Predictive Feedback Augmentation for Manual Control of a UAV with Latency

J. Cox*, KC Wong
The University of Sydney, Sydney, Australia

ABSTRACT

Teleoperation of unmanned aerial vehicles is hampered by communication delay, which causes feedback from command inputs to take considerable time to be displayed to the operator. For an international internet connection, round trip latencies can reach 500ms. The satellite connections used for military UAVs can have latencies on the order of seconds. This delay presents a substantial control problem which has been solved in the past by control abstraction (instead of "roll left" the aircraft might be instructed "go to these coordinates"). Manual control remains difficult. This study borrows the client-side prediction concept from multiplayer video games to attempt to address the control delay to allow manual control. An estimate of the change in the vehicle state due to the commands that are yet to affect the feedback is computed and then the feedback that the pilot receives is modified to reflect this predicted change. Because of this change, the pilot can see immediately the effect of the control inputs. This study has explored the concept and built a prototype system functional in real time for flight testing.

1 INTRODUCTION

Unmanned aerial vehicles (UAVs) are seeing widespread growth in many existing and new applications. Presently, UAVs are often operated within Line of Sight (LoS), where the control delays come primarily from the control radio system in use. Entry level hobby transmitters commonly achieve latencies on the order of 30ms. Advanced hobby radio systems boast ranges of 60km.

When operating via a local video link, camera latencies range from 40ms for the fastest analog systems to hundreds of milliseconds for digital systems.

Recently, hobby grade low latency digital transmission has been introduced, with latencies below 50ms achievable. Latency of wifi based transmission systems can exceed 100ms. Adoption has been low, likely due to cost and fragility.

*Email address: jeremy.cox@sydney.edu.au

Some military UAVs are operated from thousands of kilometres away via encrypted satellite connection, which imposes a round trip delay on the order of seconds. Often the terminal flight phases are controlled from ground crews near the runway. Due to the short time scales on which the state of the vehicle can change during takeoff and landing, the latency presents difficulty in control. Outside of terminal flight phases, the latency is acceptable as manouevres and disturbances occur on timescales much larger than the latency.

1.1 Internet Latency

The latency of packets sent over the internet varies depending on the source and destination. One way latency from Australia to the US is measured at around 150ms. Two way communications between an Australian and US location will double this to find the round trip time, as well as add latency from networking within each continent.

Control of a UAV over internet has been explored previously by the aerial international racing of unamnned systems (AIRUS) student project, of which the first author was a part. The impact of the latency on the control was found to be very significant, with even small latencies rendering a quadrotor UAV barely controllable. Previous work by team members has discussed the concept of a predictive aid and shown improvement in human control of a simulated system [1].

1.2 Teleoperation with Latency

In 1967, Ferrell and Sheridan [2] detailed for the first time the problem of teleoperation. Three main components of a remote operation system were described: a remote loop which acts to process tasks remotely (increasing the abstraction of the control); a supervisory loop which consists of the operator receiving information about the remote device's environment and specifies new commands; and a local loop which represents the operator's computer locally assisting (such as by modelling the remote system to present quasi-feedback). Increasing the command abstraction by allowing the remote loop to process the perceived environment and determine subtasks to achieve a higher level goal forms the majority of the systems proposed by Ferrell and Sheridan.

The operation of a robotic arm with delayed control is investigated by Bohren [3], who presents an assisted teleoperation architecture. A virtual robotic arm with a control delay is used to perform part of an assembly task. The visualisation of the scene is enhanced by displaying the user command overlayed on the robotic arm, as well as by estimating user intent. The user intent is estimated from the user inputs and the dis-

played scene that they are in response to, and then the user input is modified to try to effect the user intent in the real scene. With the assistance, test subjects showed substantial improvement in task completion time.

1.3 Latency in Multiplayer Videogames

Online videogames have previously undergone development similar to the proposed system. Bernier [4] details the progression of architectures, from a single server handling a number of clients that are entirely ignorant of the game mechanics to more modern configurations using client-side prediction and other more advanced routines. A number of the procedures detailed by Bernier are associated with foiling untrustworthy clients (cheaters) and it is noted that for military simulators, the clients are trusted.

The most basic feature discussed by Bernier is client-side prediction. In a dumb-client system with a network latency, the inputs from the client are sent to the server, which interprets how they change the game state, which is then sent back to the client. Movement of a player through an environment is given as a notable example where client-side prediction can be used. Instead of waiting for the server to acknowledge a movement command and update the game state, the client assumes that the command will be accepted by the server and processes the expected game state change by itself. The canonical version of the game state is handled by the server, so the client prediction may need to be retroactively altered. This may occur for example when the player input has an unexpected result due to another player's input that has not had time to be transmitted to the client or due to game state information that is hidden from the client. With client side prediction, the player perceives that their movement commands are being processed instantly, rather than with the round trip latency, and so their experience appears consistent regardless of latency in their connection.

Bernier claims that the success and longevity of a videogame requires a seamless multiplayer experience.

1.4 Flight Controller Modes

Quadrotor platforms are, in all practical cases, controlled via a digital control system. In these control systems, the operator commands particular flight variables, rather than controlling the motors and other actuators directly. The actuators used are the four (other numbers can be used but for simplicity only quadcopters are discussed) propellers, with the left/right, forward/back and left/right diagonal pairs each being used to generate roll, pitch and yaw moments respectively.

A review of common control schemes for quadrotors is presented in Appendix A:. Presently, the only widespread method of combating latency in UAV control is abstraction of the pilot's controls (move to a GPS coordinate instruction replacing roll/pitch/yaw commands). With UAV technologies expecting to see substantial growth in both civilian and military sectors, the controllability of UAVs at long range (and hence with control latency) is an important area of development. Current systems are either short range and low latency or long range and high latency. The tried and tested client side prediction concept from multiplayer videogames is a strong candidate for expanding the operability of UAVs at long range.

2 Proposed System Overview

To combat latency, the client-side prediction concept is applied to a UAV. The adaptation to a UAV can be summarised as predicting and displaying the vehicle state at the time when the commands being given currently will arrive at the vehicle. The eventual goal is to implement the scheme on a micro (112mm size) quadrotor flown in rate mode using a video feed from onboard as feedback. The operator receives information about the vehicle's state from the video feed, including the angular position from the image of the horizon.

Figure 1 shows the flow of commands and feedbacks around a time t_n at the remote station. 2τ is the roundtrip latency (the one-way latencies have been assumed to be identical and equal to τ).

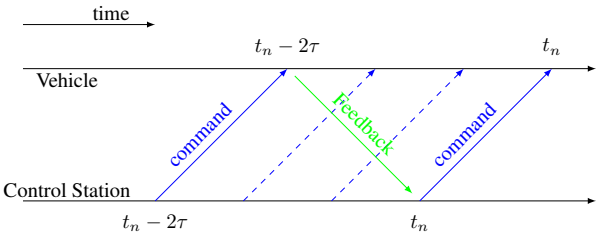

Figure 1: Flow of command and feedback.

The origin of the time axis is chosen slightly differently between the vehicle and the control station. $t = 0$ when the arm command is sent or received. This is convenient because a command occurs at the same time coordinate whether it is being sent or received.

At the control station at t_n, the remote station has the feedback that is delayed by τ, but it can also see the commands that have already been sent. The commands between $t_n - 2\tau$ and t_n will arrive at the quad after the feedback at $t_n - 2\tau$, and prior to the command being given at t_n and arriving at t_n. This command information is then used to predict the change in vehicle state, and hence the feedback that the operator would expect to see if there was no delay.

The mathematics of the problem can be described more simply as:

> Given the system state and command inputs to be executed, predict the system state after those command inputs have been applied.

which is a familiar problem from control theory with well developed means of solution.

Because the system model is likely to not exactly match the real system, and the delayed feedback is available, it is

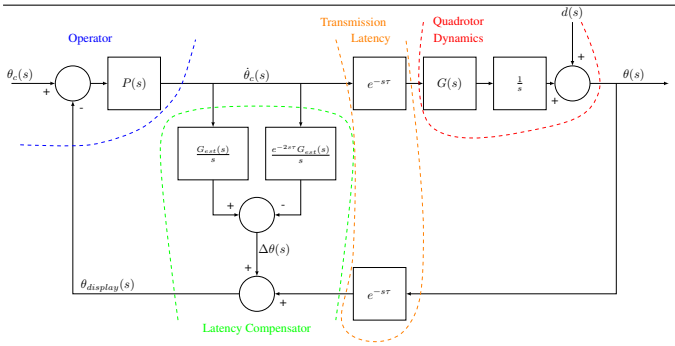

Figure 2: Block diagram with delay and compensator.

proposed to use the difference between two simulated vehicle states that are 2τ apart. One estimation is the state using all commands until time t_n and another only simulating until $t_n - 2\tau$. Taking the difference between these two states and adding the received feedback should give a more accurate estimate of the vehicle position than dead-reckoning simulation from start time to t_n. Errors in the model that accumulate prior to $t_n - 2\tau$ are contained in both terms, so are subtracted out.

2.1 System Block Diagram

Figure 2 shows the suggested system. The operator is modeled as determining a desired attitude, comparing that attitude to the feedback they are shown and then applying a controller to determine a command input. The compensator compares two simulated flight states, one of which is delayed by the round trip delay time and then augments the feedback by this amount. The remainder of the system is formed by the actual transmission delay and vehicle dynamics.

Because the implementation later in this paper deals with an angular rate controlled quadrotor, while the attitude feedback is an angular position from a video stream, some integrator blocks are included. These terms are shown for completeness and could be modified if a different control arrangement was used.

The diagram shown is for the one-dimensional case, where θ is any of the roll pitch or yaw axes. A real implementation will require propagation of these axes in three dimensions.

The compensator can be removed by setting $G_{est} = 0$, which gives the system with only the delay. The delayed system can be reduced to the typical system by setting $\tau = 0$, effectively removing the delay. Using the fact that the compensator and delay can be removed by substituting terms, the closed loop transfer functions need only be found for the delay and compensator case and the others will be readily found from those results.

2.1.1 Output and Reported Output Response to Command Derivation

First the closed loop transfer functions are found for the block diagram with the compensation system. For convenience, define

$$\begin{aligned} A(s) &\equiv P(s)\left[G_{est}(s) + e^{-2s\tau}(G(s) - G_{est}(s))\right] && (1)\\ &\approx P(s)G(s), && (2)\end{aligned}$$

which approximately is equal to the latency free open loop transfer function for the pilot gain-quadrotor system if $G_{est}(s) \approx G(s)$. The displayed feedback and actual response can then be found:

$$\frac{\theta_{display}(s)}{\theta_c(s)} = \frac{A(s)}{s + A(s)}, \quad (3)$$

$$\frac{\theta(s)}{\theta_c(s)} = \frac{e^{-s\tau}P(s)G(s)}{s + A(s)}. \quad (4)$$

The delay term in the numerator is mandatory, as the input cannot reach the output without being subject to transmission delay (and a negative delay is infeasible without predicting $c(s)$ a short time in advance).

The disturbance response can also be found:

$$\frac{\theta(s)}{d(s)} = 1 - \frac{e^{-2s\tau}P(s)G(s)}{s + A(s)}, \quad (5)$$

which makes intuitive sense as the response is the disturbance plus a delayed, negative response to the disturbance. Once again the delay in the numerator is unavoidable as the disturbance must travel once around the loop to pass through the pilot and correct the disturbance.

2.1.2 Interpretation

The transfer functions for the compensated system can be used to find the delayed system and undelayed system for comparison. The control response transfer functions are shown in Table 1 and the disturbance response transfer functions in Table 2. With $A(s)$ highlighted in green and delay terms highlighted in red.

	$\frac{\theta(s)}{\theta_c(s)}$
No Delay	$\frac{P(s)G(s)}{s + P(s)G(s)}$
Delay	$\frac{e^{-s\tau}P(s)G(s)}{s + e^{-2s\tau}P(s)G(s)}$
Delay & Compensator	$\frac{e^{-s\tau}P(s)G(s)}{s + A(s)}$

Table 1: Transfer functions for the control response.

	$\frac{\theta(s)}{d(s)}$
No Delay	$1 - \frac{P(s)G(s)}{s + P(s)G(s)}$
Delay	$1 - \frac{e^{-2s\tau}P(s)G(s)}{s + e^{-2s\tau}P(s)G(s)}$
Delay & Compensator	$1 - \frac{e^{-2s\tau}P(s)G(s)}{s + A(s)}$

Table 2: Transfer functions for the disturbance response.

The introduction of the delay causes a delay term in the numerator of the command response and the counteraction to the disturbance.

The delay term in the numerator is caused by the forward latency for the command response and by disturbances needing to be observed and then responded to for the disturbance response. Addressing these delay terms is beyond the scope of this paper.

A delay term is also seen in the denominator. This term is caused by the operator being unable to immediately see the feedback from their inputs.

The introduction of the delay compensator modifies the term in the denominator. The term $A(s)$, highlighted in green, still contains the delay term as the feedback from the vehicle is incorporated. If a suitably accurate estimate of the vehicle dynamics $G_{est}(s)$ is known, then the term in the denominator becomes approximately equal to the undelayed denominator. The compensator acts to provide feedback that shows immediately the effect of the command inputs, removing the effect of the delay from the denominator.

The change that the compensator brings to the denominator makes the transfer functions much more similar to the no delay case. The command response is affected only by the forward latency, which delays the response but otherwise does not affect its behaviour. The disturbance response is improved by the compensator, but it will still take at least the roundtrip latency before the pilot begins to correct disturbances.

2.2 *Estimation Error*

If the vehicle attitude is estimated by integrating a model which provides the vehicle attitude rate, then

$$\theta_{est}(t) = \int_0^t \dot{\theta}_{est} dt \tag{6}$$

$$= \int_0^t \dot{\theta} + \dot{\theta}_{est.error} dt \tag{7}$$

$$= \theta(t) + \int_0^t + \dot{\theta}_{est.error} dt. \tag{8}$$

Where θ represents an angular position, $\dot{\theta}_{est}$ is an estimate of the angular rate obtained from a model of the dynamics applied to the recorded inputs and $\dot{\theta}_{est.error}$ is the error in $\dot{\theta}_{est}$. The issue with a dead reckoning approach can be seen here; the size of the error will tend to grow with flight time. If the displayed attitude is based on an out of date attitude plus a prediction of its upcoming changes, then

$$\theta_{display} = \theta_{est}(t_n) - \theta_{est}(t_n - 2\tau) + \theta(t_n - 2\tau) \tag{9}$$

$$= \theta(t_n) + \int_0^{t_n} + \dot{\theta}_{est.error} dt - \left[\theta(t_n - 2\tau) + \int_0^{t_n - 2\tau} + \dot{\theta}_{est.error} dt\right] + \theta(t_n - 2\tau) \tag{10}$$

$$= \theta(t_n) + \int_{t_n - 2\tau}^{t_n} \dot{\theta}_{est.error} dt \tag{11}$$

Only errors in the model that manifest between $t_n - 2\tau$ and t_n form part of the displayed system state. Importantly, the error does not accumulate with time and large errors (such as from disturbances or rapid motions where model errors are amplified) will only manifest for the delay time. Also, so long as the latency 2τ is small, the error in the displayed information is minimal. If a dead reckoning approach is used, then the error will increase by roughly a factor of $t_n/2\tau$, and will hence grow with time.

3 System Identification

3.1 *Introduction*

To predict the motion of the vehicle, a dynamic model is needed.

For the quadrotor vehicles considered in this paper, the dynamics of the three rotational axes were assumed to be independent of each other and the flight condition (speed, altitude, throttle setting). This assumption is valid only in rate mode.

The development of high rate flight data recording [5] allows system identification by measurement rather than modelling. Quadrotor models do exist, but model measurement was chosen due to the importance of accuracy to the specific hardware and software configuration. A large number of vehicle geometry and parameters would also be needed for a derived model approach, many of which are difficult to measure or have errors in the manufacturer's specification.

3.2 *Obtaining a Model*

Finding a model of each axis from the recorded input and output data was achieved using the inbuilt `tfest` function in MATLAB. This function accepts one or more logs of system input/output data and returns a transfer function that closely maps the input data to the output data. The function works by creating an initial guess and then iterating to minimise the error in the transfer function.

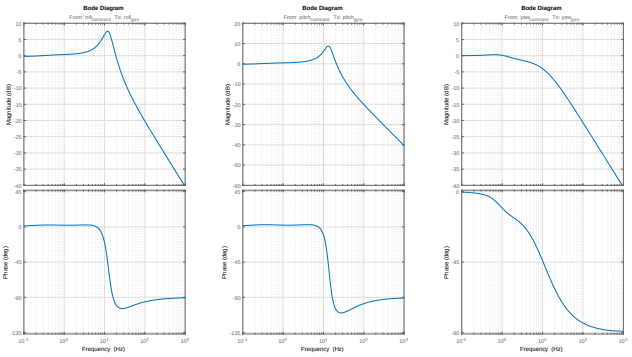

Figure 3: The models generated for the 112mm quadrotor.

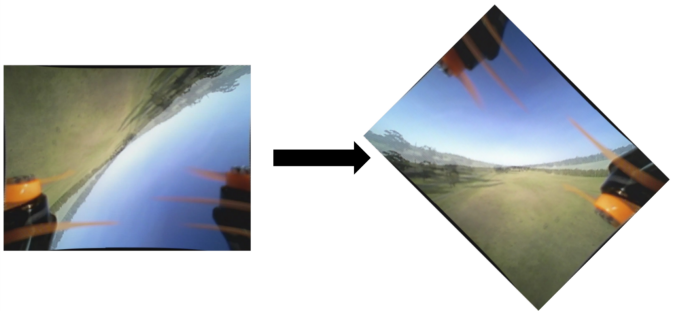

Figure 4: Example of image warping. The image is warped to reflect the compensator predicting a 135 degree roll to the right.

3.3 Identified System

For the 112mm quadrotor used, eight flight logs were used to generate a system model. The procedure was validated by generating eight models with a log each excluded and verifying that the models remained valid on the excluded log. Bode plots of the system model generated from all eight logs are shown in Fig. 3. The models map angular rate command to angular rate.

4 Augmenting the Feedback

With the change in vehicle attitude predicted, the feedback can be augmented to reflect this prediction. The video is first undistorted to remove fisheye effects from the camera lens, and then warped so that the horizon (and other objects far from the camera) appear to move immediately with operator input.

Figure 4 shows an example of image warping for a prediction of rolling. In this example the feedback shows the quadrotor in a near inverted attitude, but the command inputs that are yet to arrive will bring the quadrotor to nearly level, so the compensator warps the image such that the horizon appears level. Warping for the yaw and pitch axes is achieved by shifting the image left and right, with the camera field of view determining the rate (pixels per degree) of shifting.

5 Simulated Predictive Performance

Before implementing the compensator, it is desirable to estimate the predictive performance and to understand how the duration of the latency will affect the performance. The predicted position can be expressed in terms of the actual position and the error in the modeled system as

$$\theta_{display} = \theta(t_n) + \int_{t_n-2\tau}^{t_n} \dot{\theta}_{est.error} dt, \qquad (12)$$

from equation 11. For efficient calculation the error term is found in terms of the angular position error accumulated from the beginning of the flight

$$\int_{t_n-2\tau}^{t_n} \dot{\theta}_{est.error} dt = \int_{0}^{t_n} \dot{\theta}_{est.error} dt - \int_{0}^{t_n-2\tau} \dot{\theta}_{est.error} dt. \qquad (13)$$

The accumulated error from the start of the flight is found by integrating the difference of the measured output and the output modeled from the measured input. The difference across a moving window is then taken to find the prediction errors throughout a flight log. The predictive performance was only measured on logs not used to generate the model used to assess performance (as a model derived from the flight log after a flight cannot be used to predict during that flight).

A histogram of the prediction errors was computed to visualise the performance. A histogram of the typical prediction errors on the roll axis with a 1 second delay is shown in figure 5. The standard deviation of predictive errors with a 1 second delay for this flight is 1.1^o, and most flights show similar distributions of errors, with no more than a couple of degrees standard deviation. A more thorough analysis of the simulated performance, and variation in performance with latency can be found in [6]. With predictive performance within a few degrees, the concept is now ready for implementation and proof of concept.

6 Implementation

The latency compensation system was implemented on the 112mm size quadrotor. The quadrotor was fitted with a fixed camera with analog transmission. The real time implementation had three key parts:

- Introduce an artificial latency
- Measure the command inputs and generate the prediction of the feedback's change
- Receive and augment the video stream

The Node.js environment was chosen for implementation. Node.js is a javascript runtime, designed for internet connected applications and has event based features. The event

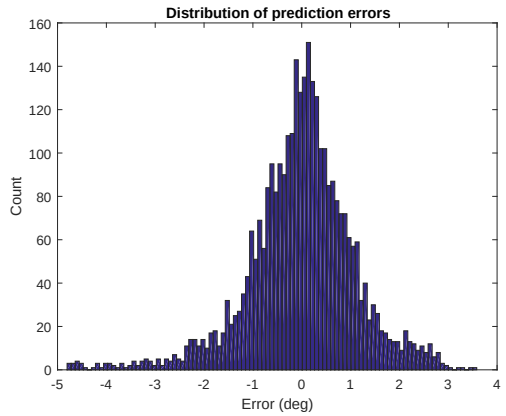

Figure 5: Histogram of predictive errors on the roll axis throughout a flight, for a delay time of 1 second.

based features are useful for running the simulation in real time, I/O in real time and introducing delays. There is a large ecosystem of pre-existing libraries for a variety of tasks, including mathematical tasks, image processing and I/O. The Node.js environment will also be ideal for future work implementing on a real source of latency. Node.js is fast, but being an interpreted javascript environment it is not fast enough to run the image warping in real time.

OpenCV was used to perform the image undistortion and warping, which was faster than Node.js because it runs as precompiled C code.

The architecture for testing the compensator is shown in Fig. 6. Regular flight uses Node.js as a passthrough to display video, to give a baseline. Flying with latency uses the Node.js environment to introduce latency to the system, and latency compensated flight uses Node.js to introduce latency and augment the feedback. All command and feedback passes through the Node.js environment.

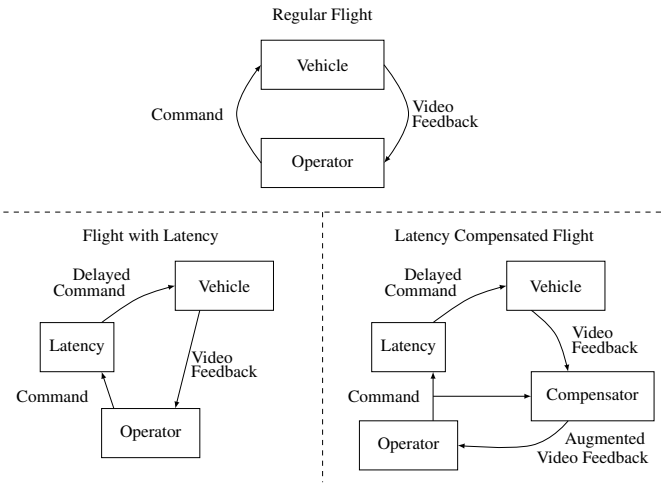

Figure 6: The flight test architectures.

Figure 7 shows the simulation arrangement used. The simulation is indicated in blue, beginning from the start of the flight. The vehicle states in red are simulated and still in memory at t_n. The simulated vehicle states at t_n and $t_n - 2\tau$ are compared to find the expected change in the feedback. A single propagation of the flight was used as it is difficult to guarantee that two simulations running side by side but separated in time will run with identical inputs, and hence may be prone to drift apart. In future work, input to the simulation from telemetry feedback may be necessary to ensure that the simulation does not drift from the true vehicle state. Because the quadrotor's angular response does not vary with flight condition, this architecture is sufficient.

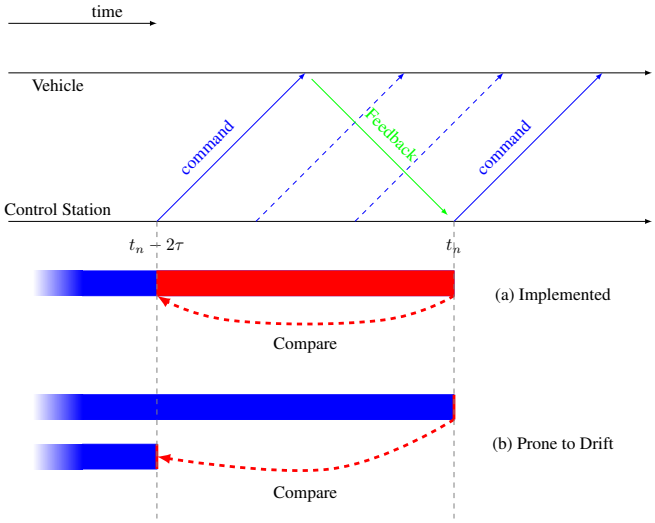

Figure 7: Simulation and computation of the predicted feedback.

7 Flight Testing and Results

The 112mm quadrotor was flown without any latency, with latency and with compensated latency.

The test pilot did not have any remarks about abnormal flight behaviors when flown without latency. The size of the quadrotor meant that its position was quite susceptible to gusts.

Latencies of 500ms and 1000ms were tested. With a 500ms latency, the test pilot struggled to maintain control. Early flights resulted in crashes. Later flights the test pilot was able to maintain flight, but only achieving the most basic maneuvers and tasks. Control was achieved by giving brief inputs and then waiting to observe the effect of those inputs before determining the next input. Controlled flight was not achieved with a 1000ms delay.

Compensated latencies of 500ms and 1000ms were tested. The test pilot remarked immediately when flying with 500ms of compensated latency that the aid allowed smooth control of the vehicle. With both 500 and 1000ms of com-

pensated latency, tasks such as following a fence line were achieved. Controlled flips in the roll and pitch axis were achieved. Figure 8 shows the pilot's view at various points through a flip at 1000ms latency. The flips highlighted that not accounting for the delays in the various command and feedback interfaces had an effect on the prediction. The compensator expected the video to show the vehicle rolling briefly before it did, so the horizon dipped from level as the quad was "catching up" to the pilot. This bug will be addressed in future flight testing.

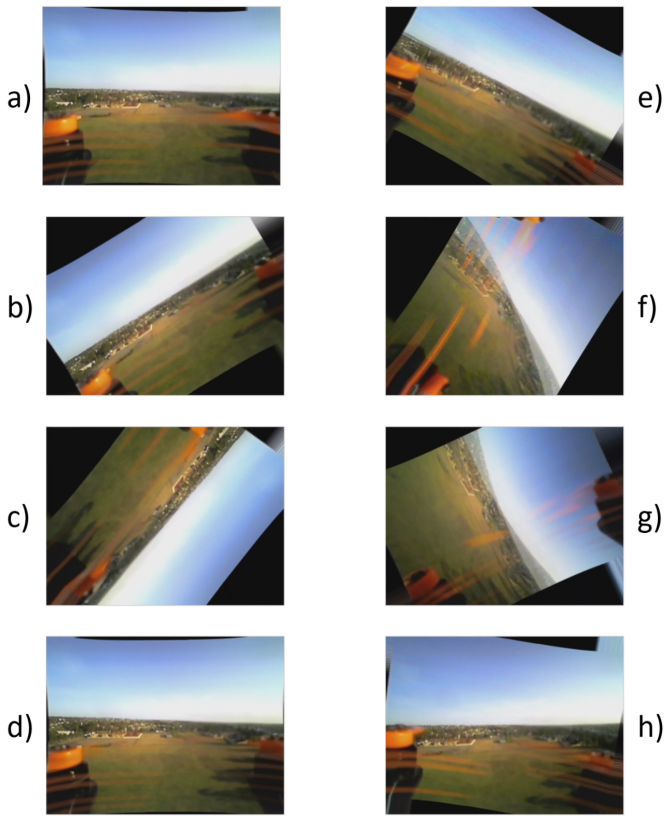

Figure 8: Frames displayed by the compensator during a flip flown with 1000ms latency. The issue introduced by the latency not accounted for can be seen in frames f) and g). Frames are labeled a)-h) chronologically. Frames on the left are taken from when the flip is commanded (vehicle remains level) while frames on the right are taken from when the vehicle executes the flip (horizon should appear to remain level, but doesn't because the compensated latency is slightly less than the true latency). The total time between the first and last frame is approximately 2 seconds.

8 Conclusion

The problem of teleoperation has been explored, and the client side prediction model has been borrowed from multiplayer videogames to address the control latency problem in UAVs. The mathematical basis for the concepts viability has been explored and a scheme that does not suffer from drift has been found. The concept has been implemented and proof of concept achieved in flight testing. Future work may refine the system, explore systems where drift of intermediate states like airspeed could affect the system response or explore implementation with a real latency source (such as an internet connection).

References

[1] Tara Bartlett. Delay compensation for international unmanned aerial vehicle control "unpublished". *AERO3711 Final Report*, 2017.

[2] W.R. Ferrell and T.B. Sheridan. Supervisory control of remote manipulation. *pp 16-17 of NEREM Record. Northeast Electronics Research and Engineering Meeting, Boston, November 2–4, 1966. Volume VIII. Newton, Mass., Ernest E. Witschi, Jr., 1966.*, Oct 1967.

[3] Jonathan Bohren, Chris Paxton, Ryan Howarth, Gregory D. Hager, and Louis L. Whitcomb. Semi-autonomous telerobotic assembly over high-latency networks. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, HRI '16, pages 149–156, Piscataway, NJ, USA, 2016. IEEE Press.

[4] Yahn Bernier. Latency compensating methods in client/server in-game protocol design and optimization. *Game Developers Conference*, 98033(425), 2001.

[5] Boris B. Betaflight. `https://github.com/betaflight/betaflight`, 2013-2018.

[6] Jeremy Cox. Adapting uav control for latency "unpublished". *Undergraduate Thesis*, 2017.

[7] GitHub Contributors. ArduPilot. `https://github.com/ArduPilot/ardupilot`, 2011-2018.

[8] GitHub Contributors. PX4 Pro Drone Autopilot. `https://github.com/PX4/Firmware`, 2011-2018.

Appendix A: Quadrotor Control Modes

The following review of control modes is based on material from the ArduPilot [7], PX4 [8] and BetaFlight [5] open source flight control codes and their associated user guides.

A.1 Basic Control Modes

The most basic control systems use the attitude or attitude rates as the process variables, as well as an overall throttle setting (which is not subject to a closed loop control system). The three angular inputs typically use a self-centring gimbal for input, and an axis which holds its position for throttle.

The inexpensive 'toy' grade models typically implement these control schemes. These control systems require only a

gyro, as well as an accelerometer for attitude control modes, which are easily and inexpensively surface mounted aboard a flight computer board.

A.2 Rate Mode

Also referred to as: acro mode.

The most basic controllers use a gyro only to control the attitude rates, requiring three axes of input for the attitude rates and one for the throttle setting. The relative thrust setting of the opposite pairs of motors is used as the actuator for each axis. The throttle input determines the overall sum of all of the motor throttle settings. A basic block diagram for this arrangement is shown in Fig. 9. While this mode is quite basic and simple to understand, it is one of the trickiest to fly (though it allows the largest set of possible maneuvers and most direct control) due to its lack of self-leveling.

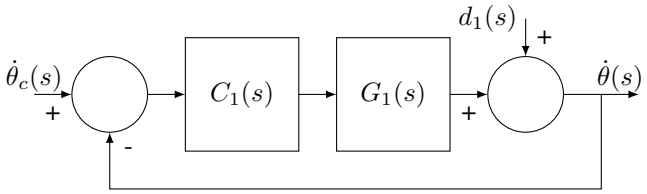

Figure 9: Rate mode controller typical block diagram.

A.3 Angle Mode

Also referred to as: stabilise, self-level mode.

Angle mode is fairly similar to rate mode, except that instead of controlling the angular rates, the pitch and roll angular position is determined by the stick inputs. The stick inputs are mapped to angular positions such that a zero input puts the quadcopter in a level, hovering attitude. Pitch/roll inputs are used to place the vehicle at an angle, which accelerates the quadcopter horizontally. This control is achieved using a new control loop, with the rate control loop from the rate mode as the plant, shown in Fig. 10.

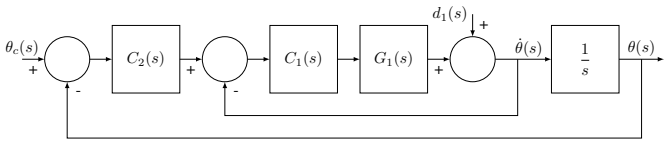

Figure 10: Angle mode controller typical block diagram.

The yaw axis remains an angular rate controller rather than angular position controller (so that a zero stick input gives a steady heading, rather than moving the quadcopter to a particular heading angle).

A.4 Advanced Control Modes

More advanced systems build on the basic control modes, providing as setpoints to basic control modes inputs based on other sensors, so that setpoints may be set for other variables. Commonly available sensors include:

- barometer, which can be inexpensively surface mounted on a flight computer board
- GPS, which is often an external module connected to the flight computer via a serial connection
- optical flow, which typically requires a substantial processor alongside the flight computer
- ultrasonic sensor, which allows sensing proximity to obstacles in a particular direction

With these additional sensors, the vehicle position and position rate (on some or all axes, depending on sensor) can be used as a process variable. This allows flight modes such as position or altitude hold. Other control modes allow preprogrammed setpoints to be used, such as orbiting a point at a constant rate or following waypoints.

Abstraction of operator control means changing the operator's control from things like "roll left" to "move to these coordinates" and can be an effective means of combating latency.