Brigham Young University

AUVSI Capstone Team (Team 45)

Autopilot Testing Procedures and Results

ID	Rev.	Date	Description	Author	Checked By
CT-002	0.1	02-12-2019	Initial conception	Andrew Torgesen	[CHECKED BY]

1 Introduction

The autopilot for the UAS is called **ROSPlane**. It is responsible for taking high-level flight path commands and translating them to low-level actuator commands (aileron and elevator servos and throttle) on the airframe. The autopilot we are using and its software architecture are documented in our team Github repository.

An intermediate step for the UAS to achieve its key success measures is to ensure that the underlying autopilot is well-tuned. The phrase well-tuned refers to the fact that the autopilot consists of a series of PID loops to control the longitudinal and lateral autopilots. Each PID loop has associated gains which must be tuned in-flight to ensure optimal performance. The following are block diagrams of ROSPlane's inner and outer loops for the longitudinal and lateral autopilots, respectively:

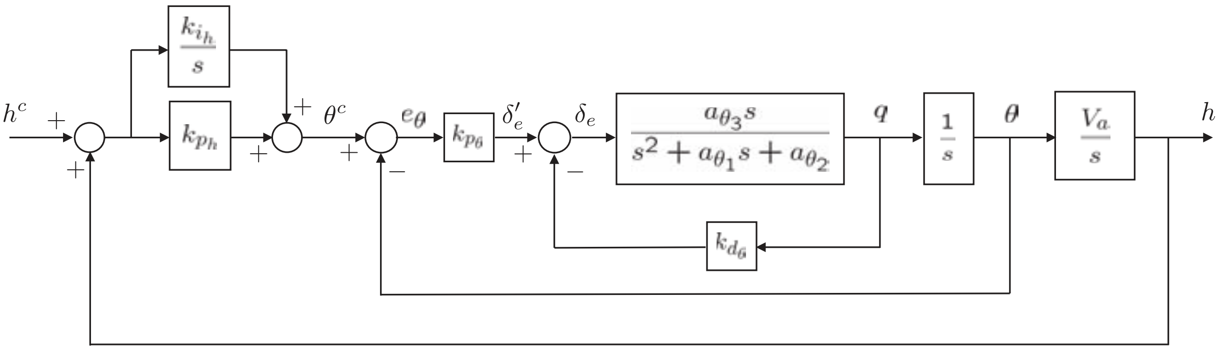

Figure 1: Longitudinal successive loop closure autopilot for the UAS. Borrowed from Small Unmanned Aircraft - Theory and Practice by Beard, McLain.

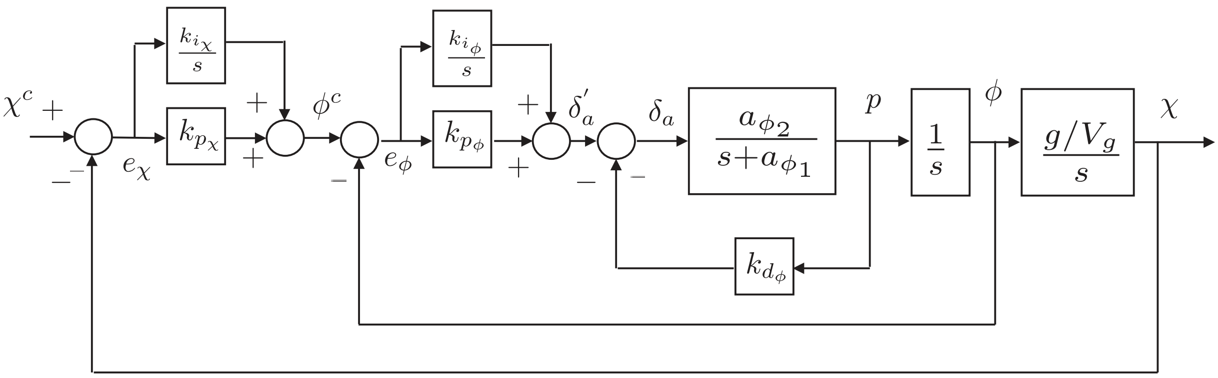

Figure 2: Lateral successive loop closure autopilot for the UAS. Borrowed from Small Unmanned Aircraft - Theory and Practice by Beard, McLain.

This artifact details our testing and evaluation procedures to ensure that autopilot performance and compatibility with the rest of the UAS.

2 Performance Testing

Performance testing of ROSPlane is divided into two parts: control algorithm testing and estimation algorithm testing. The control algorithms constitute the functionality of the autopilot, and thus must be well-tuned to ensure that the key success measures can be met. Moreover, the estimation algorithms (currently being run on the Inertial Sense hardware–see the *UAS Subsystem Interface Definition* artifact (SS-001)) provide vital information about the current dynamic state of the UAS to the autopilot, and thus must also be validated to ensure stable unmanned flight.

2.1 Control Algorithms

Control algorithm testing consists of determining how well the autopilot is able to follow commanded flight path states, such as pitch angle, altitude, roll angle, and course angle. Good performance entails timely convergence to the commanded values without subsequent oscillation or instability. To ensure good performance according to these criteria, we followed the procedures outlined in the *Autopilot Tuning* artifact (CT-001) over the course of three different flight tests. Below are plots from real flight test data demonstrating the ability of the autopilot to converge on the commanded flight path states of altitude (h) and course angle (χ):

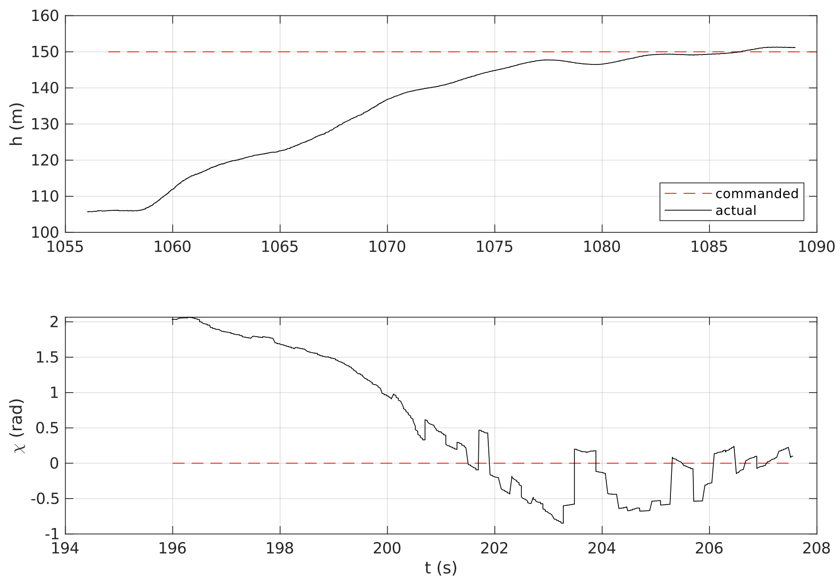

Figure 3: Flight test data demonstrating the ability of the autopilot to converge on commanded values for altitude and course angle.

As can be seen from Figure 3, the longitudinal controller converges smoothly, whereas the lateral controller appears to struggle a bit to converge. This is due to the fact that there was a bit of wind blowing from West to East the day of our flight testing. Nevertheless, the lateral controller has demonstrated the ability to overcome external wind disturbances and arrive at the commanded course angle value, which was due North in this case.

One final thing to note from Figure 3 is the chopiness of the measured course angle data. This choppiness is attributed to state estimator error from our onboard sensors, which is addressed in the following section.

2.2 Estimation Algorithms

Through flight testing and subsequent analysis of the estimated state data, we have determined that using the Inertial Sense sensor module for state estimation alone, while basically adequate for unmanned flight, is too subject to small failures which propagate into large problems for the UAS as a whole. The following are the two principal issues with the Inertial Sense Estimation that we have observed:

2.2.1 Heading Estimation

As can be seen from Figure 3, the course angle data from the Inertial Sense can be choppy. The Inertial Sense is known to have issues estimating yaw if it isn't moving, but still apparently sometimes suffers from large amount of noise while moving. This failure to produce a smooth course angle measurement could conceivably cause issues for the control algorithms, and thus must be amended.

2.2.2 Altitude Estimation

Figure 4 demonstrates another weakness with the Inertial Sense that occasionally arises. After $t \approx 75s$, the altitude estimate begins to go negative, and fails to recover. This behavior is attributable to the fact that the sensor is relying on GPS and inertial data only to estimate altitude. This failure to estimate altitude accurately is catastrophic when it arises, as the autopilot has no accurate idea of where it really is, leading to undesirable and unpredictable behavior.

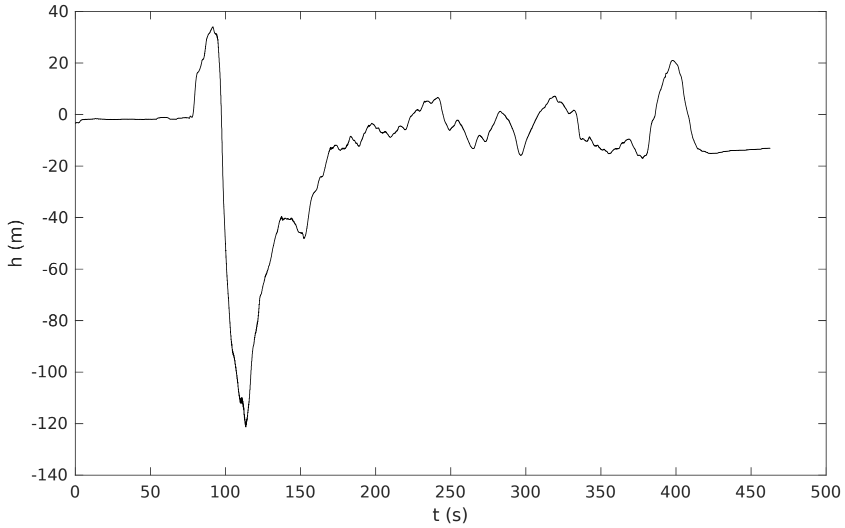

Figure 4: Altitude estimation output from the Inertial Sense sensor during a test flight.

2.2.3 Plans for Addressing Issues

To avoid the failure modes in course angle and altitude estimation discussed above, our plan is to leverage the sensor fusion capabilities of the ROSFlight board and ROSPlane to add sensor redundancy. ROSFlight and ROSPlane together constitute an estimation scheme that is specially designed for fixed-wing platforms, and they integrate sensors that the Inertial Sense does not, such as an airspeed sensor and barometer. It is anticipated that this added redundancy will greatly reduce the risk of state estimation failure due to experiences in previous years. That being said, extensive testing of the redundant system will still be carried out.

3 System Compatibility Testing

The *UAS Subsystem Interface Definition* artifact (SS-001) details that there are four interfaces between ROSPlane and other components in the UAS:

1. Flight Controller/Inertial Sense Interface (*two-way*)
2. Image Stamper Interface (*two-way*)
3. Ubiquiti Bullet Interface (*two-way*)
4. Payload Delivery Interface (*one-way*)

Table 1 communicates our testing procedures and results for each of these interfaces.

Table 1: Description of testing procedures and results for ROSPlane interfaces.

Interface	Testing Procedure	Testing Results
Flight Controller/Inertial Sense Interface	This interface runs over a USB cable using the MAVLink protocol. The MAVLink protocol itself is a tried-and-true protocol for serial communication between devices. Our particular MAVLink connection has been repeatedly tested on each flight test, whose procedure is outlined in the *Field Flight Checklist* artifact (PF-001). A working flight controller interface is required for both RC and unmanned flight.	The *Flight Test Log* artifact (AF-004) details that over the course of 13 flight tests, the flight controller interface has never posed a problem. That being said, the *Estimation Algorithms* section of this artifact details the weaknesses of the data being passed to the autopilot from the Inertial Sense hardware.
Image Stamper Interface	This interface runs over ROS, following the publisher-subscriber architecture. Testing this architecture has entailed running the ROSPlane and image stamper nodes, having each node publish messages to each other, and ensuring a constant message reception frequency in each node.	ROS network testing has shown that publisher-subscriber connectivity has never posed an issue, particularly because both nodes are running on the Odroid computer and thus not dependent on a reliable WiFi connection.
Ubiquiti Bullet Interface	Testing of this interface entails ensuring relatively constant connectivity over WiFi from the ground to the plane during flight testing. It is useful to use the *ping* command to check connection speed throughout the flight.	Over the course of several flight tests, only once has WiFi connectivity posed an issue. This was most likely attributable to failing to keep the Ubiquiti Rocket pointed at the plane constantly during flight.

Payload Delivery Interface	See the *(++++++)* artifact (...) for testing details.	A summary of the results is that the interface is reliable; the only problems with the payload delivery system are due to the behavior of the Odroid's operating system itself when ROS is not running.

4 Conclusion

Extensive testing of the autopilot subsystem, primarily in the form of flight tests and autopilot tuning, has determined that the autopilot performs up to standards and is capable of interfacing with the needed components in the UAS. Further work is needed to improve the robustness of the onboard state estimation to greatly reduce the risk of in-flight failure, since ROSPlane relies on accurate state estimation to sustain unmanned flight.