoint navigation. The difficulty level of the flight missions has increased over the years, focusing on accuracy (sensor dropping, arch crossing), and requires both off line and on line updates of the flight plan.

3.1.3. Towards a new HMI Paparazzi is not only used for MAV competitions, but now also used for real life applications, most of them consisting of video observation. Some other applications also involve the use of more sophisticated sensors, like chemo sensors and more sophisticated payload behavior, like "intelligent" searching. These applications require new interactions between the operator and the system. Moreover, the first users of the system were its designers themselves. New users can be confused with the legacy GCS, as their knowledge of the system will vary. To address this variety of needs we have focussed on the flight control activity, assuming that the rest (flight preparation, settings, debugging) can be done by experts on the legacy GCS. We also decided to limit the new HMI to the piloting of a single UAV. As the context of use is field operation, the GCS is built to run on a long battery life rugged Tablet PC. The interaction is done with a stylus on a 12.1" TFT resistive touchscreen.

The initial requirements stem in the usability of the interactive system, as (J.M.C and Scapin, 1992). The following generic criteria has been used at the design and evaluation stages:

- Adaptability: the system should reuse users' experience (desktop system, computer video games), and provide multiple methods of achieving an action;
- Guidance: legibility of the state of the system, using visual cues, animations and graphical design to ease the learning process;
- Explicit control: the system should be transparent and provide a way to cancel an action when possible;
- Protection against errors: a sound alarm system, a way to ensure compatibility between flight path patterns with a trailing action when needed (e.g. a holding pattern is added at the end of a path trajectory if there is no other pattern);
- Workload reduction with reduced information density and minimal actions.

3.1.3.1. Design process and first results The team designed the user interface independently from the current functional core, thus allowing us to assess the compatibility of the functional core adapter and overall architecture. We have used an iterative user centered design process (ISO13407, 1999), with the following steps:

- Observation of the users' activity and interviews, leading to a simple model of the tasks and a set of work scenarios;
- Participatory design with brainstormings leading to design scenarios and paper prototypes;
- Usability tests against ergonomics criteria on paper prototypes and low fidelity mock ups;
- Expert evaluations;
- Design and implementation of the chosen options.

At this stage, the new GCS layout is a three panes zoomable user interface using gesture recognition as in (Mertz and Vinot, 1999). The software is still in its early stages of development and is based on Java, the PICCOLO[4] and SATIN[5] toolkits. As proposed by (Beaudouin-Lafon, 2000), key concepts have been reified and are first class citizens of the user interface: patterns can be selected in a toolglass, their control points can be modified at any time, the sequence of selected patterns is visible in the mission timeline. The control points are available on the mission timeline, on the map and on a vertical view.

3.1.3.2. HMI requirements The HMI must not be hindered by existing functional core limitations. We have identified a set of limitations with the current architecture: there is finite set of waypoints, fixed parameter values for stages (e.g. circle radius), no notification system from the airborne system except for a periodic global flight plan parameters broadcast, etc. This can be tackled at the functional core adapter level, providing error correction, notifications, and translating user level flight plans to system level stages. However, keeping in mind that a flight can be transferred from one GCS to another, the actual flight plan parameters must get airborne. Moreover, many values are computed on board the aircraft: wind conditions, drop trigger position, etc. While it's possible to compute them in the GCS as well, it would be beneficial for the sake of safety to access this data and perform error correction and feedback.

3.2 FMS

A new application of the Paparazzi system appeared in 2006 through a project focusing on *intelligent surveillance* for forest fire monitoring, detection and localization using one or several MAVs[6]. In this application, the aircraft are equipped with chemo sensors.

In this context, we have proposed several search strategies directly inspired from biological behaviors(Balkovsky and Shraiman, 2002; Adler and Tso, 1974). The challenge is now to implement these strategies on top of the Paparazzi autopilot.

[4] `www.cs.umd.edu/hcil/jazz/`
[5] `guir.berkeley.edu/projects/satin`
[6] US army contract R&D 1042-AM-01.

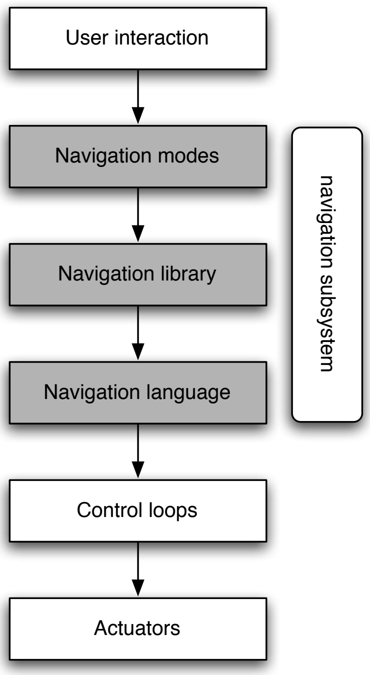

Fig. 6. The navigation subsystem is composed of three layers, the link between the interactive command of the system and the low level control loops managing the actuators.

Note that is a good idea to implement these strategies *outside* the autopilot for multiple reasons: limited complexity of the critical code, possibility to offload CPU intensive processes, higher modularity, etc.

To do so, a new component, named *FMS* for Flight Management System, has been considered to be added to the airborne Paparazzi system. This component is responsible for the planning of the mission and the execution of the *intelligent* behavior. It then requires access to the navigation component of the existing autopilot.

The strategies are described with high level directives like "go in this direction", "go to this point", "circle around this point" and generally do not require access to the basic control loops of the vehicle.

3.3 Autopilot Evolution

The two previous examples, rich human interaction with the navigation and FMS integration, expose common requirements, i.e. the needs for an interface level. This is not currently provided by the autopilot where the only available control, the scripting navigation language, is much too static.

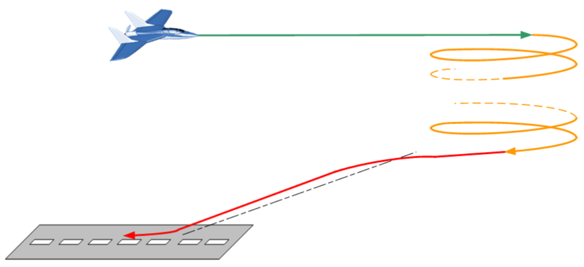

Fig. 7. The landing circuit pattern is composed of three steps which are join, descend and glide.

4. AUTOPILOT ARCHITECTURE

In the light of the previously mentioned needs, the navigation subsystem was completely rearchitectured and is now in the process of being implemented. A three layers architecture (figure 6) was formalized in order to keep the flexible and complete flight plan language that already exists while allowing the construction of a navigation library providing high level "patterns" that can be used interactively.

4.1 Navigation Language Layer

This layer implements the basic trajectories following behaviors as well as the basic flow control (sequence, jump and conditional). It resides above the stabilization subsystem. The trajectories' primitives are organized into three classes:

horizontal: roll_dot, roll, course, go, route and circle.
vertical: pitch_dot, pitch, climb, alt and glide.
speed: airspeed, ground and speed.

This layer can be adapted to the actual capabilities of the vehicle (e.g. VTOL).

4.2 Navigation Library Layer

It is used by the actual navigation algorithm found in the upper mode layer. It adds a notion of completion to the basic trajectories listed in the language layer. It also takes advantage of the flow control features available in the navigation language layer to provide complex patterns which can mix control among the three above mentioned classes (horizontal, vertical and speed).

For example, this layer includes a landing procedure pattern (figure 7) defined as a sequence of three basic trajectories:

- A go to an approach point;
- An helix made of a circle mixed with a climb until the start of descent altitude is reached;
- A glide to reach the touch point.

4.3 Navigation Mode layer

The navigation mode layer is also an addition of the new architecture. During the flight, the autopilot can be switched between different "modes". The "modes" define additional features needed to perform a particular type of navigation. This is the place where user interactions are dealt with.

Auto1 is the stability augmented mode. The operator directly steers the vehicle either under attitude or rate control. The interactions can be performed through the radio control security link or through the datalink. This mode can be seen as an advanced "rescue" mode or used to off-load the guidance algorithm.

Auto2 is the fully autonomous mode. In this mode, the aircraft executes the pre-compiled flight plan script. This mode is well suited for the description of complex behaviors including computation and decision making. Nevertheless, the complexity of the language makes this mode poorly suited for graphical real time operator interaction.

Auto3 is the new high level interactive mode. The available "language" is kept minimal to allow graphical representation and edition, including only the notion of patterns sequences. On the contrary, the interactions are rich and they enable the addition and configuration of all supported navigation library patterns. Those interactions are performed through the datalink from the graphical user interface.

Auto4 is mostly the same as Auto3 except that the interactions are performed by a payload task. This mode is used to let an external application take control of the navigation.

Home is a basic robust navigation mode that is entered upon failure of another mode. It brings back the vehicle to its starting location using conservative maneuvers. It can be triggered if the aircraft leaves its allowed area, in case of communication loss in an interactive-only mode or by any other defined failure condition.

This mode layer is the placeholder for future enhancements.

5. CONCLUSION

We have described a very pragmatic bottom-up approach. The first UAV was airborne and functional within a few months after the beginning of the project in 2003. As the mission requirements have evolved, we have taken advantage of the open architecture, added and updated subsystems, and thus proved the validity of the overall design.

The re-engineering proposed in this article is at its early stages and addresses the current needs. We claim that the new architecture is more flexible, has a greater potential, without cutting loose previously available functionalities. We believe it satisfies requirements both driven by user interaction and advanced autonomous behavior.

REFERENCES

Adler, Julius and Wung-Wai Tso (1974). Decision-making in bacteria: Chemotactic response of escherichia coli to conflicting stimuli. *Science*.

Albert, Nicolas (2005). Certification du code embarqué d'un micro-drone. Master's thesis. University of Toulouse.

Balkovsky, Eugene and Boris I. Shraiman (2002). Olfactory search at high reynolds number. *Proceedings of the National Academy of Sciences* **99**(20), 12589–12593.

Beaudouin-Lafon, Michel (2000). Instrumental interaction: An interaction model for designing post-wimp user interfaces. In: *Proceedings of the ACM CHI 2000 Human Factors in Computing Systems Conference*. ACM Press. pp. 446–453.

Brisset, P., A. Drouin, M. Gorraz, P.-S. Huard and J. Tyler (2006). The Paparazzi solution. In: *MAV2006*. Sandestin, Florida.

Brisset, P. and A. Drouin (2004). PaparaDzIY: do-it-yourself UAV. In: *Journées Micro Drones*. Toulouse, France.

Buisson, M., A. Bustico, S. Chatty, F-R. Colin, Y. Jestin, S. Maury, Ch. Mertz and Philippe Truillet (2002). Ivy : Un bus logiciel au service du développement de prototypes de systèmes interactifs . In: *IHM 2002, Poitiers*. ACM Press. http://www.acm.org/. pp. 223–226.

ISO13407 (1999). Human-centered design processes for interactive systems. Technical report.

J.M.C, Bastien and D. Scapin (1992). A validation of ergonomic criteria for the evaluation of human-computer interfaces. In: *International Journal of Human-Computer Interaction*. Vol. 4. pp. 183–196.

Mertz, C. and J. Vinot (1999). Touch input screens and animations: more efficient and humanized computer interactions for ATC.