Design and Implementation of UAV Autoland Control Law Scheme Utilizing LIDAR for Precise Altitude Determination

A Project Presented to

The Faculty of Aerospace Engineering

San José State University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Aerospace Engineering

by

Frank Sanders

December 2015

ABSTRACT

This paper presents the design and flight testing methodology to land a fixed wing Unmanned Aerial Vehicle (UAV) autonomously using MultiWii hardware and Arduino software. Due to inherent hardware inaccuracies and limitations, a suite of sensors including barometric pressure, GPS, and LIDAR (LIght Detection and Ranging) are used to determine altitude for the increasing precision needs during landing. This autoland control law scheme operates independent of ground support systems and depends solely on aircraft based systems.

Table of Contents

I. Introduction

The subject of autonomously landing an aircraft has been widely studied as a means to prevent pilot error or aid a crew in the event of an emergency. Many approaches to the design problem have been purely theoretical and few have made it to full-scale aircraft. Most of the successful implementations of autonomous landing control schemes have been applied to small unmanned aerial vehicles. The reasons for the use of small UAVs are simple: several iterations of the control scheme can be implemented quickly without extensive robustness studies to ensure pilot safety; the relative cost of hardware is kept low for both the initial investment and for damages/ losses of aircraft; and the collateral damage risks are significantly lower due to lack of an on-board pilot and the relatively low mass of the system.

Autolanding an aircraft is an important challenge to overcome not only because it is one of the last remaining phases of flight to be automated, but also because landing is one of the most dangerous phases of flight. In favorable conditions, landing an aircraft is trivial to any pilot. However, once atmospheric and physiological abnormalities begin to accumulate against the pilot, the landing procedure can easily become an incredibly difficult task. Due to the proximity of the ground during landing, a high level of precision is demanded from the pilot, but is often difficult to attain under adverse conditions.

The autoland control law scheme presented in this paper will alleviate the problems associated with landing an aircraft and, since there is no pilot input, will do away with human piloting error. This system is also entirely contained within the aircraft. Most autoland designs depend on a ground station sending glide-slope data to the aircraft; thus, making the aircraft dependent on airports equipped to send the required data. This aircraft only requires GPS coordinates of the landing zone, making the system independent of specific airports and useful for every airport.

II. Literature Review

A review of the literature on the topic of autonomously landing aircraft has provided many interesting approaches to the design problem and many starting points for the different phases of the landing approach. The most promising designs have come from Sperry Flight Systems, German Aerospace Center (DLR-Oberpfaffenhofen), and Boeing Commercial Airplane Company. Each institution has developed similar systems that are ground dependent, but these three are unique in that they have all flown full scale aircraft under their autoland control laws schemes. Although not all of their designs were successfully tuned and implemented, they all paved the path towards a system that operates completely independent of the ground.

The Boeing Aircraft Company developed and flight tested automatic landing flare control laws for the B-737 and B-747 aircraft. Although their control laws did not handle the approach phase of landing, they did develop a system to control the most critical phase of landing. If the aircraft is not aligned down the runway, has excessive pitch or roll angles or rates, or has too much or too little sink rate, then the landing flare will cause excessive control inputs and may lead to damage of the aircraft or ground items (runway lights, buildings, cars, etc.). Boeing specifically targeted the flare in an effort to eliminate these problems that any pilot may encounter on landing.

The German Aerospace Center designed and flew an aircraft they designated the ATTAS (Advanced Technologies Testing Aircraft System). The project was government funded and was ultimately canceled before being totally problem free. However, their design procedure provides an excellent framework for future autoland development because they clearly defined the bounds on their autoland controller; outlined several different design methodologies (dynamic inversion,

total energy control, Monte-Carlo simulations, etc.); and, most importantly, they discussed the pitfalls they experienced when flight testing the system.

Sperry Flight Systems was able to develop and perfect an autoland control law scheme for NASA's Space Shuttle. Their architecture was unique in that they used a total energy management system since the Shuttle is a glider. Their methodology utilized different assumptions than the other systems, but their method was quite useful in designing a ground independent system since the glide slope determination is well documented and defined.

No cases similar to this autoland control scheme were found while doing research on the topic. Every case encountered depended on the ground in some way. Either the system depended on glide slope projections from ground stations, or the aircraft touched down with a known, and usually quite large, descent rate. No other system uses precision equipment to refine the descent rate as the aircraft approached the runway, and no other system used variable altitude determination systems as the precision requirements change.

III. Problem Definition

The autoland system must be able to guide the aircraft from a position in the sky, establish a glide slope, align itself with the runway until just before touchdown, and arrest the rate of descent to an acceptable level at touchdown. While the problem seems relatively simple, there is an overarching parameter that affects every aspect of the landing and poses the biggest difficulty in the design process: altitude. Altitude is the parameter that is present in every single phase of the landing process, and it is the most difficult parameter to estimate, especially with the changing precision requirements during landing.

Commercially available hardware has its own inherent uncertainties in its measured values. One cannot fully depend on one single method for determining altitude, because the

hardware uncertainties and operational limits of each method has its own pitfalls if depended upon solely during the landing process. For example, the GPS used in this system has an uncertainty of about 10 feet. An error of this magnitude is acceptable if one wants to estimate the altitude in the alignment phase of the landing. However, for the flare phase, if the altitude information is 10 feet higher than the actual altitude, the aircraft will plunge into the ground with an unacceptable sink rate. If the altitude information is 10 feet too low, then the aircraft will arrest its sink rate too high and stall. To remedy the hardware pitfalls, this control law scheme depends on three different altitude determination systems: barometric pressure, GPS, and LIDAR.

IV. Hardware

The aircraft used for the testing the autoland system is a commercially available model of the Cessna 182. This aircraft platform was chosen because of the docile characteristics flight characteristics of the classic Cessna design, the stable behavior of the tricycle gear aircraft on the runway, and the high wing, which affords a voluminous cabin for flight computer and sensor equipment. The physical characteristics of the aircraft are tabulated in Table 1.

The flight computer used in the system is a MultiWii running Arduino software (

Figure 1). The flight computer measures 2in x 2in x 1in and weighs a mere 46 grams. Sensors incorporated into the custom daughterboard include: Accelerometers, Gyros, Barometric Pressure sensor, and SD card slot for data collection. Airspeed, GPS, and the LIDAR-Lite sensor (Figure 3) are connected via I^2C, and finally a Weight on Wheels switch is connected via PWM. The aircraft profile and sensor layout is presented in Figure 2.

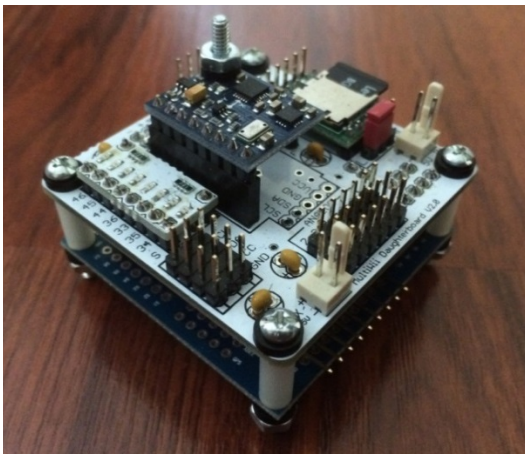


Figure 1. MultiWii Flight Computer

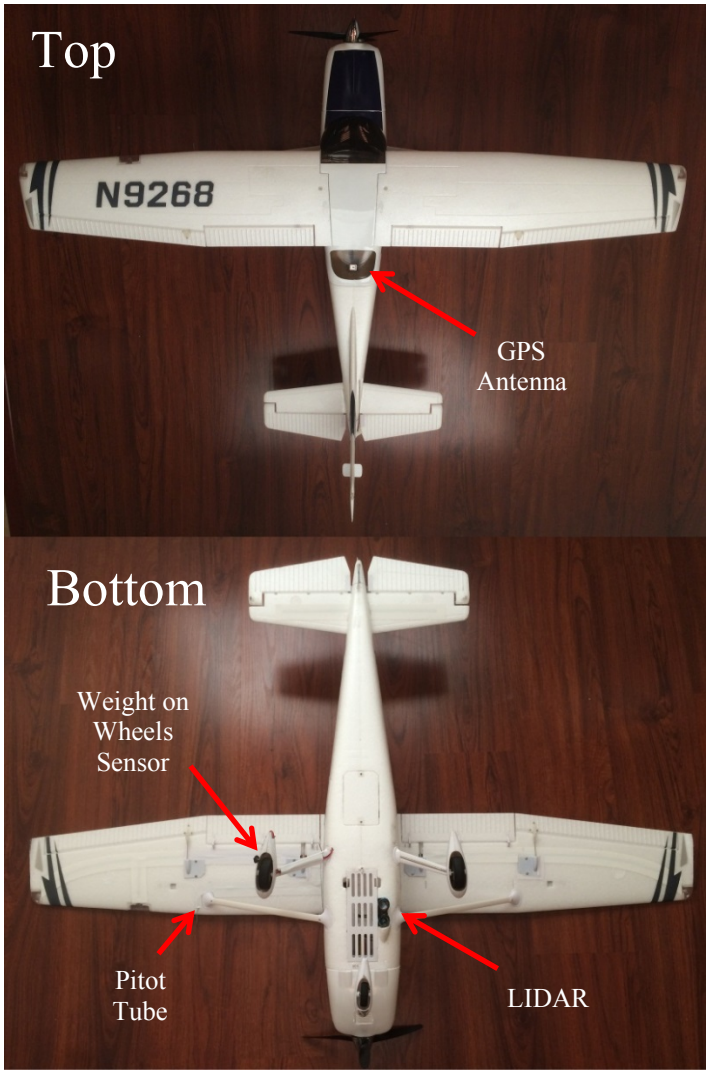


Figure 2. Profile and Sensor Layout of the C182 Test Aircraft

Cessna 182 Physical Characteristics	
Wing Area (ft^2)	2.48
Wing Span (ft)	4.27
Weight (lb)	2.76
Construction	EPO Foam

Table 1. Physical Characteristics of the C182 Aircraft.

Figure 3. Detail of the LIDAR-Lite Installation.

V. System Identification

In order to safely choose gains for the various control laws, MATLAB/Simulink was used to generate the necessary block diagrams for each controller and simulate the aircraft's response to inputs and disturbances. To do this, the transfer functions for C182 needed to be defined in order to properly model the aircraft. Extracting transfer functions from flight test data using the program CIFER (Comprehensive Identification from FrEquency Response) was the method chosen to generate the necessary transfer functions.

CIFER is a computer program developed by the U.S. Army Aeroflightdynamics Directorate at NASA Ames to identify dynamic systems using the response of the system to frequency inputs. For example, if one wants to determine the stability and control derivatives of the two longitudinal modes of an aircraft, the pilot sweeps the elevator across a frequency range that includes the natural frequencies of both the short period and phugoid. The designer then extracts the necessary flight data to generate the transfer function that relates the elevator input to pitch rate. CIFER's relative simplicity and accuracy makes it the better method for determining the transfer functions of a flyable aircraft, as opposed to the lengthy and assumption-filled numerical approximation process based on the aircraft's geometry.

Since rates and inputs are collected with the flight computer on board the C182, the rates due to input transfer functions were created. Angles were not collected during the frequency response flying. However, if angles are necessary for the control laws, then the aircraft integrates the gyro and accelerometer signals to provide the computer with the Euler angle. CIFER also has the ability to provide Bode plots and a coherence plot (quantifying the accuracy of the identification) for given data. Figure 4 and Figure 5 provide Bode and coherence data for the roll and pitch data sets, respectively. Coherence is the metric relating the attributability of the response to the input, ranging from 0 to 1. For example, if one were to investigate the roll rate due to elevator response of the C182, the coherence would be 0 since there is no coupling between roll and elevator. However, as shown in Figure 4 and Figure 5, pitch rate due to elevator and roll rate due to elevator have coherence values of approximately 1 because those inputs are directly related to those responses. Coherence values greater than 0.6 are considered acceptable for system identification, and the C182 responses show excellent coherence across a broad frequency range.

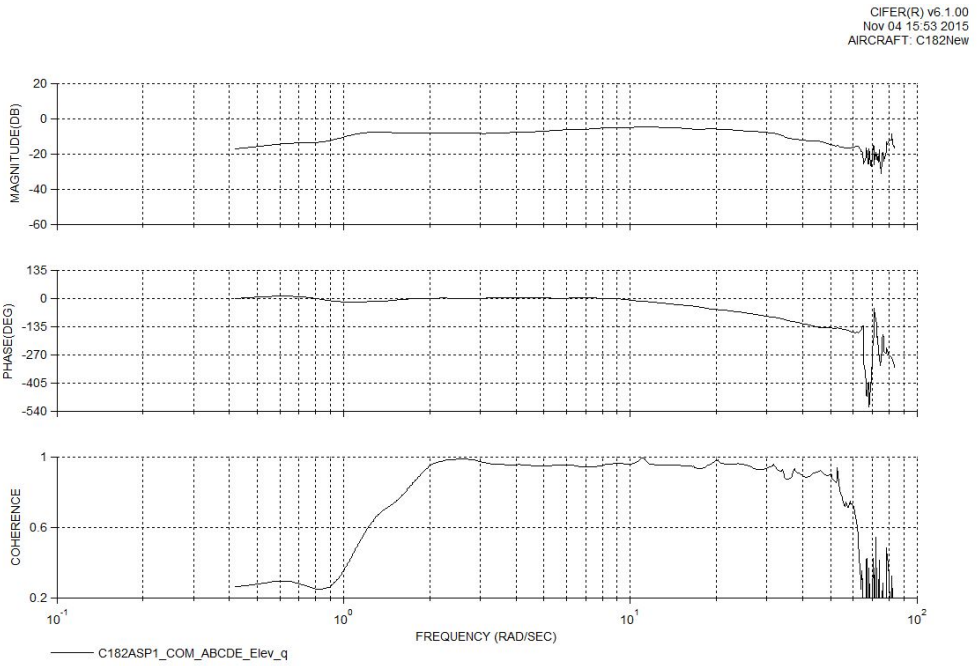


Figure 4. Bode and Coherence Plots for q/δe

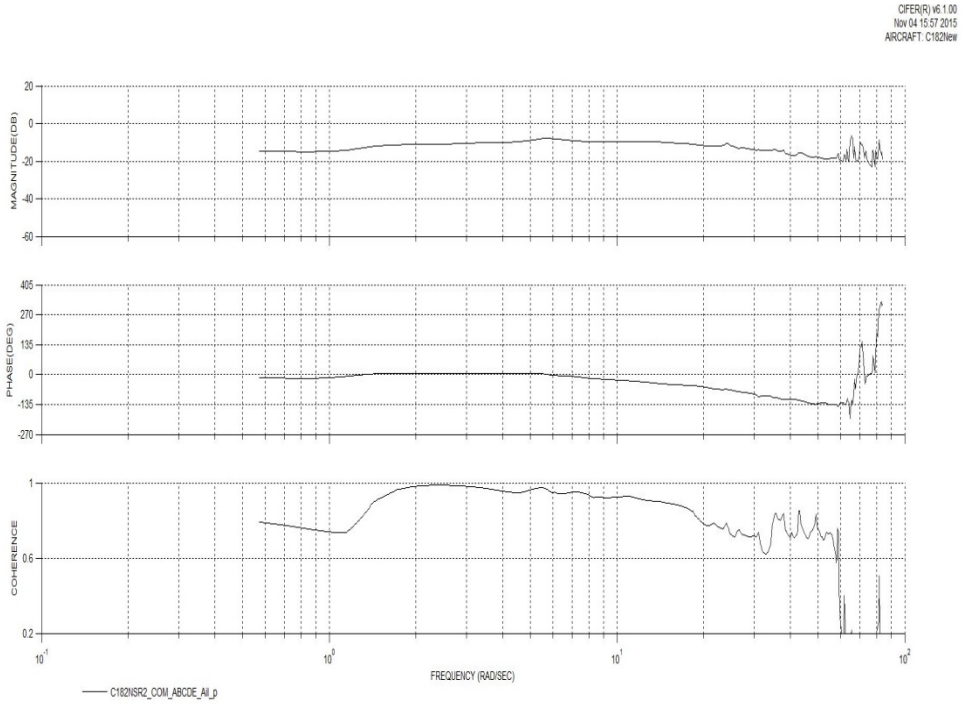


Figure 5. Bode and Coherence Plots for p/δa

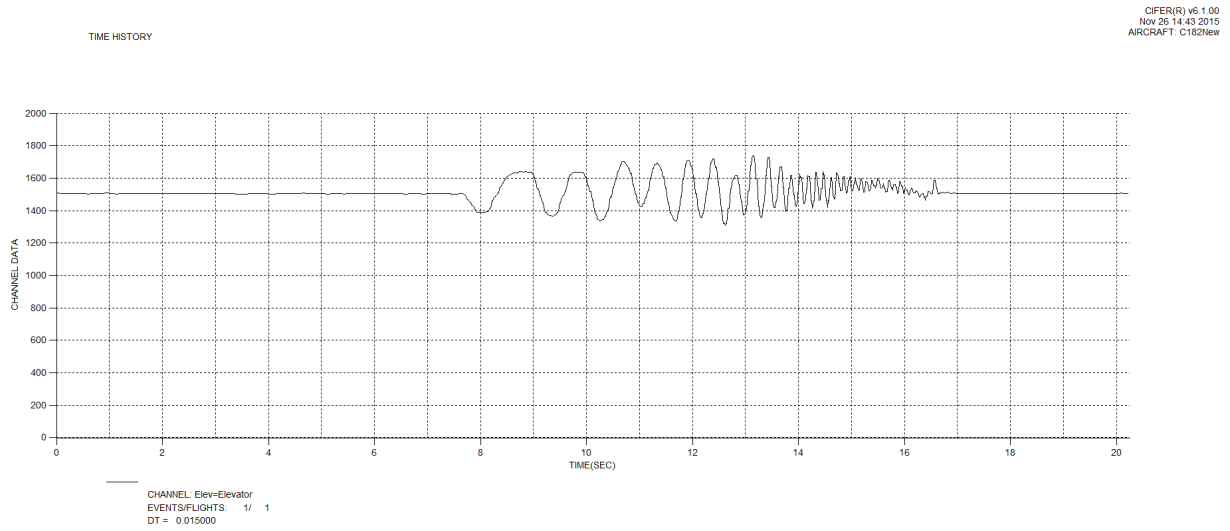


Figure 6. Typical Elevator Input Sweep

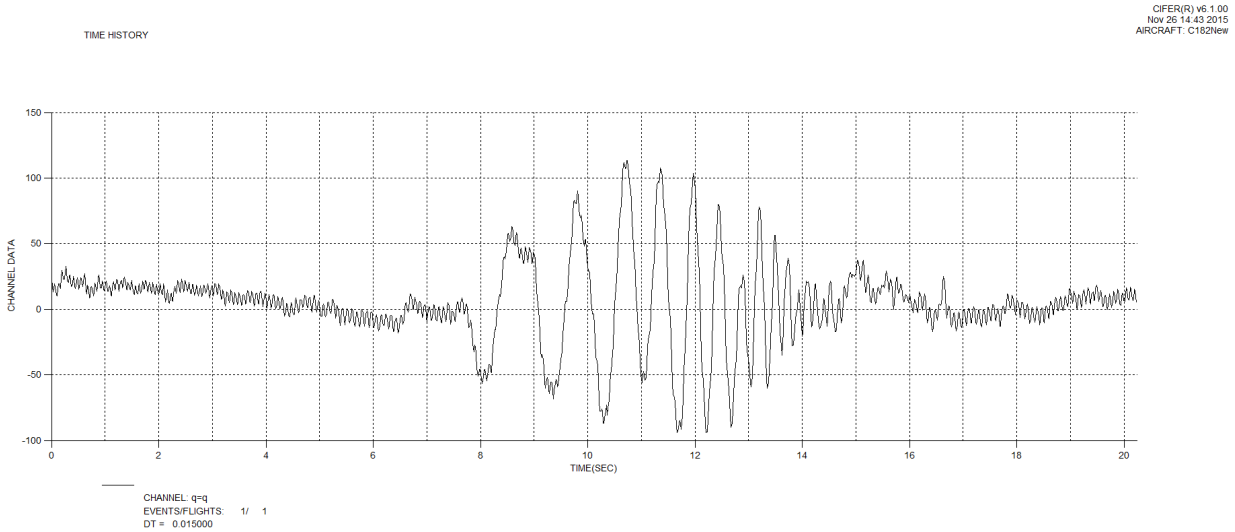


Figure 7. Typical Pitch Rate Response (Actual Response to Figure 5 Input).

The flight data was then used with the CIFER tool NAVFIT. Given the flight data and orders of magnitude of the numerator and denominator, NAVFIT will generate a transfer function that matches the flight data as closely as possible. NAVFIT was used to determine transfer functions for the pitch and roll axes.

For system identification purposes, the short period mode was focused upon more heavily while the phugoid mode was neglected. The phugoid mode is difficult to capture because the low

frequency and time necessary to capture the dynamics exceed the limitations of UAV flying. The aircraft spends little time in steady state flight during the landing sequence, so the phugoid mode transfer functions were completely omitted from the data analysis. The transfer function for $\frac{q(s)}{\delta_e(s)}$ using NAVFIT is provided in the following form:

$$\frac{q(s)}{\delta_e(s)} = \frac{(U_1 M_{\delta_e} + Z_{\delta_e} M_{\dot{\alpha}})s + (M_\alpha Z_{\delta_e} - Z_\alpha M_{\delta_e})}{U_1\left\{s^2 - \left(M_q + M_{\dot{\alpha}} + \frac{Z_\alpha}{U_1}\right)s + \left(M_q \frac{Z_\alpha}{U_1} - M_\alpha\right)\right\}} \quad (1)$$

A first order over second order transfer function was then defined in NAVFIT, and

$$\frac{q(s)}{\delta_e(s)} = \frac{0.421926s + 363.924}{s^2 + 27.119s + 852.541} \quad (2)$$

The Bode magnitude and phase plots are presented in Figure 8 and Figure 9 respectfully. The dashed lines represent the transfer function model fit, while the solid line is the flight data.

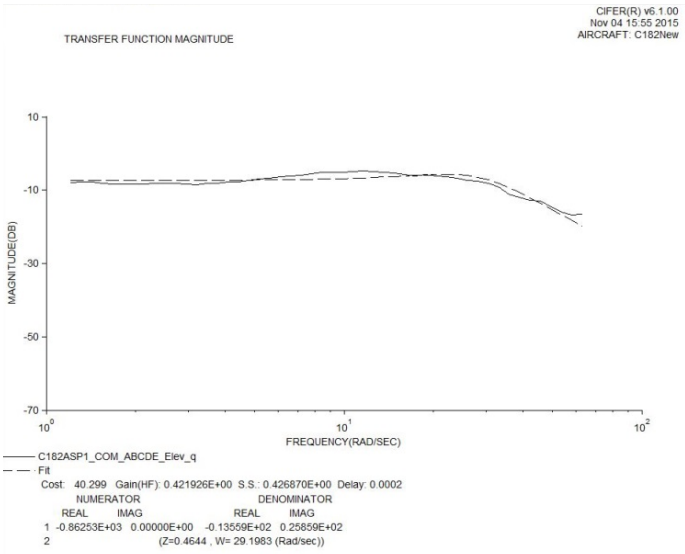


Figure 8. Magnitude Plot for the Short Period Approximation Transfer Function Model Compared to the Flight Data.

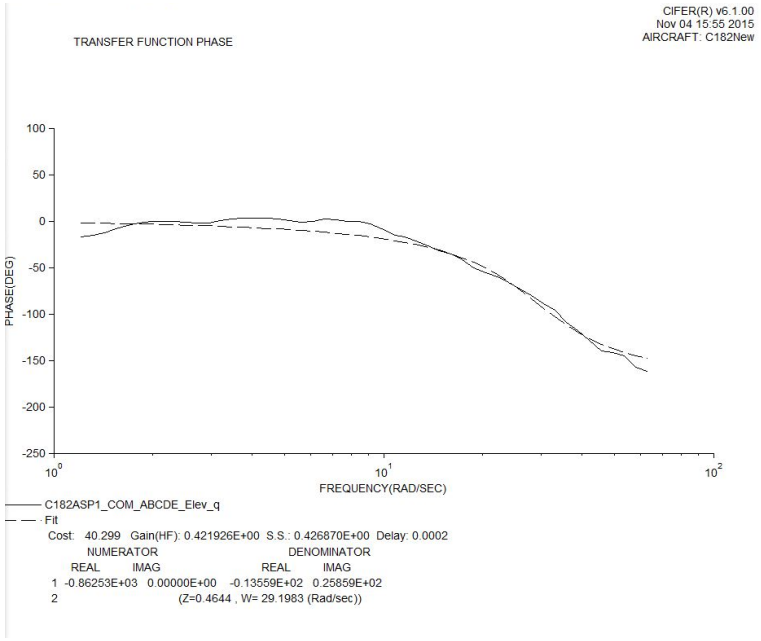


Figure 9. Phase Plot for the Short Period Mode Approximation Transfer Function Model Compared to the Flight Data.

Due to the simplicity of the roll controllers, the roll mode approximation was used to generate a transfer function model in NAVFIT, and

$$\frac{p(s)}{\delta_a(s)} = \frac{L_{\delta_a}}{s - L_p} \tag{3}$$

A zero order over first order transfer function was then defined in NAVFIT, and

$$\frac{p(s)}{\delta_a(s)} = \frac{11.9795}{s - 37.2144} \tag{4}$$

The bode magnitude and phase plots are presented in Figure 10 and Figure 11 respectfully. The dashed lines represent the transfer function model fit, while the solid line is the flight data.

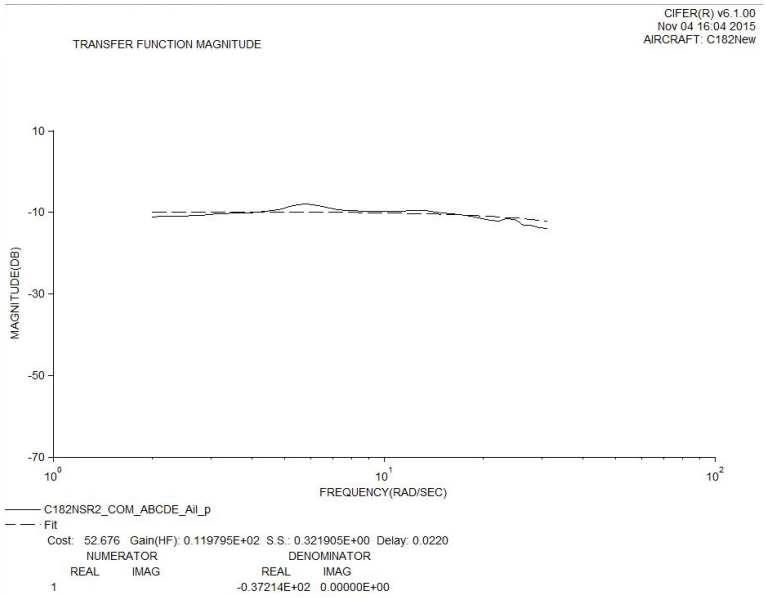


Figure 10. Magnitude Plot for the Roll Approximation Transfer Function Model Compared to the Flight Data.

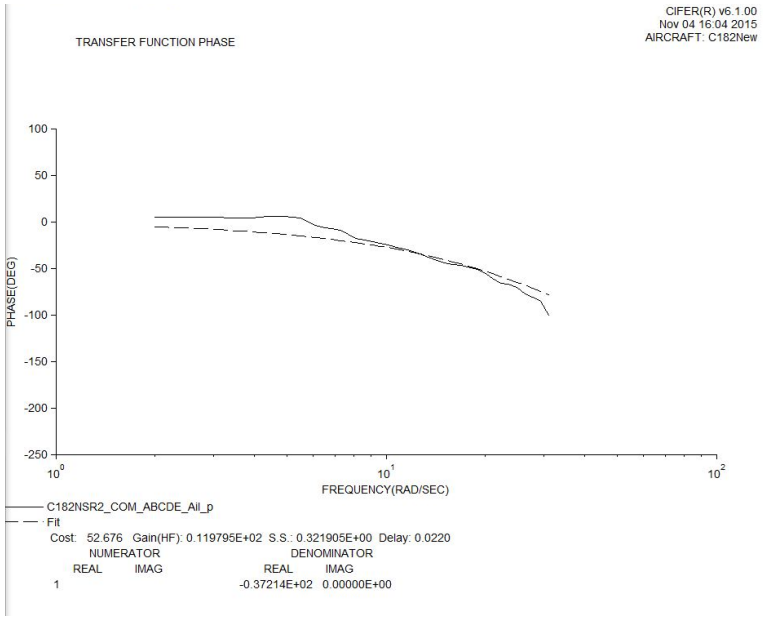


Figure 11. Phase Plot for the Roll Approximation Transfer Function Compared to the Flight Data.

VI. System Architecture

There are three main phases in the landing sequence: Approach, Descent, and Flare. Each phase has its own specific altitude precision requirements, and each phase's control laws and systems for altitude determination will be discussed in detail.

A. Approach

The first phase of the landing sequence is the approach. The aircraft needs to be in steady state flight prior to intersecting the initial position (IP) where the next phase is initiated. The aircraft is aligned with the runway approximately 600 feet away from the approach end of the runway and at approximate altitude of 50 feet above the ground when the controller is engaged. The aircraft uses GPS to fly through one waypoint, approximately 400 feet away from the approach end of the runway, before arriving at the IP in steady state flight. Since the altitude and alignment with the runway heading of the aircraft is arbitrary when the controller is engaged, it is necessary to properly align the aircraft and reach a steady state altitude at an appropriate airspeed for intersection with the first waypoint. The control laws of this phase are altitude hold, heading hold, and airspeed hold. The overview for this phase is presented in Figure 12.

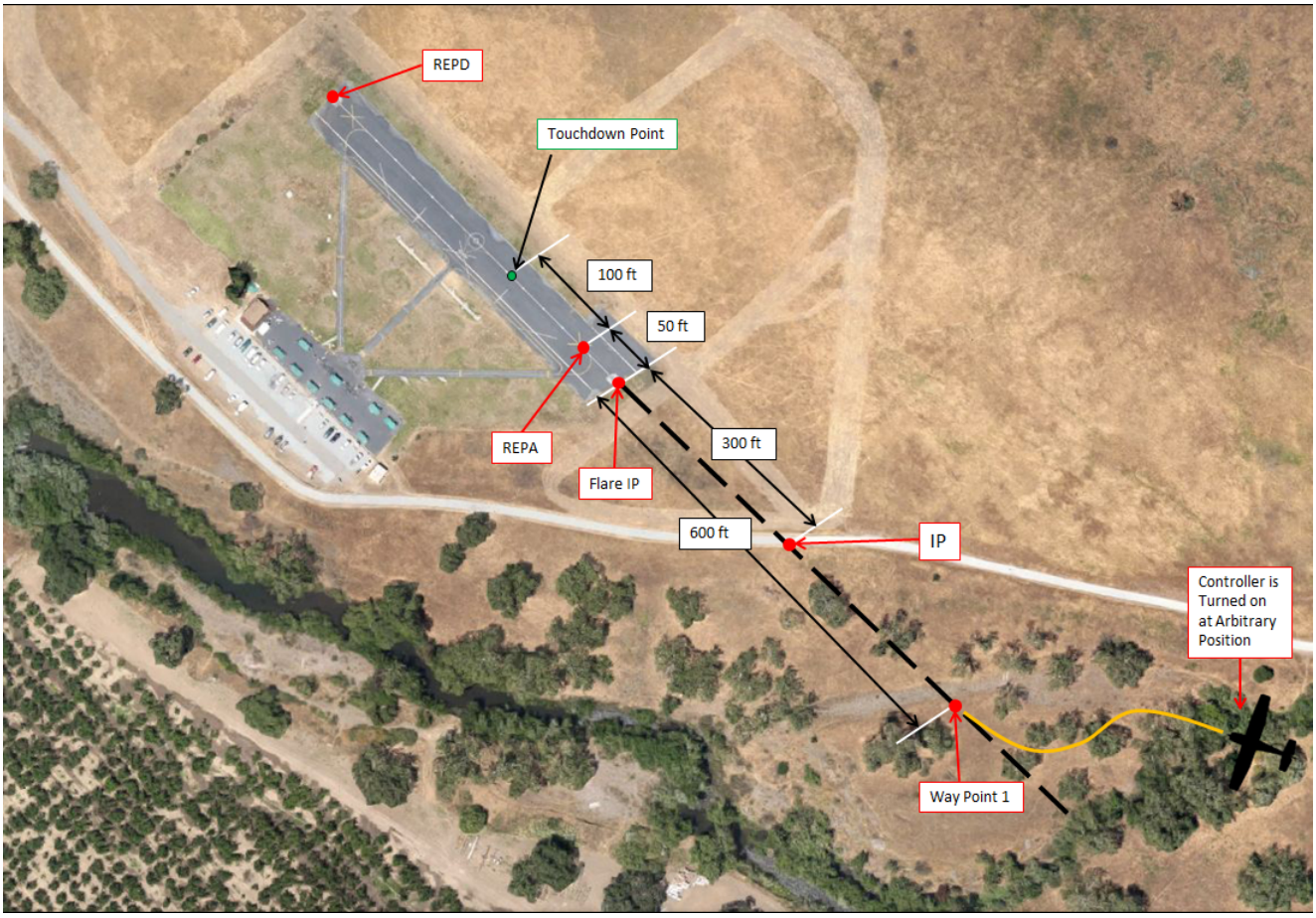


Figure 12. Overview of Approach Phase.

For altitude determination in this phase, the system uses a combination of the barometric pressure altitude and GPS altitude. The barometric pressure altitude is calculated using methodology outlined by Barber, Griffiths, McLain, and Beard in Equation $hag = \frac{P_{ref} - P_{actual}}{\rho g}$, (5 [2]. A reference pressure is taken before flight and defined within the computer in order to generate a height above the ground. The barometric altitude and GPS altitude are then averaged and fed to the flight controller.

$$hag = \frac{P_{ref} - P_{actual}}{\rho g}, \tag{5}$$

B. Descent

When the GPS location data of the aircraft falls within the bounds of the IP defined in the code, the aircraft initiates the descent phase. During the descent phase, the aircraft reduces power and begins a descent with a 9.5 degree flight path angle towards the Runway End Point Approach (REPA) located 50 feet inside the physical end of the runway. Upon reaching the IP, the aircraft compu

$$\boldsymbol{Alt_{desired} = \sin(Glideslope) * Dist_{REPA}} \tag{6}$$

This method provides a more robust descent command compared to commanding a pitch angle or rate of descent. If the aircraft is perturbed vertically, the airspeed hold ensures the total energy is maintained and the aircraft does not overspeed or stall. Since the distance value is in an inertial frame, longitudinal wind perturbations to the aircraft have no effect on the undershoot or overshoot of the touchdown point. The control laws in this phase are identical to the control laws in the previous phase.

Altitude information is still reliant on the combined GPS and barometric altitude until the aircraft's altitude is within range of the LIDAR; approximately 15 feet. At this height, the LIDAR will provide the sole altitude information to the flight computer.

C. Flare

Once the aircraft reaches the flare IP at the physical end of the runway and 5.5 feet high, based on the LIDAR measurement, the aircraft begins the flare routine. The flare routine is based on Roskam's method (Figure 13) in which a touchdown point, flare point, and time constant are defined. With these parameters, the descent rate at touchdown can be derived (Appendix A).

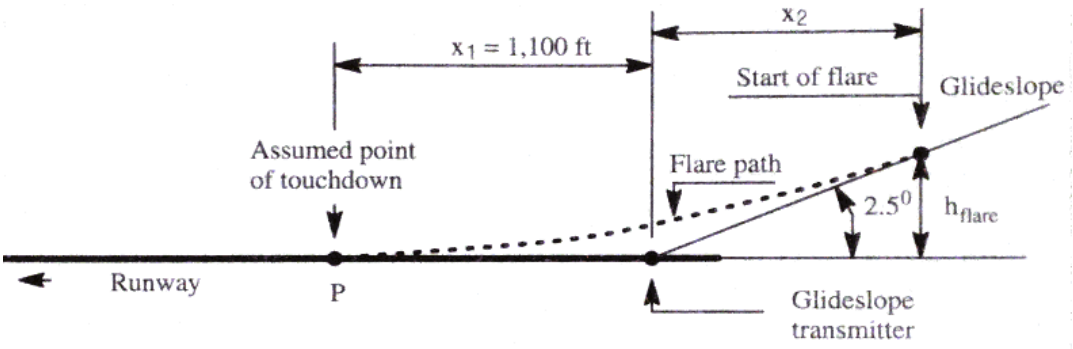


Figure 13. Flare Routine Outlined by Roskam [3]

Due to obstacles at the end of the runway where flights are conducted, a glideslope of 9.5 degrees is used, the touchdown point is defined as 100 feet after the flare initialization, and a time constant of 4sec was used for this aircraft. The flight control laws of this phase are heading hold, roll hold, descent rate command, and airspeed hold. It is important to note that the heading hold during flare is not controlled by the ailerons, but rather the rudder. The wings maintain level attitude while small changes in heading are corrected by the rudder.

D. Additional Modes

Two more features of the autoland system require discussion: the failure mode and the touchdown mode. The failure mode is enabled when the aircraft loses GPS signal, which is not uncommon for this hardware. If the aircraft should lose GPS signal, the aircraft commands a pitch angle of 5 degrees nose up, levels the wings, and commands 75% throttle. The premise of

the failure mode is to direct the aircraft away from the runway without stalling in order to give the pilot time to switch off the flight computer. The touchdown mode is enabled when the aircraft physically touches the runway and trips the Weight on Wheels switch, located on the left main landing gear (Figure 14) inside the wheel fairing. Once the switch is tripped, the aircraft commands zero throttle and nose down elevator deflection (to put further weight on the main landing gear and prevent bouncing), and turns off the roll angle controller.

Figure 14. Weight on Wheels Switch Nestled Inside the Left Main Landing Gear Fairing.

VII. Control Law Design

MATLAB/ Simulink was used to design the control laws for the aircraft because the user is able to input the transfer functions for a given system in the form of CIFER's NAVFIT output. With the control laws written in appropriate block diagram form, starting gains for the various controllers could then be selected with relative confidence in the stability of the system once the control laws had been coded into the flight computer.

A. Altitude Hold Design

A simple PID controller was chosen for controlling the altitude of the aircraft throughout the flight. While the frequency response system identification generated the relationship between pitch rate and change in elevator deflection, the $\frac{q(s)}{\delta_e(s)}$ transfer function alone is insufficient for holding altitude. The altitude to pitch angle transfer function, $\frac{h(s)}{\theta(s)}$, is well defined by most textbooks. However, the input to the flight controller is written in terms of elevator deflection. This discrepancy is easily remedied by multiplying the integrated pitch rate to elevator deflection transfer function $\frac{q(s)}{s\delta_e(s)}$. The altitude to pitch angle transfer function was then analytically derived

u

s

i

n

$$\frac{\boldsymbol{h(s)}}{\boldsymbol{\theta(s)}} = \frac{\boldsymbol{U_1}}{\boldsymbol{s}}\left[\frac{-\boldsymbol{Z_\alpha}/\boldsymbol{U_1}}{\boldsymbol{s}-\boldsymbol{Z_\alpha}/\boldsymbol{U_1}}\right] \tag{7}$$

g While U_1, the steady state velocity, was known, the Z-force due to angle of attack, Z_α,

w

t a

h s

e

u

p n

i k

t n

e o

h w

n

r

$$-\frac{\boldsymbol{Z_\alpha}}{\boldsymbol{M_{\delta_e}}} = \mathbf{1.59651} \tag{8}$$

The block diagram for the final altitude hold control law is presented Figure 15, and the controller's response to a 20 foot step input is presented in Figure 16. The system has mild overshoot with a rather extreme step and, more importantly, settles in 9 seconds.

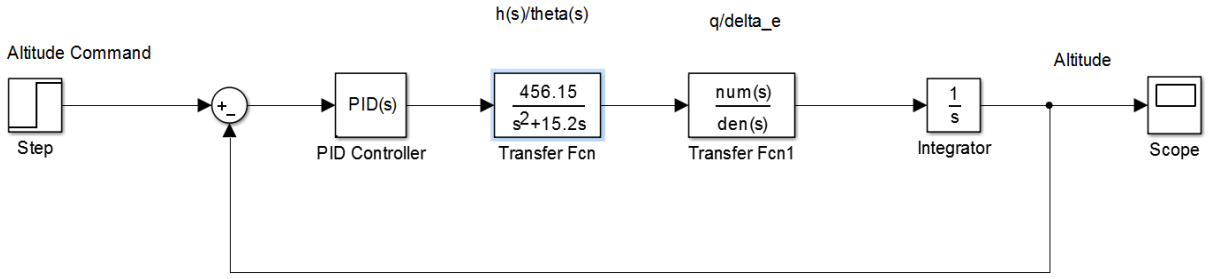


Figure 15. Simulink Block Diagram for PID Altitude Hold Controller.

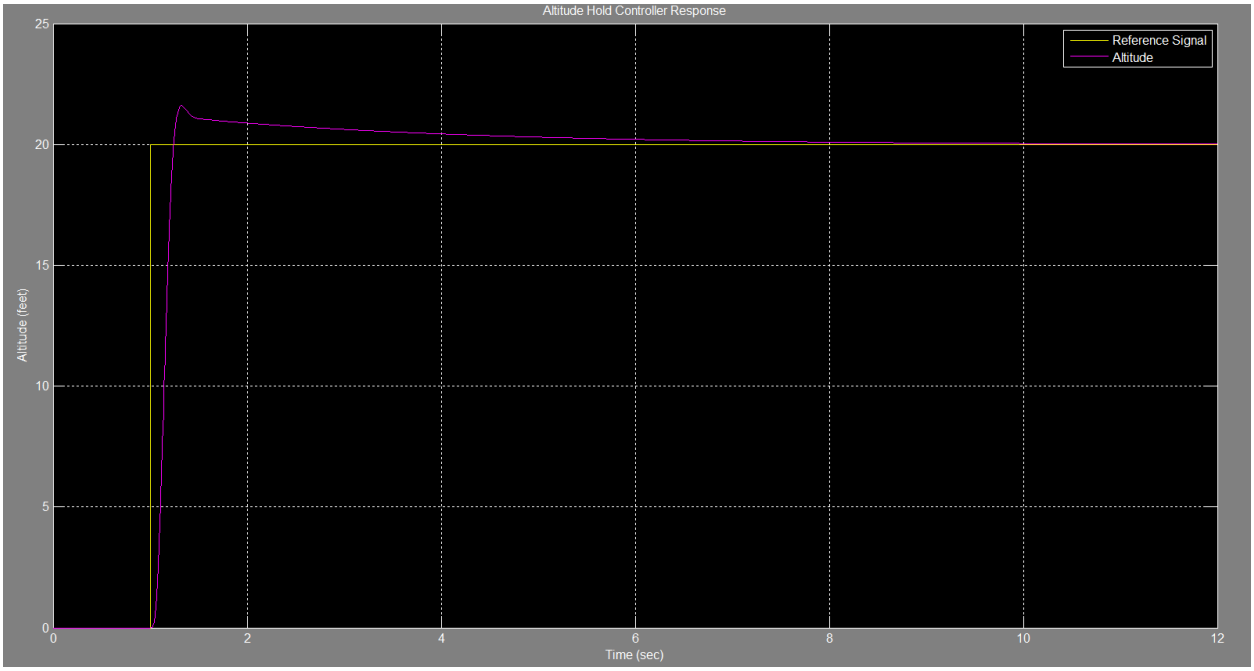


Figure 16. Altitude Hold PID Controller Response to 20 Foot Step Input.

B. Autothrottle Design

The autothrottle controller design is the simplest of the control laws. The controller uses airspeed as the input and multiplies the error by a simple gain. Since the airspeed is an incredibly noisy signal, the raw airspeed signal is run through a low pass filter and the filtered airspeed is then fed back to the controller. Attempts were made to identify the transfer function that relates airspeed to change in throttle, however, the noisy airspeed data proved unusable for CIFER. Therefore, the Simulink block diagram was not used for design purposes, rather the block diagram presented in Figure 17 serves as a visual reference for how the controller was implemented in the code.

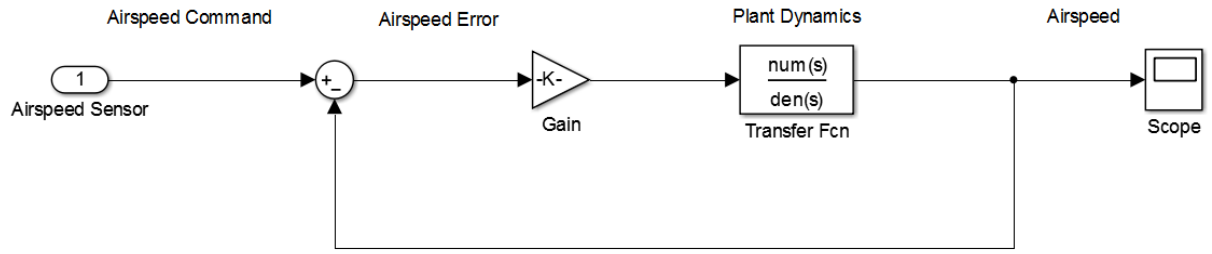


Figure 17. Autothrottle Reference Block Diagram.

C. Heading Hold Controller Design

The heading hold controller is based on Roskam's method [3] that uses both roll and yaw angle feedback (where yaw attitude is the heading angle of the aircraft). In this controller, the transfer function $\frac{p(s)}{\delta_a(s)}$ is used in the block diagram to represent the aircraft dynamics. However,

the control scheme calls for roll angle, so the roll rate is integrated before it is fed back to the controller. GPS ground course is used for the yaw angle feedback. The block diagram for this controller is presented in Figure 18 and the system's response to a 90 degree heading step input is presented in Figure 19. The system exhibits excellent behavior with a rapid rise time and reasonable setting time at only 2.3 seconds.

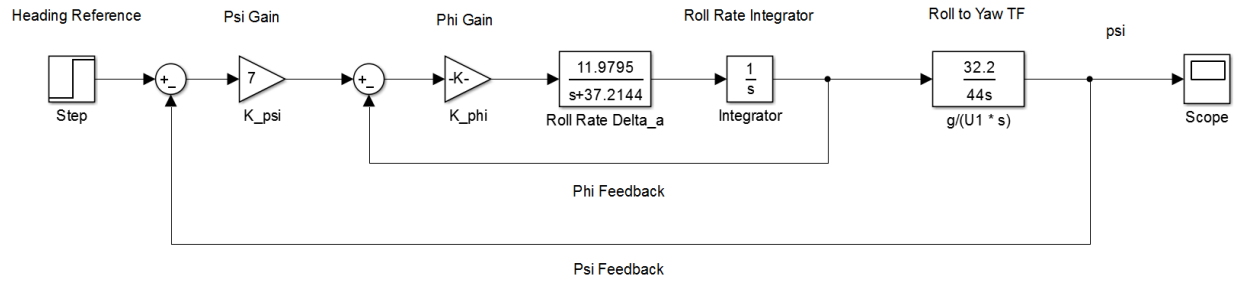


Figure 18. Simulink Block Diagram of the Heading Hold Controller.

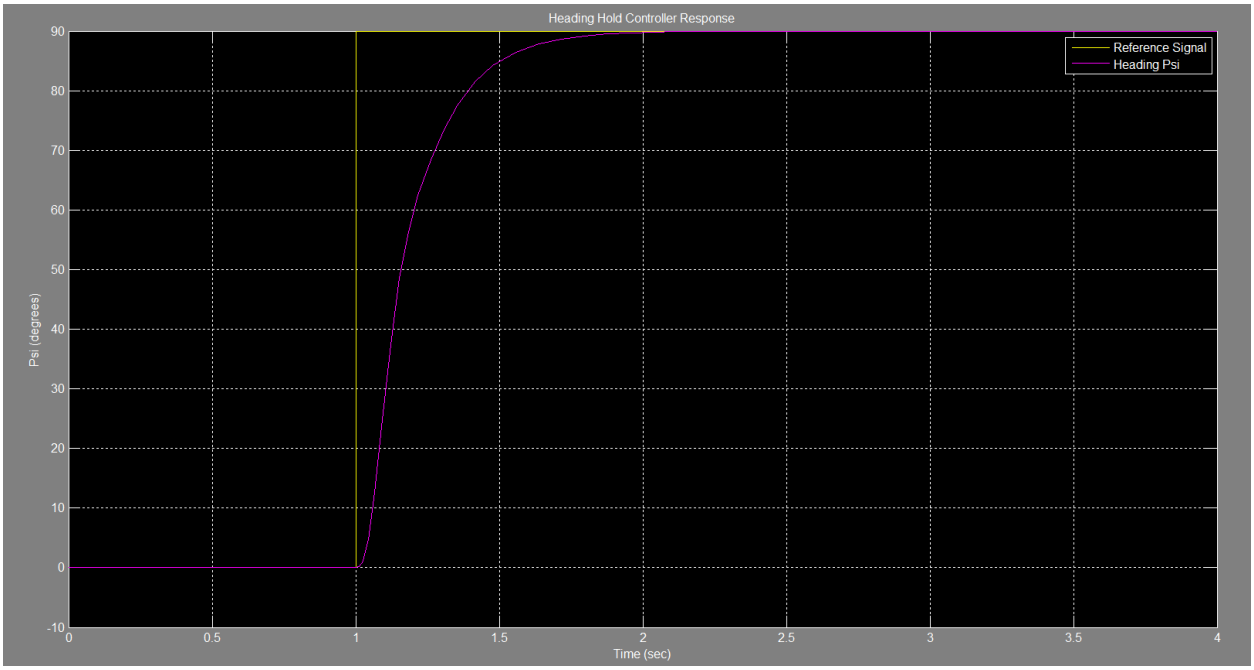


Figure 19. Heading Hold Controller Response to 90 Degree Heading Step Input.

D. Roll Angle Hold Controller Design

In order to hold the wings level, a roll angle hold with rate feedback controller was designed based on Roskam's method [3]. The block diagram and system response are presented in Figure 20 and Figure 21 respectively. This controller has desirable performance with no overshoot and a settling time of 1.4 seconds.

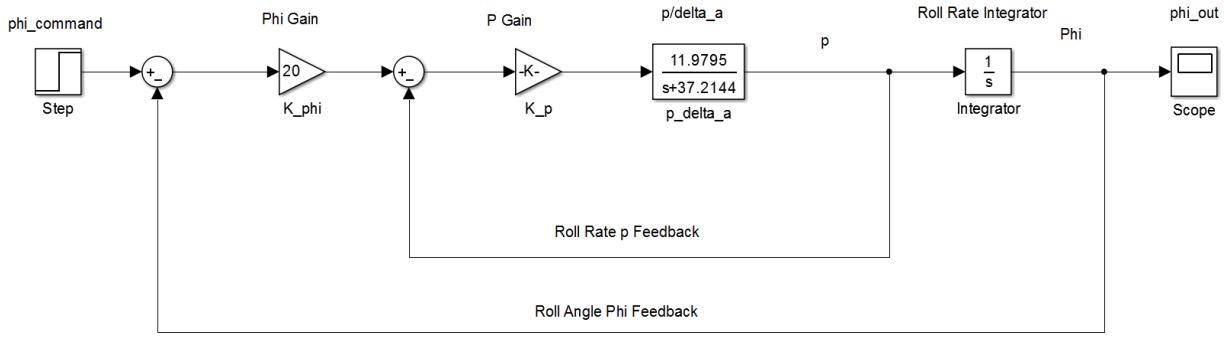


Figure 20. Simulink Block Diagram for Roll Angle Hold with Rate Feedback Controller.

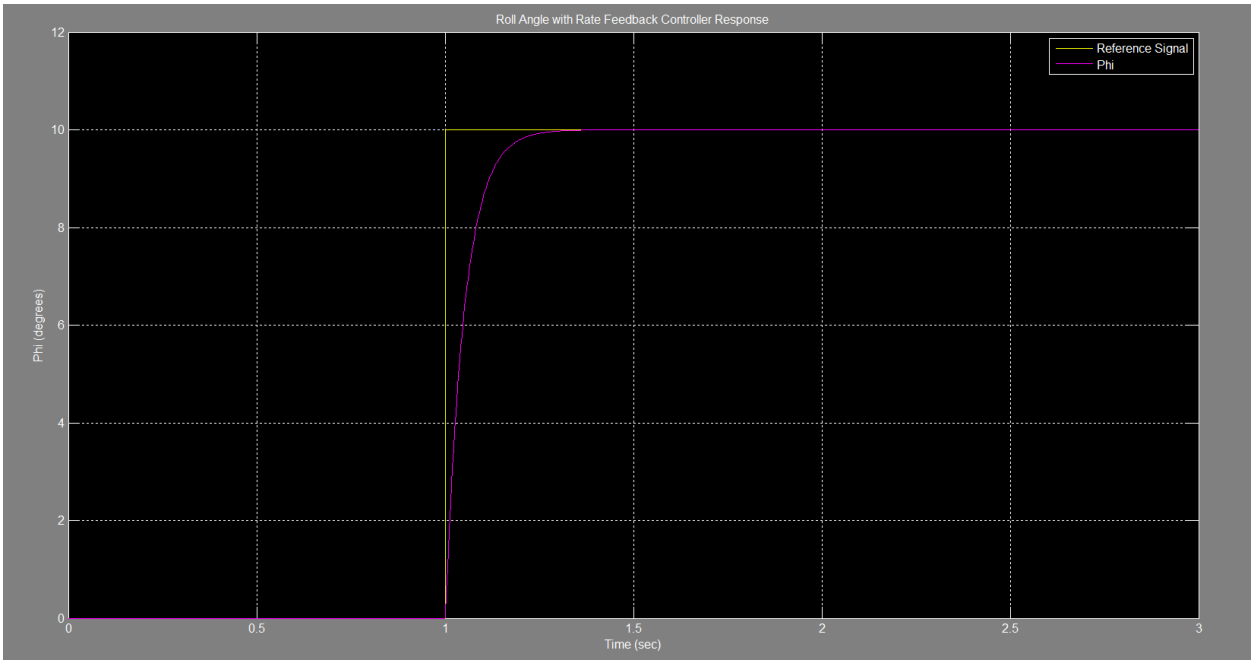


Figure 21. Roll Angle Hold with Rate Feedback Controller Response to 10 Degree Step Input.

E. Pitch Angle Hold Controller Design

The final control law is also designed using Roskam methodology and is a pitch angle hold with rate feedback controller. The block diagram is presented in Figure 21. This controller has great performance with no overshoot and a settling time of 1.3 seconds. However, there are slight oscillations while rising, but given the time over which the oscillations occur their actual influence on the aircraft can be considered negligible.

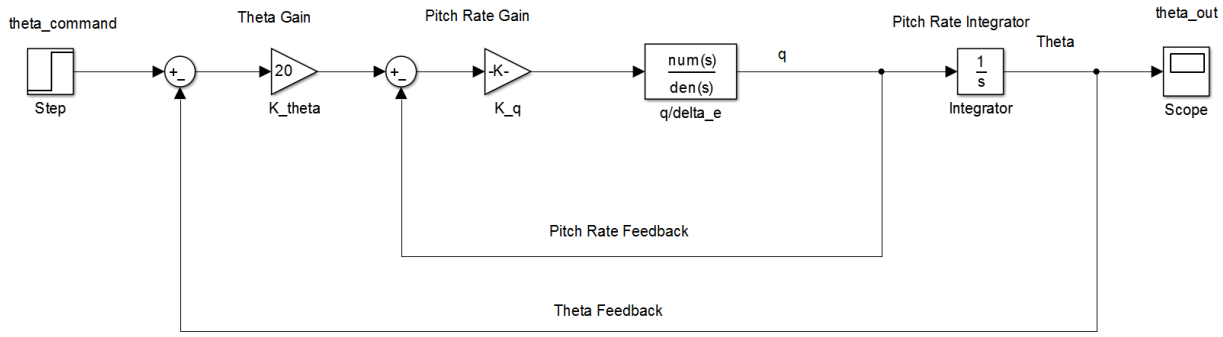


Figure 22. Simulink Block Diagram of the Pitch Angle Hold with Rate Feedback Controller.

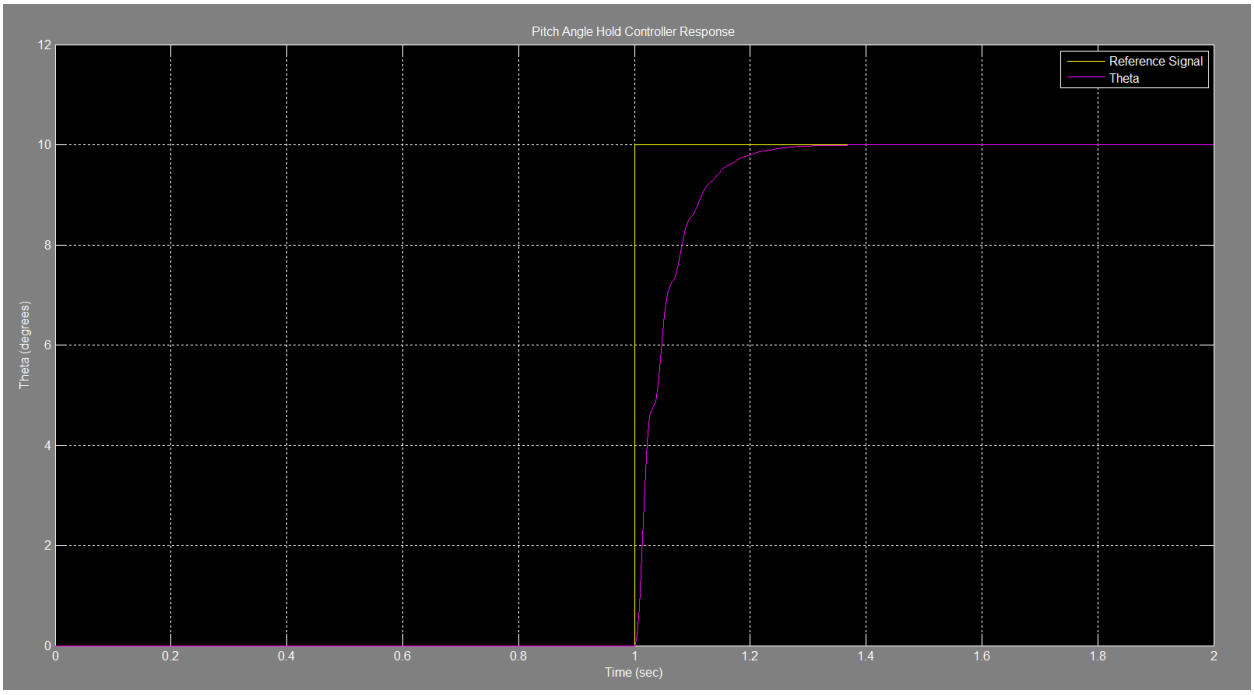


Figure 23. Pitch Angle Hold with Rate Feedback Controller Response to a 10 Degree Step Input.

VIII. Code Structure

The flight code is broken up into the four main control categories: Throttle, Aileron, Elevator, and Rudder. Each section is then broken down into the phases of the landing: approach, descent, and flare. The sections are broken apart using *if statements* to switch between control laws as each phase is ended or begun. At any point during the autoland sequence three out of four control laws are running, since rudder commands are reserved for the flare.

Throttle, for example, is broken into four *if statements*. The first *if statement* is an autothrottle control law that is commanding a cruise velocity, 40 feet per second. If the controller is active and the GPS coordinates of the aircraft are outside of the first waypoint, then the aircraft is holding a constant cruise velocity. Once the parameters of the next *if statement* (GPS is within the bounds of the IP) are satisfied, the controller switches to the autothrottle commanding descent velocity. Once the next *if statement* is satisfied (LIDAR altitude is 5.5 feet), then the controller switches to a constant throttle value of 10% in order to lessen the deceleration rate (as opposed to rapidly decelerating while gliding). Then, when the Weight on Wheels switch is tripped, the throttle command is set to zero. A sample of the throttle code is presented in Figure 24.

```
if(enableClaw && lidarData.distance <= 5.5)
{
    ThrottleOutput = 1150;
}

if(!WoWGain.WoW)
{
    //WEIGHT ON WHEELS SHUTS OFF THROTTLE COMPLETELY
    ThrottleOutput = 1000;
}

if(enableClaw && gpsLongZero && gpsLatZero)
{
    //LOSS OF GPS FIX CODE
    ThrottleOutput = 1750;
}
```

Figure 24. Sample of Throttle Flight Code Showing Parameters Used to Switch Between Control Laws.

It is necessary to outline the process by which the Simulink block diagrams were converted to usable commands in the MultiWii flight computer, for it is an important process that is seldom mentioned. The signal heading into the transfer function $\frac{p(s)}{\delta_a(s)}$ (outlined in Figure 25) is the aileron command, which the MultiWii uses as a control variable in the code. This signal was written in terms of the relevant variables within the flight code and converted to a usable output to the servo. The code section, Figure 26, is the multiple loop closure with the feedback signals and gains written according to the block diagram in Figure 25.

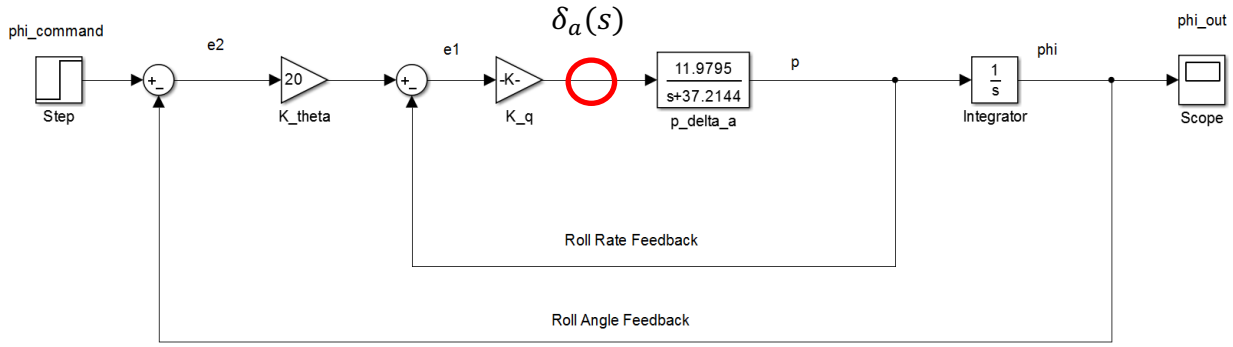


Figure 25. Roll Angle Hold with Rate Feedback Block Diagram with Aileron Command Outlined.

```
int phi_command = 0;
int angle_rate_ail = -(K_phi*(K_p*(phi_command-inertial.roll))- ((float)inertial.gx * GyroResFloat));//Angle hold with rate feedback

AileronOutput = constrain((angle_rate_ail+1500), ServoMinMillis, ServoMaxMillis)+ CommandedAileron-1500;//ANGLE HOLD WITH RATE FEEDBACK
```

Figure 26. Code Snippet for Roll Angle Hold with Rate Feedback.

The middle line of Figure 26, angle_rate_ail, is the line representing the highlighted portion of Figure 25. The AileronOutput line is the data as sent to the servo. In that line of code the variable angle_rate_ail is added to 1500 to center the error value within the servos' pulse width range, then constrained between the upper and lower limits of the servos' range to protect the servos from signals that could potentially cause damage to the servo. Each control law is structured in the code in this fashion.

IX. Results and Discussion

The autoland control system was tested successfully. Some portions of the system worked as expected, while other parts required and continue to require tuning and development:

1. Many of the gains required moderate to extensive tuning. The autopilot gains given by MATLAB/Simulink were slightly higher than the final, tuned gains derived from flight testing. For example, not only were the initial controller gains too large for the roll controller, but the ratio between the angle and rate gains was also different from to model predictions. These same kinds of discrepancies were found with the other controllers and their respective gains. Despite the differences in gain values, the aircraft was not unstable with the Simulink autopilot gains. The controllers simply did not perform as rapidly or accurately as predicted. For example, the roll hold controller would arrest roll rate perturbations satisfactorily, however, if the roll angle was less than 5 degrees the controller could not level the wings and the aircraft would gently circle. Increasing the angle gain and reducing the rate gain eliminated the problem experienced with small angles while keeping the aircraft stable. The discrepancy in gain values is most likely due to the identified model. CIFER does have its complexities that can result in errors in the model that and these errors are eventually rectified with user experience. Identifying the C182 aircraft using CIFER was definitely the best option for this project and the program definitely supplied a great starting point for control law design.

2. The most critical phase, the flare, was also successful after extensive tuning. The aircraft slowed its rate of descent proportional to its height above the ground as expected. It was apparent during initial testing that holding one airspeed all the way to touchdown was improper as the energy at touchdown was too high. When the aircraft touched down with airspeed hold, the Weight on Wheels switch would engage and cause the aircraft to bounce back into the air

instead of placing more weight on the wheels. So in the later stages of flight testing a throttle retard law was written in the code to smooth out the aircraft's energy as it neared the ground. When the aircraft is four feet above the runway, the throttle retards proportional to the aircraft's height. This law ensures that energy is bled at a reasonable rate as the aircraft nears the runway, as opposed to chopping the throttle completely and having drag rapidly bleed energy at an uncontrolled rate. When landing with the autoland control laws and the throttle retard law enabled, the Weight on Wheels switch operated perfectly as expected. As soon as the wheels touched the ground the throttle cut off completely, the nose dropped due to the elevator command, and the roll controller switched off.

3. The navigation code, while operable, would benefit from optimization as a follow-on project. The navigation control laws are not nearly as refined or straightforward as the other control laws and are quite bluntly written in the code. The navigation code is also fairly limited. Given the time frame for this project, the navigation routine was truncated to just the latter portions of the total autoland scheme; when the aircraft is on long final approach. The navigation routines were originally intended to take the aircraft from any point in the sky and direct it through the nominal landing path of downwind leg, base leg, and then final approach. Future development of this code should incorporate the entire landing path, as well as refine the existing portions of the navigation code.

3. The GPS failure mode and the touchdown mode work perfectly. To test the GPS failure mode, the GPS was disabled in the code in order to generate the error signal to the control laws. Once the controller was enabled, the aircraft responded exactly as expected: the throttle advanced to 75%, the roll angle went to zero, and the pitch angle increased to 5 degrees nose up.

X. Conclusion

The design and implementation of flight control laws using a MultiWii flight computer to autonomously land a small UAV has been presented. The MultiWii flight computer was installed in a commercially available model of the Cessna 182 aircraft and frequency response data was collected through flight testing. This frequency response data was then analyzed by CIFER to identify transfer functions for computer simulation of the control laws. The block diagrams for each control law were created in Simulink and simulations were run to determine starting point gains that were later refined for better aircraft performance during flight test.

Many lessons were learned in the course of this project.

1. While the project was comprised of rigid body mode identification, autopilot design and hardware implementation, programming and debugging of the hardware was the most difficult and most time consuming. For example, there were many instances where the system would operate nominally prior to implementing the LIDAR, and once the LIDAR was plugged in a new source of noise was introduced in the other systems. These new noise sources required addressing before flight testing could be attempted.

2. MATLAB/Simulink provides a good starting point for autopilot gains, but those gains are ideal and thus not appropriate for real world flight. As such, they cannot be directly transferred from the block diagram to the hardware, but require some adjustment. The differences in data types are one possible source of the gain discrepancy. Simulink runs ideal calculations, whereas the MultiWii is bound by physical hardware restrictions.

3. Problems in the flying characteristics of the aircraft revealed that too many variables defined as ‘float’ type were causing significant delays in the flight computer because the size of a float variable is 32 bits, compared to the similar type ‘int’ (integer) which is only 16 bits. A

refinement of the code to optimize speed resulted in many variables being redefined as integers at the expense of the accuracy of the calculations performed. The rounding errors incurred by the differing data types may be cause of the difference between the predicted and flight test based gain values.

In summary, the autoland system as a whole works satisfactorily thanks to the high level of precision afforded by the LIDAR. With the total system cost of about $400, the LIDAR-Lite and MultiWii system opens the door to many unique possibilities. In the burgeoning field of home delivery via UAVs, this system gives the capability for precise, autonomous delivery of payloads without fear of returning to earth with a potentially damaging rate of descent. This hardware also allows the design and use of UAVs specifically for topographical mapping, thanks to the LIDAR-Lite's excellent speed, range, and accuracy. The design methodology provides a clear pathway for anyone to design or purchase any UAV, and program any conceivable mission that ends with a perfectly executed autonomous retrieval with no repairs necessary.

XI. References

[1] J. Roskam, Airplane Flight Dynamics and Automatic Flight Controls Part I, Lawrence, KS: DARCorporation, 2011.

[2] D. B. Barber, S. R. Griffiths, T. W. McLain and R. W. Beard, "Autonomous Landing of Miniature Aerial Vehicles," American Institute of Aeronautics and Astronautics, Provo, UT.

[3] J. Roskam, Airplane Flight Dynamics and Automatic Flight Controls Part II, Lawrence, KS: DARCorporation, 2011.

[4] J. Hunter, "Aerospace Vehicle Dynamics and Control," Maple Press, San Jose, CA, 2014.

XII. Appendix A Flare Calculations

$U_1 := 30 \quad \frac{ft}{sec}$ Landing Velocity

$G_s := 9.5 \quad deg$ Glideslope angle

$x_1 := 100 \quad ft$ Touchdown distance after flare

$$h_{dot\_flare} := -\left(\frac{G_s}{57.3}\right) \cdot U_1$$

$$h_{dot\_flare} = -4.974 \quad \frac{ft}{sec}$$

$t_c := 4$ Time Constant

$$x_2 := \frac{-h_{dot\_flare}}{\tan\left(\frac{G_s}{57.3}\right)}$$

$$x_2 = 29.725$$

$$\tau := \frac{-x_1}{\langle x_2 - \langle t_c \cdot U_1 \rangle \rangle}$$

$\tau = 1.108 \quad sec$

$$h_{dot} := \frac{-1}{\tau}$$

$h_{dot} = -0.903 \quad \frac{1}{sec}$ multiplied by height to give ft/sec Descent Rate at Touchdown

$h_{flare} := -h_{dot\_flare} \cdot \tau$

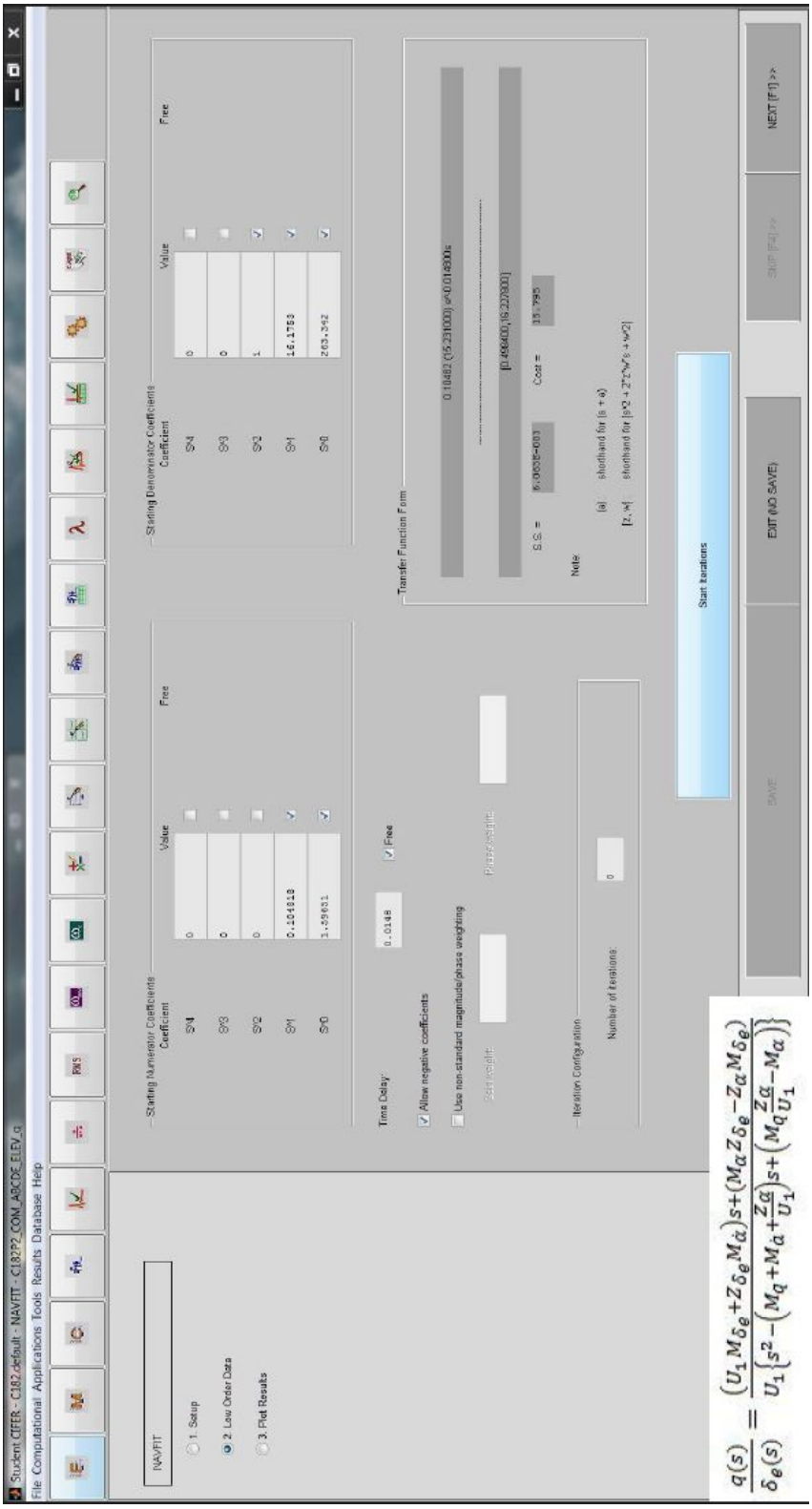
$h_{flare} = 5.51 \quad ft$ Flare height

XIII. Appendix B LIDAR-Lite Sensor Specifications [5]

Signal/Power Interfaces	Specifications
Power	4.7 - 5.5V DC Nominal, Maximum 6V DC
Weight	PCB 4.5 grams, Module 16 grams with optics and housing
Size	PCB 44.5 X 16.5mm, Housing 21 X 48.3 X 35.5mm
Current Consumption	<100ma continuous operation, <2ma @ 1Hz (power off between acquisitions)
Max Operating Temp.	70° C
External Trigger	3.3V logic, high-low edge triggered
PWM Range Output	PWM Signal proportional to range, 1msec/meter, 10µsec step size
I2C Machine Interface	100Kb - Fixed, 0xC4 slave address. Internal register access & control
Supported I2C Commands	Single Distance Measurement, Velocity, Signal Strength
Mode Control	Busy status using I2C, External trigger input PWM Outputs

System Parameters	Laser/Pin(1) Class 1 Laser Product
Transmitter	905nm, 75um, 1watt, 8mrad, 14mm optic
Receiver	Surface mount PIN, 3° FOV wi 14mm optics
Detector Gain	1X
Max Range @ 1Hz 30% Target	30 Meters
Max Range @ 1Hz 90% target	40 Meters
Accuracy	+/- 0.025 meter
Acquisition Time	<0.02 sec
Max Rep Rate	100Hz (2)

XIV. Appendix B Typical CIFER NAVFIT Transfer Function Page



$$\frac{q(s)}{\delta_e(s)} = \frac{(U_1 M_{\delta_e} + Z_{\delta_e} M_{\dot{\alpha}})s + (M_\alpha Z_{\delta_e} - Z_\alpha M_{\delta_e})}{U_1\left\{s^2 - \left(M_q + M_{\dot{\alpha}} + \frac{Z_\alpha}{U_1}\right)s + \left(M_q \frac{Z_\alpha}{U_1} - M_\alpha\right)\right\}}$$