ACS - Autopilot

This document briefly describes the autopilot chosen for the Aerial Combat Swarm project and details the modifications made to it. We briefly mention the hardware behind the autopilot, but focus on software modifications. Detailed documentation on the systems described here is available at:

http://dev.ardupilot.com

http://plane.ardupilot.com/

Note that all documentation on those sites assumes the use of the Mission Planner ground control station (GCS). We use an alternate GCS called MAVProxy:


as well as custom GCS software for swarming which is documented elsewhere as part of this ACS documentation repository.

Pixhawk Autopilot using ArduPlane software "stack"

The autopilot we are using is the Pixhawk as provided by 3DRobotics in 2015. An overview of the wiring of this autopilot is available here:


The firmware running on the autopilot is a combination of PX4Firmware, PX4NuttX, and ArduPlane. They each exist in these github repositories:

https://github.com/diydrones/PX4Firmware

https://github.com/diydrones/PX4NuttX

https://github.com/diydrones/ardupilot

The ardupilot repository has configured PX4Firmware and PX4NuttX as git submodules. A discussion of that architecture is here:

http://dev.ardupilot.com/wiki/git-submodules/
ArduPlane exists as a part of the ardupilot repository. That repository also contains two other autonomous systems: ArduCopter and APMrover. Each exists in separate libraries labeled as such in the root of the ardupilot repository. Also in that repository are a set of libraries (in the libraries directory) that are shared between all 3 systems. The arudiplot repository also includes utility scripts (Tools folder) and a build system (mk folder).

We have only made modifications to the ArduPlane software and will limit the discussion to it and some of the libraries we modified.

**AP_ACS library.**

We created a library inside the ardupilot system called AP_ACS. It encapsulates most of our modifications.

**AP_ACS Features**

**Failsafe modifications**

Default failsafe behaviors for ArduPlane are documented here:


We do not employ the throttle failsafe because swarming scenarios require that planes not be connected to a manually radio Tx at all times. The failsafe is easily disabled via parameter, so it was not necessary to modify autopilot firmware to achieve the desired behavior.

Other failsafes did require some modification or additional behavior.

- We introduced an autoland failsafe. We employ it for low battery or if the communication link is lost for more than 2 minutes (the communication link failsafe is called the "GCS failsafe" in ardupilot documentation).
- The GPS failsafe was modified. The ArduPlane default behavior when GPS is lost is to attempt to dead reckon back to a rally point. We have experienced situations where the INS was not able to dead reckon accurately after a loss of GPS. Therefore, to maintain containment inside the airspace, when GPS is lost for more than 5 seconds the plane goes to LOITER mode. If GPS does not return after 20 seconds then the throttle is cut. If GPS should return, the plane reverts to previous mode.
- We use the geofence as described in the above link, but a secondary geofence failsafe was introduced as well. If the plane does not reach a point inside the geofence within 20 seconds of a fence breach we assume something is wrong with the control surfaces and the plane will not be able to return so the throttle is cut. If the plane does happen to later reach a point inside the fence throttle is restored and all breaches cancelled.
- Motor failsafe. If > 60% throttle and < 2 amps for more than 5 seconds this sends the plane to a Rally point. Note that this failsafe will not fire if the batteries are low as we expect the plane to autoland.
- Companion computer failsafe added. Loss of communications with the companion computer for more than 20 seconds results in the plane flying to the rally point.
- A parameter was added, ACS_WATCH_HB, that controls whether to listen for the companion
computer heartbeat. If set to 0, the companion computer failsafe is disabled.

- A parameter was added, ACS_KILL_THR, that allows killing throttle from GCS in an emergency.
- All failsafes are disabled in MANUAL, FBWA (fly by wire A), and FBWB modes. When a pilot has manual control of the aircraft we desire no interference from the autopilot.
- All failsafes except fence and GPS disabled when on approach for landing. All failsafes disabled when on final landing approach.

**Modifications neccessary for swarming**

- A custom MavLink message, GLOBAL_POS_ATT_NED is sent out periodically to the companion computer (also required small change to GCS_Mavlink library).
- A custom MAVLink message, MAV_CMD_OVERRIDE_GOTO, that allows the companion comptuer to modify the current waypoint. In this way, the companion computer has control over the navigation outer loop, while the autopilot maintains control of the inner loop. Note that this was not implemented inside the AP_ACS library -- see the next section.

**Modifications outside the AP_ACS library**

It was necessary to make some modifications outside of the AP_ACS library to fully implement the above features.

- (ArduPlane: ArduPlane.cpp, GCS_Mavlink.cpp, commands_logic.cpp, events.cpp, system.cpp) Support for custom failsafes and autolanding allowed by AP_ACS library.
- HAL_SITL library: modification to keep simulator clock in lockstep with wall clock in order to stay synchronized with the simulator clocks (implemented in ROS) aboard the companion computer.
- (ArduPlane: events.cpp, geofence.cpp) All failsafes disabled in MANUAL, FBWA, and FBWB modes.
- (ArduPlane: GCS_Mavlink.cpp) Handler added for MAV_CMD_OVERRIDE_GOTO mavlink message that we use to allow the payload to change the lat/lon/alt of the current waypoint.
- GCS_MAVLink library: increased log download speeds over a serial link. We never download logs over the radio and we need log downloads to the payload to take full advantage of the serial line.
- AP_Arming: ignoring some pre-arm checks on accelerometers as they were resulting in too many false positives.
- Dataflash: log the GLOBAL_POS_ATT_NED message occurrences to the Pixhawk log file (on a micro SD card).

**900 MHz radio and atcommander.py**

Pixhawk uses a 900 MHz radio for telemetry and communication between the single-plane GCS:


The firmware that runs on this radio is hosted on github:

[https://github.com/Dronecode/SiK](https://github.com/Dronecode/SiK)

We have added a small utility, atcommander, to this repository in a forked version of the SiK firmware made available at the ACS gitlab site. Documentation of this utility is elsewhere in the acs-docs repository.
Pixhawk background

The hardware for the Pixhawk was primarily developed at ETH Zurich. The first generation device was named "PX4." ETH Zurich and many other groups operate a different software stack on that hardware (the "PX4 stack" vice the "ArduPlane stack") though they are still based on the PX4Firmware and PX4NuttX repositories. Their web site is here:

https://www.pixhawk.ethz.ch/