PD Altitude Control of a Parrot Mambo Quadrotor

Trevor Burgoyne, AEM 4602W, Lab Group 3Bi

13 November 2022

Abstract

This report details the experiments performed using Proportional-Derivative (PD) Altitude Control of a Parrot Mambo Quadrotor. The drone was piloted autonomously to a desired reference height, and the settling time and steady-state error were investigated using the flight data recorded by the Parrot Mambo software. Three pairs of control gains were tested, and two of those pairs were tested a second time with an added 10g payload. The same experiments are compared to both a Low-Fidelity and a High-Fidelity MATLAB simulation that were generated using the same flight plan. The relationship between the control gains, mass, and the dynamic response is discussed and compared to theory. An expansion of the test methodology with the addition of integral control or other more advanced techniques is recommended as a potential way to improve performance.

Introduction

Quadrotors provide much needed utility in various applications, due to their capacity for vertical take-off and landing (VTOL) as well as the ability to carry sensor payloads to accomplish different tasks. A large body of research exists exploring various control solutions for such rotorcraft, with PID control being a well-established solution [1]. Simpler solutions, like the PD control testing in this experiment, are usually relegated to serving as a benchmark for comparison to more robust methods, as seen in the work by Kaplan et al [2]. Yet starting with the straightforward PD control can serve as a useful introduction to fundamental concepts that form the basis of many of the more sophisticated control algorithms in use today.

The PD control used in this experiment follows the second order differential equation shown below, where h is the current altitude, $\dot{h}$ is the vertical velocity, $\ddot{h}$ is the vertical acceleration, m is the mass, h^* is the reference height, and $\widehat{Kp}$ and $\widehat{Kd}$ are the proportional and derivative control gains, respectively:

$$m\ddot{h} + \widehat{Kd}\dot{h} + \widehat{Kp}h = \widehat{Kp}h^*$$

The proportional (P) control parameter scales according to the current position, whereas the derivative (D) control is proportional to the current velocity, meaning that it has a damping effect on the response. Additionally, the performance metrics used to analyze the effect of different control gains on the response will be the steady-state error (h_{ss}) and the settling time (t_s). The steady-state error is defined as the difference between the average steady-state altitude of the quadrotor and the desired reference position. The settling time is the amount of time it takes for the response to stay within ± 5% of the same steady-state altitude.

The experiments described hereafter aim to measure the response of a drone to different pairs of PD control constants, and to compare that to both high and Low-Fidelity simulations of the same flight plan. The steady state error and settling time of each test will be presented, and the level of damping for each pair of control variables will be estimated.

Apparatus & Methodology

The experiment was divided into three parts, which will be referenced throughout this report. Part 1 involved using a Parrot Mambo quadrotor at three different pairs of controller gains, with two runs for each test, for a total of six flights. Part 2 consisted of testing the same three pairs of gains in a High-Fidelity simulation, in addition to testing three more arbitrarily selected pairs of control gains for a total of six simulations. Lastly, Part 3 added a ~10g payload to the drone, and two sets of controller gains were tested in both High-Fidelity simulations as well as test flights. In each part, the same flight plan was used. The drone would take off to an altitude of 0.7 m and hover for 20 seconds, after which it would ascend to 1.25 m and remain at that altitude for 25 seconds. Finally, it would descend to 0.7 m briefly before landing. The area of investigation was the drone's response to the control model when moving from 0.7 m to 1.25 m, since that portion of the flight avoids the additional disturbances associated with taking off from rest. The test flights and simulations were run off of laptops in the Drone Lab in Shepherd Laboratories at the University of Minnesota. The flight area was surrounded by a black mesh for safety, and all participants wore safety glasses. One team member was always present at the computer that was controlling the drone so that a flight could quickly be terminated should a problem arise. After these experiments were concluded, additional Low-Fidelity simulations were performed outside of the lab for comparison.

For the Low-Fidelity simulations, several assumptions were made to simplify the calculations, the largest of which was not taking into account any sort of disturbances to the system. This gives insight into the ideal response from particular control gains, which in turn can help quantify the impact of disturbances when compared to the actual flight data. It was also assumed that all rotors would provide equal thrust, which is a fair approximation for a flight plan like the one above, where the drone only moves vertically. Because of these assumptions, there was a slight discrepancy between the way Kp and Kd were defined in the Low-Fidelity model, versus $\widehat{Kp}$ and $\widehat{Kd}$ in the Parrot Mambo flights and High-Fidelity model. These two sets of control gains are related by $\widehat{Kp} = 4\widehat{kT} * Kp$ and $\widehat{Kd} = 4\widehat{kT} * Kd$, where $\widehat{kT}$ is a proportional constant that determines the thrust output given a rotor command, and is defined (using the assumption of equal thrust in all rotors) as $\widehat{kT} = \frac{mg}{4\bar{u}}$, where $\bar{u}$ is the average rotor control input. This relationship is used in the Appendix when generating the Low-Fidelity equivalents to the $\widehat{Kp}$ and $\widehat{Kd}$ pairs used during the experiment.

The metrics used to compare the dynamic response in each test were steady state error (h_{ss}) and settling time (t_s). These values were calculated from the graphs themselves, and the response from 20 seconds to 45 seconds was what was investigated, with h_{ss} being the difference between the reference altitude (1.25 m) and the actual steady state altitude. As illustrated in the Appendix, the steady state altitude was taken to be the average altitude of the response between 30 and 45 seconds, since qualitatively it was observed that this region was free of any major oscillations for all cases. The settling time (t_s) was then taken to be the latest time at which the response exceeded ± 5% of the average steady state altitude.

Uncertainty in measured quantities is considered throughout the analysis. Depending on the source, the value of uncertainty is taken from the instrument specification or derived based on the level of precision of the measurement device. Some other sources of disturbance that were less quantifiable include random sensor noise, the remaining power available in the drone battery, and environmental effects (ground effect, air pressure, etc). These uncertainties are discussed in the Results section to help explain variation between runs. The main defined source of uncertainty comes from the mass of the drone itself, which was measured from the average of three or more measurements using a table scale. The resulting masses discussed in this report have an uncertainty of ± 0.05 g.

Since more significant assumptions are made in the Low-Fidelity model, it was anticipated that those results would stray considerably from reality, while the general shape of the dynamic response would still be similar to the experimental results. The High-Fidelity simulation, because it attempted to model some disturbances in the system, was expected to provide a flight trajectory closer to what we would actually see in the flight data, but still somewhat idealized. Across all three methods, it was anticipated that the overall response for different types of systems (underdamped, critically damped, and overdamped) would be consistent in terms of shape, but not necessarily in terms of accuracy. In terms of the control gains, a higher proportional gain should result in higher overshoot, since a higher control input will be modeled for the same difference in height. If the overshoot actually exceeds the reference value, then oscillations characteristic of an underdamped system are expected. For a higher derivative gain, a more damped response is anticipated, because the derivative constant acts according to the velocity of the vehicle, thus lessening the control input when the drone is starting to gain too much speed.

Results

Flight Experiments

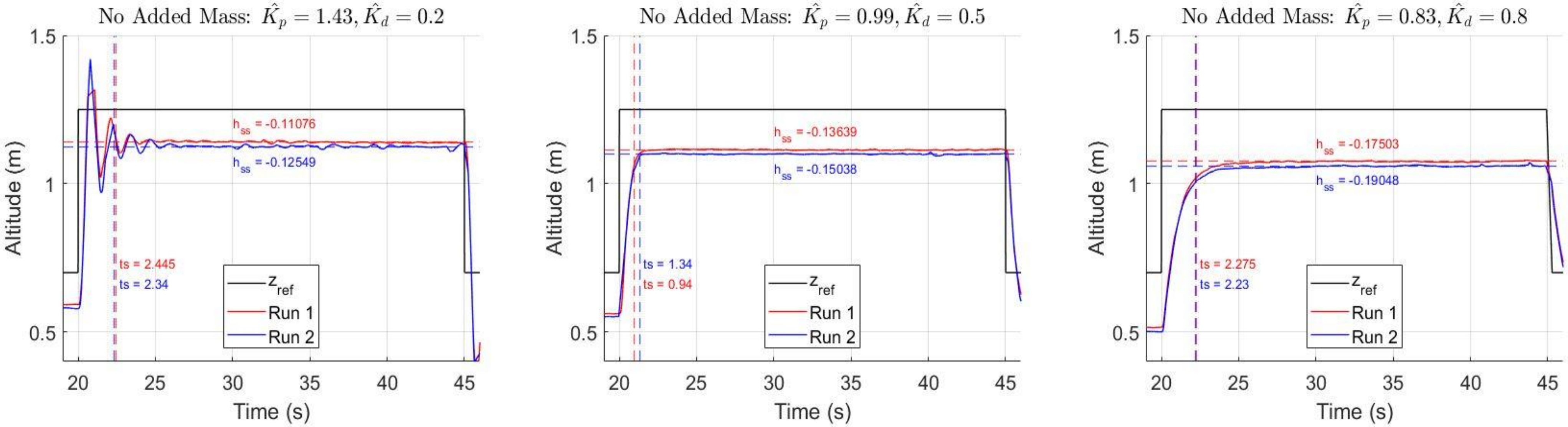
Three tests were conducted at different control gains with two flights each. Then, two of these tests were performed again with an added payload, with two flights each. In total, ten flights were recorded, with two for each test case. For brevity, each flight test will be referred to by its $\widehat{Kp}$ value, and whether or not an additional mass was added (ie, $\widehat{Kp}$ = .99 with no mass). For each test, both runs are shown on the same graph, with the target height noted in black. h_{ss} and t_s are also presented for each run. Looking at Figures 1-3, the three test cases show three different categories of dynamic response: underdamped, critically damped, and overdamped, respectively. Figure 1 illustrates an underdamped response as evidenced by the large oscillations in altitude that occur from ~20-23 seconds. This is expected in situations where $\widehat{Kp}$ is much larger than $\widehat{Kd}$, since the proportional part of the control overreacts to the large jump in reference altitude, and the low derivative control has less ability to slow the high rate of change. Settling time is therefore large, but the steady state error ends up being the lowest of the three tests. Figure 2 appears critically damped, since it quickly reaches its steady state with no overshoot or oscillation. The settling time is by far the lowest in this case, since the proportional gain is high enough to provide a rapid climb, and the derivative control is sufficient to completely prevent any large overshoot. Figure 3, the overdamped case, exhibits a similar response to Figure 2, but with a longer settling time and larger steady state error. It can be seen that $\widehat{Kp}$ is low enough that the derivative overdamps the system, causing a slower rise to steady state that ends up the farthest away from the target altitude. The tests with added mass still agree very well with the analysis of the no mass cases, with Figure 4 appearing critically damped and Figure 5 being overdamped. The magnitudes of both the steady state error and settling time are also very similar, with the exception of Run 1 in Figure 5. It should be noted that in between Run 2 of Figure 4 and Run 1 of Figure 5, the drone had been exhibiting erratic behavior and had to be emergency stopped in-flight, which did cause some damage to the drone body itself. Components that fell off were subsequently reattached, but it is likely that some slight imbalance in the weight of the drone itself led to the slow, upward drift that is not present in any of the other flights. But overall the flights with added mass behaved very similarly to those without, with the only difference being perhaps a very slight increase in settling time in the tests with the payload.

High-Fidelity Simulations

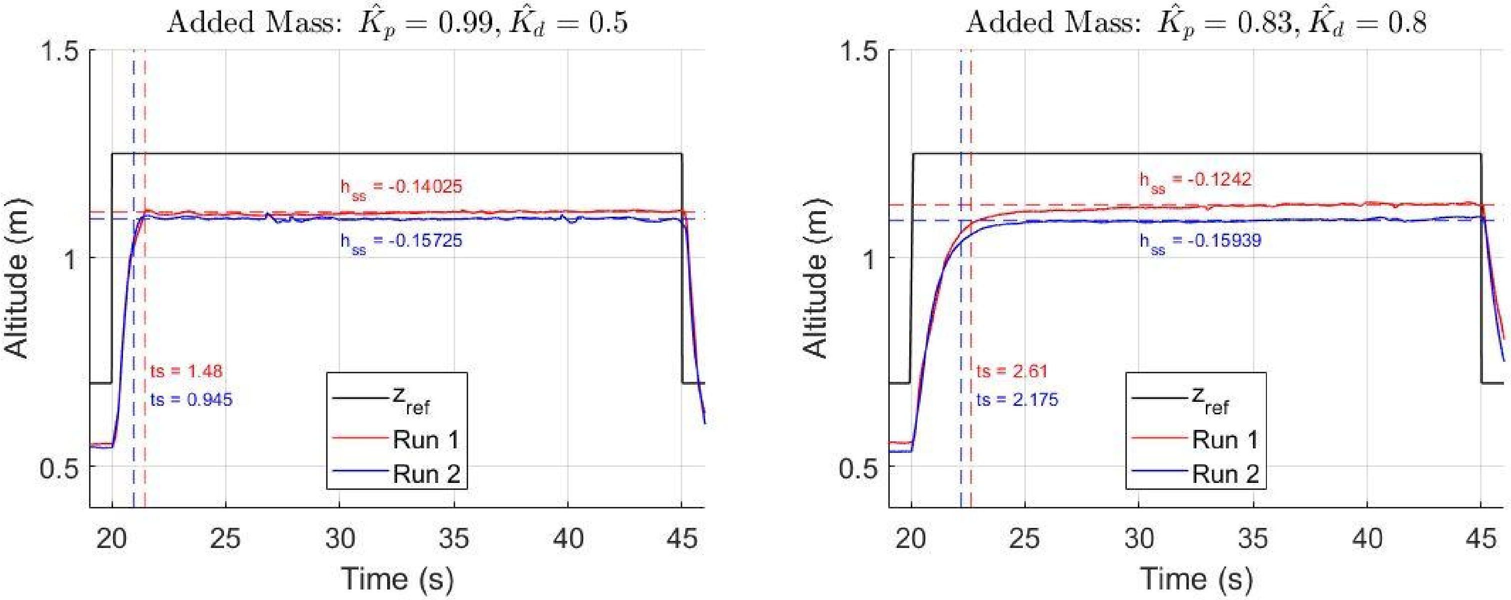
The same three test sets of control gains were also run through a High-Fidelity simulation of the Parrot Mambo flight path. The same two sets of control parameters ($\widehat{Kp}$ = 0.99 and 0.83) were also simulated with the additional mass, and the results for all five tests are shown in Figures 6-10. In comparison to the actual flight results, the simulations all exhibit very little steady-state error, due to the fact that very few disturbances are fully captured by the model. Additionally, the settling times for Figures 6-8 appear to be consistently lower than those in Figures 1-3, which is likely also due to the fact that the simulation shows an idealized trajectory. However the High-Fidelity model does manage to capture some amount of oscillation even in the steady-state region of the response, something that will be lacking in the Low-Fidelity simulation results that will be presented next.

Low Fidelity simulations

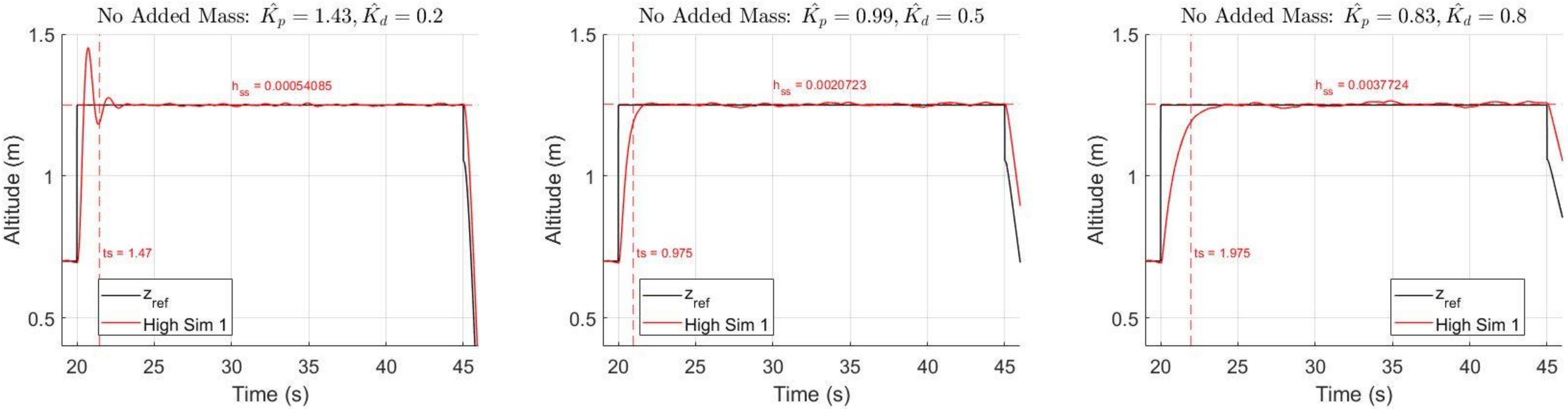
Lastly, the same five scenarios were also analyzed using a Low-Fidelity model as shown in Figures 11-15. In addition to the steady-state error and settling time, the natural frequency (w_n) and damping ratio (zeta) were calculated for each case, as detailed in the Appendix. As was discussed in Apparatus & Methodology, the inputs to the Low-Fidelity model were $\widehat{kT}$, Kp and Kd , which were derived from the motor control recorded during the flight tests. Looking at zeta for Figures 11-13, we can confirm that these three pairs of gains do in fact represent three different types of damped response, namely overdamped (zeta < 1), critically damped (zeta ≈ 1), and overdamped (zeta > 1), respectively. The shape of the responses are very similar to the actual flights we see in Figures 1-3, which supports the fact that our conversions between the two types of control parameters are valid. Steady-state error is very nearly zero, which is to be expected from a model that assumes no disturbances, since those are what give rise to such errors. Settling times are also predicted to be much lower than those observed in the actual flight tests.



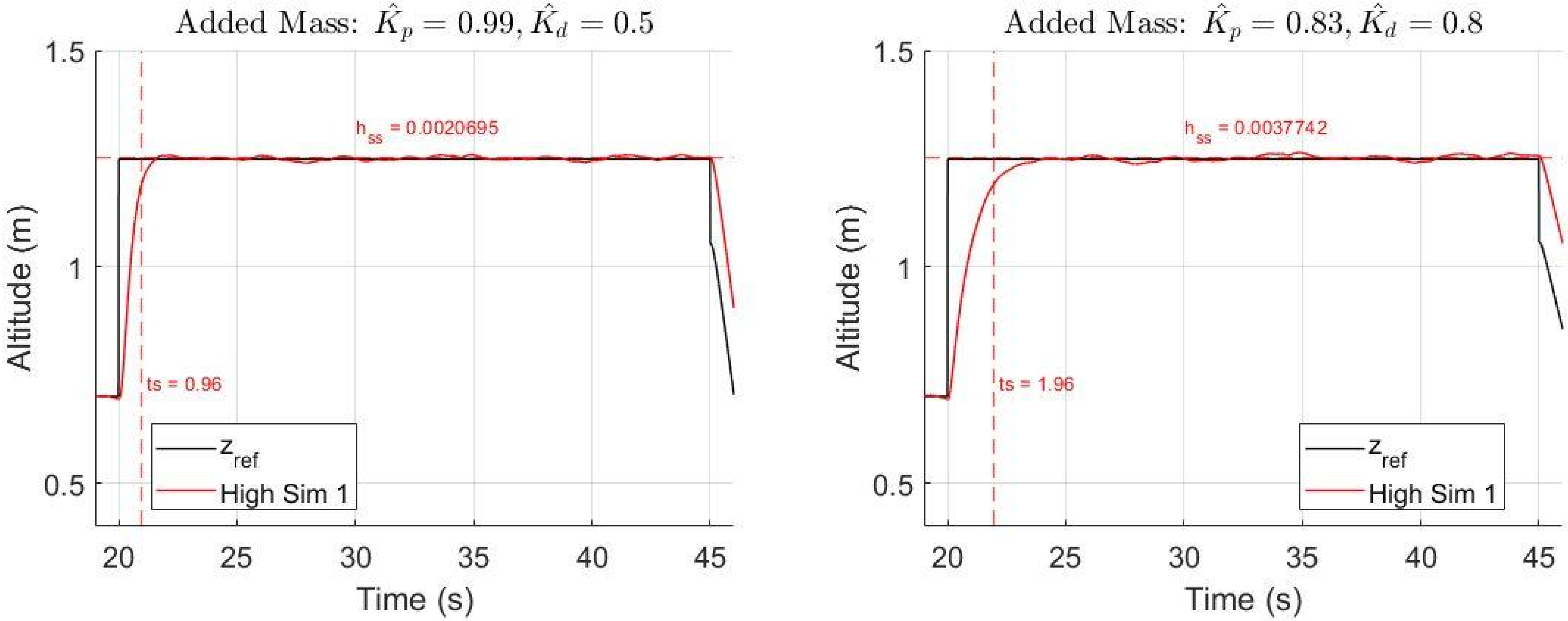
Figures 1-3: Flights with no added mass and $\widehat{Kp}$ = 1.43, 0.99, and 0.83, respectively.



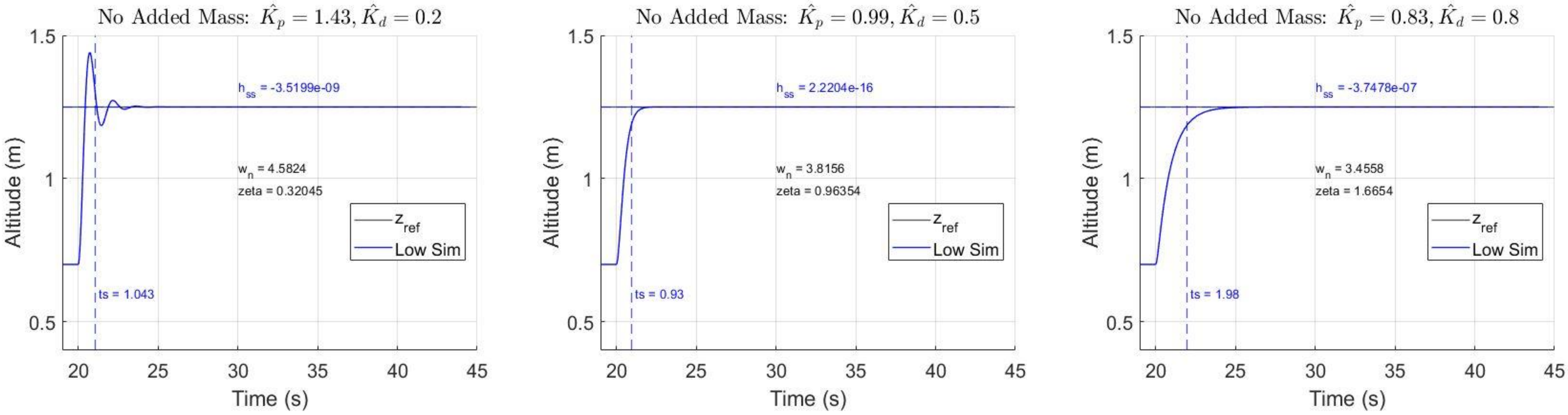
Figures 4-5: Flights with added mass and $\widehat{Kp}$ = 0.99 and 0.83, respectively.



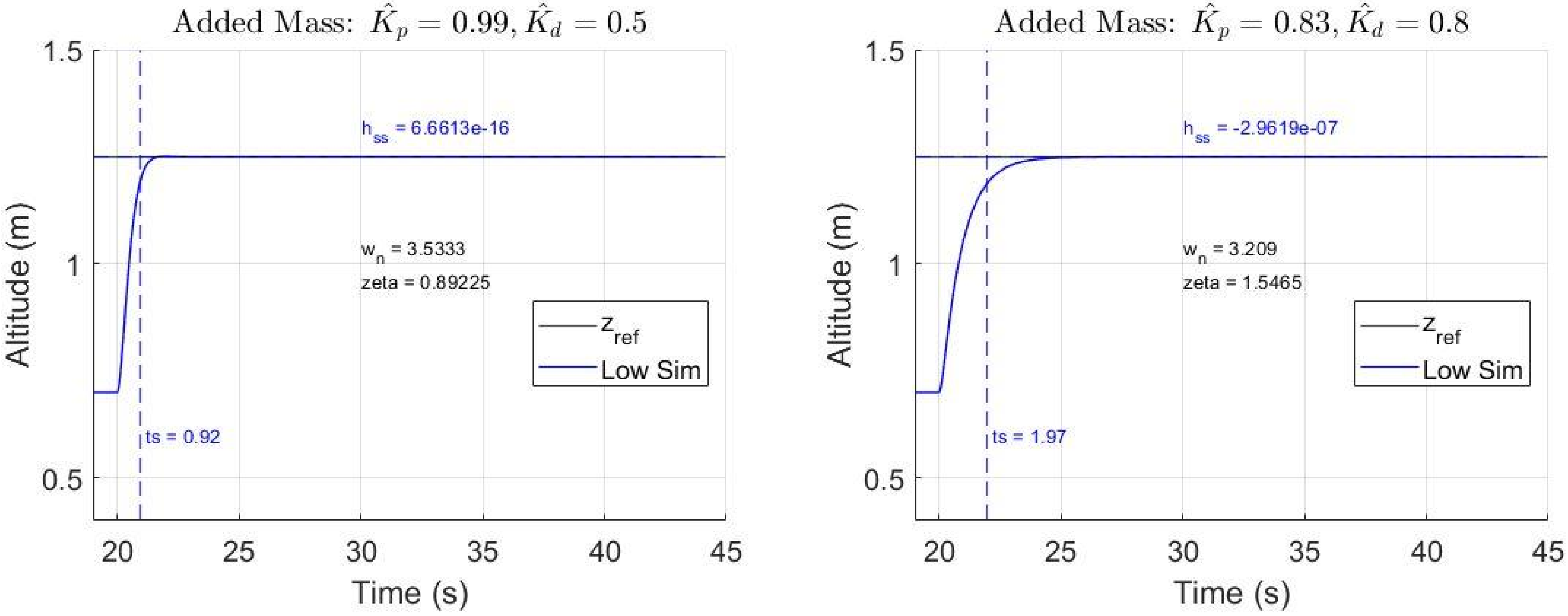
Figures 6-8: High-Fidelity simulations with no added mass and $\widehat{Kp}$ = 1.43, 0.99, and 0.83, respectively.



Figures 9-10: High-Fidelity simulations with added mass and $\widehat{Kp}$ = 0.99 and 0.83, respectively.



Figures 11-13: Low-Fidelity simulations with no added mass and $\widehat{Kp}$ = 1.43, 0.99, and 0.83, respectively.



Figures 14-15: Low-Fidelity simulations with added mass and $\widehat{Kp}$ = 0.99 and 0.83, respectively.

Overall discussion

The collective results from all of the different simulations and flight tests performed are presented in Table 2 below.

Control Gains	Mass (g)	Test Type	h_{ss} (m)	var h_{ss} (mm)	t_s (s)
$\widehat{Kp}$ = 1.43, $\widehat{Kd}$ = 0.2	68.1	Low-Fidelity	0.000	N/A	1.04
		High-Fidelity	0.001	N/A	1.47
		Flight	-0.1181	0.1084	2.39
$\widehat{Kp}$ = 0.99, $\widehat{Kd}$ = 0.5	68	Low-Fidelity	0.000	N/A	0.93
		High-Fidelity	0.002	N/A	0.98
		Flight	-0.143	0.0977	1.14
$\widehat{Kp}$ = 0.99, $\widehat{Kd}$ = 0.5	79.5	Low-Fidelity	0.000	N/A	0.92
		High-Fidelity	0.002	N/A	0.96
		Flight	-0.149	0.1446	1.21
$\widehat{Kp}$ = 0.83, $\widehat{Kd}$ = 0.8	69.5	Low-Fidelity	0.000	N/A	1.98
		High-Fidelity	0.004	N/A	1.98
		Flight	-0.183	0.1194	2.25
$\widehat{Kp}$ = 0.83, $\widehat{Kd}$ = 0.8	80.6	Low-Fidelity	0.000	N/A	1.97
		High-Fidelity	0.004	N/A	1.96
		Flight	-0.142	0.6192	2.39

Table 2: Results for Low-Fidelity, High-Fidelity, and Flight tests for each set of control gains and masses.

Across each of the different types of tests conducted for each pair of control gains, the same overall trends were observed, which evidences the validity of the models used in both the Low and High-Fidelity simulations. The first pair ($\widehat{Kp}$ = 1.43, $\widehat{Kd}$ = 0.2) prompted an underdamped response, the second ($\widehat{Kp}$ = 0.99, $\widehat{Kd}$ = 0.5) was critically damped, and the third ($\widehat{Kp}$ = 0.83, $\widehat{Kd}$ = 0.8) resulted in an overdamped system. But even though the simulations and flight tests all point to the same overall dynamical characteristics, the actual values of the performance metrics (h_{ss}, t_s) differ significantly with the flight test data, and, to a lesser extent, vary between the two types of simulation. Since both of the simulation models exclude many potential sources of disturbance, it is to be expected that they would predict much lower magnitude h_{ss} and t_s, since they assume an ideal test environment. Interestingly, the Low and High-Fidelity simulations generated very similar results, with the only major difference being a small but noticeable steady-state error present in the High-Fidelity model while basically zero error was predicted in every Low-Fidelity test. The presence of this error in the High-Fidelity model is a consequence of the attempt it makes to model at least some of the disturbances expected in a real-life system. Since $\widehat{Kp}$ and $\widehat{Kd}$ were changed inversely at the same time, it is difficult to parse out exactly what is responsible for what, but it seems that increasing $\widehat{Kp}$ causes the overshoot to increase, since the controls react more strongly to the same difference in current and reference position. Alternatively, increasing $\widehat{Kd}$ decreases the overshoot, because the derivative control has a damping effect of the system which dials down the output of the proportional control.

The significant and consistent negative h_{ss} of all of the flight tests points to the Parrot Mambo drone itself perhaps having some sort of sensor bias that causes the system to overestimate its current altitude, an interesting observation that is also noted in [2] and [3]. Kaplan et al present an altitude graph (Figure 6 in [2]) that appears very similar to the ones generated by this experiment, with a significant negative steady-state error for PD altitude control of a Parrot Mambo.

Given the same set of control gains, the effects of mass across all tests were very negligible. The overall dynamic response maintains the same shape in both the simulations as well as the actual flight data, and the performance metrics have very slight and contradictory effects. While the simulations predict a slight decrease in settling time with the added mass, the actual flight tests show a slight increase in settling time. The steady-state error shows no clear trend in either direction. This is expected, since the dynamic equations governing the drone’s motion should scale proportionally with the mass, given that the mass input into the model is accurate.

Conclusion

This report explores the effectiveness of PD altitude control for a Parrot Mambo Quadrotor. Several flight tests and simulations were presented at a variety of control gains and flight vehicle masses, illustrating overdamped, critically damped, and underdamped responses. The response of a given test was found to be influenced primarily by the control gains themselves, with very little, if any, effect from adding additional mass. The flight tests showed good agreement with the general shape of both the Low and High-fidelity simulations, but showed greater steady-state error and settling time. Increasing $\widehat{Kp}$ caused the overshoot to increase and introduced oscillations to the response, while increasing $\widehat{Kd}$ decreased the overshoot and damped the system. Overall, the data discussed here indicates that choosing a set of control gains for a particular application can have significant effects on the dynamic response of the system. More investigation could be made into adding integral control into the Parrot Mambo altitude control, such as is seen in [1] and [2], and seeing how that improves upon the performance metrics presented here. Additionally, it could be interesting to deduce the source of the consistent negative steady-state error in the PD control as seen here and in [2], and confirming whether or not the altitude sensing onboard the Parrot Mambo has a measurable bias, like what is explored in [3].

The concepts explored in this report build upon the discussion of control parameters in AEM 4601 Instrumentation Laboratory, where time was taken to experiment with the response of a servo/arduino device. The expansion of those same ideas to the area of rotorcraft control helps to show the wide range of applications that control theory can have in the Aerospace industry and beyond. With an ever-increasing demand for automated vehicles and systems, it is clear that the fundamental principles of PID control will be continually explored for the foreseeable future.

References

[1] Bayisa, A.T., “Controlling Quadcopter Altitude using PID-Control System,” IJERT Journal, Vol. 8, Issue 12, December 2019, pp. 195-199.
https://doi.org/10.17577/IJERTV8IS120118

[2] Kaplan, M. R., et al., "Altitude and Position Control of Parrot Mambo Minidrone with PID and Fuzzy PID Controllers," 2019 11th International Conference on Electrical and Electronics Engineering (ELECO), 2019, pp. 785-789.
https://doi.org/10.23919/ELECO47770.2019.8990445

[3] Veedhi, C. C., Yeedi, V. S. D., “Estimation of Altitude : using ultrasonic and pressure sensors” (Dissertation), 2020.
http://urn.kb.se/resolve?urn=urn:nbn:se:bth-19950

Appendix

Make graphs

Trevor Burgoyne 13 Nov 2022

```
function [avg_hss, var_hss, avg_kt, var_kt, kphat, kdhat, kp, kd] = make_graphs(ROOT_DIR, PREFIX, DISP_NAME, LABEL_NAME, N_TESTS, N_RUNS, masses)
    COLORS = ["red", "blue", "green", "black"];
    T_START = 20; % s
    T_STEADY = 30; % s, Chosen as start of steady-state response from observation
    T_END = T_START+25; % s
    HREF = 1.25; % m
    g = 9.81; % m/s^2

    % Plotting options
    font_size = 12;
    line_size = 15;
    line_width = 1;
    avg_hss = zeros(1,N_TESTS);
    var_hss = zeros(1,N_TESTS);
    avg_kt  = zeros(1,N_TESTS);
    var_kt  = zeros(1,N_TESTS);
    kphat   = zeros(1,N_TESTS);
    kdhat   = zeros(1,N_TESTS);
    kp      = zeros(1,N_TESTS);
    kd      = zeros(1,N_TESTS);
    for test_n=1:N_TESTS
        ts = zeros(1,N_RUNS);
        kt = zeros(1,N_RUNS);
        hss = zeros(1,N_RUNS);
        zs = zeros(1,N_RUNS);
        figure(test_n)
        hold on
        for run_n=1:N_RUNS
            % Load data
            path = ROOT_DIR + PREFIX + test_n;
            if N_RUNS > 1 % Add runs suffix if more than one run was done
                path = path + "R" + run_n;
            end
            path = path + ".mat";
            load(path);

            if run_n == 1 % Only plot reference once
                plot(time,-z_ref,'Linewidth',line_width,'Color',COLORS(4),'DisplayName','z_{ref}');
            end
            plot(time,-z_est,'Linewidth',line_width,'Color',COLORS(run_n),'DisplayName',LABEL_NAME + " " + run_n);

            % Steady-state error
            idxs = find(time >= T_STEADY & time <=T_END); % Indices of steady state region
            z_arr = -z_est(idxs); % Z values being investigated
            zs(run_n) = double(mean(z_arr)); % m, Experimental settling value
            hss(run_n) = zs(run_n) - HREF; % m, Steady state error

            % Settling time
            idxs = find(time >= T_START & time <=T_END); % Idxs of investigated response
            start_idx = idxs(1);
            z_arr = -z_est(idxs); % Z values being investigated

            % Find last time z dipped below 95% of z_settle
            ts_idxs = find(z_arr <= 0.95*zs(run_n));
            if isempty(ts_idxs)
                ts1 = 0;
            else
                ts1 = time(ts_idxs(end) + start_idx);
            end

            % Find last time z rose above 105% of z_settle
            ts_idxs = find(z_arr >= 1.05*zs(run_n));
            if isempty(ts_idxs)
                ts2 = 0;
            else
                ts2 = time(ts_idxs(end) + start_idx);
            end

            ts(run_n) = max(ts1,ts2); % s, Settling time (use the later time)

            % Kt calculation
            motors = abs([motor1' motor2' motor3' motor4']); % Get all motor values
            u = mean(motors);
            kt(run_n) = (masses(test_n)*g) / (4*u) * 1000; % N

        end

        scale = [1.05 0.95]; % scalar values used to position text
        for run_n=1:N_RUNS
            % Steady-state error
            if zs(run_n) == max(zs) % Position labels above if the line is higher
                y = scale(1);
            else
                y = scale(2);
            end
            line_name = "h_{ss} = " + hss(run_n);
            text(T_STEADY,y*zs(run_n),line_name,'Color',COLORS(run_n))
            yline(zs(run_n),"--",'Linewidth',line_width,'Color',COLORS(run_n),'HandleVisibility','off')

            % Settling time
            if ts(run_n) == max(ts) % Position labels above if the line is higher
                y = scale(1);
            else
                y = scale(2);
            end
            line_name = "ts = " + (ts(run_n)-T_START);
            text(1.01*max(ts),y*.7,line_name,'Color',COLORS(run_n))
            xline(ts(run_n),"--",'Linewidth',line_width,'Color',COLORS(run_n),'HandleVisibility','off')
        end
        title(sprintf('%s: $\\hat{K_{p}} = %s, \\hat{K_{d}} = %s$',DISP_NAME,num2str(Kp),num2str(Kd)),'Interpreter','latex');
        xlabel('Time (s)','fontsize',font_size);
        ylabel('Altitude (m)','fontsize',font_size);
        legend('show','Location','best');
        set(gca,'XMinorGrid','off','GridLineStyle','-','FontSize',line_size)
        xlim([T_START-1 T_END+1]);
        ylim([0.4 1.5]);
        grid on

        avg_hss(test_n) = mean(hss);
        var_hss(test_n) = var(hss);
        avg_kt(test_n)  = mean(kt);
        var_kt(test_n)  = var(kt);
        kphat(test_n)   = Kp;
        kdhat(test_n)   = Kd;
        kp(test_n)      = Kp/(4*avg_kt(test_n));
        kd(test_n)      = Kd/(4*avg_kt(test_n));
    end
end
```

```
Not enough input arguments.

Error in make_graphs (line 16)
    avg_hss = zeros(1,N_TESTS);
```

Published with MATLAB® R2020b

Contents

- PID Control of Quadcopter Altitude Near Hover
- Vehicle Parameters
- Step change in altitude reference, m
- Initial Conditions
- PD Control With Gravity Feedforward
- Closed-Loop Transfer Function (hdes to h)
- Convert Gains
- Simulate Closed-Loop
- Plot results

PID Control of Quadcopter Altitude Near Hover

Acknowledgement: Prof. Peter Seiler

```
%--------------------------------------------------
function [wn, zeta] = Low_Fidelity_Model(DISP_NAME, m, kT, Kp, Kd)
```

Vehicle Parameters

close all; clear all; m = 65e-3; % Mass, kg

```
    T_START = 19; % s
    T_STEADY = 30; % s, Chosen as start of steady-state response from observation
    T_END = T_START+25; % s
    g = 9.81; % m/s^2
    % kT = ; % Thrust coefficient, N
    % kT = input('Enter the estimated thrust coefficient, N, kT: ');
    umax = 500; % Maximum motor input command, unitless
    umin = 0; % Minimum motor input command, unitless
```

Step change in altitude reference, m

```
    Tf    = 25;          % Final simulation time, sec
    hdes0 = 0.7;         % Initial altitude, m
    hdesf = 1.25;        % Final altitude, m
```

Initial Conditions

```
    h0 = hdes0;        % Initial altitude, m
    hdot0 = 0;         % Initial altitude velocity, m/s
    hddmax = (4*kT*umax-m*g)/m;    % Maximum upward acceleration, m/s^2
```

```
Not enough input arguments.

Error in Low_Fidelity_Model (line 26)
    hddmax = (4*kT*umax-m*g)/m;    % Maximum upward acceleration, m/s^2
```

PD Control With Gravity Feedforward

Consider the following PD control law with gravity feedforward u = Kp*(hdes-h) - Kd*hdot + (mhat*g)/(4*kThat) Here u is the ***individual motor command*** input (unitless). Also, (mhat,kThat) indicate that these are estimates used by the controller. The parrot drone Simulink diagram uses:5 uhat = Kphat*(hdes-h) - Kdhat*hdot + mhat*g where uhat is the ***total thrust*** command (N). These are related by uhat = (4*kT)*u Thus gains computed using the simplified second-order model must be scaled by (4*kT) before implementing in the parrot Simulink diagram.

NOTE - An integral gain is included below for testing but this can be set to zero to simply to the PD control given above.

```
% Select Gains -- Gains are specified using simplified second order model

GainCase = 2;
switch GainCase
    case 1
        % Select closed-loop natural frequency and damping ratio
        wn = input('Desired natural freq, rad/sec, wn: ');      % Desired natural freq, rad/sec
        zeta = input('Desired damping ratio, unitless, zeta: ' );  % Desired damping ratio, unitless

        % Closed-loop ODE with PD control + perfect gravity cancellation:
        %   m hdd = (4*kT)*( Kp*(hdes-h) - Kd*hdot )
        % --> hdd + (4*kT*Kd/m) hdot + (4*kT*Kp/m) h = (4*kT*Kp/m)*hdes
        Kp = wn^2*m/(4*kT);
        Kd = 2*zeta*wn*m/(4*kT);
        Ki = 0;

    case 2
        % Kp = input('Enter the proportional gain, Kp: ');
        % Kd = input('Enter the derivative gain, Kd: ');
        Ki = 0;

        wn = sqrt(Kp*4*kT/m);
        zeta = 4*kT*Kd/(2*wn*m);
end
```

Closed-Loop Transfer Function (hdes to h)

This constructs the closed-loop model from altitude reference to altitude. It assumes perfect cancellation of the gravitational force by the feedforward term. It also neglects the saturation that limits the motor command.

```
% Plant dynamcs (neglecting gravity term)
%   m d^2(h)/dt^2 = (4*kT)*u
% The state-space model below has input motor command and outputs [h;hdot]
Ag = [0 1; 0 0];
Bg = [0; 4*kT/m];
Cg = eye(2);
Dg = 0;
G = ss(Ag,Bg,Cg,Dg);
time = [];
h_arr = [];

tstep = 1;   % Step time, sec
% Controller: PI and D terms
if Ki==0
    Cpi = ss(Kp);
else
    Cpi = tf([Kp Ki],[1 0]);
end
Cd = ss(Kd);

% Form closed-loop from r to h, T
systemnames = 'G Cpi Cd';
inputvar = '[r]';
outputvar = '[G(1)]';
input_to_G = '[Cpi+Cd]';
input_to_Cpi = '[r-G(1)]';
input_to_Cd = '[-G(2)]';
%T = sysic;

if false
    % Code to debug:  Represent controller
    % (*)  u = Kp e + Kd edot + Ki integral(e)
    % The actual implementation only uses rate-feedback, i.e. Kd*edot is
    % replaced by -Kd*hdot.  However, the form in (*) above is much easier
    % to construct with standard functions and will yield the same poles
    % (only the closed-loop zeros will be different)
    Gb = tf(4*kT/m,[1 0 0]);
    if Ki==0
        Cpid = tf([Kd Kp],1);
    else
        Cpid = tf([Kd Kp Ki],[1 0]);
    end
    Tb = feedback(Gb*Cpid,1);
end
```

Convert Gains

There are two sets of gains: 1) (Kd,Kp,Ki) for simple second order model with motor command input 2) (Kdhat,Kphat,Kihat) for parrot drone simulink model with total thrust command input. These are the gains that should be used in the parrot drone Simulink model.

```
Kdhat = (4*kT)*Kd;
Kphat = (4*kT)*Kp;
Kihat = (4*kT)*Ki;
```

Simulate Closed-Loop

```
% Parameter estimates
mhat = m;
kThat = kT;

% Disturbance Force, N
Fd = 0;

% Simulate system

% Allow sim to be called as a function
% and have proper variable scope
options = simset('SrcWorkspace','current');
sim('QuadPID',[0 Tf],options);
```

Plot results

```
font_size = 12;
line_size = 15;
line_width = 1;
color = 'b';

figure();
hold on
text(T_STEADY,1,"w_{n} = " + wn + newline + "zeta = " + zeta);
yline(hdesf,'Linewidth',line_width,'Color','black','DisplayName','z_{ref}')
plot(tsim+T_START,h,'Linewidth',line_width,'Color',color,'DisplayName','Low Sim');
title(sprintf('%s: $\\hat{K_{p}} = %s, \\hat{K_{d}} = %s$',DISP_NAME,num2str(Kphat),num2str(Kdhat)),'Interpreter','latex');
xlabel('Time (s)','fontsize',font_size);
ylabel('Altitude (m)','fontsize',font_size);
legend('show','Location','best');
set(gca,'XMinorGrid','off','GridLineStyle','-','FontSize',line_size)
xlim([T_START T_END+1]);
ylim([0.4 1.5]);
grid on;




time  = [time tsim];
h_arr = [h_arr h];

% Steady-state error
idxs = find(time >= T_STEADY-T_START); % Indices of steady state region
z_arr = h_arr(idxs); % Z values being investigated
```

```
    zs = double(mean(z_arr)); % m, Experimental settling value
    hss = zs - hdesf; % m, Steady state error

    line_name = "h_{ss} = " + hss;
    text(T_STEADY,1.05*zs,line_name,'Color',color)
    yline(zs,"--",'Linewidth',line_width,'Color',color,'HandleVisibility','off')

    % Settling Time
    if zeta < 0.5 % Underdamped case
        ts = 3/(zeta*wn);
        line_name = "ts = " + (ts-1) + " (theory)";
        text(1.01*(ts+T_START),.6,line_name,'Color','r')
        xline(ts+T_START,"--",'Linewidth',line_width,'Color','r','HandleVisibility','off')
    end

    % Find last time z dipped below 95% of z_settle
    ts_idxs = find(h_arr <= 0.95*zs);
    if isempty(ts_idxs)
        ts1 = 0;
    else
        ts1 = time(ts_idxs(end));
    end

    % Find last time z rose above 105% of z_settle
    ts_idxs = find(h_arr >= 1.05*zs);
    if isempty(ts_idxs)
        ts2 = 0;
    else
        ts2 = time(ts_idxs(end));
    end

    ts = max(ts1,ts2); % s, Settling time (use the later time)

    line_name = "ts = " + (ts-1);
    text(1.01*(ts+T_START),.6,line_name,'Color',color)
    xline(ts+T_START,"--",'Linewidth',line_width,'Color',color,'HandleVisibility','off')
```

```
end
```

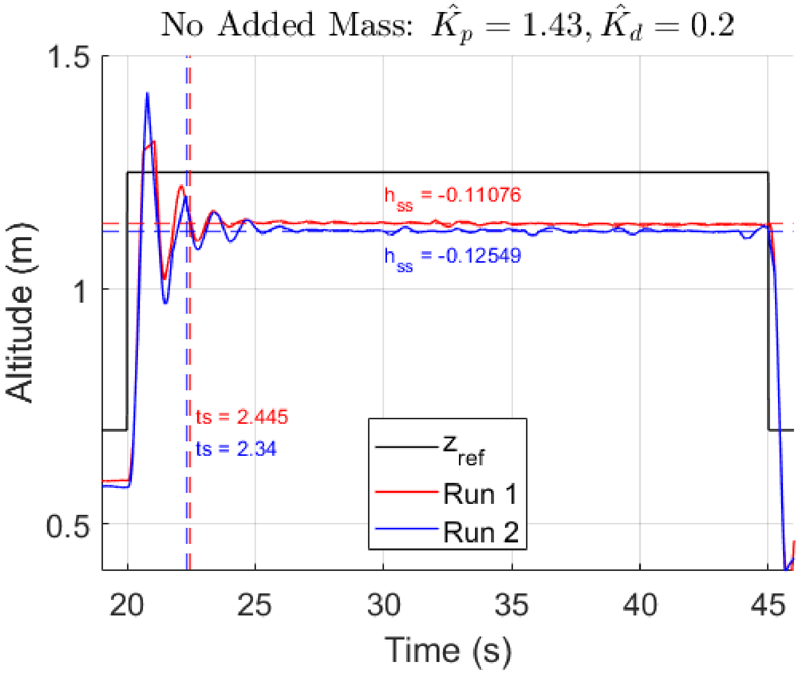
Published with MATLAB® R2020b

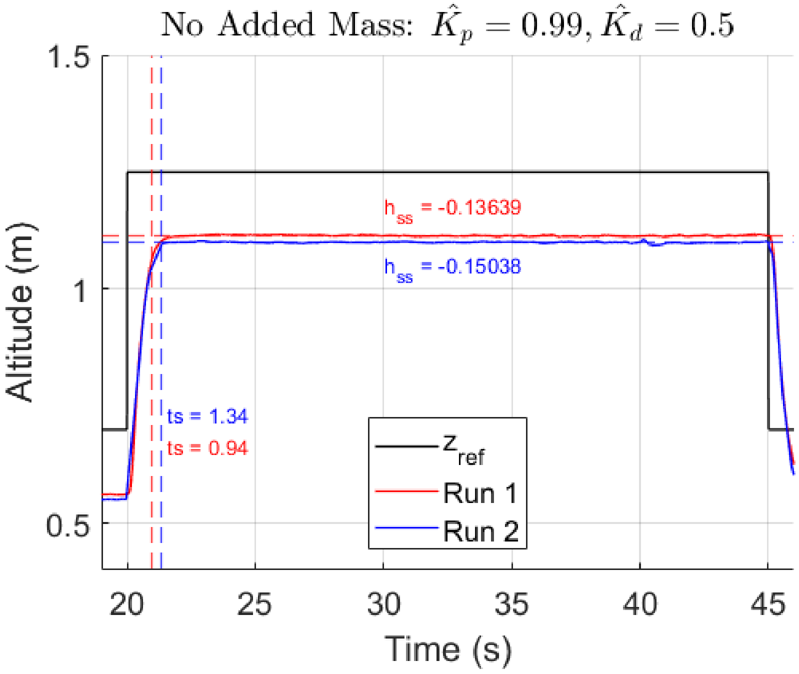
Flight Experiments

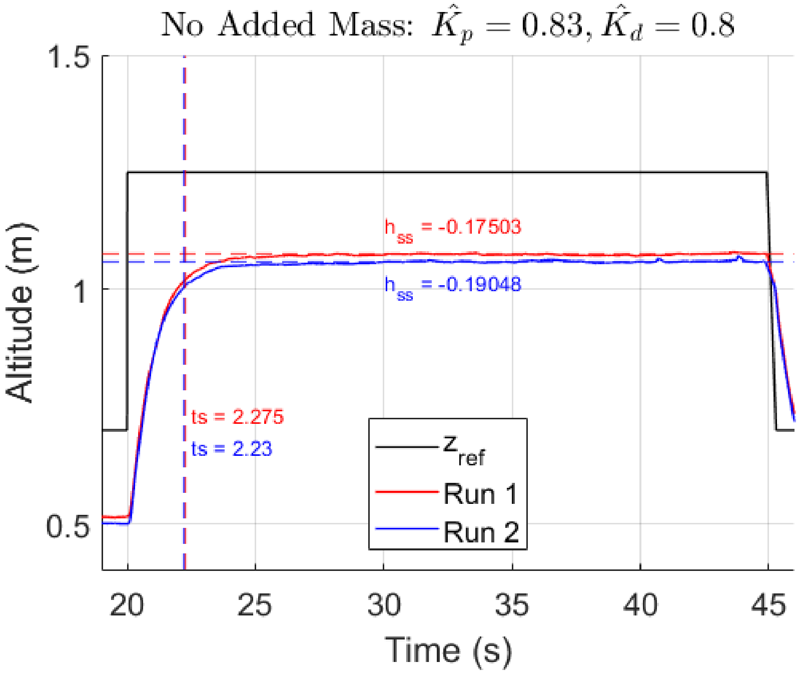
Trevor Burgoyne 13 Nov 2022

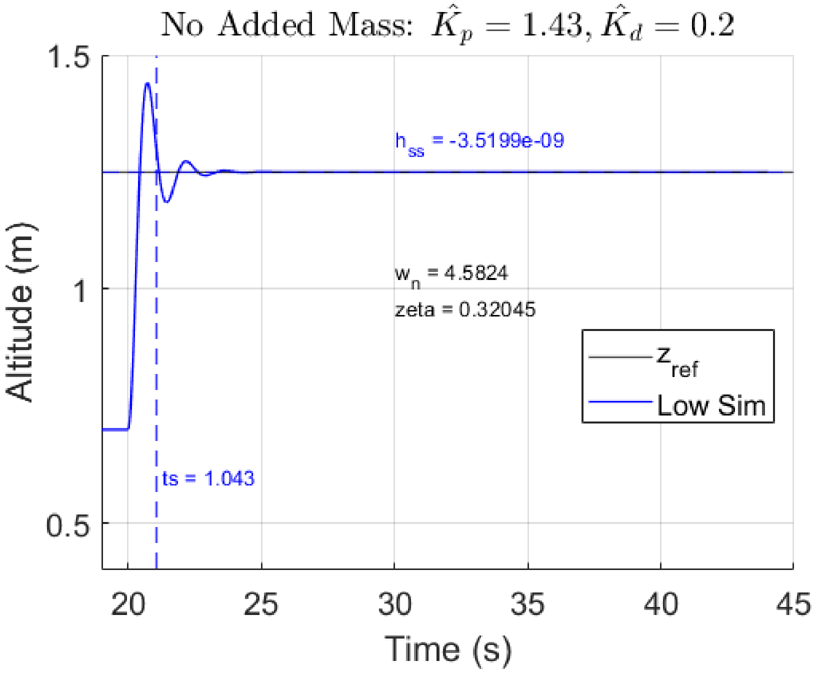
```
% Paths for data loading
ROOT_DIR = "C:/Users/Trevor/Desktop/AEM 4602W/Controls-Lab/Data/Part 1/";
PREFIX = "pjsdata_T";
DISP_NAME = "No Added Mass";
LABEL_NAME = "Run";
N_TESTS = 3;
N_RUNS = 2;
masses = [68.1, 68, 69.5] / 1000; % kg
[avg_hss, var_hss, avg_kt, var_kt, kphat, kdhat, kp, kd] = make_graphs(ROOT_DIR, PREFIX, DISP_NAME, LABEL_NAME, N_TESTS, N_RUNS, masses)
% Generate Low Fidelity Model Plots
for test_n=1:N_TESTS
    [wn, zeta] = Low_Fidelity_Model(DISP_NAME, masses(test_n), avg_kt(test_n), kp(test_n), kd(test_n));
end
```

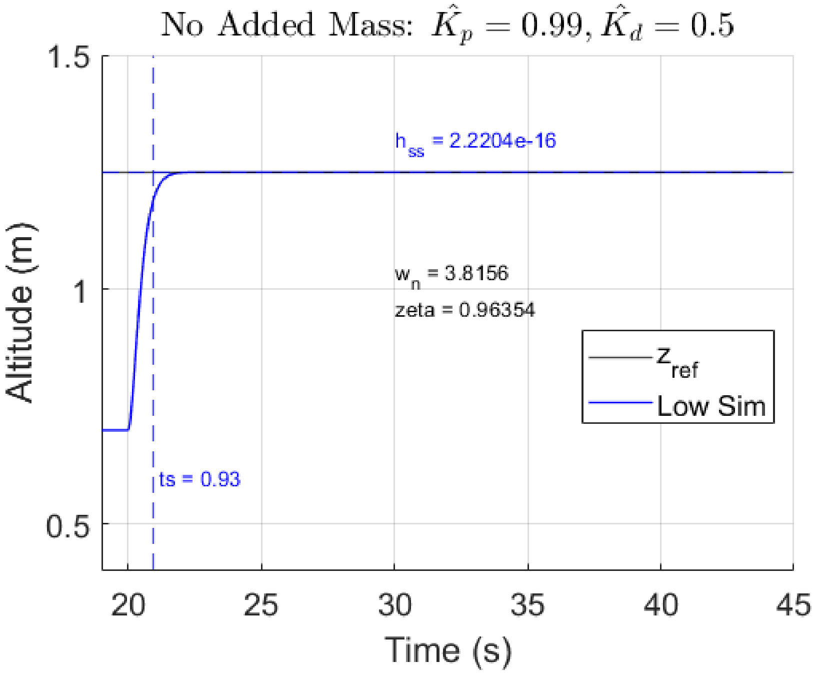
```
avg_hss =

   -0.1181   -0.1434   -0.1828


var_hss =

   1.0e-03 *

    0.1084    0.0977    0.1194


avg_kt =

    0.5240    0.5358    0.5237


var_kt =

   1.0e-04 *

    0.6498    0.4380    0.5865


kphat =

    1.4300    0.9900    0.8300


kdhat =

    0.2000    0.5000    0.8000


kp =

    0.6823    0.4619    0.3962


kd =

    0.0954    0.2333    0.3819
```

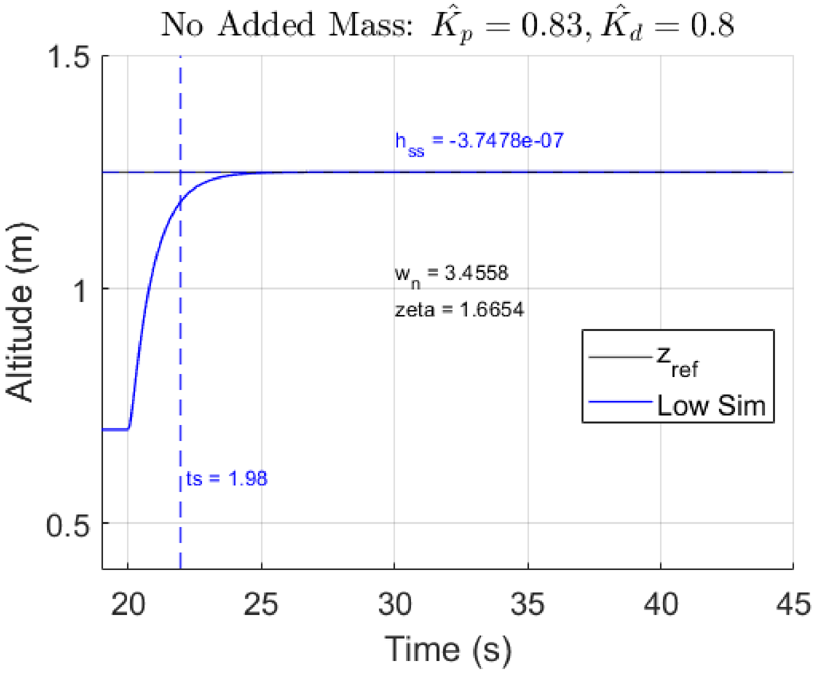

No Added Mass: $\hat{K}_p = 1.43, \hat{K}_d = 0.2$
Altitude (m)
Time (s)
1.5
1
0.5
20
25
30
35
40
45
h_{ss} = -0.11076
h_{ss} = -0.12549
ts = 2.445
ts = 2.34
z_{ref}
Run 1
Run 2


No Added Mass: $\hat{K}_p = 0.99, \hat{K}_d = 0.5$
Altitude (m)
Time (s)
1.5
1
0.5
20
25
30
35
40
45
h_{ss} = -0.13639
h_{ss} = -0.15038
ts = 1.34
ts = 0.94
z_{ref}
Run 1
Run 2


No Added Mass: $\hat{K}_p = 0.83, \hat{K}_d = 0.8$
Altitude (m)
Time (s)
1.5
1
0.5
20
25
30
35
40
45
h_{ss} = -0.17503
h_{ss} = -0.19048
ts = 2.275
ts = 2.23
z_{ref}
Run 1
Run 2


No Added Mass: $\hat{K}_p = 1.43, \hat{K}_d = 0.2$
Altitude (m)
Time (s)
1.5
1
0.5
20
25
30
35
40
45
h_{ss} = -3.5199e-09
w_n = 4.5824
zeta = 0.32045
ts = 1.043
z_{ref}
Low Sim





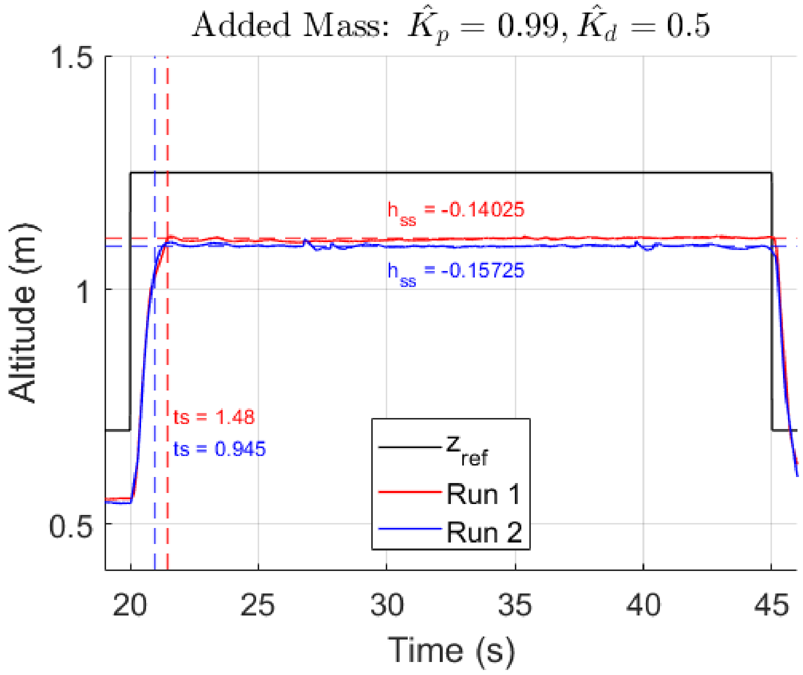
Published with MATLAB® R2020b

Added Mass Experiments

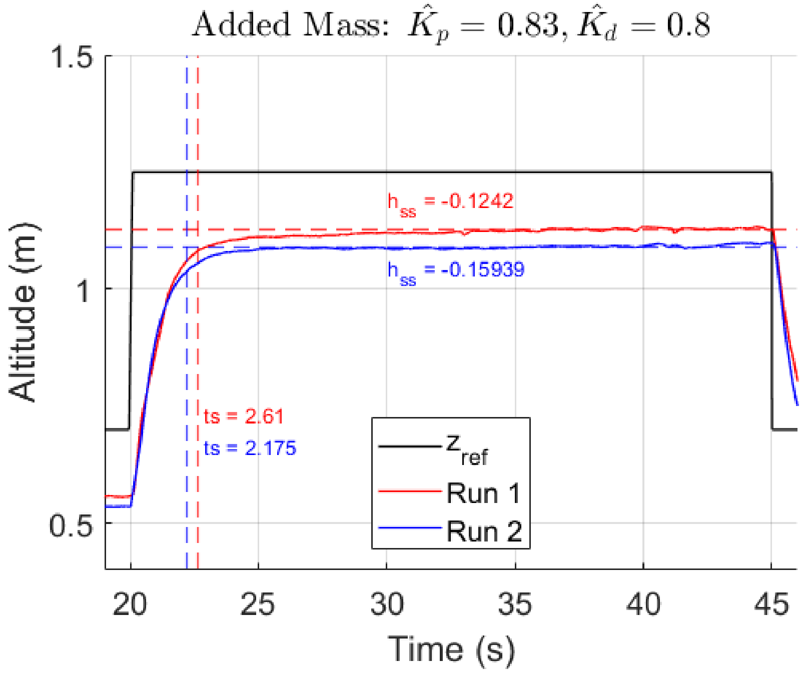
Trevor Burgoyne 13 Nov 2022

```
% Paths for data loading
ROOT_DIR = "C:/Users/Trevor/Desktop/AEM 4602W/Controls-Lab/Data/Part 3/";
PREFIX = "pjsdata_P3T";
DISP_NAME = "Added Mass";
LABEL_NAME = "Run";
N_TESTS = 2;
N_RUNS = 2;
masses = [79.3, 80.6] / 1000; % kg
[avg_hss, var_hss, avg_kt, var_kt, kphat, kdhat, kp, kd] = make_graphs(ROOT_DIR, PREFIX, DISP_NAME, LABEL_NAME, N_TESTS, N_RUNS, masses)

for test_n=1:N_TESTS
    [wn, zeta] = Low_Fidelity_Model(DISP_NAME, masses(test_n), avg_kt(test_n), kp(test_n), kd(test_n));
end
```

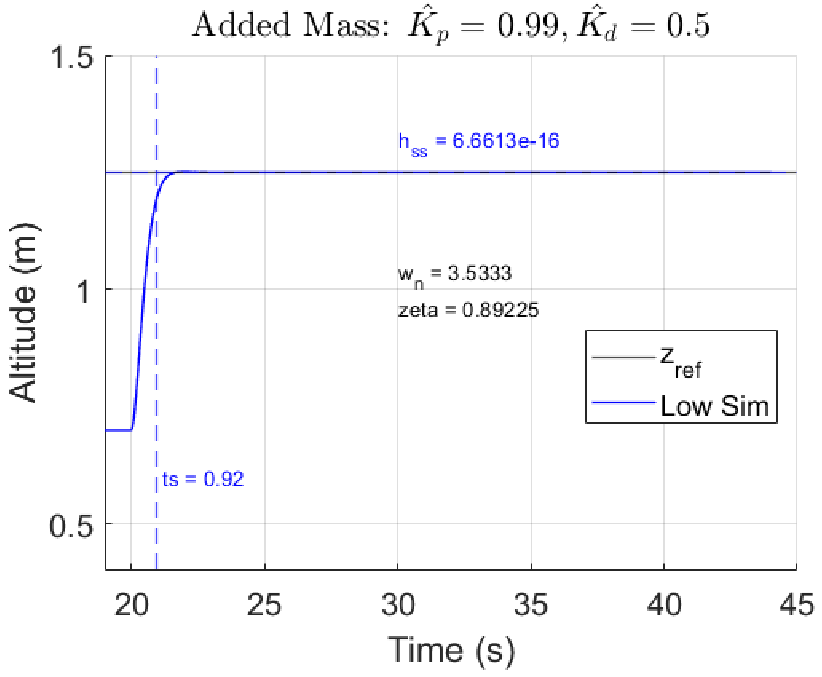
```
avg_hss =

   -0.1487   -0.1418


var_hss =

   1.0e-03 *

    0.1446    0.6192


avg_kt =

    0.5450    0.5636


var_kt =

   1.0e-03 *

    0.0114    0.1419


kphat =

    0.9900    0.8300


kdhat =

    0.5000    0.8000


kp =

    0.4541    0.3682


kd =

    0.2294    0.3549
```

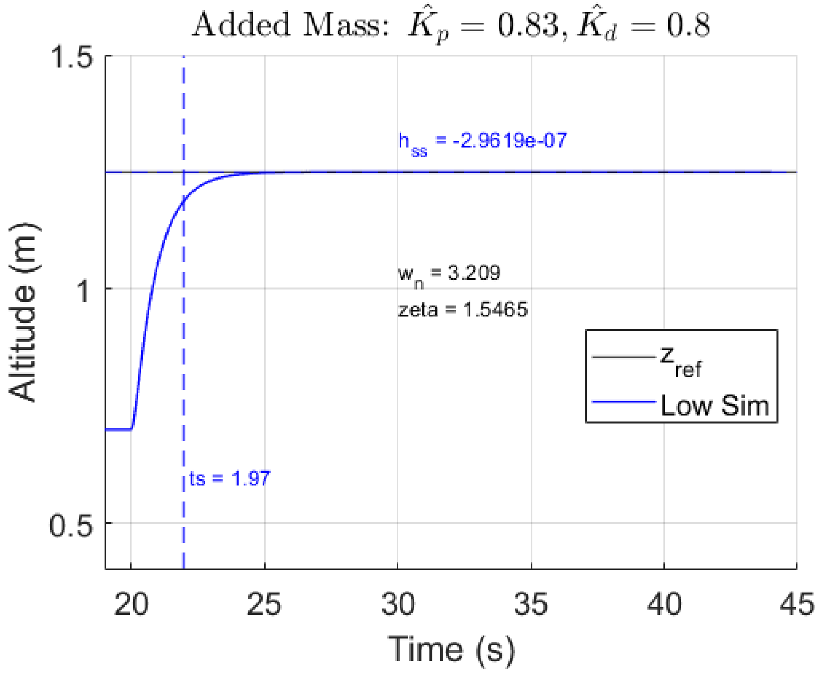


Added Mass: $\hat{K}_p = 0.99, \hat{K}_d = 0.5$



Added Mass: $\hat{K}_p = 0.83, \hat{K}_d = 0.8$





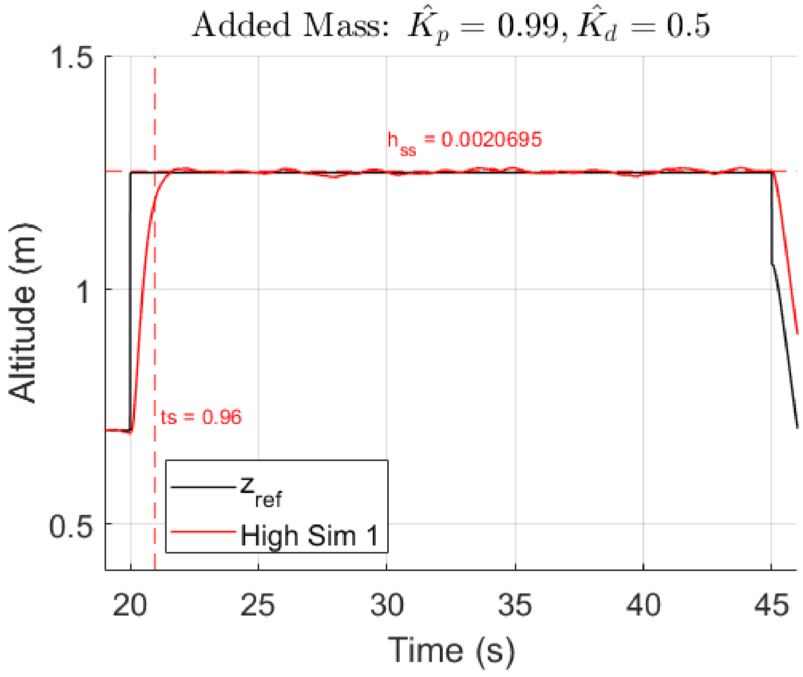
Published with MATLAB® R2020b

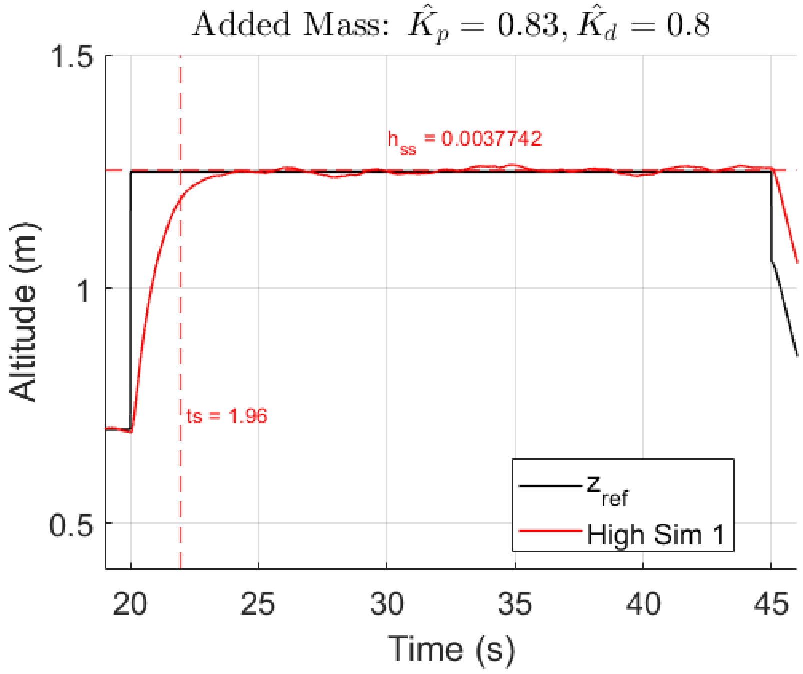
High-Fidelity, Added Mass Sims

Trevor Burgoyne 13 Nov 2022

```
% Paths for data loading
ROOT_DIR = "C:/Users/Trevor/Desktop/AEM 4602W/Controls-Lab/Data/Part 3/";
PREFIX = "part3_t";
DISP_NAME = "Added Mass";
LABEL_NAME = "High Sim";
N_TESTS = 2;
N_RUNS = 1;
masses = [79.3, 80.6] / 1000; % kg
[avg_hss, var_hss, avg_kt, var_kt, kphat, kdhat, kp, kd] = make_graphs(ROOT_DIR, PREFIX, DISP_NAME, LABEL_NAME, N_TESTS, N_RUNS, masses)
```

```
avg_hss =

    0.0021    0.0038


var_hss =

     0     0


avg_kt =

    0.6585    0.6660


var_kt =

     0     0


kphat =

    0.9900    0.8300


kdhat =

    0.5000    0.8000


kp =

    0.3758    0.3116


kd =

    0.1898    0.3003
```





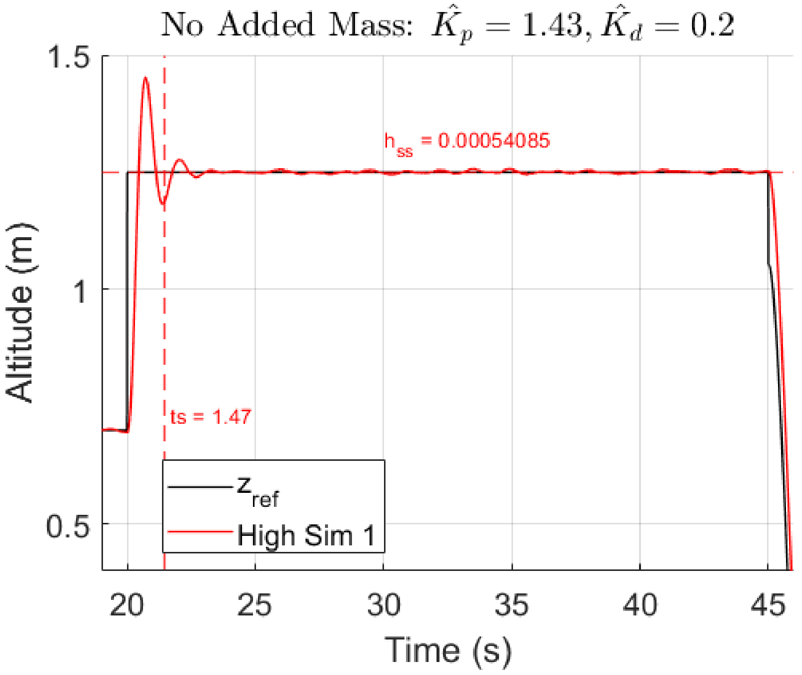
Published with MATLAB® R2020b

High-Fidelity, No Mass Sims

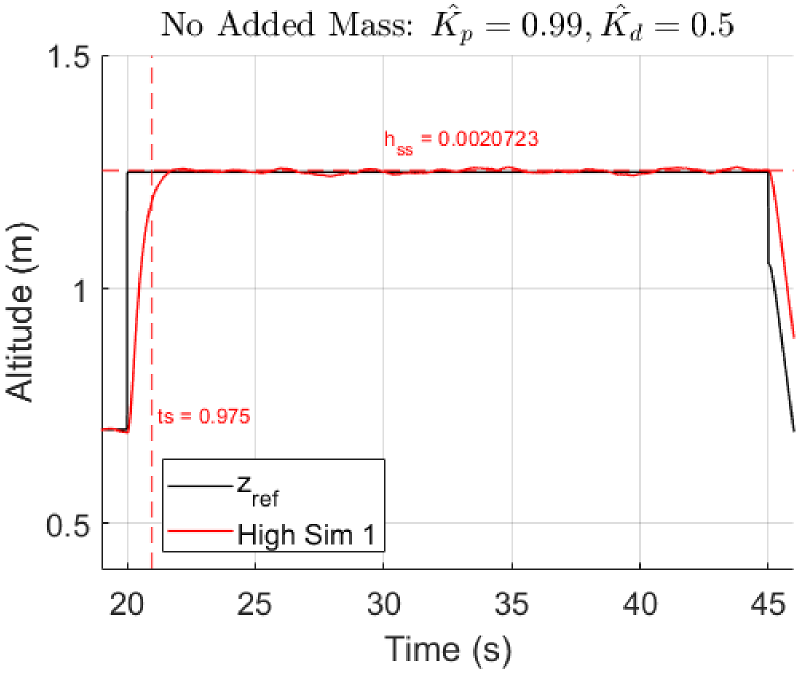
Trevor Burgoyne 13 Nov 2022

```
% Paths for data loading
ROOT_DIR = "C:/Users/Trevor/Desktop/AEM 4602W/Controls-Lab/Data/Part 2/";
PREFIX = "sim_t";
DISP_NAME = "No Added Mass";
LABEL_NAME = "High Sim";
N_TESTS = 6;
N_RUNS = 1;
masses = 69*ones(1,N_TESTS)/1000; % kg
[avg_hss, var_hss, avg_kt, var_kt, kphat, kdhat, kp, kd] = make_graphs(ROOT_DIR, PREFIX, DISP_NAME, LABEL_NAME, N_TESTS, N_RUNS, masses)
```

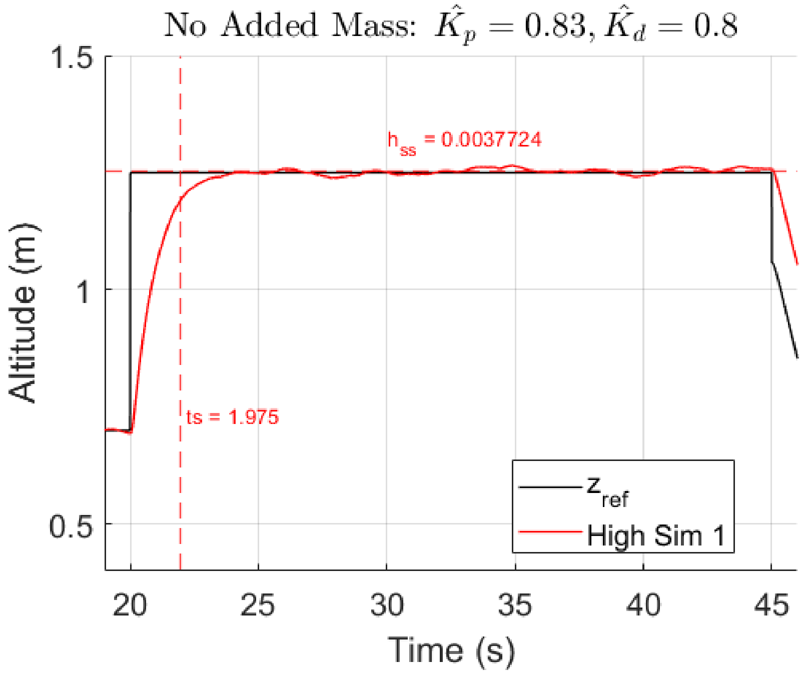
```
avg_hss =

    0.0005    0.0021    0.0038    0.0017    0.0062    0.0042


var_hss =

     0     0     0     0     0     0


avg_kt =

    0.6815    0.6669    0.6489    0.6615    0.6534    0.6536


var_kt =

     0     0     0     0     0     0


kphat =

    1.4300    0.9900    0.8300    1.6900    0.5000    0.1000


kdhat =

    0.2000    0.5000    0.8000    0.6900    0.9900    0.1000


kp =

    0.5245    0.3711    0.3198    0.6387    0.1913    0.0383


kd =

    0.0734    0.1874    0.3082    0.2608    0.3788    0.0383
```

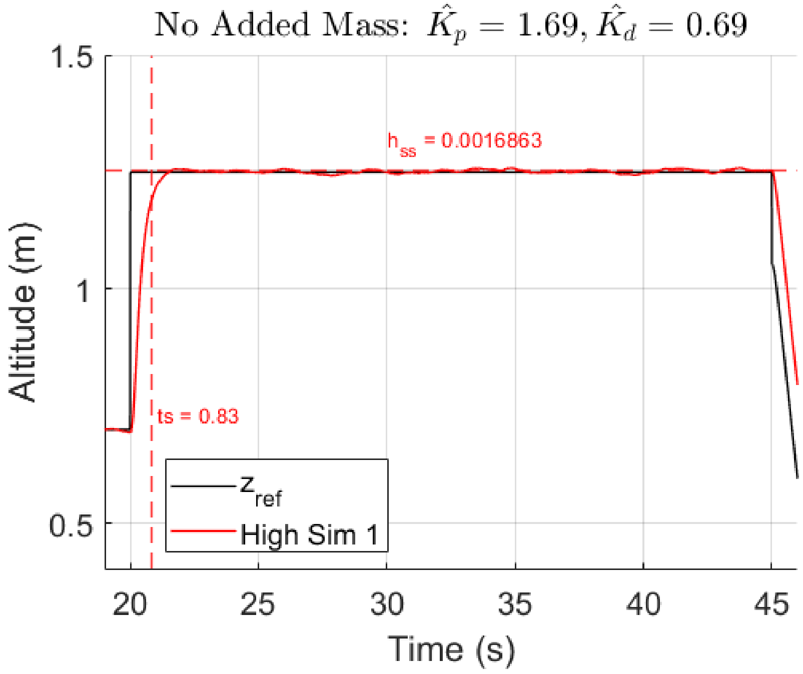


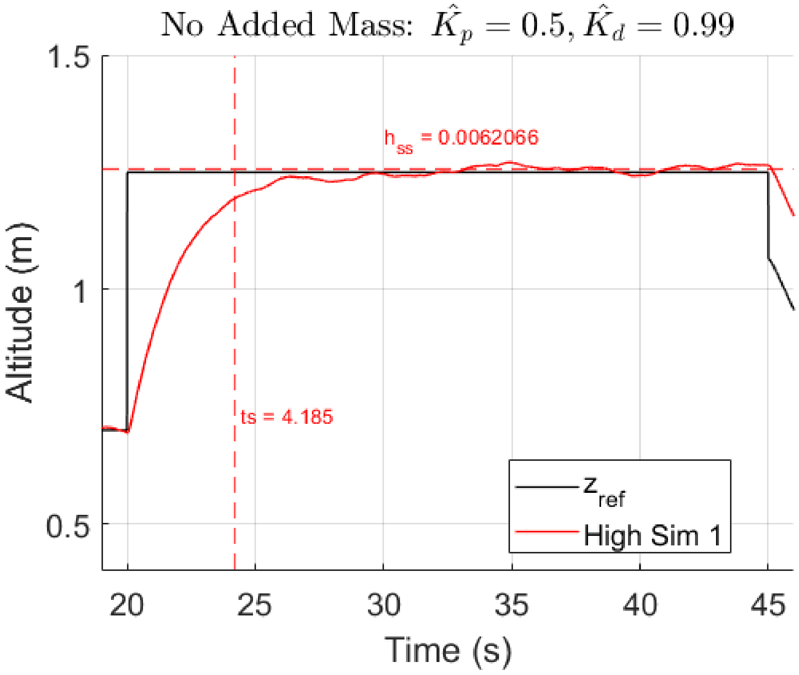
No Added Mass: $\hat{K_p} = 1.43, \hat{K_d} = 0.2$

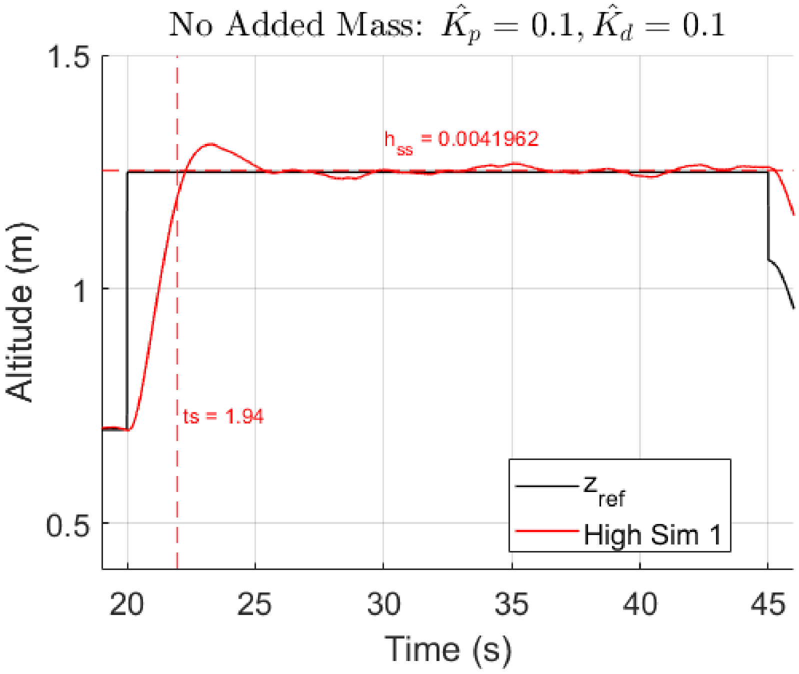


No Added Mass: $\hat{K_p} = 0.99, \hat{K_d} = 0.5$


No Added Mass: $\hat{K}_p = 0.83, \hat{K}_d = 0.8$
h_{ss} = 0.0037724
ts = 1.975
z_{ref}
High Sim 1
Altitude (m)
Time (s)
1.5
1
0.5
20
25
30
35
40
45


No Added Mass: $\hat{K}_p = 1.69, \hat{K}_d = 0.69$
h_{ss} = 0.0016863
ts = 0.83
z_{ref}
High Sim 1
Altitude (m)
Time (s)
1.5
1
0.5
20
25
30
35
40
45





Published with MATLAB® R2020b