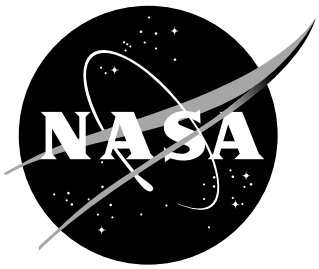
NASA/TM–2015–



Urban SAFE50: MARTI Project 2015

Jack Casey - University of Arizona
Devin Cody - Yale University
Alyssa Deardorff - Oregon Institute of Technology
Pakasutha "Pear" Dhiantravan - Northwestern University
Lauren Kolkman - Purdue Unversity
Anthony Markowitz - Princeton University
Simon Martinex - Embry Riddle University
Steve Schafer - University of Missouri
Mario Sebasco - University of Miami
NASA MARTI

August 2015

NASA STI Program ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

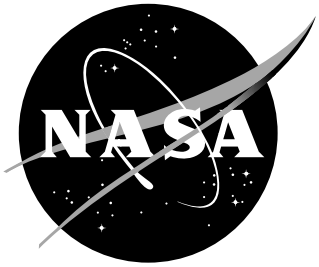
- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English- language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI Program, see the following:

- Access the NASA STI program home page at ***http://www.sti.nasa.gov***
- E-mail your question to help@sti.nasa.gov
- Phone the NASA STI Help Desk at 757-864-9658
- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681–2199

NASA/TM–2015–



Urban SAFE50: MARTI Project 2015

Jack Casey - University of Arizona
Devin Cody - Yale University
Alyssa Deardorff - Oregon Institute of Technology
Pakasutha "Pear" Dhiantravan - Northwestern University
Lauren Kolkman - Purdue Unversity
Anthony Markowitz - Princeton University
Simon Martinex - Embry Riddle University
Steve Schafer - University of Missouri
Mario Sebasco - University of Miami
NASA MARTI

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, Mountain View, CA 94035

August 2015

Acknowledgments

Thank you to our mentors Kalmanje KrishnaKumar, Alfredo Bencomo, Chetan Kulkarni, Corey Ippolito, Shankar Sankararaman, Vahram Stepanyan, Carl Russell, Ben Nikaido, and John Melton, and William Warmbrodt!

This report is available in electronic form at
http://

Abstract

In NASAs quest to develop Unmanned Aerial Systems Traffic Management (UTM), there are many problems which must be solved. Some of the most interesting challenges come from small UAS which can operate below 50 feet AGL. While these UAS offer many benefits to society, including the potential for same-day package delivery and infrastructure monitoring, they must be thoroughly studied and tested to ensure they do not present a danger to the National Airspace or those on the ground. Enter Urban SAFE50. By performing CFD and trim analysis on a COTS quadcopter, studying wind flow around buildings, performing battery prognostics, and implementing and tuning control algorithms, we have begun the necessary detailed analysis which must be performed on small multirotors to ensure they are safe to fly below 50 feet.

Simulating the airflow around these platforms allows us to predict problems which might arise due to the wake of the aircraft. For example, if one multirotor flies into the wake of another, it may stall and fall onto the ground below, damaging property or hurting people. Using RotCFD, we performed computational fluid dynamics analysis on the effect of a commercially available quadcopter within the fluid domain. In particular, a trimming scheme was developed and implemented which aimed to place 3 different representative quadrotor cases in trim (Hover OGE, Hover IGE, and steady forward flight at 20 mph). The trim coefficients for forward flight at 20 mph were found, as well as the required motor speeds for hover in and out of ground effect. This will allow for robust, representative fluid dynamics modeling so that a better understanding of the behavior of these craft within the fluid domain may be better understood, and may also be used in the development of more accurate and safe autopilot controls.

Studying the wind flow around buildings allows us to identify problem areas in urban cities where UAS should not fly. For example, if all areas within a few feet of buildings present very turbulent winds, it should be regulated that all UAS stay out of that area. After analyzing the CFD of a model of Ames Research Center in OpenFoam, we determined that potential problem areas can include corners of buildings facing the free stream, wakes caused by architecture, and urban canyon settings. Using the wind field obtained from CFD results, we also developed an optimal route planner, which minimizes time by taking into account distance travelled and wind velocity. By running more thorough CFD studies on larger city scapes, the route planner can be used to calculate the best route around a windy city, potentially saving a UAS battery power and time. Ultimately, the goal is to design a data base from which a UAS user can extrapolate a wind field for any starting condition, and given a set of GPS coordinates, determine the optimal flight route that can be taken.

In order to enable the UAS to maintain flight and successfully navigate

urban environments, end of discharge and end of usable life events need to be accurately predicted. Battery prognostics enables better designed batteries, more robust decision making protocol in the face of failure events, and therefore more efficient and safer UAS. An improved battery model in Reflection was developed for the Iris+ LiPo battery to enable a more accurate energy system representation for UAS flight simulation. Current control systems for small UAS are not very robust. First, current control systems cannot operate once one or more motors fail. This is especially a problem when UAS are flying over populated areas. Should a motor fail, the UAS must be able to recover and land in a controlled fashion. Second, current control systems do not account for the dynamic drag coefficients or wind disturbance terms present in flights in turbulent areas. This is especially problematic in urban areas, where the many buildings can create turbulent areas and accelerate wind flow dramatically. To address these issues, we implemented, simulated, and adjusted a novel motor failure protocol which can save a multirotor from a failed motor by detecting a drop in current and changing the motor command allocation. We also implemented a dynamic wind estimator, which allowed a simulated quadcopter to fly smoothly despite wind disturbance in translation and rotation.

Contents

1 Motivation **5**
- 1.1 Failure Protocol . . . 5
- 1.2 Motor Failure Protocol . . . 6
- 1.3 Collision Avoidance . . . 6
- 1.4 Wind Response . . . 7
- 1.5 CFD Modeling of Rotorcraft . . . 7
- 1.6 CFD Modeling of Urban Landscapes . . . 8
- 1.7 Implementations . . . 9
- 1.8 Emergency Response . . . 9
- 1.9 Police/Crime . . . 9
- 1.10 Delivery . . . 10

2 Rotorcraft CFD Team **11**
- 2.1 Abstract . . . 11
- 2.2 Background . . . 12
  - 2.2.1 Motivation . . . 12
  - 2.2.2 Introduction to RotCFD . . . 12
  - 2.2.3 Introduction to Rotorcraft Trimming . . . 13
- 2.3 Summary of Research . . . 14
  - 2.3.1 CAD Modeling of a Quadrotor Platform . . . 14
  - 2.3.2 Creation of the Airfoil Table Data . . . 15
  - 2.3.3 Setting Up, Learning, and Implementing RotCFD 16

2.3.4 Process of Trimming a Quadrotor Using RotCFD . 16
2.4 Methodology . . . 18
2.4.1 Trimming Process . . . 18
2.4.2 Isolated Rotor Studies . . . 18
2.4.3 Finding Aerodynamic Forces on the IRIS+ Body . 21
2.4.4 Determining Tilt Angle . . . 21
2.4.5 Determining Rotor Tip Speed . . . 23
2.4.6 Trimming . . . 24
2.4.7 Trimming Full Quadrotor Configurations: Forward Flight . . . 24
2.4.8 Trimming Full Quadrotor Configurations: Hover . 25
2.5 Final Conclusions, Lessons Learned, and a Discussion of Future Work . . . 25
2.5.1 Conclusions . . . 25
2.5.2 Lessons Learned . . . 26
2.5.3 Future Work . . . 27

3 Urban CFD Team **28**
3.1 Abstract . . . 28
3.2 Introduction . . . 28
3.3 CFD of Urban Environments . . . 29
3.4 CFD Data Analysis . . . 30
3.5 CFD Discussion . . . 32
3.6 Route Optimization Algorithm . . . 33
3.7 Path Optimization Discussion . . . 38
3.8 Conclusion . . . 38

4 Controls Team **39**
4.1 Abstract . . . 39
4.2 Background . . . 40
4.3 Motor Failure Procedure . . . 40
4.3.1 Simulink model . . . 43
4.3.2 Hardware Validation . . . 45
4.4 Drag and Wind Disturbance Estimation and Rejection . . 46
4.4.1 Simulink Model . . . 47
4.5 Battery Modeling . . . 50
4.6 Reflection Software Validation Environment . . . 52
4.6.1 Equivalent Circuit Battery Model Integration . . . 52
4.6.2 Motor Failure Procedure . . . 53
4.7 Conclusion . . . 55
4.7.1 Future Work . . . 56

5 Hardware Team **57**
5.1 Background . . . 57
5.2 Introduction . . . 57
5.3 Multi-rotor Design and Integration . . . 58

5.4 Motor Failure Detection 60
5.4.1 Electrical Model of Motor 60
5.4.2 Current Detection 62
5.5 Conclusions 64

6 Appendix A Rotorcraft: Case Data 67
6.1 Case parameters 68
6.2 Calculation of Tip Speed Required for Hover 68

7 Appendix B Rotorcraft: Input Parameters 70
7.1 Rotor Data 70
7.2 Flow Data 70

8 Appendix C Rotorcraft: Isolated Rotor Analysis 70
8.1 Thrust Data for Isolated Rotor 71

9 Appendix D Rotorcraft: Isolated Body Analysis 71
9.1 Calculation of the Tilt Angle During Forward Flight . . . 71
9.2 Calculation of Tip Speed for a Calculated Tilt Angle . . . 73

10 Appendix E Rotorcraft: Forward Flight Trim 75
10.1 Calculation Procedural Overview 76
10.2 Data 77

11 Appendix F Rotorcraft: Hover Trimming 78
11.1 Out of Ground Effect 78
11.2 In Ground Effect 79

12 Appendix G Rotorcraft: Convergence Data from RotCFD 80

13 Appendix A Urban: CFD results 84

14 Appendix B Urban: Optimization Path 89

15 Appendix C Urban: Get Time From Path 96

16 Appendix D Urban: Way Points to Path 97

17 Appendix E Urban: Ames Section 98

18 Appendix F Urban: Obstacle Generator 101

19 Appendix Controls: Code 102

1 Motivation

The proposal and implementation of a Safe Autonomous Flight Environment for the last 50 feet (SAFE 50) of unmanned aerial systems (UAS) presents challenges unique from the overarching Unmanned Aerial System Traffic Management (UTM) project. Unmanned vehicles will play a large role in the integration and development of towns and countries far from necessary resources, and will further increase the day-to-day efficiency and convenience of normal societal operations. As the understanding and acceptance of these systems grows, the documentation and control of all aircraft involved will be increasingly vital.
UTM encompasses concerns ranging from collision with larger aircraft to provision of a framework for drone fleet management. Its task is devoted to the initialization and autonomization of UAS operations. SAFE 50 is an internal project that will tackle the last division in the development of this management system. The constraints that emerge when modeling safe autonomous flight near the ground are distinct from those found in other flight regimes. Design challenges in this region reflect the dynamicity of the environment and incorporate the imminent necessity of precise trajectories, failure protocols, and object recognition and avoidance. This is further exaggerated in urban settings where the presence of buildings, ground objects, and general infrastructure present numerous engineering hurdles. As with all autonomous operations, stability and control are of the utmost priority (stable, controlled flight is safe flight). The applications of UASs within urban areas range from firefighting to medical emergency relief to package delivery. Across every mission, a set of challenges remains universal and the response to these challenges can be applied unilaterally. Defining possible obstacles and proposing response strategies will be the focus of this study. These challenges include: Motor failure Obstacle collision Wind influence In order to best create a network of procedures to respond to these challenges, building a framework of basic fluid dynamics models will allow for an initial generalization of problem scenarios and thus serve as a basis on which to solve these problems.

1.1 Failure Protocol

Operation within fifty feet of the ground introduces a plethora of potential internal and external obstacles to completing a safe mission. Given UASs close proximity to property and the public during flight, safety measures must be in place for any foreseeable in-flight issues. Drones need to be reliable within their motor, control, and communication systems. A comprehensive analysis of possible ways that a UAS could be unsuccessful is necessary in order to comprehend the imperative system redundancies and failure mode procedures. Without a human operator, the machine will need to be able to independently determine the safest

response procedures. Safety features may include an autonomous return mode, a manual override, and built in hardware redundancy.

1.2 Motor Failure Protocol

Motor failure is one of the more well-understood problems that may arise. With multirotor UASs, failure of motors can be catastrophic if proper contingencies are not in place. The heavy payload, spinning blades, and/or blunt body of an aircraft could easily damage structures, cars, and people caught in its freefall. This study aims to minimize the damage done by a multirotor UAS that has lost one or more of its motors. One dramatic case might include power loss to all but three motors. In order to maintain aircraft control, all but two opposing motors must be shut down, or the net thrust will cause the UAS to flip and fall to the ground. If only two opposing motors remain functional, the UAS will be sent into a spin caused by the net torque of the motors.

Before UASs can safely be integrated into society, an on-board autonomous response system must exist to direct the vehicle in its semi-controlled state to the best possible landing location. Optimizing UAS power allocation and maximizing the safety of humans in the vicinity of UASs is key to their incorporation into daily routines.

1.3 Collision Avoidance

With the many possible applications for UASs, it is not difficult to imagine a situation in which a large number of third party drones occupy the same airspace. In such a scenario, an air traffic management system will need to be developed for the UASs. One of the primary reasons UASs have become so popular is the ease with which they can be developed and piloted. However, the implementation of more advanced collision avoidance hardware (cameras, rangefinders, LiDAR) and computationally intensive software (computer-vision, object recognition) on board will not likely occur in the near future. Rather, building a communications network seems to be the more viable solution.

The best solution will then be to develop air management software able to communicate with all the UAS in the local airspace and relay to the control system on board which routes they are and are not allowed to follow. The architecture of such a system might be similar to that of the current US highway system, or perhaps it might be able to handle a larger volume of vehicles if a more decentralized approach is taken. Alternately, it might be possible to implement a drone-to-drone collision avoidance algorithm using Nash bargaining. In such a case, drones would largely be free to fly along any path desired so long as collisions are actively avoided.

A final solution for this problem is for all drones to carry an onboard suite of sensors to do their own collision avoidance calculations. This would

ensure the greatest information privacy but also the least reliability given the huge cost associated with both building a drone and purchasing the collision avoidance hardware and software.
While collision avoidance is beyond the scope of this study, it must be integrated into the flight controls in future work in order to allow for autonomous, beyond-line-of-sight UAS tasks.

1.4 Wind Response

Below fifty feet, wind presents a major problem for drones. Given that trees, houses, buildings, vehicles, and people heavily populate the lowermost fifty feet of airspace, there are many objects that may generate unpredictable gusts and cause complex turbulent wind profiles which UASs will have to account for during their missions. Without a robust control system in place, UASs will not be able to fly efficiently or safely in these airspaces. These hindrances can easily result in a loss of control, leading to an abort of the mission, damage to the drone, and possible damage to property and people in the surrounding environment.
Wind presents a particularly difficult problem because unlike motor failures, its disturbance cannot be measured directly by an onboard sensor. While traditional fixed-wing aircraft can measure airspeed directly with pitot tubes, rotorcraft like multirotor drones cannot use the same sensors because their spinning rotors create local flows that would interfere with any onboard measurements. This problem can be solved, however, by utilizing the fact that wind disturbs a UAS by changing the net force acting on it. Its effect can thus be measured indirectly using other sensors such as accelerometers, magnetometers, and optical cameras.
Unfortunately, these indirect measurements are insufficient to control most small UASs when used separately. Defining the on and offline measurements necessary for the rotorcraft will then be the first task. These measurements will ensure that the control system will have enough information for safe and efficient UAS flight. The second challenge to address is the proper integration of these measurements to provide UASs with the best control possible in the turbulent environments present below fifty feet. Steps taken towards solving these goals will be steps closer to enabling UAS navigation in the windy spaces present in the last 50 feet of urban areas.

1.5 CFD Modeling of Rotorcraft

Initial development of a dynamic control system will be dependent on the cohesion of a set of model situations under which the UAS operates. This model will include both the external flight environment as well as the aircrafts own design. Understanding the flow dynamics around an aircraft is pivotal for creating an optimized platform. The quadcopter will be the subject of study as it is the most commonly used platform

in this field. As of current, the flow characteristics and fluid dynamics of quadcopter UASs have not been well documented. In order to better understand the behavior and dynamics of these aircraft, a comprehensive CFD analysis should be performed outlining some baseline flight regimes that quadcopter might typically encounter. This would include:

- Hovering flight
- Steady forward flight
- Flight in ground effect (IGE)
- Flight in strong wind
- Formation flight

The gathered baseline data will then be imported into the Reflections simulation software. This will increase the accuracy and fidelity of the simulation, which in turn will lead to a stronger analysis of the quadcopter platform. Once the effect of the rotorcraft in different flight regimes is further understood, design optimization- whether through physical alterations or control protocol strengthening- can begin. It should also be noted that a general, scalable model of multi rotor platforms will be valuable in the future as further work is done on this subject.
Independent of both the simulation and scalable model, CFD analysis of the UAS is necessary as a first step towards understanding its behavior and outlining preliminary regulatory measures.

1.6 CFD Modeling of Urban Landscapes

In addition to modeling the aircraft design, establishing relevant parameters stemming from its flight environment is essential to designing an efficient controls algorithm. An important aspect in these parameters is the velocity profile of wind as it travels around cityscapes. In order to implement a viable UAS flight system, CFD modeling of the urban landscape is necessary. Although the computer algorithm designed for this experiment is responsible for the estimation of external forces, it is necessary to know if there are any particular areas in the city for which the UAS simply cannot handle the forces present.
This information can be extended to provide data regarding the safest or most efficient path that a UAS can take in order to reach its destination without encountering any problems. Further progression into this field permits the integration of more intricate systems for the future, such as potential time-dependent geo-fences, specific architectural avoidance strategies, and more. Data emerging from any urban CFD simulations should be incorporated into flight simulations in order to amplify the fidelity of the wind response algorithm generated from the controls aspect

of the project. Such data can account for specific scenarios, possible downfalls, and potential improvements to the model.
It is clear that a code must be tested numerous times in order to be truly considered viable, and a simulation of the flight using CFD-imported data from numerous different environments is potentially the best non-experimental way to confirm a fully functional system. Computational fluid dynamics provides insight into the potential difficulties that distinct geometric figures may have. This can range from increased velocity between buildings to unpredictable turbulence around a corner, and is essential information to the future of UAS flight within 50 feet. The overall adaptations of urban CFD simulations into the SAFE 50 project can yield positive results. The outcomes of such experiments can range from practical information, to necessary input for the UAS testing environment, to potential future investigations/applications.

1.7 Implementations

The benefits of utilizing UASs within an urban setting are immense; they offer new capabilities otherwise not realizable. With emerging missions of the next generation of UASs, safety is the number one priority. SAFE 50 is the first step towards realistic use of UASs in an urban setting.

1.8 Emergency Response

With their immense population densities and poor traffic flow, urban environments present a unique problem in the event of emergencies. First responders may find themselves trying to navigate through blocks of bumper-to-bumper traffic to reach a victim. A UAS can navigate independent of the citys road system, reach an emergency, and begin providing aid long before first responders may arrive. Even if the UAS does not engage the victims directly, the ability to scout out the emergency and potential routes that the first responders should take is a critical capability which may save countless number of lives. In an emergency, every second is crucial, and UASs can allow for faster, more detailed response.

1.9 Police/Crime

In many high intensity situations, a police departments number one asset is their eye in the sky. While helicopters bring a wide array of functionality to a crisis, they are incredibly expensive to deploy and maintain. In many scenarios, a UAS may take the place of the helicopter, generating huge cost savings. Additionally, the removal of the human pilot from the equation makes missions where the UAS are deployed inherently safer, as there is no risk to the pilots life.

1.10 Delivery

Drone Delivery has been a socio-political buzzword for the past couple of years, and with good reason: the ability of UASs to deliver packages faster is of huge convenience to the general public. The days where we can expect to see a mail drone dropping off a package are not far out. In order for this service to be readily available, the UASs must pose absolutely no threat of harm to any persons or property. Urban settings provide unique challenges for delivery services with their crowded design, dense populations, and dynamic environments. Further, the prospect of numerous third party UASs operating in close proximity requires beyond state-of-the-art collision avoidance technology and traffic management schemes.

While the utilization of UASs is an advent not too far in the future, there is still much that needs to be studied and optimized before their widespread use. While the use of UASs within an urban setting is not a particularly novel concept, our understanding of the dynamics and behaviors of both the vehicles and the environments in which they will operate is in its infancy. The goal of the MARTI team working on the SAFE 50 project is to begin a baseline study of some of the numerous design challenges brought about in operating UASs in an urban environment. This includes modeling wind turbulence around various buildings/cityscapes, modeling the fluid dynamics of the UAS platforms themselves, and creating preliminary protocols for safe, stable flight through collision avoidance, motor failure handling, and wind rejection. Once these factors are better understood, UASs may be realized as a part of daily urban life.

2 Rotorcraft CFD Team

2.1 Abstract

The progression of technology constantly fosters changes in the mechanisms within societies, generating the need for regulations that address the safety and wellbeing of its citizens and infrastructure. Under the overarching project of Unmanned Aerial System Traffic Management (UTM), the Safe Autonomous Flight Environment within 50 feet of the ground (SAFE50) addresses this concern for the time when small unmanned aerial vehicles (UAVs) become densely distributed in an urban environment. To better understand the hazards of a densely populated, low-altitude flight condition, computational fluid dynamics (CFD) analysis needs to be performed to model the outwash, downwash, and ground effects of a small UAV. The quadrotor in particular requires specific attention as a rotorcraft platform due to its increased prevalence for recreational and commercial use. A commercial off the shelf (COTS) quadrotor (3DR IRIS+) was modeled using a combination of CAD software and a rotorcraft CFD solver RotCFD. A series of isolated body and rotor cases were run in order to place the quadrotor in trim at 3 flight conditions: hover out-of-ground-effect, hover in ground effect, and steady, forward flight at 20 miles per hour.

2.2 Background

2.2.1 Motivation

Implementation and integration of autonomous aircraft into high density environments requires an elemental understanding of flight characteristics. Gathering a complete database of flow patterns including wake, outwash, downwash, and ground effect created by UAV flight will allow policymakers to create necessary and effective flight regulations to ensure safe in-air coordination between aircraft. Pressure gradient visualization and eventually acoustic data, all computed in CFD software, can assure both UAV users and the general public of the safety and functionality of these systems.

The fundamental implementation of this data requires consistent analytical procedures to ensure a uniform understanding of its implications. All flight regulators must be operating with the same standards in order for safety measures to be effective. Designing and testing a process for assuring that CFD modeling is representative of the actual flight conditions experienced by these small UAS is pivotal to ensuring a robust understanding of these craft. This work therefore explores a process for modeling a small, unmanned aerial system (UAS), and will provide procedural information on the characterization of such rotorcraft. The platform used in this study is the 3DR IRIS+, which is commonly manipulated as a quadcopter platform for UAV hardware testing. A selection of flight and hover cases of the IRIS+ is analyzed in a computational fluid dynamics solver, RotCFD, which is a graphic user interface (GUI) CFD program specifically created for use on analyzing rotorcraft platforms. At the heart of this study is an exploration of an iterative method for performing trim analysis on a quadrotor platform. Results can be used not only in assuring representative flight conditions are accurately modeled, but may also be utilized at a higher level for implementation in controls protocols for small UAS and auto piloting functions.

2.2.2 Introduction to RotCFD

RotCFD is a software developed by Sukra Heliteck, Inc. under Ganesh Rajagopalan. Its capability domain is unique in that it contains a comprehensive analytic package [1]. For this specific application, the Rotor UNStructured Solver Application (RotUNS) was used. This solver utilizes fit to body tetrahedrals near the solid geometries within the fluid domain and Cartesian unstructured gridding far from the body. Permitted by this definition is grid adaptability which can assist flow visualization [2] [3] [4].

RotCFD is unique in the way it initiates the interaction from the rotor on the fluid. By representing each rotor as a momentum source

distribution, the effects of the rotor are taken into account in the flow profile. There is thus is no need to mesh the physical geometry of the rotor; this allows for RotCFD to be run on computers of lower computational capability.

The assumptions made within the solver for the Navier-Stokes equations were unsteady, laminar, and incompressible flow. The following three equations are used as the governing equations of the fluid flow. Of note are the additional S_x and S_y terms which take into account the aforementioned momentum sources.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{1}$$

$$\frac{\partial (\rho u)}{\partial t} + \frac{\partial}{\partial x}(\rho u^2 - \mu \frac{\partial u}{\partial x}) + \frac{\partial}{\partial y}(\rho uv - \mu \frac{\partial u}{\partial y}) = -\frac{\partial p}{\partial x} + S_x \tag{2}$$

$$\frac{\partial (\rho v)}{\partial t} + \frac{\partial}{\partial x}(\rho uv - \mu \frac{\partial v}{\partial x}) + \frac{\partial}{\partial y}(\rho v^2 - \mu \frac{\partial v}{\partial y}) = -\frac{\partial p}{\partial y} + S_y \tag{3}$$

Unique to RotCFD and RotUNS is the capability to visualize rotor to rotor and rotor to body interactions within the flow regime. Flow characterization and blade performance are parameters automatically included in the analysis run by these programs. This feature allows for the trim of a new rotor system to be performed much more rapidly through the use of these parameters. Further detail on the theory of RotCFD may be found in references [1] [2] [3] [4].

2.2.3 Introduction to Rotorcraft Trimming

Trimming is the act of balancing all forces and moments on a rotorcraft such that the net force and net moment produced sum to zero [5]. This data is used to adjust aircraft in numerous ways, sometimes leading to a design change or mechanical correction. It may also be used to control flight via pilot inputs, or by onboard computers or autopilot functions. The trim process is one of nontrivial importance to rotorcraft design and control. Further details on helicopter trim may be found in references [6] [7].

While similar in design to helicopters, quadrotors have unique properties which make their trimming process inherently different, for instance, the lack of pitch and roll control via cyclic or collective inputs. In a quadrotor, the thrust output of the rotors is controlled via the tilt angle of the craft (the rotors are rigid and do not pitch about their shaft axis) and the speed of the rotors, not by a deflection of a main rotor hub. Thus, to place a quadrotor in trimmed hover flight, the thrust force from the motors must exactly balance the weight force of the craft. Since this hover flight regime is assumed to be in steady state, there are no other

inputs necessary; matching the thrust of the motors to the weight of the quadrotor is sufficient to place the craft in trim.

In forward flight, quadrotors require a specific tilt angle to generate forward thrust. This tilt presents a different challenge due to the dependence of generated forward thrust and therefore forward velocity on the specified angle. The craft is thus influenced not only by the amount of vertical thrust generated by the rotors, but also by the effect of the tilt angle on its drag and forward thrust. The thrust generated by the rotors is additionally dependent on the relative wind created by the forward velocity, as well as on the motor speed. Thus, trimming a quadrotor in forward flight requires an understanding of the thrust output of the motors at different tilt angles and motor speeds, which is valid at one specified velocity. Likewise, the aerodynamic forces which act on the craft (lift and drag) at a specified forward flight speed with regards to tilt angle must be defined.

2.3 Summary of Research

2.3.1 CAD Modeling of a Quadrotor Platform

The initial conception of the quadrotor platform analysis involved the use of an IRIS+ quadcopter produced by 3D Robotics. An approximated model was created in Autodesk Inventor utilizing hand measurements of the physical quadrotor blade. Several different models of the IRIS+ were created in this software. The first included the landing feet on each of the four legs. However, these feet were very small when compared to the overall geometry of the craft, and the mesh generated created distorted, small elements that would have resulted in diverging flow information in the area. A new model without feet was then created. This simplified geometry made the generated elements more evenly shaped, and removed the problem areas near the feet where grid resolution was an issue. The model was then exported as an stl file which could subsequently be imported into RotCFD for the CFD analysis of this specific quadrotor.

Figure 1. 3DR Iris+

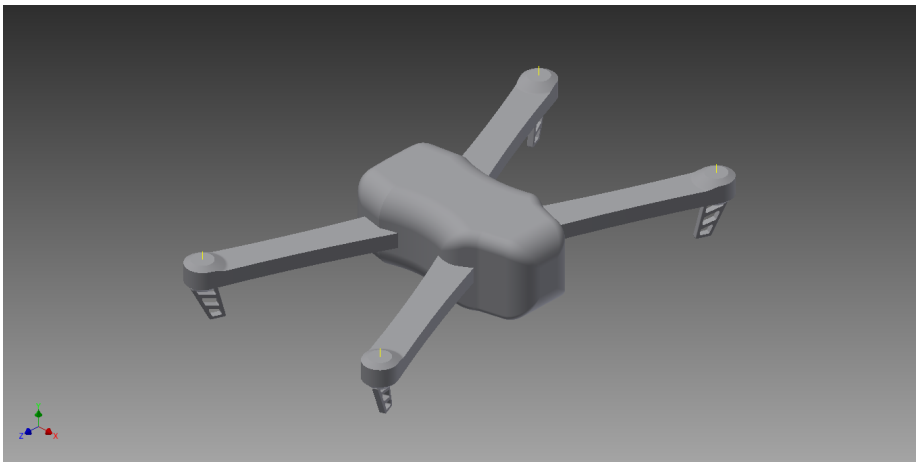


Figure 2. CAD model of the 3DR Iris+ created in Autodesk Inventor with feet

2.3.2 Creation of the Airfoil Table Data

RotCFD uses airfoil table data (C81 format) and blade geometry inputs to calculate the effects of the rotors in the fluid domain using the momentum source term method described earlier. Thus, it was necessary to measure and reverse engineer the blade data out of the IRIS+ propellers.

The IRIS+ uses a TigerMotor Navigator Series 2212 V2.0 920Kv motor/propeller (MN2212 920Kv V2.0). The propeller data was extracted using hand measurements, and this data is shown in Appendix A. Figure 2.3.2 shows the propellers as they are seen by RotCFD in comparison with the actual propellers.

Figure 3. Comparison of the actual and modeled rotor

After the necessary blade contour data was collected, a series of C81 airfoil tables was created using AFTGen, an airfoil table generator present within the RotCFD suite [8] [9] [10]. These were then used to specify the airfoil stations throughout the span of the blade, which is shown in Appendix A.

2.3.3 Setting Up, Learning, and Implementing RotCFD

Isolated rotor and body configurations were used to gain a basic understanding of how to generate acceptable grids and configurations within RotCFD. This was very much a learning experience; many of the first cases did not converge, or were met with gridding issues. A full directory of the cases run in this work is available in Appendix A. The grid options, boundary sizes, and boundary conditions are also listed in the various Appendices corresponding to the case run. A refinement box was implemented for each case to further resolve the grid near the rotors and geometries. The refinement of the rotors and bodies was set to one level higher than that of the refinement box, creating a finer mesh near the body, which dramatically aided convergence. Body fitting was not utilized in this work, as it created complications with grid resolution near the quadrotors legs and in turn made convergence difficult. The grid generated without the use of the fit bodies option was found not only to converge much more easily, but also drastically lowered the computational time required per case.

2.3.4 Process of Trimming a Quadrotor Using RotCFD

The majority of this work focuses on collecting trim data for three different flight regimes which are believed to be indicative of nominal conditions experienced by quadrotors. They are: hover flight out of ground effect (OGE), hover flight in ground effect (IGE), and forward flight at 50% max speed (which in this study is approximately 20 mph). In

order to accurately model the quadrotor in a configuration which is indicative of what it will experience in actual forward flight at 20 mph, a trimming scheme is executed in which numerous flight parameters (tilt angle, motor speed, disparity of motor speed between fore and aft rotors) are approximated to values which place the craft in trim. The iterative trimming scheme is outlined in later sections.

The quadrotor in hover, both in and out of ground effect, is assumed to have two degrees of freedom within the trimming analysis: vertical motion and motor speed. In steady state hover, the quadrotor is said to be in trim when the craft experiences no vertical motion. The only control input for trimming the craft is then motor speed. The motor speed which results in a net force of zero acting on the quadrotor is said to be the trimmed condition; this is true both in and out of ground effect. While it could be said that the presence of the ground only presents another parameter within the design, these two conditions are analyzed as two completely different flight regimes. A motor speed which yields a net zero vertical aerodynamic force on the craft in hover within ground effect regimes and one which creates a net zero out of ground effect regime represent trim constants for the rotorcraft.

A number of correlations and parameters exist in forward flight which are unique from the hover cases. These differences necessitate that additional analysis is completed in developing an accurate trim scheme. The prominent difference is the necessity of the craft to fly at some forward tilt angle in order to generate the thrust required to maintain a steady forward velocity. In this sense, the trimming scheme becomes much more involved; the forces on the quadrotor will change with respect to tilt angle in steady flight at a given speed, likewise, the thrust generated by the rotors will change as a function of tilt angle as well as motor speed. This means that the aerodynamic forces on the quadrotor body must be expressed as a function of tilt angle, and that the thrust generated by the rotors must be expressed as a function of tilt angle and motor speed. This leads to the creation of an iterative trimming scheme which implements RotCFD in order to place a hovering quadrotor (IGE and OGE) and a quadrotor in steady forward flight in trimmed flight.

It is important to note that throughout this analysis there are many assumptions made which allow for the process to be possible. The first is that the center of gravity of the quadrotor body is coincident with the origin of its geometry. The .stl geometry was modeled in a way that aims to make this assumption as valid as possible, however, a more robust analysis should consider determining the precise center of gravity of the rotorcraft. The rotor blades are also assumed to be completely rigid, and do not contain any drag or flapping hinges, and no flapping data is used in modeling them.

2.4 Methodology

2.4.1 Trimming Process

In order to analyze the performance of the quadrotor, a trim scheme was executed which allowed for its parametric offsets to be approximated for given flight conditions. This data is then used to analyze the behavior and performance of the quadrotor in three different flight regimes: hover out of ground effect, hover in ground effect, and steady forward flight at 20 mph.

It was necessary that the behavior of each of the components of the quadrotor be understood in order to place the quadrotor in trim. As such, the performance of the crafts rotors was analyzed under a series of different parameters. These cases studied the effect of the tilt angle and motor speed in both hover (no wind) and steady flight (20 mph relative wind). This data was then used to see how much thrust was generated by the rotor under the influence of these differing parameters.

Likewise, an understanding of the quadrotors geometry and the effects of the fluid domain on it are pivotal to being able to operate the quadrotor in trim. In order to gain a more complete understanding of the aerodynamic forces acting on the quadrotor during steady forward flight at 20 mph, the IRIS+ body was modeled in RotCFD at 0, 5, 10, and 15 degree tilt angles in 20 mph wind. The data gathered in these studies allowed for a more accurate representation of the drag forces and vertical aerodynamic forces are acting on the geometry, and ultimately allowed for the trimmed tilt angle to be approximated. From this, and using the rotor data also found, the quadrotor may be placed in approximate trim. All thrust and force data curves from the cases studied in this trimming process are available in Appendix G.

2.4.2 Isolated Rotor Studies

In order to better characterize the performance of the motors in the computational domain, a study was performed in which the rotors were modeled in hover and in 20 mph wind at different tilt angles.

Figure 4 shows a thrust/motor speed curve taken from data provided by TigerMotors, the manufacturer of the motors and propellers used on the IRIS+. The free body diagram in Figure 5 of the quadrotor in hover shows that the only two forces assumed to be acting on the quadrotor in hover are the thrust from the rotors and the weight of the craft itself. In order for the quadrotor to be in steady hover, then, the thrust generated by the motors must exactly match its weight.

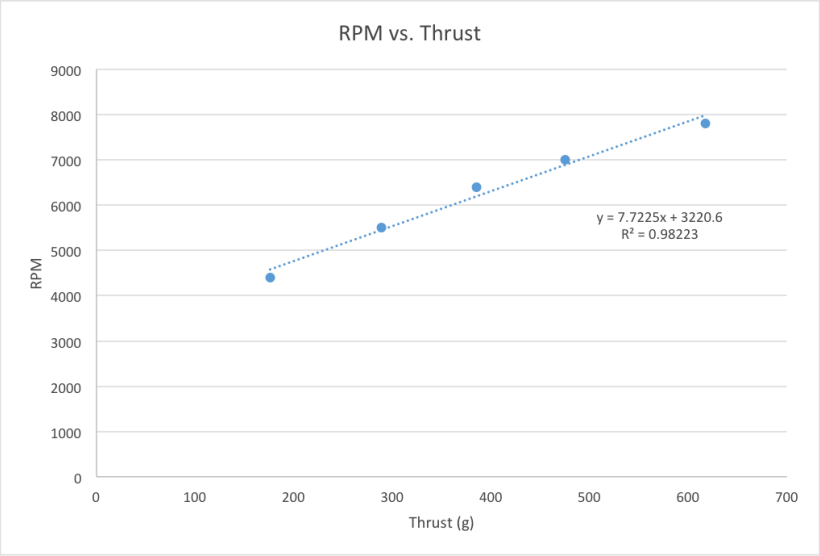


Figure 4. TigerMotor thrust and RPM data

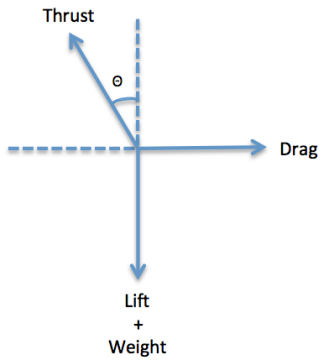


Figure 5. Free body diagram of forces on the quadcopter in hover

In steady forward flight there are additional considerations that must be taken into account, for instance, the effect of the relative wind on the thrust generated by the rotors, as well as the effect of tilt angle on the thrust output, and the direction of these forces. In order to place the craft in trimmed forward flight, the aerodynamic and weight forces acting on it must be precisely balanced by the thrust output of the rotors. In order to understand the effect of the tilt angle on the thrust generated by the rotors, a study was performed in which an isolated rotor was modeled in 20 mph wind at 5, 10, and 15 degree tilt angles. Isolated rotors were also run in no wind at 0 degree tilt, which is indicative of hover. This series was run at two different motor speeds, one which was the motor speed necessary for hover flight, based on the manufacturers thrust data, and one which was 20% higher than this value. This provides a series

of curves that describes the performance of the rotors at different tilt angles. This data is shown in Figures 6 and 7.

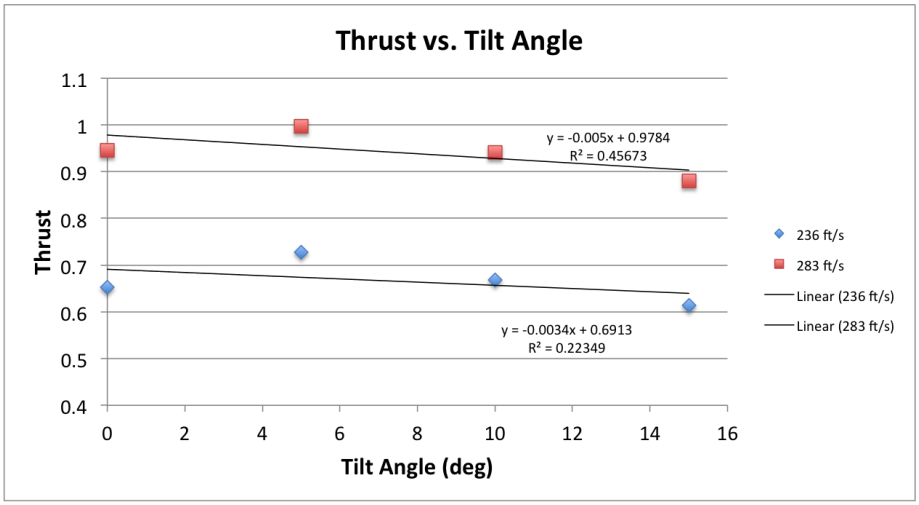


Figure 6. Thrust vs. tilt angle for two different rotor speeds

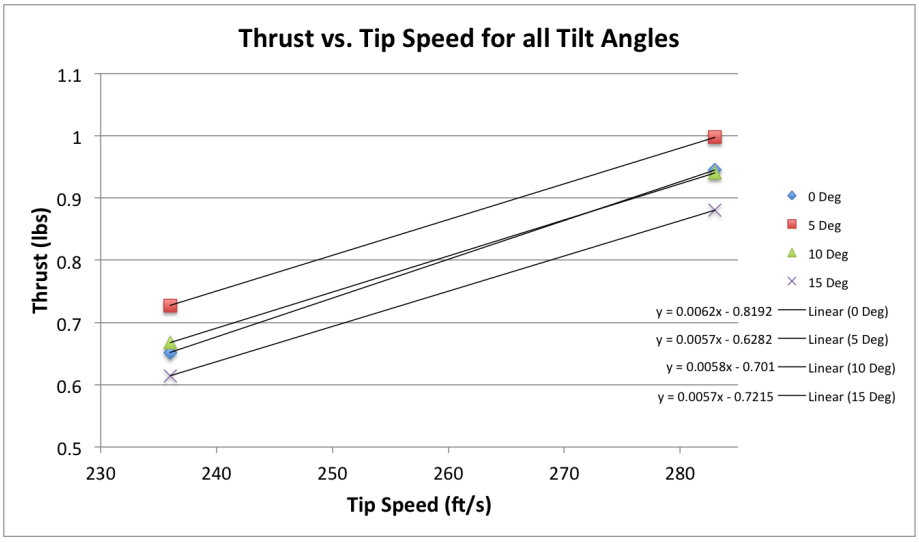


Figure 7. Thrust vs. tip speed for four different tilt angles

This data shows that the thrust generated by the rotors at each of the two motor speeds can be approximated as a function of tilt angle, which is described by the expressions:

$$T_{236} = -0.0034\theta_{tilt} + 0.6913 \tag{4}$$

$$T_{283} = -0.005\theta_{tilt} + 0.9784 \tag{5}$$

2.4.3 Finding Aerodynamic Forces on the IRIS+ Body

In order to place the quadrotor in steady, trimmed flight, all forces and moments must be equalized such that there is no net force on the craft. In this work the effect of tilt angle on the net force output on the quadrotor geometry is studied in one forward flight condition (20 mph). This allows for the force required from each rotor to be found at any tilt angle, since they must counteract the aerodynamic forces present on the quadrotors body in order for the craft to be in steady flight.

To determine the thrust required as a function of tilt angle on the quadcopter body, four separate cases were studied in 20 mph wind at 0, 5, 10, and 15 degree tilt. These cases were run without rotors to obtain information which is only indicative of the aerodynamic forces acting on the IRIS+ body.

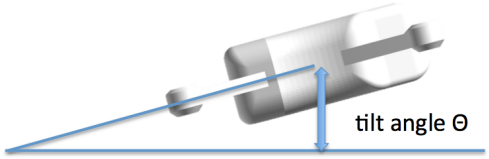


Figure 8. Iris+ at a tilt

Figure 9 shows the drag and lift forces which act on the IRIS+ body at 0, 5, 10, and 15 degrees as computed in RotCFD, while Figure 10 below shows the total net force acting on the quadrotor at the same tilt angles (this is a combination of the lift, drag, and weight forces). This force vector must be exactly counteracted by the thrust vectors of the rotors to place the craft in trim.

2.4.4 Determining Tilt Angle

Data obtained from these isolated body studies was used in the first iteration of the calculation for the proper tilt angle for steady flight. Each run provides information about the horizontal drag force as well as the vertical aerodynamic force acting on the body. The data was extracted

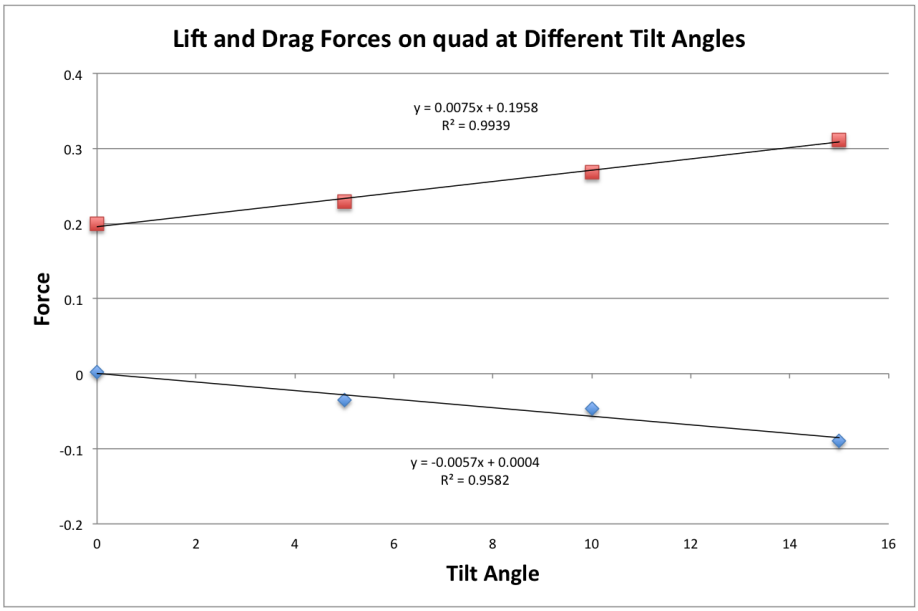


Figure 9. Lift and drag forces on the body at various tilt angles

from the force and moment solutions in RotCFD for forward flight at all tilt angles studied (see Appendix D). The magnitude of horizontal drag and vertical lift forces used for thrust calculations were taken from the last time step (ts = 399) to maximize time allowed for convergence.

A regression of each of these forces was used to generalize the drag and lift on the IRIS+ body for preliminary thrust calculations. Figure 9 above displays the regressions used for drag and lift characterization. Assuming linear behavior, the magnitude of the lift and drag forces can be written as a function of tilt angle:

$$D = 0.0075\theta_{tilt} + 0.1958 \tag{6}$$

$$L = -0.0057\theta_{tilt} + 0.0004 \tag{7}$$

For steady forward flight at constant velocity, the forces on the quadcopter must be balanced by adjusting to the correct tilt angle. This angle alpha describes the vector direction of the quadcopters thrust, which should exactly counteract the drag, lift, and weight forces on the body if the craft is truly in steady forward flight. The tangent of alpha then gives the non-dimensional ratio of drag to lift and weight.

Figures 9 and 10 from the previous section indicate that the lift and drag forces acting on the quadrotor body are directly affected by the tilt angle of the craft. There is exactly one angle at which the direction of the quadrotors thrust (which is the same as the tangent of the tilt

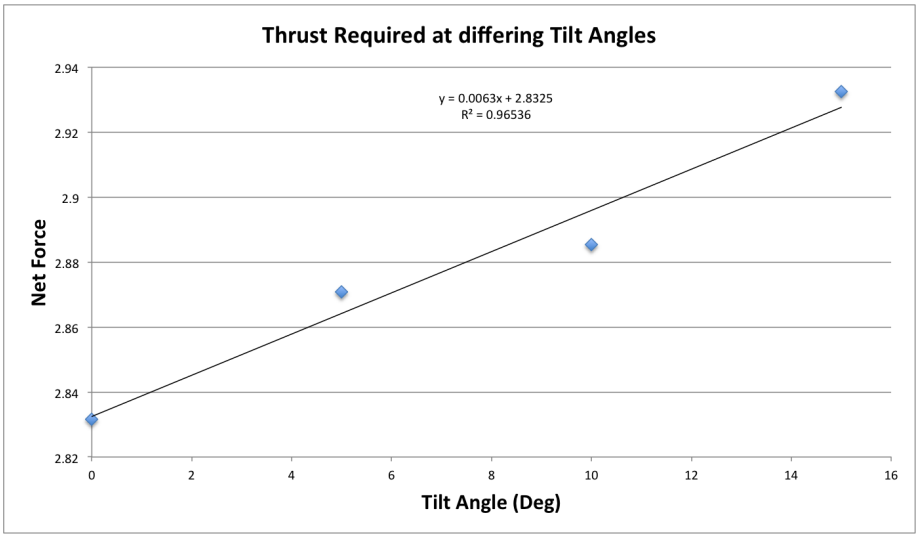


Figure 10. Net force on the body at various tilts

angle, since the rotors all generate thrust orthogonal to the longitudinal plane of the quadrotor, and are all assumed to be operating at the same speed) will match the resultant force acting on the quad through the aerodynamic force and weight force vectors. Thus, the correct tilt angle may be approximated by minimizing the expression:

$$tan(\theta_{tilt}) - \frac{D}{(L-W)} \tag{8}$$

The calculations for this condition are shown in Appendix D. From the expression above, and utilizing the isolated body data gathered and shown in Figures 9 and 10, it is found that the approximate tilt angle for trimmed flight is 4.62 degrees. It is understood that this value still makes numerous assumptions on the lack of body/rotor effects and rotor/rotor effects, and that at this point all four rotors on the craft are assumed to be operating at the same motor speed.

2.4.5 Determining Rotor Tip Speed

In forward flight, a rotor-rotor effect as well as a body-rotor effect causes differences in the required thrust between the front and rear rotors. This effect is not taken into account in the isolated rotor cases which are used to set the tilt angle and motor speed for the initial trimmed case. All four rotors were set to a uniform motor speed in this study to quantify the errors in horizontal drag and vertical aerodynamic forces when the thrust produced by each individual rotor is the same. Using the initial tilt angle found for steady forward flight, and understanding that the thrust from the rotors must match the total thrust required at that

initial tilt angle, shown in Figure 10 , the total thrust required at 4.62 degrees was found to be 2.86 lbs. This value was divided by 4 to find the average thrust required from each individual rotor, which was found to be 0.715 lbs.

The required motor speed to achieve this thrust was found by linearly interpolating between the thrusts given by tip speeds of 236 ft/s and 283 ft/s. This data can be found in Figure 6 under the section Isolated Rotor. Equations 4 and 5 were used to calculate the thrust produced at 236 ft/s and 283 ft/s. The tip speed required for one rotor to produce 0.715 lbs of thrust is 243 ft/s. All of the calculations used to achieve these trim values may be found in Appendix D.

2.4.6 Trimming

2.4.7 Trimming Full Quadrotor Configurations: Forward Flight

After finding the motor speed required to counterbalance the drag, lift, and weight forces acting on the rotorcraft at 4.62 degree tilt in 20 mph forward flight, a full configuration run in which all four rotors operate at this speed (243 ft/s) was studied. The error between the thrust generated by each rotor and the thrust required to keep the quadrotor in steady flight was then determined. This process takes into account the rotor-rotor and rotor-body effects as calculated by RotCFD, and is therefore an iterative procedure.

Thrust data from the Rotor Performance History in Case 4A (the preliminary, forward flight full configuration study) shown in Appendix G depicts that there is an imbalance between the thrust generated by the front and rear rotors. A visual analysis of the data shows that the rear rotors are generating too little thrust and the front rotors too much thrust in reference to the calculated thrust requirement for steady flight at 20 mph in a 4.62 degree tilt (which should be 0.715 lbs).

Convergence occurs in Case 4A near timestep 70; the average thrust generated by each rotor was taken between timesteps 200 and 225. It is seen that the thrust generated between both of the rear rotors is essentially the same, likewise, the thrust generated by each of the front rotors is essentially the same, and thus an average value for the front rotors and that for the rear rotors was calculated. These values were quantitatively compared to the thrust required of each rotor for steady forward flight at 4.62 degree tilt. The percent deviation between the actual thrust and the required thrust was found, and corrections in the tip speed of each rotor are displayed in Tables 17, 18, and 19.

Using the calculated force balance errors and motor speed corrections

to compensate for the net negative vertical thrust and net positive forward thrust, a second iteration trimmed case (4B) was run.The front rotors were prescribed a tip speed of 218 ft/s, while the rear rotors were given a speed of 257 ft/s. Appendix G displays the solution graph of the net thrust being exerted by each rotor in this configuration. It was seen that in this case the correction caused an exaggerated result; the rear rotors now generated too much thrust, while the front rotors generated too little. A final trimmed case (4C) was created which aimed to interpolate between the two cases. It is seen that the thrust output of these rotors is very near the value required for the craft to be in steady forward flight, and as such, the quadrotor appears to be (approximately) in trim. The rear rotors operate at a tip speed of 250 ft/s, and the front rotors operate at a tip speed of 230 ft/s. This produces a 1.46 percent error between the thrust that was thought to have been needed from the analysis in Determining Rotor Tip Speed.

2.4.8 Trimming Full Quadrotor Configurations: Hover

Similar to the full configuration cases which were used to trim the craft in forward flight, a series of cases were run to trim the quadrotor in hover. Two cases were run in ground and out of ground effect: one with motors operating at the perceived hover velocity given by the manufacturers data (see Figure 4 in the section Isolated Rotor Studies) and another at the perceived hover speed plus 20% (236 ft/s and 283 ft/s, respectively). In ground effect the quadrotor is said to operate 0.5 ft off of the ground. These values were chosen under the assumption that hover flight could be sustained between these two tip speeds. It was seen that the rotor-body effects also create a force on the quadrotor body in these flight regimes; this data was combined with the weight force of the craft in order to better approximate the net forces present on the quadrotor in hover (IGE and OGE). This force was not considered in the forward flight case for simplicity. The required motor speed is then found by interpolating between the thrust values found at both test motor speeds. These calculations are shown in Appendix E under Data. It is then seen that the approximate rotor tip speed required to place the quadrotor in trim in hover OGE is 270.67 ft/s, for hover IGE it is 246.57 ft/s. Appendix F details the data found in these cases, as well as the data trends and convergence seen in the hover cases.

2.5 Final Conclusions, Lessons Learned, and a Discussion of Future Work

2.5.1 Conclusions

A procedure for performing trim analysis on quadrotor platforms is presented and analyzed. A COTS UAS, the 3DR IRIS+, is used as the example craft in this work. The quadrotor was modeled using CAD

software and analyzed using the computational fluid dynamics solver RotCFD. A series of airfoil tables was generated, and the rotors of the IRIS+ were measured and modeled in RotCFD. From there, a trim analysis of the craft was performed, in which the effect of tilt angle and motor speed were examined through a series of isolated body/rotor studies. This provided enough aerodynamic data and rotor performance data to allow an initial trimmed configuration of the IRIS+ to be studied. From here, the effects of rotor-rotor and rotor-body interactions were approximated, and a final series of cases were run resulting in trim data for IGE and OGE hover, as well as steady forward flight at 20 mph. The approximate trimmed conditions for these three cases is shown in Figure1 below.

Table 1. Flight Parameters for Trimmed Cases

Trim Coefficients Found				
Study	Velocity (u)	Tilt Angle (i)	Motor Tip Speed (Front)	Motor Tip Speed (Rear)
Steady Forward Flight at 20 mph	20 mph	4.62 deg	230 ft/s	250 ft/s
Hover In Ground Effect	0 mph	0 deg	247 ft/s	247 ft/s
Hover Out of Ground Effect	0 mph	0 deg	271 ft/s	271ft/s

2.5.2 Lessons Learned

Due to the nature of computational fluid dynamics, a large quantity of time is required to run each solution set. Thus, a rigid run schedule and case organization is penultimate to the success of any rigorous CFD analysis process. Achieving convergence typically proves to be more difficult than had been originally planned for. In these studies, much time was spent determining what was causing divergence. From this, many lessons were learned. The feet were removed from the geometry input to RotCFD; this helped with grid generation, and helped with uniform grid generation. Convergence proved difficult while the fit bodies option was chosen in RotCFD, and while capturing the geometry in high resolution was a desirable goal, it proved unrealistic with the given time and computational resources. Different grid refinement specifications would cause divergence, and a few cases were run to find what levels of refinement can be achieved within reasonable amounts of time. It was learned that the general body refinement and general rotor refinement should be higher than the refinement level in the refinement box. 3D scanning the propellers was also pursued, however, this was later abandoned as taking hand measurements and graphically modeling the blade contours proved to be more accurate.

The entire trimming process was full of lessons as well. It was seen that the isolated rotors and bodies provided a very good baseline for finding the trim data, however, the rotor-rotor and rotor-body interactions that were present in the full configurations proved significant. This made the trim process more of an iterative one than originally intended. Eventually, this lead to trimming the full configurations via interpolation

like the other cases. In the forward flight studies, this required trimming the tip speed with regards to the front and rear rotors.

2.5.3 Future Work

A number of assumptions were made in both the set-up and in the analysis of the cases studied. Case parameters assumed constant flow conditions, which would change dynamically depending on location and weather conditions. Changes in these parameters would largely vary the fluid domain and its behavior in the presence of a rotorcraft. Further detail could be given to the overall dynamics of the system: an accurate placement of the center of gravity of the craft would allow for a very robust dynamic model to be created. Finally, a more detailed analysis, involving more flow parameters, could be examined. This would include modeling at different flight speeds, more tilt angles, and more motor speeds in order to form more accurate expressions for drag, lift, and thrust throughout differing flight regimes. Studying cases run at higher grid/parameter densities would require much greater computational power, but would provide more thorough and accurate solutions.

In the analytical process quantifying vertical lift and forward thrust disparities between the required and the generated, moment data was not considered. A more accurate trim process would include balancing the torque generated about the quadrotor at varying motor speeds and tilt angles. In steady state, it should be equilibrated to a net zero.

Computational characterization of the induced drag generated by the front rotors and affecting the performance of the rear rotors was not completed in this study. An analysis of the pressure gradient around the rotors and the thrust that they each generate in each of the trimmed cases would provide more useful information to determining the optimal thrust and tilt angle with which to operate, as induced drag changes the forces that the quadrotor experiences in flight. Similarly, future studies should explore the characteristics of quadrotor wakes and a second quadrotors ability to fly close by. Developing a robust analysis for the external airflow beneath and behind these quadrotors will be integral in performing safety calculations for autonomous flight.

Collected data may also be used in the creation of autopilot controls. Once generalized expressions for the performance of the craft are found, more robust autonomous controls can be created. Coupling this autopilot controller with other IMU or sensor controllers may aid in the generation of fully autonomous control.

Further work which could build upon these more detailed analyses would likely involve wind tunnel testing as a form of validation. Physical test-

ing would solidify the trimming process and complete the creation of a uniform trimming procedure for these relatively new rotorcraft.

3 Urban CFD Team

3.1 Abstract

Ensuring the safe flight of Unmanned Aerial Systems (UASs) is a crucial step as the commercial industry introduces more of these systems into civilian populations. The fear of wind turbulence damaging the trajectory of a UAV, and possibly even causing it to crash, is a real concern at the current stage of development of the systems. Additionally, the need for the maximization of route planning efficiency for companies handling these UAVs is a topic that can lead to considerable savings in energy and money. This paper will analyze what potentially hazardous areas a low flying UAV will have to interact with when traversing through a city. Furthermore, conclusions will be drawn regarding the feasibility of building a database of CFD test case results that can be pulled up and used whenever a UAS is sent on a mission. This data can be used in compliment with route planning algorithms in order to minimize travel time given a wind field and a final destination. This paper builds on past works in order to conduct testing, it is meant to serve as a stepping stone to the establishment of a more robust UAS route planning system.

3.2 Introduction

The future expectations for small UAVs are that they will be able to provide many services within urban areas in various environmental conditions while maintaining their safety and reliability. One of the biggest expected problems that UAVs will encounter is the variation of wind and turbulence at various locations around buildings while still maintaining their stability. Wind prediction by CFD analysis of various city landscapes has been done in the past for various biological and engineering applications such as expected spreading of diseases [11] or building aerodynamics [12], however the application of CFD in urban settings is relatively new and as such there isnt a wide variety of literature available for study. For applications concerning UAVs, it is important to be able to have an understanding of various wind environments that could be experienced during flight in order to better prepare for upcoming stability problems.

This paper examines the CFD results obtained from several cases ran using a model of the Ames Research Center, and how the data can be interpreted in order to allow for the safe flight of low flying UAVs in the area. The paper then goes on to use the CFD data and couple it with computer programming in order to find the minimal travel time it

takes to reach a destination accounting for distance traveled and wind velocity profiles. Ideally a company using a UAV (such as for package delivery) will receive a GPS destination and wind information from a nearby weather station (such as wind speed and direction). Then, from this information, the UAV would extrapolate within a reasonable amount of time a current wind field of the city, as well as create an optimized travel route. This paper details the CFD and the route optimization aspect of UAV flight as well as ties them together in order to summarize any results or specific areas that need to be researched further.

3.3 CFD of Urban Environments

The CFD program used for this application was OpenFOAM. This choice was made due to OpenFOAM being a reliable and heavily used open source code that offers an atmospheric boundary layer condition. The state of the wind was treated as incompressible due to the low atmospheric wind speeds being analyzed, steady-state to save computing time, and turbulent flow to mimic realistic flow characteristics. A slip wall boundary condition was applied to the buildings, although this is not accurate in a real scenario, for such a large scale test as this one, the error arising from making this assumption is negligible. For these flow conditions, the simpleFoam flow solver within OpenFOAM was used [13]. The geometry used for this test was a section of the Ames Research Center campus.

Several test cases were run in order to analyze the effects of wind speed and direction on the selected urban environment. Using online wind data, it was discovered that the most common wind direction in the Mountain View, California area came from north-northwest (this is to be expected as it lines up with the Moffett field runway), and the wind velocities peaked at around 20-25mph in the late afternoon (this data more closely represents the summer/ fall seasons) [14]. Using this data, a total of 12 CFD cases were processed. These cases were split into four categories based on direction: West, Northwest, North-northwest, and North. Each of these directions were then run at various wind velocities: 15, 20, and 25 mph. For each of these test cases, a cut plane was used 1 meter from the ground to capture flow characteristics of all buildings used within the geometry. One additional case (NW wind at 15mph) was set up with a finer grid in order to ensure that the errors caused by using a coarser mesh were not significant (local velocity differences found were less than 5 percent in value). Representation of all the data cases obtained is given in the form of figures in Appendix A of this paper (all units are in meters per second unless otherwise stated).

3.4 CFD Data Analysis

The objective of this data is to analyze how a UAS behaves as it traverses through urban environments. Particular areas of interest are those that can potentially cause flight trouble for a UAV, such as high velocity or high acceleration zones. Focusing the scope on high velocity zones gives rise to two main regions arising from the test results. The first are the regions where the buildings create an urban canyon setting. In these cases, a decrease in flow area causes an increase in velocity as is expected by the continuity equation of fluid dynamics. As shown in Figure 11, these wind funnels are particularly hazardous when previously uninterrupted by other buildings (as displayed by the arrow). The wind direction also plays an important role in determining the velocity of these canyons. In this case, wind coming in from the Northwest will provide a higher velocity in the area displayed than wind coming in from the North (which can be compared using Figure 12 (left)). This is expected to be caused due to the fact that the source of flow of the wind is aligned with the narrow passageways of the buildings within the Ames environment. For the case shown in Figure 11, wind speeds of up to 12 percent higher than the free stream were recorded in the urban canyon setting. Other high velocity locations arise in the eddies of building corners facing the free stream. These areas introduce velocities as high as 22 percent greater than the velocity of the free stream. Additionally, the velocity vectors wrap around the building introducing a high velocity gradient in the area. This can be potentially hazardous to any UAV passing by the area, such as causing it to hit a building if turning around a corner due to the sudden wind changes. These types of potentially hazardous Wind fields and their effects on small UAV's should be experimentally tested in order to determine their stability within these environments.

Although the most dangerous areas provided by the CFD cases appear to be the ones with highest velocity profiles, the wakes created by the buildings do have the potential to generate high acceleration magnitudes. The wakes, which are the areas of low velocities behind obstacles, can experience changes in velocities of up to 31 percent relative to the streets which they are directly next to. This introduces a considerable velocity gradient, and a potentially dangerous region. These situations become particularly difficult to deal with when combining the low speed of the wakes with the high velocities experienced within the urban canyons and the building corners. Thus, these situations present another possible situation for which UAVs flight capabilities should be tested in in order to ensure safe flight.

One interesting phenomenon can be observed as the same wind direction is viewed at different wind speeds. As shown in Figure 12, as the free stream velocity increases, the wind disturbance caused by the buildings in the lead becomes larger. This, in turn, creates a low velocity (relative to the free stream) region in front of and around the city model. This

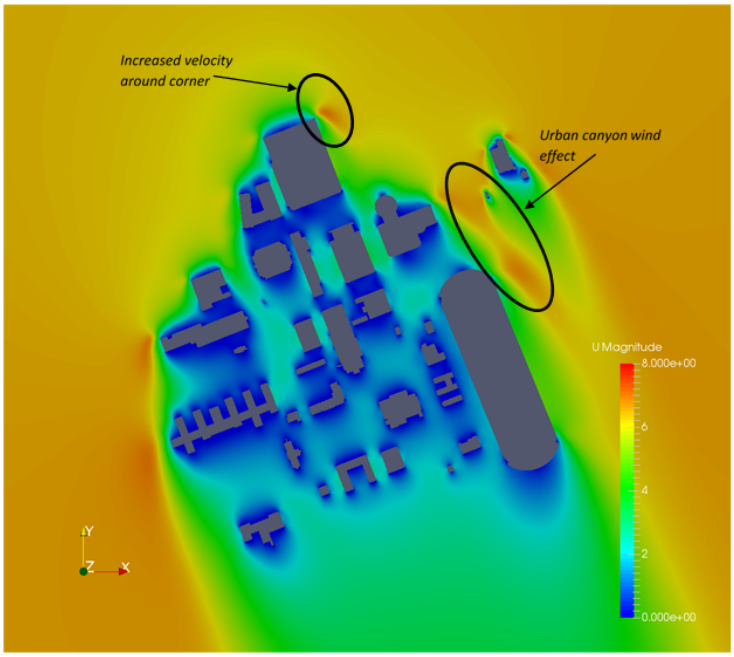


Figure 11. NW wind at 20mph showcasing high velocity regions

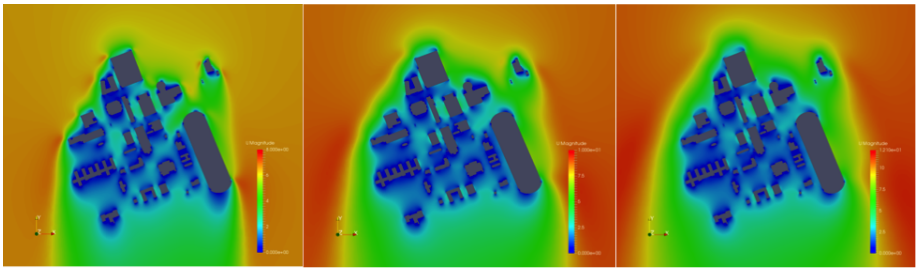


Figure 12. Northern wind at 15mph (left), 20mph(middle), 25mph(right)

bubble-like layer impedes the formation of the high speed wind canyons described above (or at least the ones directly facing the free stream), along with the high velocity points at the corners of the buildings facing incoming wind. Effectively, there seize to be any locations where the wind velocity inside of the city model is higher than that of the velocity of the free stream. As a result of this phenomenon of a protective layer created when the free stream velocity is high, UAVs flying can actually have a better chance of avoiding the higher velocity gradients within the urban environment.

Analysis of the three northern wind cases shown in Figure 12 can also be used to test how the velocity inside the model varies as the free stream speed is increased. Ideally an increase in the velocity of the free stream would result in a proportional increase to the velocity inside of the city model so that any set of data could be interpolated from a baseline set

of known profiles. Taking a closer look at the data however (the figures can be seen more closely in the appendix), does not appear to reinforce this assumption. There does not seem to be a clear linear relationship between a change in the velocity of the free stream and that of the flow interacting with the model. In order to make a proper conclusion, many more test cases need to be run along a larger range of varying speeds. Although it is unfortunate that a clear solution concerning wind flow prediction did not present itself in this case, it is not surprising. As described above, an increasing free stream velocity actually saw a relative decrease in the velocity at the leading edge of the model. Thus, hinting at a nonlinear nature between varying flow speeds. Nevertheless it does beg the question of what the most efficient method would be to obtain a particular wind field when needed.

3.5 CFD Discussion

Overall, urban CFD is not a highly evolved area when compared to the applications of computational fluid dynamics in aerospace. As a result, significant emphasis must be placed on the fact that the data presented from these results is but a rough average of what the estimated wind velocity in Ames actually is. Actual wind profiles are subject to the sporadic tendencies of nature. Although the data presented in this paper leans on the North-Western wind directions (which are predominant most of the time in this region), there are occasions (particularly during the winter) where the wind can originate from the South and East. Thus, instead of testing every possible scenario, it is more efficient to test a few representative settings and then apply the data gathered to all other comparable circumstances. Another inherent result of the incipient stages of city CFD is the lack of CAD models available to study. The model provided to us of the Ames campus was not completely accurate. Several geometrical details were left out of the building designs and a large chunk of the buildings were not included in order to save computational time. Although using a model of Ames can provide unique advantages to engineers within the region, such as being able to test actual wind data with ease and applying fairly high wind velocities that are actually present on site, errors can arise from a basic constructed CAD model. As this area of CFD application grows, it will be more important to raise awareness of the lack of detailed urban CAD models and their need for them, as well as establishing a system to update these models as time progresses due to new construction and other causes. As UAVs become more popular, and the CFD becomes more necessary, these issues will need to be addressed in order to properly progress and maximize efficiency in the area.

Regarding the analysis described, it is important that UAS's are tested and their responses recorded in areas where high wind velocities and accelerations are present. The original goal for these test cases was orig-

inally to test them on the NASA Ames in-house developed simulation program Reflection. However, due to the time constraint within the MARTI program, this was not accomplished. Continuance of this experiment, therefore, would include the testing of small UAS's in these simulation environments with the final step being the completion of real tests to validate the CFD data. Another sub-team of the MARTI 2015 NASA academy is working in parallel to this project on a wind rejection control algorithm, which could be implemented on-board a UAV. It would be beneficial to test this control algorithm along with the CFD data in order to see if any of the identified hazardous areas can be made safer. Ultimately, urban CFD data can be used in order to create a large database from which companies can draw information from whenever needed. As mentioned previously, interpolating data from benchmark cases is not guaranteed to work, however the wind directions in a city do not vary widely in a day (except in abnormal conditions such as storms from which UAVs should refrain from flying in), and small changes in the free stream velocity do not create considerably large changes anywhere in the model. Therefore, it is not necessary to have an extremely large amount of data in order to closely represent every possible wind field a city can experience. Additionally, a substantial amount of memory can be saved by simplifying the CFD results once obtained. Mesh sizes can include up to millions of cells, while a company attempting to obtain a rough estimate of the wind in an urban setting may not need such a precise accuracy. Therefore, obtaining the CFD results and then averaging the values to reduce the number of data points can be a useful technique when data storage is an issue. Conclusively, using the example of a drone delivery company, if a business obtains wind information from a local meteorologist and it wants to know what the expected wind fields within the city will look like, all the company needs to do is hold a database consisting of CFD data for several different wind directions and at different speeds.From this method, the company can have a close estimate of what the wind looks like in the city without needing an unreasonably large amount of data storage.

3.6 Route Optimization Algorithm

The optimization code written for this project minimizes the amount of time it takes to traverse through two dimensional space given a wind velocity field. The program is a modification of a code written by MathWorks [15], and works by using the MATLAB function fmincon. This function finds the minima given a set of constraints. The reason that this code was chosen was because it was already easily accessible and worked perfectly for the purpose of this experiment. It is possible that more efficient algorithms are available however, at this moment the concern is simply a proof of concept. The function to be optimized is the travel time (as shown in appendix C) and is dependent on the way points

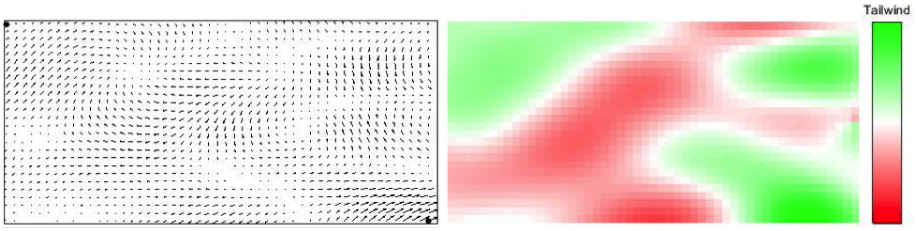


Figure 13. Randomly generated wind field as shown by vectors (left), and as shown by favorability (right)

used, the wind velocity components, the dimensions of the wind field being used, as well as the endpoints. The first step is to set up a wind field with the vertical and horizontal components of velocity. For the first case, a random wind field is generated using the MathWorks created program makeWindFun.m. Then, as shown in Figure 13 (left), the resulting field can be viewed using the quiver command in MATLAB, which displays vectors in space. Then, this code goes on to add color by distinguishing between a headwind and a tailwind (also shown in Figure 13 (right)).

Once all initial parameters have been set up (such as those shown in the beginning lines of Appendix B) and an initial guess has been input, the way points are interpolated cubically in order to form a continuous line. These values are then differentiated and the dot product is applied to the i and j velocity components (which have to be bi-linearly interpolated along the way point path) in order to calculate the velocity in the direction of travel (this process is written in Appendix C). The time it takes to traverse this infinitesimal distance is then defined as dt and is given by the equation

$$dt = dx/v \tag{9}$$

where dx is the distance between the path points and the velocity v is given by the sum of the velocity of the vehicle and the wind at that point in space. The total travel time is then the sum of dt along the entire path. It is important to note that in order to provide more accurate information regarding the actual speed of the vehicle, precise measurements of mass and other body constants have to be calculated. Simply adding the velocity of the UAV and the wind does not accurately deliver the actual pace, although for this purpose it is a good approximation. The optimization script then calls the way points as a function handle and applies a lower and upper bound in order to ensure that the values do not leave the dimensions of the velocity matrix. Finally, the fmincon function is called to iteratively solve for the optimum position of the way points in order to minimize travel time. Figure 14 shows an initial estimate of the fastest route versus the optimum calculated route. It

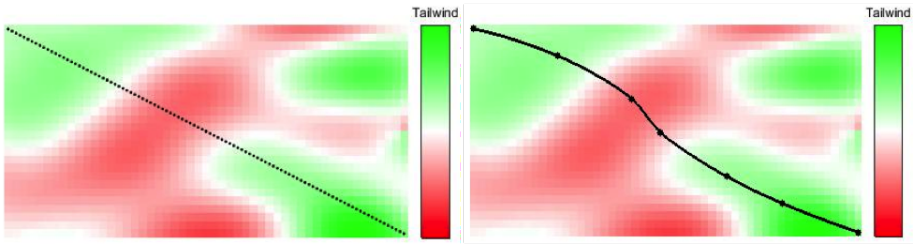


Figure 14. Expected fastest route (left), optimized route (right). The optimized route shows a 2.45% savings in time

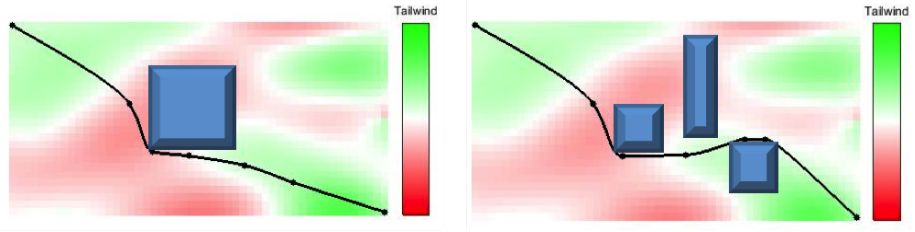


Figure 15. Obstacle implementation example, and the effect on the path

can be seen from the results that the algorithm attempts to avoid any areas of strong headwind, while at the same time taking advantage of any tailwinds. What is originally the shortest path does not necessarily correlate to the shortest time due to wind interference; the result (as seen in the figure) corresponds to a saving of 2.45 percent of the travel time. Although it is not necessarily a considerably large value, it can mean substantial energy savings when thousands of UAVs are flying daily. Furthermore, understanding various flight paths for different wind fields could potentially lead to larger savings in time.

The original program can be modified (as is the case in the OptimizationPath.m file shown in Appendix B) in order to implement obstacles along the trajectory path. These obstacles in turn can act as buildings that a UAS traversing the path will have to avoid. In this way, the code can be modified to implement cities or small towns and create optimized routes for UAVs and other small, low-flying vehicles. For this code, the obstacles are implemented one at a time, and are treated as extremely high velocity wind fields that oppose the trajectory of the path. Figure 15 depicts some examples of different routes as the obstacles are varied along the trajectory path. It is important to note that the fmincon function allows for the use of constraints and can thus allow for more robust methods of implementing this idea. However, again, the objective here is simply a proof of concept, and is in no way claimed to be the most efficient or effective way of adding obstacles.

Once the MATLAB command has been thoroughly understood, the final step is then to implement a full scale model into the simulation. For this

case it is possible to export CFD data obtained of the NASA Ames model described previously, and save it as an Excel file; specifically, the data being looked at consists of the u and v components of velocity and their position in two-dimensional space. Importing this data into MATLAB and applying the quiver command, displays a vector field of the wind velocity, as shown in the left hand side of Figure 16. It is important to note that due to the nature of CFD meshes the vector coordinates are not uniform, while the MATLAB code relies on a uniform vector field in order to function properly; as a results the data has to be modified in order to be used. For this scenario, a loop was written that averaged the values of the nearest wind velocities at each grid coordinate; Using this principle, it is possible to create a uniform mesh, although the buildings themselves have to be implemented completely separate, as carefully defining their boundaries in MATLAB is necessary for accurate results. Tackling the problem of implementing the buildings is more difficult in real scenarios such as this one due to the fact that they are seldom perfect rectangles aligned with the coordinate axes. In order to properly implement buildings the function buildingAdder was written (which can be found in appendix F), for which the user inputs the four coordinate positions (in the x and y) of a buildings corners and the program then connects the lines and generates a wind vector of very high speeds along those lines; it is important to note that this code still needs to be improved before being efficiently implemented.

As a proof of concept, a small section of the Ames map (as seen in Figure 16) was taken and edited until it was ready to be implemented into the main algorithm (note that for this example the buildingAdder function was not necessary since it is very small scale); the parameters were adjusted to match the new set of data, and the code ran successfully (The result is shown in Figure 17). The reason for using only a small section of the Ames map is due to the current lack of an efficient method for correctly depicting all of the buildings and generating a uniform mesh, even with the use of the codes described above, the overall process is still time consuming and some bugs need to be fixed.

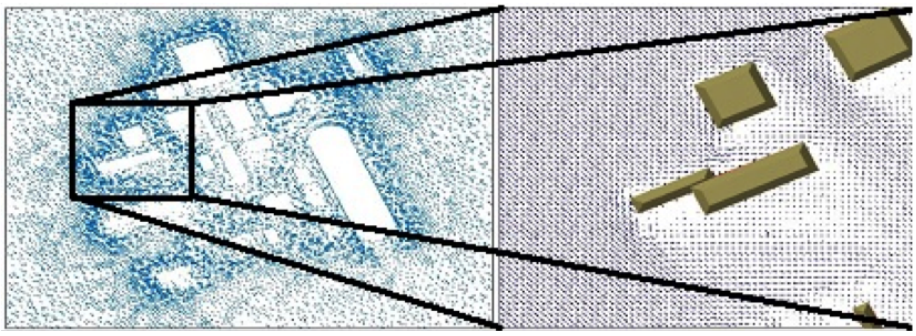


Figure 16. Ames wind velocity vector map obtained from CFD results (left), smaller section (right)

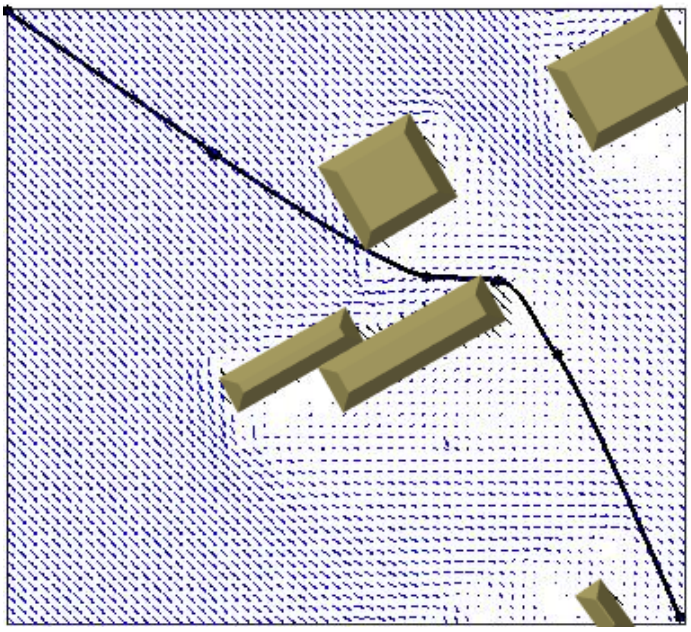


Figure 17. Application of optimization algorithm on Ames section

3.7 Path Optimization Discussion

It is important to note that this code is not perfect. It uses an iterative solving mechanism and can thus, at times, not converge to a correct solution. Furthermore, another limitation of this code is that it is only a two dimensional solution. A full solution would require looking into the height at which the UAV travels as well, although doing this would cost considerably more computing power, and may not necessarily be effective in all environments. The current method used to add in obstacles uses high wind speeds in order to encourage the algorithm to avoid that area. However, if the code is to be studied further, it may be more effective to set a constraint in order to avoid the obstacles so that the program does not, under any circumstance, place a way point in a building. Lastly, in order to create a more robust model, it may be necessary to integrate a time dependent system. The wind can never be completely predicted, but it may be capable of being efficiently averaged. Even as wind directions change throughout the day, or throughout the UAVs flight, it is possible, through the use of transient CFD codes, to adapt the program to account for changes in way points even throughout flight (the original code written by MathWorks has an example of this). In any case, even with rough predicted wind averages, given by forecast information and steady-state CFD data, it is highly likely that vehicles are capable of saving time by choosing a more optimum path.

The proliferation of low flying vehicles in urban environments introduces more systems to areas of highly varying wind speeds. This presents new challenges that have not been heavily studied before, and thus provides the opening for possibilities of new solutions. Overall the implementation of a wind optimization code, can, in the future, lead to considerable savings in energy and money. The number of UAVs in the sky today is not necessarily large, but as the rules and guidelines are set up by the FAA and NASA [16], and commercial industries begin to utilize the systems, there will be an exponential increase in UAVs, making for a good reason to maximize efficiency wherever possible. The code used for this study does still need to be improved at several stages, as described in previously. However, it can serve as a initial stepping stone for any future work within this field.

3.8 Conclusion

The amalgamation of the two subtopics covered, including urban CFD and the route optimizing algorithm in MATLAB, is essential, as neither can be fully utilized without the other also being taken into account. Thus, the discussion of how these two fields interact together should be further studied. It is interesting to note that in the CFD discussion it was pointed out that corners around buildings facing the wind experience particularly high wind speeds, while the optimization code passes

extremely close to building corners (as seen in figure 15) in order to conserve time, and perhaps even to use any tailwinds in that area. This can be, of course, particularly dangerous as the UAS can be trapped in a strong wind and lose its track or crash into the side of a building. In order to prevent this hazard, a larger geo-fence can be placed around any such areas in order to introduce a factor of safety when flying. Furthermore, preventative measures such as geo-fencing can also be used in order to avoid areas where the acoustic vibrations of the UAVs can be potentially disruptive. Strategies such as these however, would depend on the specifics of the vehicle that is flying. Furthermore, it is necessary to analyze at what speeds a particular UAV is incapable of flying in. Even if the wind is a tailwind, it can potentially cause trouble for any UAV that is not designed for these conditions. It will also be of good practice to perform flight experiments in order to test this for each particular UAV, and ,depending on the results, adapt the route optimization algorithm accordingly in order to avoid locations where average wind velocities exceed dangerous levels.
Ultimately the data obtained from this paper can really be of use to any company or organization that will use a large amount of UAVs to travel around a particular setting. As brought up previously, a database of CFD scenarios can be coupled with the optimization algorithm presented here in order to obtain the most efficient route if the company merely knows the GPS destination and easily accessible wind data. If a system such as this is well built, it can account for quite a large amount of energy and money savings. Further improvements can be made through the implementation of a wind rejection algorithm (which was briefly mentioned as being worked on by a subgroup of the NASA MARTI 2015 academy), which may allow the UAV to traverse through paths that it was previously incapable of doing, potentially saving even more time. Testing of this entire system can be accomplished in a simulation environment (such as through NASA's in house simulation software: Reflection) before actual flight testing. A robust test should include the dynamics of the wind obtained from CFD results, the wind rejection algorithm, and the way points derived from the optimum route program. If this is successfully accomplished, then further tests can examine the use of transient CFD data or the addition of a third dimension of space.

4 Controls Team

4.1 Abstract

Operation of Unmanned Aerial Systems (UAS) within fifty feet of the ground requires an added level of redundancy and safety from all subsystems on board. Given the higher density of obstacles and civilians at lower altitudes, extra care must be taken by designers of UAS meant to fly close to the ground. UAS must be able to recover or land safely after a

motor failure, maintain controlled flight in turbulent environments, and monitor their own health in order to predict potential failures. We build on past work to address each of these areas of concern for UAS by focusing on a motor failure detection algorithm, wind estimation algorithm, and battery prognostics.

4.2 Background

As UAS's continue to be developed for use in the commercial, research, and military worlds, a wide variety of application scenarios must be considered. Since small, unmanned, multirotor vehicles are the platform of choice for planned aerial delivery applications, multirotors are an appropriate focus for the safe autonomous flight environments within fifty feet project (SAFE50). Specifically, this project focused on quadcopters in order to develop control algorithms to reject wind disturbances and detect and account for failed motors.

Multirotors with more than four motors allow for system redundancy, so quadcopters with failed motors offer an interesting control system challenge. Even with a control algorithm, if a quadcopter has one motor failure, it can no longer control its altitude and attitude, and so it must maintain a reduced attitude.

Wind disturbance rejection was also chosen as it represents a significant challenge for UAS flight in low-altitude environments. The extra obstacles present below 50 feet can cause unexpected gusts of wind that can have strong effects on small multirotors if their control systems do not account for the strong drag forces. Developing controls to reject these disturbances will allow UAS's to fly more safely through urban environments.

UAS battery health monitoring was selected as it is a vital component of ensuring reliable energy systems. Accurately predicting end of discharge and end of usable life events will allow the UAS to maintain safe flight and successfully navigate urban environments.

4.3 Motor Failure Procedure

Current control allocation algorithms expect all motors on-board a multirotor vehicle to be operating nominally. However, motors can fail in several ways such as by having a propeller break or vibrating loose and the motor becoming jammed. With a normal flight controller, either of these problems will render any multirotor uncontrollable and the UAS will fall spiraling out of the sky.

Figure 18 shows the free body diagram of a typical quadcopter system. This graphic demonstrates that cutting out one of the motors will cause an imbalance in both the thrust profiles and torque on the vehicle. If left unaccounted for, these unbalanced forces will cause the quadcopter to flip and spin uncontrollably. If there is a timely response to a failure,

the vehicle can be controlled and landed safely. Using a reconfigurable controller, like the one described in Stepanyan 2016, an impaired motor can be detected and the UAS can be controlled and landed safely [17].

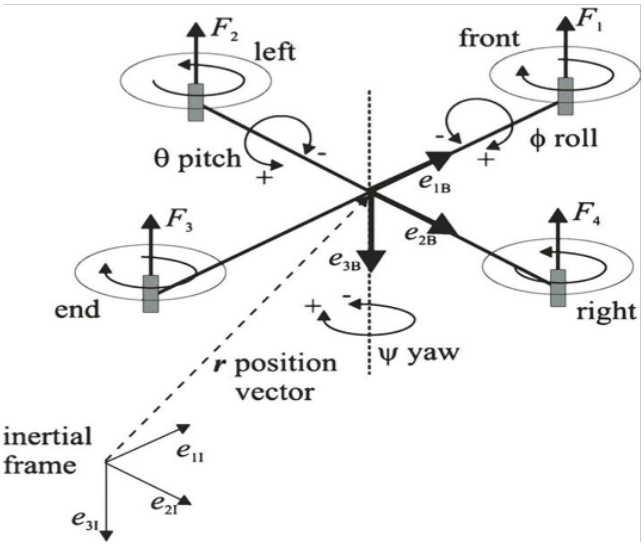


Figure 18. A FBD of quadcopter from sysmagazine.com/posts/183964/

The motor failure algorithm developed for this project is broken up into three different blocks.

- The first is an identification block that identifies any failed motors. This is done by detecting the current drawn by each motor and comparing it with an expected value. Each motor's expected current draw can be computed based on motor parameters, the instantaneous voltage input, and a mathematical model of the motor. If there is a large difference between the expected and actual current being used by a motor, the motor is assumed to have failed. This allows for a simple way to ID a failed motor.
- After a failed motor is identified, this information is sent to a decision making block that updates the motor mixer matrix with a new control matrix without the failed motor.
- The last block is a motor mixer that converts the controller outputs of commanded thrust, roll torque, pitch torque, and yaw torque into individual motor commands using the updated motor mixer matrix.

This algorithm allows for the detection and safe control of a multirotor with one or more failed motors. Additionally, if a motor comes back online, this algorithm can update the motor status vector and return full control to the UAS.

The controller designed to accompany this motor failure algorithm was also developed in Stepanyan 2016. It is a cascaded dynamic inversion controller which is a common control method in aviation. [17] The controller is based off of the translational dynamics in equation 10 and the rotational dynamics in equation 11.

$$\begin{aligned} M\ddot{x}(t) &= T(t)[\cos\phi(t)\sin\theta(t)\cos\psi(t) + \sin\phi(t)\sin\psi(t)] - k_t\dot{x}(t) \\ M\ddot{y}(t) &= T(t)[\cos\phi(t)\sin\theta(t)\sin\psi(t) - \sin\phi(t)\cos\psi(t)] - k_t\dot{y}(t) \\ M\ddot{z}(t) &= T(t)\cos\phi(t)\cos\theta(t) - Mg - k_t\dot{z}(t) \end{aligned} \tag{10}$$

$$\begin{aligned} J_1\dot{p}(t) &= -(J_3 - J_2)q(t)r(t) - J_{r3}q(t)\Omega(t) + \tau_1 - k_r p(t) \\ J_2\dot{q}(t) &= -(J_1 - J_3)p(t)r(t) - J_{r3}p(t)\Omega(t) + \tau_2 - k_r q(t) \\ J_3\dot{r}(t) &= -(J_2 - J_1)p(t)q(t) + \tau_3 - k_r r(t) \end{aligned} \tag{11}$$

As in Stepanyan 2016, we neglect the moment of inertia of the motors, (J_{r3}), as it is considered much smaller than the principle moments of inertia of the multirotor. [17]
Once the dynamic inversion controller calculates the commands for T, τ_1, τ_2, and τ_3, those commands must be translated into motor forces. By using the geometry of the multirotor, a B matrix may be populated which will determine the net thrust and torques on the multirotor given the forces generated by each motor. A general B matrix is shown in equation 12.

$$\underbrace{\begin{bmatrix} T \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix}}_{\text{u}} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ b_{11} & b_{12} & b_{13} & \dots & b_{1n} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2n} \\ \pm a & \mp a & \pm a & \dots & \mp a \end{bmatrix}}_{\text{B}} \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}}_{\text{f}} \tag{12}$$

T is the total thrust generated by the motors. τ_1, τ_2, and τ_3 are the torques generated by the pitching, yawing, and rolling moments. The b_{nn} coefficients in the matrix are derived from the moment arm of each motor on the multirotor, while the a coefficients are determined by the lift to drag ratio of the propellers. f_n are the forces generated by each individual motor.
In order to obtain the matrix which allocates motor forces based on the four commands, we follow Stepanyan 2016 and simply use Moore-Penrose inverse to calculate an optimized "inverse" matrix, as in equation 13.

$$\boldsymbol{f} = B^T(BB^T)^{-1}\boldsymbol{u} \tag{13}$$

In the case of a quadcopter with no motor failures, there are exactly as many motors as there are inputs, so a simple inversion will produce the motor commands, as in equation 14.

$$\boldsymbol{f} = B^{-1}\boldsymbol{u} \tag{14}$$

As motors on the mulitrotor fail, the algorithm simply eliminates the appropriate column from the B matrix and updates the allocation matrix according to the appropriate inversion technique. That said, the remaining motors must be able to produce enough thrust in order to overcome the weight of the platform and stay in the air if the mission is to continue.
Once less than two counter-clockwise rotating motors or less than two clockwise rotating motors are available, the algorithm eliminates the fourth row of the B matrix to sacrifice yaw in order to maintain control of altitude and the reduced attitude. This way, even a quadrotor with a failed motor or an octocopter with no available clockwise motors can still land in a controlled fashion. In simulation, a controlled landing was achieved despite the reduced attitude control.
Though beyond the scope of this project, if yaw mode needs to be sacrificed on board, an algorithm needs to be developed such that a UAS can determine where and how to land safely once the failure occurs.

4.3.1 Simulink model

To do our analysis, we modeled a quadcopter, the controller, and the algorithm in Simulink and Matlab. Figure 19 shows the fully integrated disturbance estimator and motor failure algorithm in a Simulink diagram. The Simulink model is capable of running both the motor failure program and the wind estimator simultaneously.

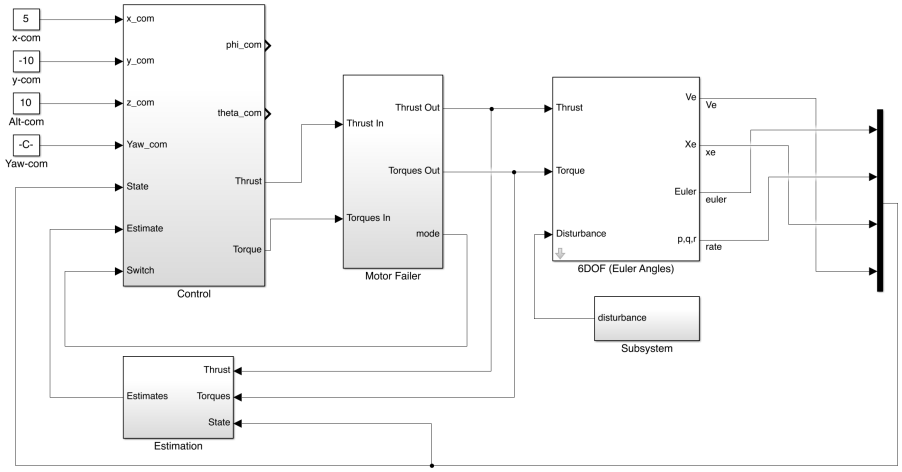


Figure 19. Simulink Model.

Shown in Figure 20 is a simulation of a motor failing while in flight. The quadcopter is ordered to ascend to 10 meters. At 15 seconds, a motor fails. Then, the quadcopter is ordered to hover at 5 meters while holding a 0 degree roll and 0 degree pitch, so it descends. It then maintains a stable hover for the rest of the simulation.

Figure 21 shows the positions of the quadcopter in the X and Y axis. This figure is pulled from the same simulation as shown in Figure 20. After the motor has failed at 15 seconds, control in the X and Y directions is less stable than before the motor failure. However, it is still able to control the reduced attitude and it is stable in this state. The sinusoidal characteristics of the X and Y positions is due to the wind disturbance forces being sinusoidal inputs.

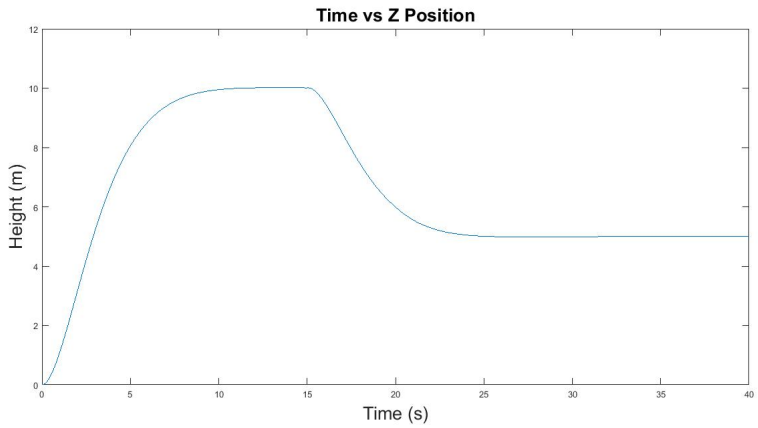


Figure 20. Simulated motor failure.

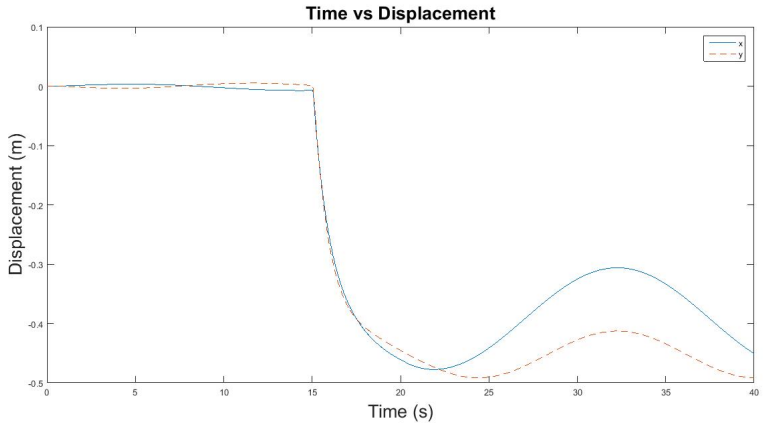


Figure 21. Positions in the x and y axis during the same simulation as in Figure 20

Figure 21 shows the position of the quad during the failure event. The quad maintains its position at the origin, the motor fails at 15 seconds, and then the quad translates slightly in both axes. Oscillations are present in X and Y after the failure because the reduced attitude controller cannot reject the wind disturbances as well as the full attitude controller.

After 15 seconds, the quadcopter gives up yaw control in order maintain controlled flight. Figure 22 shows the yawing of the quadcopter after the motor failure event at 15 seconds. The spinning accelerates until it reaches a terminal rotational velocity due to rotational drag on the vehicle balancing with the unbalanced yaw torques from the motors.

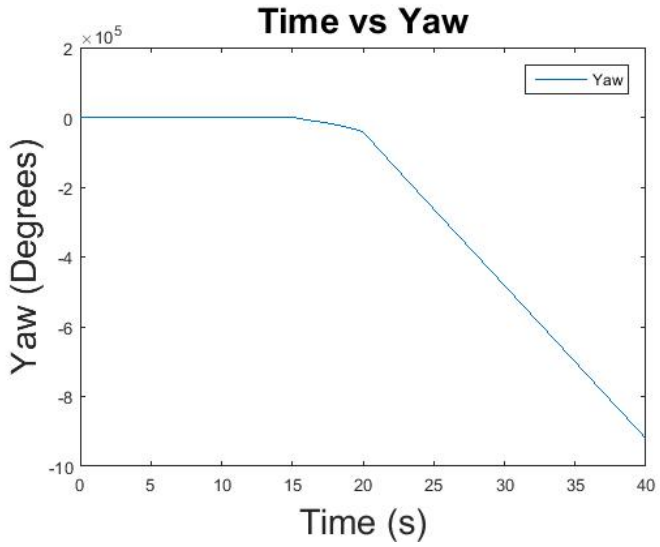


Figure 22. After the motor fails at 15 seconds, the quad yaws out of control as expected.

4.3.2 Hardware Validation

Hardware testing was conducted using the Robot Operating System (ROS) and a Parrot AR.Drone 2.0. Using the Robotics Toolbox from MATLAB, we are able to issue velocity commands to the AR Drone over Wi-Fi. In this way, we can simulate the dynamics of motor failure on a vehicle that is not equipped to fail a motor.

ROS works on the concept of publishing and subscribing data to various "topics". ROS nodes are able to publish commands or sensor data as well as subscribe to topics to collect this data. The AR Drone acts as a ROS node and subscribes to a topic called "cmd_vel" which is the commanded linear and angular velocities (V_x, V_y, V_z, p, q, r). These commands follow the topic format given by the "geometry_msgs/Twist". Once the command is received, the AR Drone mixes the command and allocates the forces/torques to each motor by itself. At the same time, the Drone measures and publishes information about its current 12-variable state (positions, euler angles, rates) as well as a number of other pieces of data.

From the computer side of ROS, Matlab/Simulink sets up a global node and connects into the ROS network using the computer's IP address. From there, it has the same publishing and subscribing capabilities as

any other node. To publish data from Simulink, we use the "publish" block from the Robotics Blockset as shown in Figure 23.

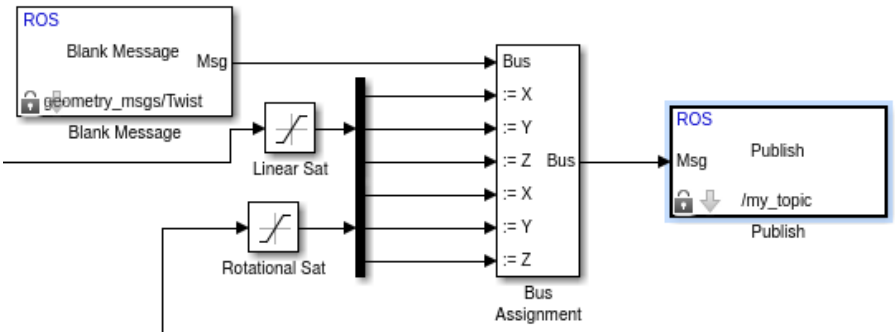


Figure 23. Block wiring to publish linear and angular velocity commands from Simulink

The "Blank Message" block sets the data type formatting and the "Bus Assignment" block populates the message with useful data.

4.4 Drag and Wind Disturbance Estimation and Rejection

Another important aspect of flying UASs in low-altitude environments is compensating for disturbances caused by wind. In urban environments in particular, corridors of large buildings along the streets create urban canyons that can amplify existing winds and generate unstable eddies in wind flow. In order to compensate for such disturbances, the cascaded dynamic inversion controller must be able to model and account for the atmospheric disturbances and drag coefficients in real time. This is done by using an adaptive estimator described in [18].
The implemented estimator aims to estimate 12 terms: the instantaneous drag coefficients in the three translational directions and the three rotational directions and the instantaneous disturbance forces in the three translational directions and the three rotational directions.
From [18], the UAV's translational atmospheric drag can be modeled as:

$$\big[D(t)\big] = \begin{bmatrix} -k_{tx}[v_x(t) - w_x(t)] \\ -k_{ty}[v_y(t) - w_y(t)] \\ -k_{tz}[v_z(t) - w_z(t)] \end{bmatrix} \tag{15}$$

Where k_{tx}, k_{ty}, and k_{tz} are the unknown drag coefficients, $w_x(t)$, $w_y(t)$, $w_z(t)$ are the unknown local wind velocities. The rotational drag coefficients and local vorticities can be modeled similarly.
The adaptive estimator that was developed estimates the twelve desired terms iteratively in two steps [18]. In the first step, estimates for the current translational and rotational accelerations are calculated using

the dynamics of the system. An example in the x-direction is shown in equation 16.

$$\begin{aligned} M\dot{\hat{v}}_x(t) = T(t)[&\cos\phi(t)\sin\theta(t)\cos\psi(t) + \sin\phi(t)\sin\psi(t)] \\ &- \hat{k_{tx}}(t)v_x(t) + \hat{d}_x(t) + \lambda\tilde{v_x}(t) \end{aligned} \tag{16}$$

Here, $\dot{\hat{v}}_x$ is the estimated acceleration in the x-direction, $\hat{k_{tx}}$ is the estimated drag coefficient in the x-direction, $\hat{d}_x$ is the estimated disturbance force in the x-direction, $\tilde{v_x}$ is the error between the estimated x-velocity and the actual x-velocity, and λ is a design parameter.

Once the estimated accelerations are calculated, they are integrated to get velocity estimates. These velocity estimates are used to calculate the derivatives for the 12 estimated drag terms. The derivatives were determined using a Lyapunov function in [18]. Once the derivatives are calculated, they are also integrated and fed into the cascaded dynamic inversion controller so it can account for the drag and disturbance terms. At each time step of computation, either onboard a UAS or in simulation, these derivatives will be calculated and used to update the appropriate state and disturbance terms. By incorporating this estimator into the controller, the UAS will have a smoother flight in windy conditions and will be less likely to lose control in a turbulent environment.

4.4.1 Simulink Model

Using the concepts and equations from [18], a Simulink Model of the system was developed. A sinusoidal disturbance term in all 6 axis of motions was implemented into the Simulink model. Figure 24 shows the results of a Simulink simulation, with the instantaneous drag in the X, Y, and Z directions plotted against time. The exact translational drag given was set to be .3. The quadcopter was ordered to move 10 units in the X, Y, and Z directions. In Figure 24, the spike in instantaneous drag can be seen between zero and ten seconds. After the multi-rotor stops moving, the drag estimates settle down to a lower position. While the estimator does not converge to the set value of 0.3, it does allow the simulation to smoothly execute its mission with despite unknown drag coefficients and forces.

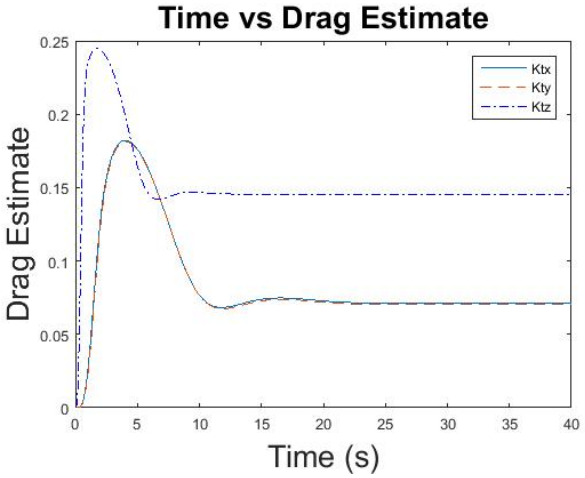


Figure 24. Instantaneous Translational Drag Estimates

Figure 25 shows the same case as Figure 24, except with rotational drag coefficients graphed. The true value of the rotational drag coefficient was set at 0.5. Again, while the estimator does not converge to the set value, the simulation does smoothly execute the mission.

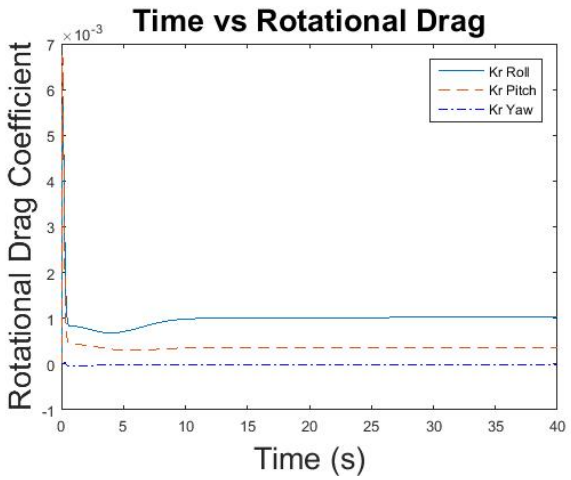


Figure 25. Instantaneous rotational Drag Coefficient Estimates

Figure 26 shows the instantaneous disturbance forces on the UAS as it moves. Although the estimator does not converge to the true simulation value of 1 Newton, it clearly shows a sinusoidal oscillation is present. The same is true for the disturbance torques shown in figure 27.

Using this type of control system would result in an accurate wind disturbance estimator, allowing for finer control of the vehicle. If flight is to occur in an urban environment, being able to effectively model the disturbance terms

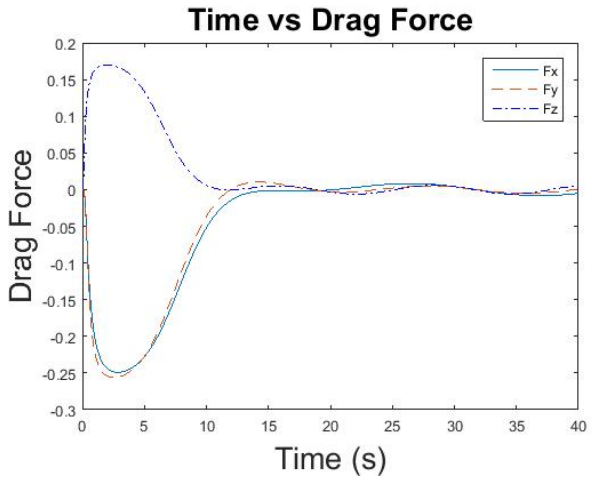


Figure 26. Instantaneous Drag Force Estimates

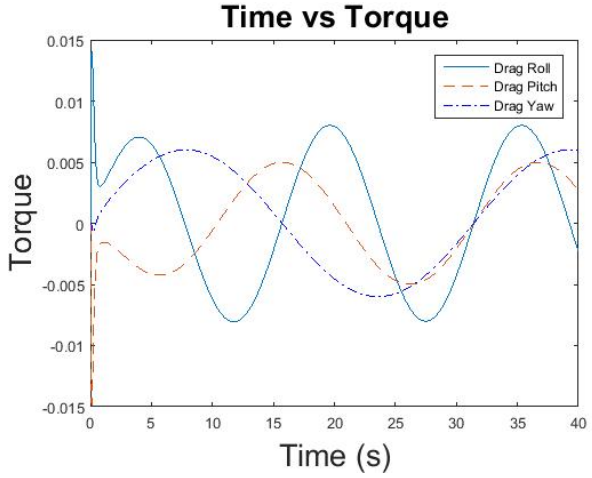


Figure 27. Instantaneous rotational drag torque estimates

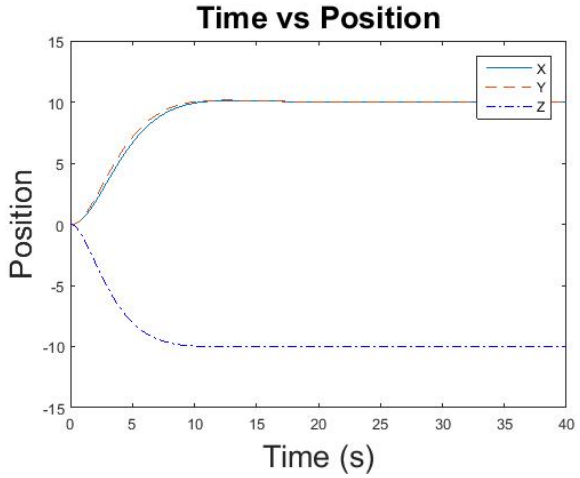


Figure 28. X,Y, and Z positions with the drag estimator

Figure 28 shows the position against time. The drag estimates allowed for smooth flight to the desired locations, even with wind disturbance terms inputted into the simulation.

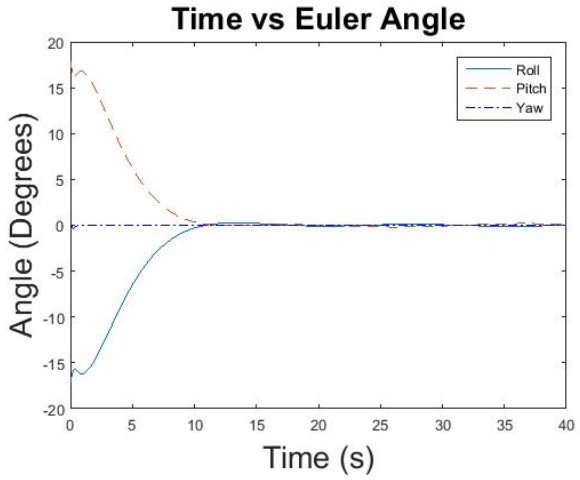


Figure 29. The Euler angles vs time during the flight

Figure 29 shows the Euler angles during flight. The angles change smoothly during movement,but settle quickly after desired location is reached.

4.5 Battery Modeling

Electric UASs need to have reliable energy systems in order to ensure that they fly safely. Battery health monitoring is vital for accurately predicting end of discharge and end of usable life events. Mission planning and decision for UAS needs to be able to rely on battery health estimation and prediction in order to understand what types of missions are achievable. For example, a package delivery system that lifts a 100 g package will require a smaller amount of energy than a 500 g package. In this manner, the UAS will need to recognize that it will not be able to transport the 500 g package as far as the 100 g package and adjust its mission plan accordingly.

UAS battery conditions throughout flight need to be assessed in order to determine the extent of deviation or degradation that the battery system endures throughout flight from expected normal operating conditions during different failure scenarios. Test flight simulations of the battery can inform an energy storage fault tree analysis that would determine UAS protocol during failure events.

Internal battery cell failure mechanisms such as dendrite growth and plating will occur over time in accordance with the particular chemistry and construction of the battery. These failure events will need to be assessed to help characterize their behavior and timing. Externally caused

failure mechanisms in the mechanical and electrical realms can be primarily prevented through charging management and component design. While the end of typical battery health is vital to assess, these cell level failure mechanisms may trigger a relatively shortened battery life. These events may not be as consistent, but they necessitate intelligent controls in order to predict these events early and adjust mission way-points and tasks accordingly.

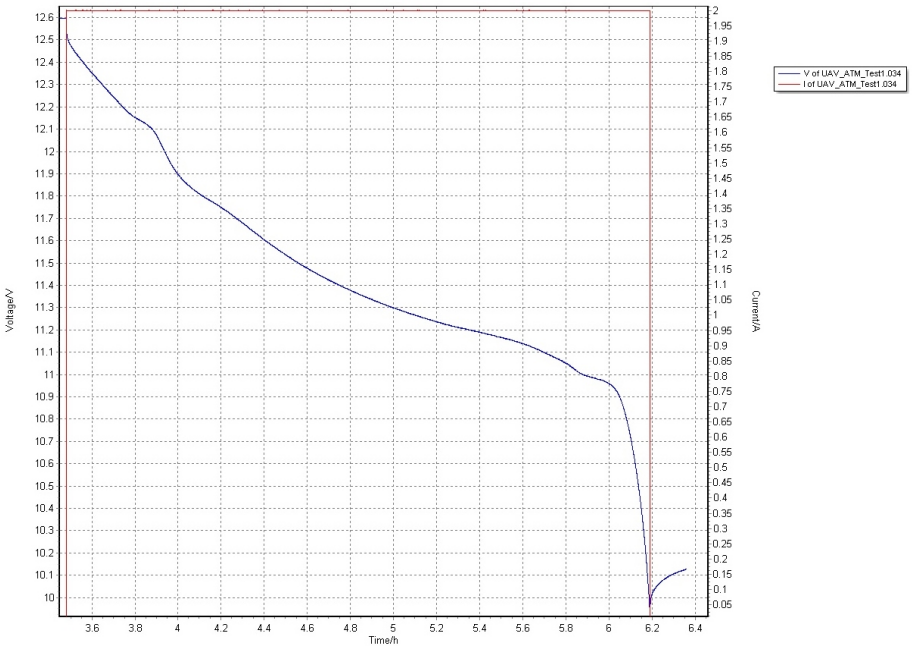


Figure 30. Iris+ 2A Discharge profile

Battery performance depends on how the battery is used and the environmental conditions in which they are placed. The Iris+ lithium polymer (LiPo) battery was selected from 3DRobotics as the test battery [19]. The MACCOR instrument in the NASA Ames Systems Health, Analytics, Resilience, and Physics modeling (SHARP) laboratory was utilized in to obtain a 2A discharge profile of the LiPo battery as in Figure 30. By continually discharging the battery at 2A, the voltage response over time was recorded. This matches a new LiPo battery characteristic discharge curve and represents a low charge rate due to the long time duration. The effective capacity of the cell is reduced when discharged at high charge rates. This capacity offset can be characterized through a series of discharge curves to inform the battery health management system. Further discharge over the life of the battery will inform the cycle life and continual performance degradation.

4.6 Reflection Software Validation Environment

Reflection is a C/C++ program for distributed embedded systems development component-based architecture simulation from Corey Ippolito. This software program simulates a multirotor platform and is implemented to for control system validation. The UAS flight simulation environment in Reflection has a rotorcraft dynamic model which simulates several rotorcraft subsystems and the rotorcraft's interaction with the surrounding environment as represented in Figure 31.

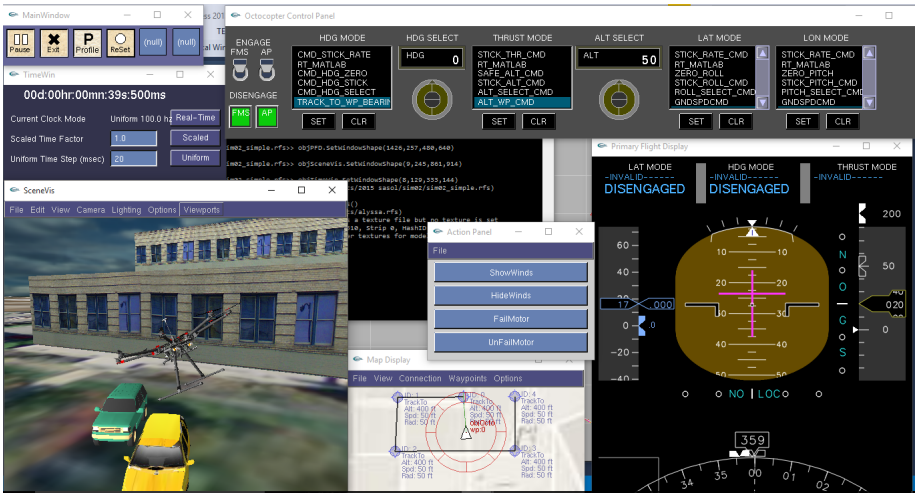


Figure 31. Reflection flight simulation user interface

Reflection was utilized as a validation environment for implementing a battery model and testing a motor failure algorithm. In future work, Reflection can be used to simulate wind disturbance rejection and integrate the NASA Ames wind velocity CFD analysis data. Through the use of this simulation program, the algorithms may be tested in a low risk environment before being assessed with dangerous hardware and several missions may be simulated to show their effect on battery health.

4.6.1 Equivalent Circuit Battery Model Integration

The octocopter modeled in Reflection originally was powered from a constant 22.2V power source line as in Figure 32 that did not vary based off of the current drawn from each of the 8 motors. Therefore, each of the motors was able to draw as much current and voltage as necessary to ideally maneuver through each set waypoint. However, this battery modeling technique misrepresents the amount of power available to the UAS in real life. As the octocopter maneuvers, the current drawn from each motor will fluctuate and this should in turn vary the total voltage drawn.

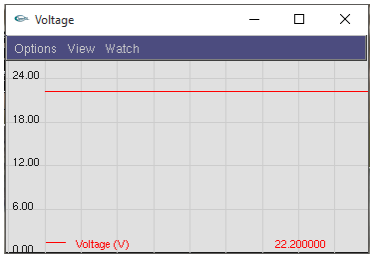


Figure 32. Constant 22.2 V line in Reflection

The Iris+ has a 3S 5.1 Ah 8C lithium polymer battery with a maximum voltage of 12.6 V [19]. Integrating this battery in to Reflection will enable the flight simulation program to more accurately represent Iris+ flights. The lithium ion equivalent circuit battery model as developed by Sankararaman 2014 was translated to represent lithium polymer batteries [20]. This Matlab model was then converted to C++ to be integrated in to Reflection.
Rather than representing a constant voltage as in Figure 32, including this model in Reflection caused the voltage available to the UAV more accurately represent the LiPo battery of the Iris+. Figure 33 has a maximum voltage of 12.6 V as does the Iris+ and as each of the 8 motors draws current, the voltage drops accordingly.

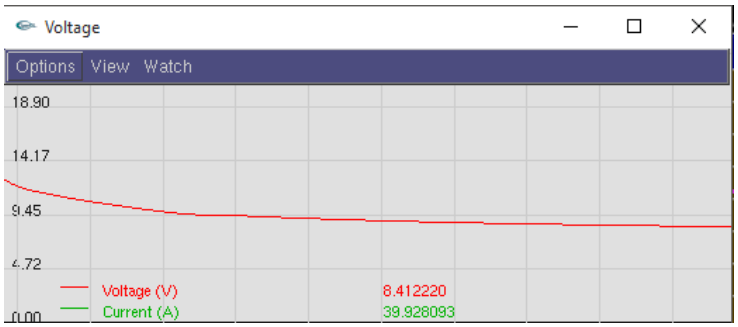


Figure 33. Corrected Voltage during takeoff in Reflection

Figure 34 demonstrates the increased current drawn during take off and the slight variance between the motors as the octocopter orients itself towards the first destination and gains altitude. The current levels off and balances between the motors as the octocopter then transitions into steady flight towards the first waypoint.

4.6.2 Motor Failure Procedure

When one of the eight motors fails on an octocopter, the rotorcraft spirals out of control because of a mis-allocation of motor inputs. This behavior was modeled in Reflection by commanding the motor to produce 0 thrust. As a result, the previously balanced and steady motor commands on

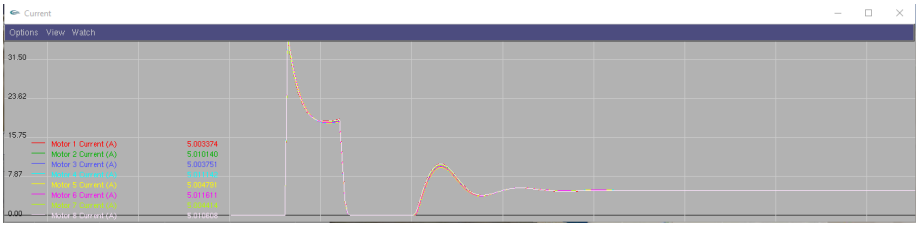


Figure 34. Octocopter motor currents during takeoff

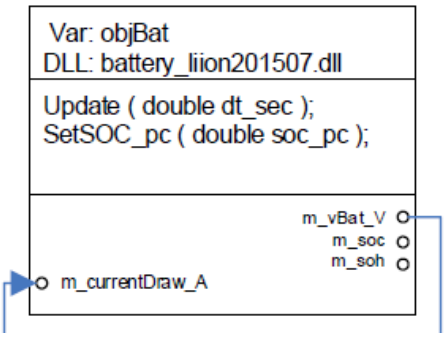


Figure 35. Battery module within the Reflection control system

the octocopter during steady forward flight, shown in the first half of the graph in Figure 36, suddenly begin to oscillate haphazardly. Each motor's erratic current demonstrates the octocopter's futile attempt to regain control while falling and spiraling downwards.

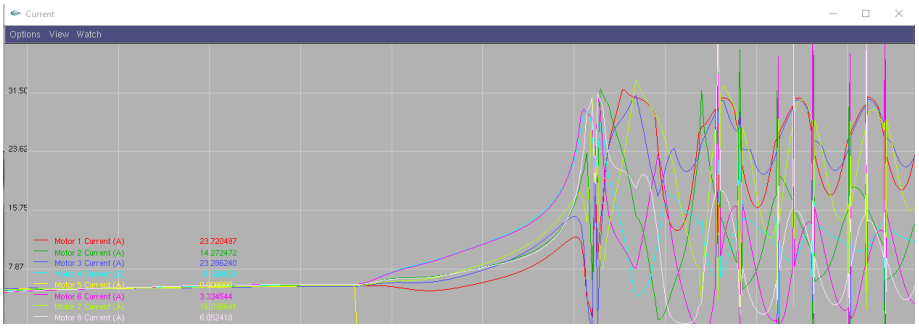


Figure 36. Current drawn from each octocopter motor without control algorithm

The motor failure algorithm as previously developed was transferred into Reflection and implemented with the octocopter's dynamic control system. As a result, instead of losing control, the octocopter should now be able to continue its mission despite the loss of one motor. Figure 37 demonstrates the same octocopter set of motor current readings and waypoint sequence as in Figure 36, but in this scenario, the motor failure controller is integrated. Once motor 5 is failed, its current immediately

drops to 0 A and the current on the remaining motors is briefly affected and they immediately oscillate. However, these oscillations damp out quickly and each motor current regains stability. The varying levels of current in the steady-state after failure illustrate the effect of the new control allocation matrix which commands the working motors at different thrust levels to compensate for the lost motor.

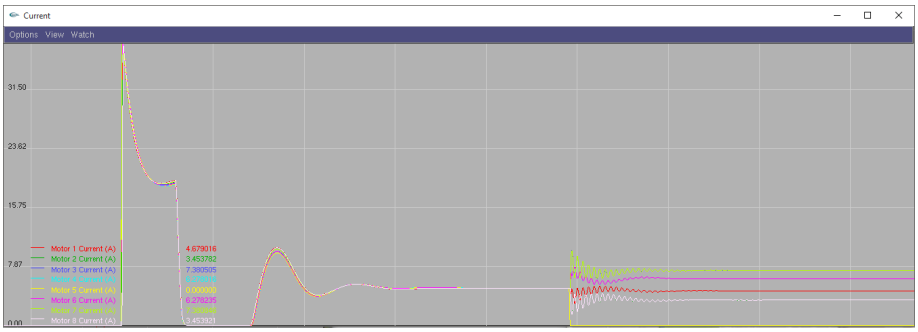


Figure 37. Current drawn from each octocopter motor with control algorithm

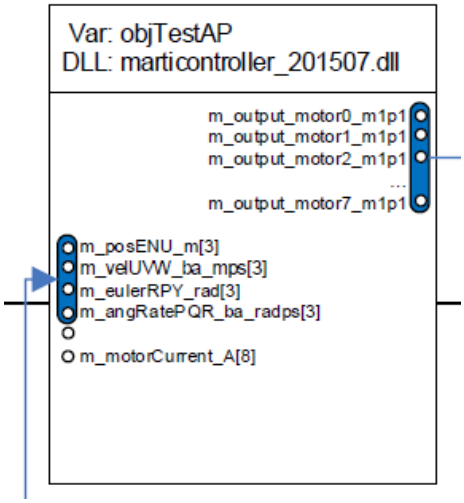


Figure 38. Motor failure dynamics test module within the Reflection control system

4.7 Conclusion

The objective of the project was to develop a control system to safely stabilize and land a multirotor during a motor failure event, incorporate estimated drag and wind disturbances into a controller, create an accurate model of a typical multirotor battery, and simulate and test our work in the Reflection software or hardware.

The motor failure algorithm was successfully implemented and tuned in a Simulink model. The algorithm allowed for the safe control and landing of a quadcopter after an engine failure.
The drag and wind disturbance estimator was successfully implemented into a Simulink model. While this model did not allow for an exact estimation of the drag coefficients and disturbance forces, it did enable the simulated multirotor to fly smoothly in a gusty environment.
Within Reflection, the equivalent circuit LiPo battery model was successfully integrated, which enables the UAS to have an accurate energy system representation. This will serve as a better simulation of real-life missions as the battery will now more accurately limit the flight time and thrust immediately available to battery motors.
Also within Reflection, the motor failure algorithm was shown to work with an octocopter. When considered with the results from Simulink, this integration demonstrates the generality of the implemented reconfigurable controller.

4.7.1 Future Work

Future work should include fully integrating the developed subsystems of motor failure protocol, drag and wind disturbance estimator, and battery models into the Reflection software platform. By integrating all of these components, they can be validated in simulation and their interaction can be observed.
The motor failure algorithm should be tested explicitly on hardware by forcing a motor failure during flight with the algorithm on-board. By iteratively testing and adjusting parameters, the system can be tuned and the limits of the algorithm can be determined.
The drag and wind disturbance estimator could be tested by implementing it on hardware and flying in a windy environment. Ideally ,this test would take place indoors with a controlled airflow pattern generated by fans. The results of this test would be used to improve and tune the estimation algorithms.
Prognostics of batteries in UAS could be further explored through the SHARP lab to better assess the battery conditions throughout flight. By modeling test flight simulations of the battery, an energy storage fault analysis may be produced in order to determine UAV protocol during failure events.
An improved battery model in Reflection could present further constraints to the UAS that more closely resembles those posed by batteries in real life UAS scenarios. These features could include minimum voltage thresholds for flight and integrating charging time in to mission planning.
Further work in Reflection may be done to more closely simulate actual flight environments by integrating computational fluid dynamic analysis of rotorcraft movement and regional wind profiles. This will enable more

dynamic, realistic, and challenging scenarios such as navigating through urban canyons and interacting with other UAS.

5 Hardware Team

5.1 Background

In the last ten years, Quadcopters and Multi-rotor systems have become ubiquitous platforms for hobbyists and professionals alike. Their popularity is the result of low cost and simple design. The ease of production inevitably leads to situations in which low-quality components fail and cause the vehicles to depart from their intended trajectories. This ultimately, can cause damage to personnel, buildings, and property. To address the issues surrounding loss of control events, route management, and collision avoidance, NASA has invested resources into the Unmanned Traffic Management (UTM) system and into the SAfe Flight Environment under 50 feet (Safe50) Project. These projects are designed to allow for safe integration of UAVs into the National Airspace.

5.2 Introduction

The purpose of the hardware team was to develop a multi-rotor platform that could be used to test the control algorithms being developed. We were primarily interested in examining was the motor failure recovery protocol. In this situation, a multi-rotor vehicle flying any given mission suddenly and unpredictably loses control of one or more actuators. By nature, quadcopters and multi-rotors are aerodynamically unstable and the loss of any number of motors will result an imbalance of torques. This imbalance, if left uncorrected, will cause the Unmanned Arial System (UAS) to rapidly enter an uncontrolled descent.

Total loss of control, however, is not assured if the actuator failure is detected and appropriate action is taken. The actions that need to be taken are discussed more fully in the control section of this paper, but in essence, the control matrix needs to updated and optimized. A control matrix is traditionally used to allocate forces/torques to individual motors, however, when a motor is lost, the control matrix needs to be changed such that the new configuration is represented. When this update occurs, the resulting matrix, might be singular (non-invertible) or might produce negative values. In such a scenario, the best course of action is to reduce the rank of the matrix and find an invertible solution. This is most commonly done by sacrificing control of yaw while maintaining control of horizontal attitude and vertical altitude. However, before these actions can be completed, the motor failure must be detected in hardware.

5.3 Multi-rotor Design and Integration

One benefit of the control allocation method that we describe here is that it works with any configuration and with any number of motors. So when choosing a proof of concept frame design, we wanted to use the minimum number of motors that would allow us to demonstrate all 3 possible failure modes, in addition to proving that the algorithms will work with any possible configuration (even with the unusual choice of an odd numbers of motors). The five-motor pentacopter allows us to do just that. For the sake of argument, we will assume that this UAS has 3 clockwise motors (numbered 1,3, and 5) and 2 counterclockwise motors (numbered 2 and 4) as shown in Figure 39.

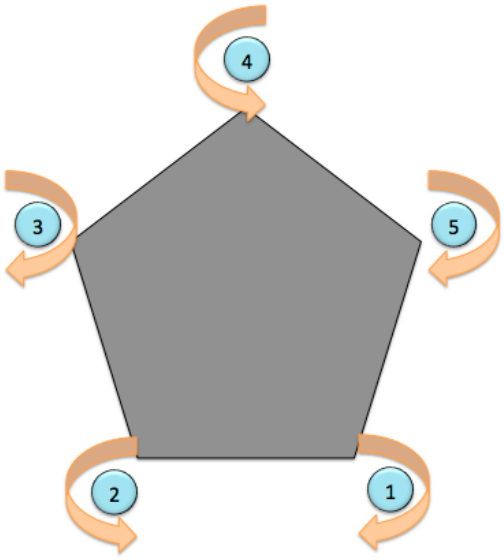


Figure 39. Motor numbering and direction diagram

If motor 1 or 5 fails, then the control algorithm can adjust for the failure, but will be able to continue flight. If motor 2 or 4 fails, then the controller will produce negative numbers. Such negative thrust values are numerically possible, but mechanically is impossible, so control over yaw must be sacrificed. If motor 3 fails, then the control matrix will not be invertible and the UAS will again be forced to sacrifice yaw.

$$B = \begin{bmatrix} 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 \\ -0.0059 & 0.0059 & 0.0095 & 0.0000 & -0.0095 \\ -0.0081 & -0.0081 & 0.0031 & 0.0100 & 0.0031 \\ 0.0240 & -0.0240 & 0.0240 & -0.0240 & 0.0240 \end{bmatrix}$$

Figure 40. Control matrix for pentacopter

With this knowledge in mind, we set out to build a pentacopter that met the following design parameters shown in Table 2.

$Specification$	$Value$
$Weight$	$0.5Kgs$
$Diameter$	$0.03m$
$Thrust$	$> 180g$

Table 2. Design Parameters

Electronics were picked online and a chassis to house them all was designed in SolidWorks as shown in Figure 41.

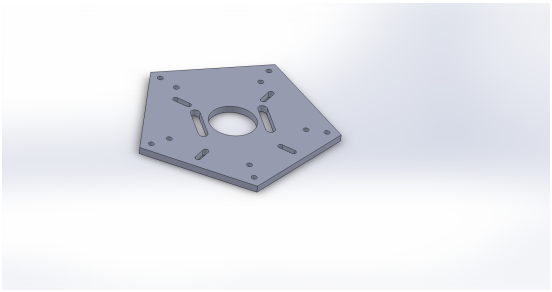


Figure 41. Pentacopter Chassis

To meet the modularity aspect of this of this multi-rotor platform, we also developed a quadcopter chassis for additional testing as shown in Figure 42.

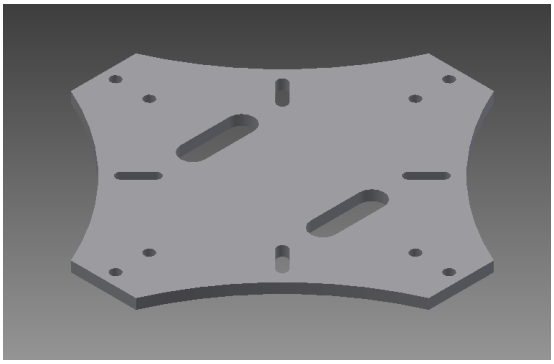


Figure 42. Quadcopter Chassis

The final product is shown in Figure 43

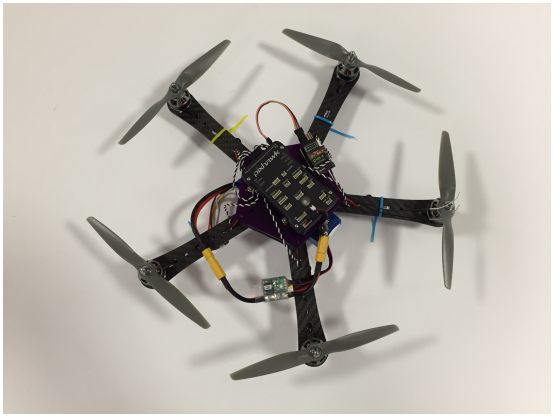


Figure 43. Final Pentacopter design

5.4 Motor Failure Detection

5.4.1 Electrical Model of Motor

The most common multi-rotor systems use four or more identical motors to provide the thrust needed. Here we analyze the mathematical models for a brushed DC motor as a proxy for a brushless motor and ESC system. Derivation is from [17]. Figure 44 shows a circuit diagram for a brushed DC motor.

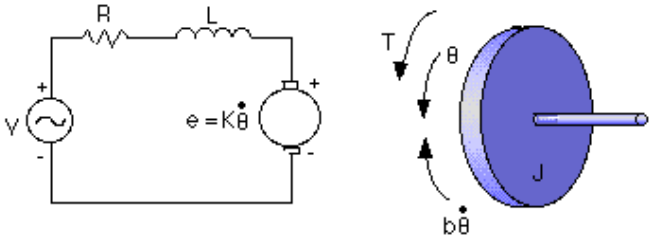


Figure 44. Circuit Diagram for a brushed DC motor. We assume that the brushless motor and Electronic Speed Controller (ESC) behaves according to this model.

From this diagram, we can use Kirchhoffs voltage law to derive equation 1:

$$L\dot{i}_j(t) + Ri_j(t) = v_j(t) - k_b\Omega(t) \tag{17}$$

Where L is the motor inductance, R is the motor resistance, $i_j(t)$ is the current flowing through the jth motor, k_b is the back EMF constant, and $\Omega_j(t)$ is the angular rate of the jth motor. We can relate this angular rate to the produced torque through the differential equation:

$$J_{r3}\dot{\Omega}_j(t) + k_d\Omega_j(t) = k_m i_j(t) - \tau_j^l(t) \tag{18}$$

Where J_{3r} is the inertia of the motors, k_d is the friction coefficient, and $\tau_j^l(t)$ is the torque. Combining the two equations, and converting to the laplace domain, this can be written as:

$$(Ls + R)i_j(s) = v_j(s) - \frac{k_b}{J_{r3}s + k_d}\left[k_m i_j(s) - \tau_j^l(t)\right] \tag{19}$$

Solving for i_j and assuming that the inductances of the motor are small, we can write:

$$i_j(s) = \frac{J_{r3}s + k_d}{RJ_{r3} + Rk_d + k_b k_m} v_j(s) + \frac{k_b k_m}{RJ_{r3}s + Rk_d + k_b k_m} \tag{20}$$

Now we have two linearly combined equations which can both be decomposed as the products of two functions. To find the time-domain solutions, we must individually find the impulse response functions and convolve the results. When the motor stops working, its torque ($\tau_j^l(t)$) will sharply drop to 0. This means that the term on the right side of the equation will change while the left half will stay constant. To understand what this change will do, we solve for the impulse response function:

$$h(t) = \frac{k_b k_m}{RJ_{r3}} e^{-\frac{Rk_d + k_b k_m}{RJ_{r3}}t} \tag{21}$$

We note that this equation will always be positive and that there is a direct relationship between torque and current. This suggests that when the torque drops to zero, the current will experience a similar fall. And indeed if we model the torque drop off using a Heaviside step function ($U_c(t)$, where c represents the step time) as:

$$\tau_j^l(t) = 6 - 6 * U_5(t) \tag{22}$$

We take the convolution of the impulse response with the torque function:

$$\int_0^t \frac{k_b k_m}{RJ_{r3}} e^{-\frac{Rk_d + k_b k_m}{RJ_{r3}}t} \left[12 - U_5(t - \tau)\right] d\tau \tag{23}$$

Note that at 5 second, the current sharply drops and levels off at 0.4 amps.

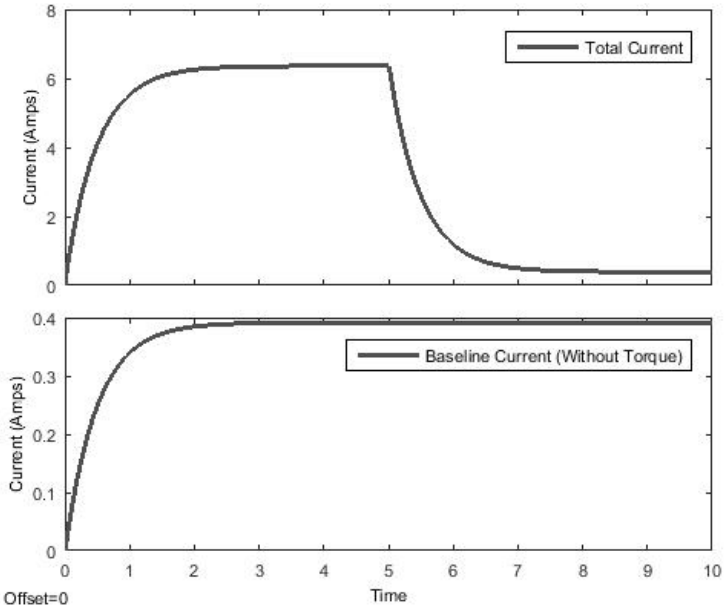


Figure 45. Motor current during motor failure. Both curves asymptotically reach ≈ 0.4 amps

5.4.2 Current Detection

To determine the amount of current being drawn by each motor, we attached AttoPilot (150V, 90A) current sensors (Such as the one shown in Figure 46) to each of the ESCs.

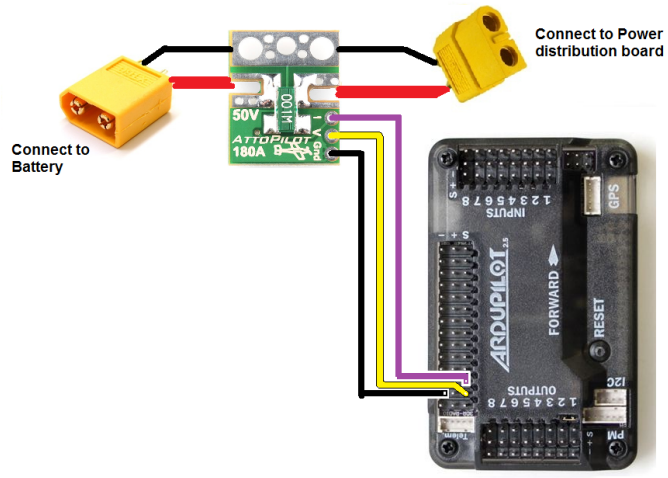


Figure 46. Picture of typical AttoPilot connection. In our case, the board separates the power distribution board from the ESCs. Figure courtesy http://copter.ardupilot.com/

Note that when working with a 4-in-1 ESC such as the one that is pro-

vided with the IRIS+, such a connection is not possible. These boards provide signal voltages that correspond to the voltage and current given to the motors. Before the boards could be used, they were calibrated so we could compare the instantaneous current to the predetermined threshold voltages.
To artificially "fail a motor", we used 50Amp power N-type Mosfets in series with the ammeter boards as shown in Figure 47.

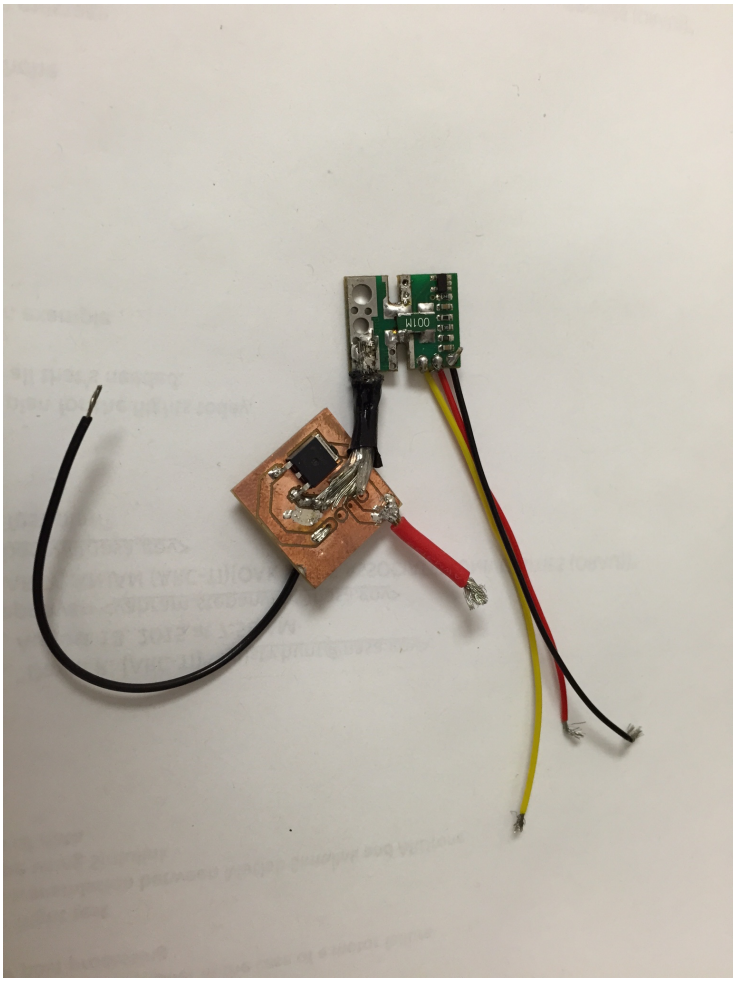


Figure 47. Ammeter board with a Power Mosfet Transistor that is used to sever the electrical connection and cause the motor to fail

For our multi-rotor, we used T-Motor MN1806 motors. Without propellers, they usually draw anything between .4 and .6 Amps. With propellers they draw between 3 and 7 Amps.
To detect the signal voltages coming from these boards, we used analog to digital converters that communicate via an I2C bus (http://www.mouser.com/ds/2/268/21805a-74229.pdf). Because we wanted this technology to be modular, we decided to use individual chips instead of one chip with many differential

pairs. The PCB was fabricated using a Roland milling machine found in the NASA Ames Space Shop and was designed in eagle as shown in Figure 48.

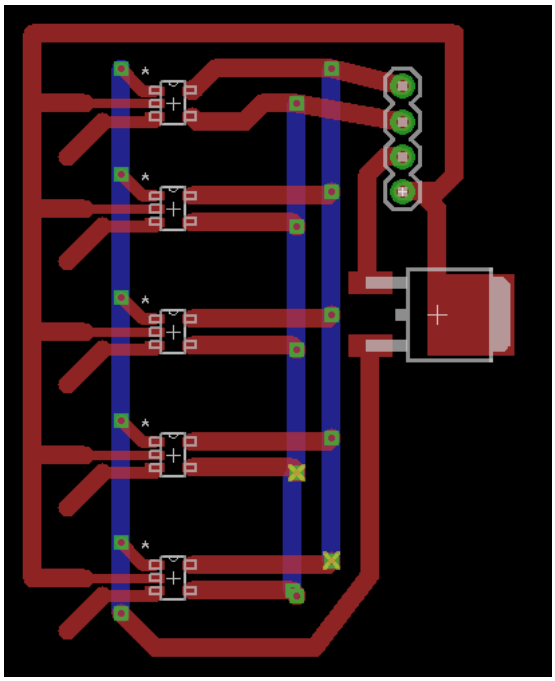


Figure 48. Eagle board schematic drawn up to visualize the placement of integrated circuits on the printed circuit board

5.5 Conclusions

We have produced a multi-rotor vehicle that will hopefully be used to test the motor failure recovery algorithm. This vehicle was specifically designed to meet the NASA Ames safety standards as well as provide full functionality for testing the algorithm. In the future, we hope to more fully integrate the ammeters and ADCs and start to re-write the PixHawk software so the device will actually be able to fly.

References

1. Sukra Helitek, Inc. *RotCFD User Manual*, version 0.9.15. edition, 2015.

2. Sukra Helitek, Inc. *RotUNS Theory*, version 0.9.15. edition, 2015.

3. Sukra Helitek, Inc. *RotUNS User Manual*, version 0.9.15. edition, 2015.

4. Sukra Helitek, Inc. *RotUNS Tutorials*, version 0.9.15. edition, 2015.

5. C. Venkatesan. *Fundamentals of Helicopter Dynamics*. CRC Press, Boca Raton, FL, 2015.

6. W. Johnson. *Rotorcraft Aeromechanics*. Cambridge University Press, New York, NY, 2013.

7. J. Fries. Helicopter trim analysis. Technical report, Army Research Laboratories, 1996.

8. Sukra Helitek, Inc. *AFTGen User Manual*, version 0.9.15 edition, 2015.

9. Sukra Helitek, Inc. *AFTGen Theory*, version 0.9.15 edition, 2015.

10. Sukra Helitek, Inc. *AFTGen Tutorials*, version 0.9.15 edition, 2015.

11. H. I. Seo and B. Lee. Web-based forecasting system of airborne livestock virus spread simulated by openfoam cfd, 2014.

12. A. L. Braun and A. M Awruch. Aerodynamics analysis of buildings using numerical tools from computational wind engineering. Asociation Argentina de Mecanica Computacional, 2007.

13. C. J. Greenshiels. Openfoam the open source cfd toolbox user guide. OpenFOAM Foundation, 2015.

14. Weather history for knuq, August 2015. http://www.wunderground.com.

15. Finding an optimal path using matlab and optimization toolbox. MathWorks Inc, April 2012. http://www.mathworks.com/matlabcentral/fileexchange/36321-demo–finding-an-optimal-path-using-matlab-and-optimization-toolbox/content/PathOptimizationScript.m.

16. National Aeronautics and Space Administration. Unmanned aerial system traffic management (utm), August 2015. http://utm.arc.nasa.gov/index.shtml.

17. V. Stepanyan, K. Krishnakumar, and A. Bencomo. Identification and reconfigurable control of impaired multi-rotor drones. In *Proceedings of the 2016 AIAA Guidance, Navigation, and Control Conference*, January 2016.

18. V. Stepanyan and K. Krishnakumar. M-mrac for the uncertain drones with wind estimation (paper in progress).

19. 3D Robotics. *Iris+ Operations Manual*. October 2014.

20. S. Sankararaman, M. Daigle, and K. Goebel. Uncertainty quantification in remaining useful life prediction using first-order reliability methods, June 2014.

6 Appendix A Rotorcraft: Case Data

Table 3. Case Directory

	Flight Condition	Tilt (deg)	Tip Speed		Wind Speed (ft/s)
Isolated Rotor					
Case 1A	hover	0	236.094		0
Case 1B	general (29.333,0,0)	5	236.094		29.333
Case 1C	general (29.333,0,0)	10	236.094		29.333
Case 1D	general (29.333,0,0)	15	236.094		29.333
Isolated Rotor					
Case 1E	hover	0	283.432		0
Case 1F	general (29.333,0,0)	5	283.432		29.333
Case 1G	general (29.333,0,0)	10	283.432		29.333
Case 1H	general (29.333,0,0)	15	283.432		29.333
Isolated Body					
Case 2A	general (29.333,0,0)	0			29.333
Case 2B	general (29.333,0,0)	5			29.333
Case 2C	general (29.333,0,0)	10			29.333
Case 2D	general (29.333,0,0)	15			29.333
Full Configuration Trimmed Runs			**Front Rotor Tip Speed**	**Rear Rotor Tip Speed**	
Case 3A	general (29.333,0,0)	4.62	242.722	242.722	29.333
Case 3B	general (29.333,0,0)	4.62	218	257	29.333
Case 3C	general (29.333,0,0)	4.62	230	250	29.333
Full Configuration Out of Ground Effect					
Case 4A	hover	0	236.094		0
Case 4B	hover	0	283.432		0
Full Configuration In Ground Effect					
Case 5A	hover	0	236.094		0
Case 5B	hover	0	283.432		0
Case 5C	hover	0	283.432		0

Table 4. IRIS+ Propeller Blade Physical Input Values

Quadrotor Blade Metrics	
Blade Radius (in)	4.75
Blade Radius (ft)	0.395833333
Cutout (r/R)	0.04558643
Twist	
r/R	Angle (deg)
0.09375	18.88
1.0	9.464
Airfoil Stations	
r/R	Airfoil
0.04559	NACA 0010
0.5	NACA 0009
0.75	NACA 0008
1.0	NACA 0008
Cyclic Pitch	None
Flapping	None

6.1 Case parameters

Table 5. IRIS+ Propeller Blade Chord:Radius

Chord/Radius	
r/R	Chord/Radius
0.0455	0.12847085
0.1287	0.18483225
0.2072	0.24036482
0.2486	0.25104033
0.2942	0.24161761
0.3812	0.22362879
0.4641	0.20649658
0.5470	0.18936437
0.6324	0.17171820
0.7128	0.15509995
0.7998	0.13711113
0.8868	0.11912231
0.9697	0.10199010
1.00	0.09573685

6.2 Calculation of Tip Speed Required for Hover

The mass of the IRIS+ quadcopter is approximately 1282 grams. This requires that each motor carry 320.5 grams of the total mass. The fol-

lowing table denotes the amount of thrust generated per varying rpm for this specific model.

Table 6. Manufacturer data for thrust generated at varying rpm

Throttle	**Thrust(g)**	**RPM**
50%	176	4400
65%	289	5500
75%	385	6400
85%	475	7000
100%	617	7800

Plotting this data and creating a line of best fit results in a linear equation that can be used to determine the thrust that must be generated by each motor to maintain the quadcopter in hover.

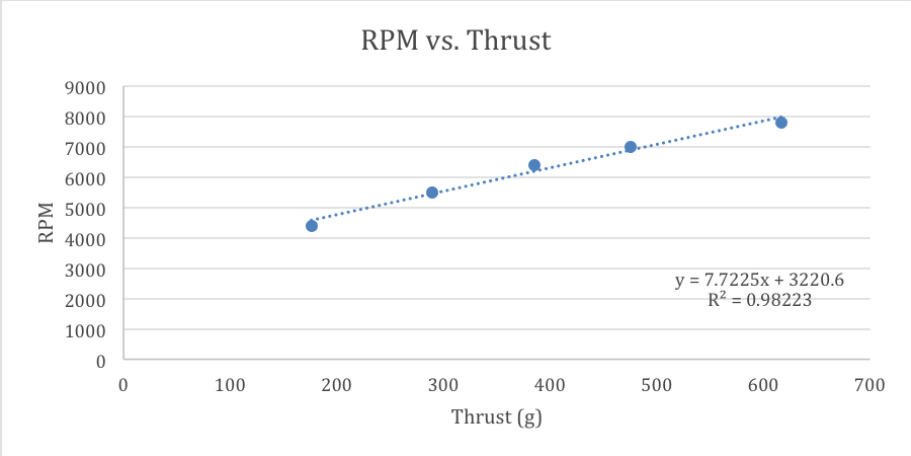


Figure 49. Plot of the quadrotor motor rpm versus the thrust generated per motor

The following equation relates this thrust per motor rpm.

$$RPM = 7.7225T + 3220.6 \tag{24}$$

Inputting a thrust of 320.5 grams into the above equation results in an RPM of 5695.66125 for the hover case.

The tip speed can then be calculated by determining the perimeter of the circle one blade sweeps out during one rotation (the perimeter of a circle).

$$P_b = 2\pi * r_b \tag{25}$$

The tip speed can then be determined in feet per second

$$s_t = \frac{(RPM * P_b)}{60} \tag{26}$$

7 Appendix B Rotorcraft: Input Parameters

7.1 Rotor Data

7.2 Flow Data

Table 7.

Flow Properties		
Density	0.002377	slugs/ft^3
Temperature	518.69	Rankine
Gas Constant	1718	ft-lb/slug*degree*Ra
Specific Heat Ratio	1.4	
Dynamic Viscosity	3.74E-07	lbf*sec/ft^2
Pressure	2118.17	lbf/ft^2

8 Appendix C Rotorcraft: Isolated Rotor Analysis

Table 8. Isolated Rotor at Hover RPM

	Flight Condition	**Tilt (deg)**	**Wind Speed (ft/s)**	**Boundary Conditions**	**Boundary**	**Grid**	**Refinement Box**
Case 1A	hover	0	0	pressure	5x5x5	20x20x20	2x2x2
Case 1B	general (29.333,0,0)	5	29.333	velocity	7.5x5x5	30x20x20	3x2x2
Case 1C	general (29.333,0,0)	10	29.333	velocity	7.5x5x5	30x20x20	3x2x2
Case 1D	general (29.333,0,0)	15	29.333	velocity	7.5x5x5	30x20x20	3x2x2

**all cases studied with clockwise rotation at 236.094 ft/s
For all cases:
Refinement Box Refinement = 5
Rotor Refinement = 6
Time Grid = 300 time steps in 3 seconds

**all cases studied with clockwise rotation at 283.432 ft/s
For all cases:
Refinement Box Refinement = 5
Rotor Refinement = 6
Time Grid = 300 time steps in 3 seconds

Table 9. Isolated Rotor at Hover + 20% RPM

	Flight Condition	Tilt (deg)	Wind Speed (ft/s)	Boundary Conditions	Boundary	Grid	Refinement Box
Case 1E	hover	0	0	pressure	5x5x5	20x20x20	2x2x2
Case 1F	general (29.333,0,0)	5	29.333	velocity	7.5x5x5	30x20x20	3x2x2
Case 1G	general (29.333,0,0)	10	29.333	velocity	7.5x5x5	30x20x20	3x2x2
Case 1H	general (29.333,0,0)	15	29.333	velocity	7.5x5x5	30x20x20	3x2x2

8.1 Thrust Data for Isolated Rotor

9 Appendix D Rotorcraft: Isolated Body Analysis

Table 10. Isolated Body

	Flight Condition	Tilt (deg)	Wind Speed
Case 2A	general (29.333,0,0)	0	29.333
Case 2B	general (29.333,0,0)	5	29.333
Case 2C	general (29.333,0,0)	10	29.333
Case 2D	general (29.333,0,0)	15	29.333

For all cases:
Boundary Size = 20x10x10
Grid = 40x20x20
Refinement Box Size = 8x4x4
Refinement = 5
Body Refinement = 6
Time Grid = 400 time steps in 2 seconds

9.1 Calculation of the Tilt Angle During Forward Flight

From RotCFD runs of the isolated body in forward flight, the drag and lift forces on the body could be determined. Cases were run at 0, 5, 10, and 15 degrees tilt at a steady 20mph forward speed. The data was plotted on the following graph and the lines of best fit were used to interpolate data for additional tilt angles.

Table 11. Boundary Conditions for Isolated Body Cases

X-Min Type	Velocity
X-Min Velocity	29.333
	0
	0
X-Max Type	Mass Outflow Correction
Y-Min Type	Velocity
Y-Min Velocity	29.333
	0
	0
Y-Max Type	Velocity
Y-Max Velocity	29.333
	0
	0
Z-Min Type	Velocity
Z-Min Velocity	29.333
	0
	0
Z-Max Type	Velocity
Z-Max Velocity	29.333
	0
	0

Table 12. Lift and drag force data for varying angles

Angle	Drag Force	Lift Force
0	0.199303	0.00182143
5	0.229258	-0.0354366
10	0.268806	-0.0465095
15	0.311652	-0.0895441

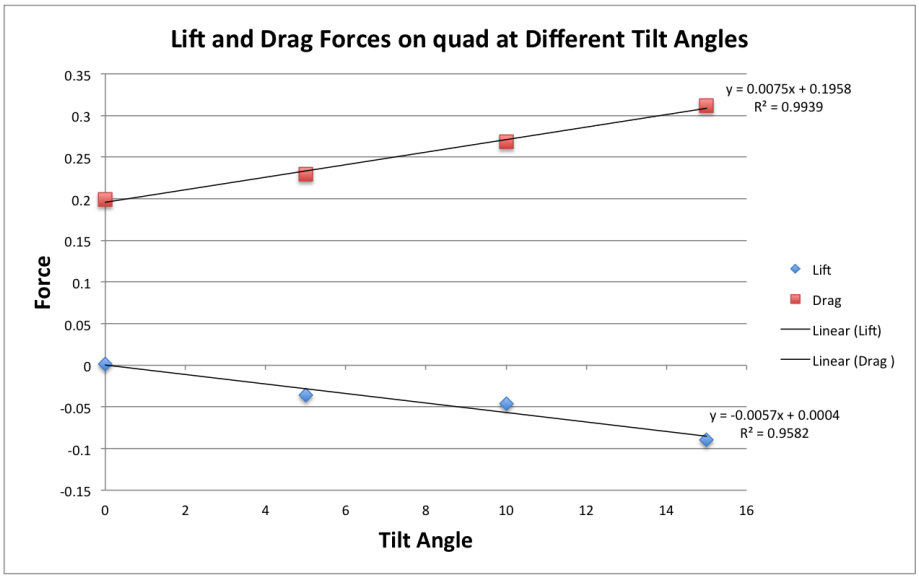


Figure 50. Isolated body RotCFD run data for the lift and drag forces at varied tilt angles

This resulted in the following equations for lift and drag:

$$L = (-0.0057)\theta_{tilt} + 0.0004 \tag{27}$$

$$D = (0.0075)\theta_{tilt} + 0.1958 \tag{28}$$

Using equations 27 and 28, a range of tilt angles were input varying from 0 to 20 in 1 degree increments yielding the corresponding lift and drag values. The resultant force vector was then determined in two methods. The first was through the use of the ratio:

$$R_1 = \frac{D}{(L + W)} \tag{29}$$

The second was found from the tilt angle.

$$R_2 = tan(\theta_{tilt}) \tag{30}$$

To find the variation between the two values of R, the absolute value was taken of the difference between the two values.

$$R_{diff} = |R_1 - R_2| \tag{31}$$

The goal seek function in excel was then used to vary θ_{tilt} until R_{diff} is minimized. This gave a value of 4.6224025°.

9.2 Calculation of Tip Speed for a Calculated Tilt Angle

From the information located in Table 13, the net force required for equilibrium for a quadrotor in forward flight could be calculated.

Table 13. Net force required for equilibrium for a quadrotor in steady, forward flight

Angle	**Net Force**
0	2.83152431
5	2.87092778
10	2.88538079
15	2.93247455

This data was used to create an interpolation curve for which the net force required as shown in the following plot.

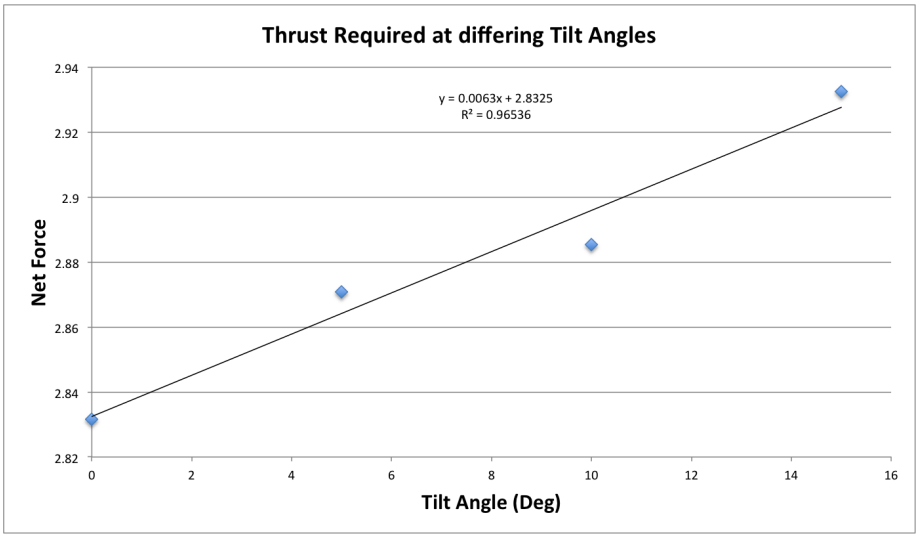


Figure 51. Best fit line for net thrust versus tilt angle

$$T = (0.0063)\theta_{tilt} + 2.8325 \tag{32}$$

The line of best fit, equation 32, can then be used to find the total force required for the condition $\theta_{tilt} = 4.622°$. This results in a thrust value of 2.8616 lbs_f. To find the force per rotor, this value was simply divided by 4 resulting in a thrust of 0.7154 lbs_f per motor. Knowing this specific thrust value, a new interpolation using the data for tip speed and thrust per motor could be made. This used the following data:

Table 14. Data of the thrust generated per motor for a given tip speed

Tip Speed	**Thrust per Motor**
236	0.675374201
283	0.9552898

Interpolating linearly:

$$\frac{283 - s_{tip}}{s_{tip} - 236} = \frac{0.955 - 0.7154}{0.7154 - 0.6754}$$

$$s_{tip} = 242.72219 \text{ rpm}$$

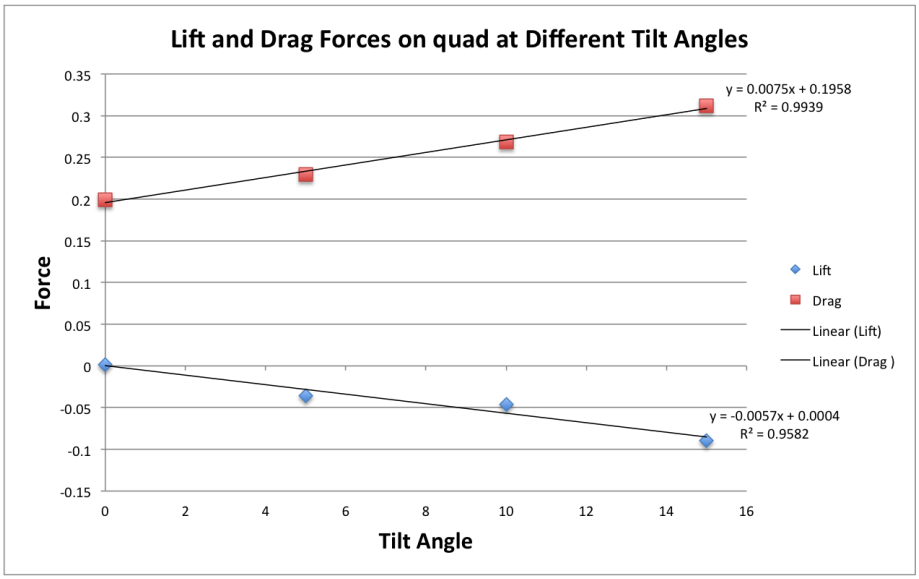


Figure 52. Lift and drag forces on the body at various tilt angles

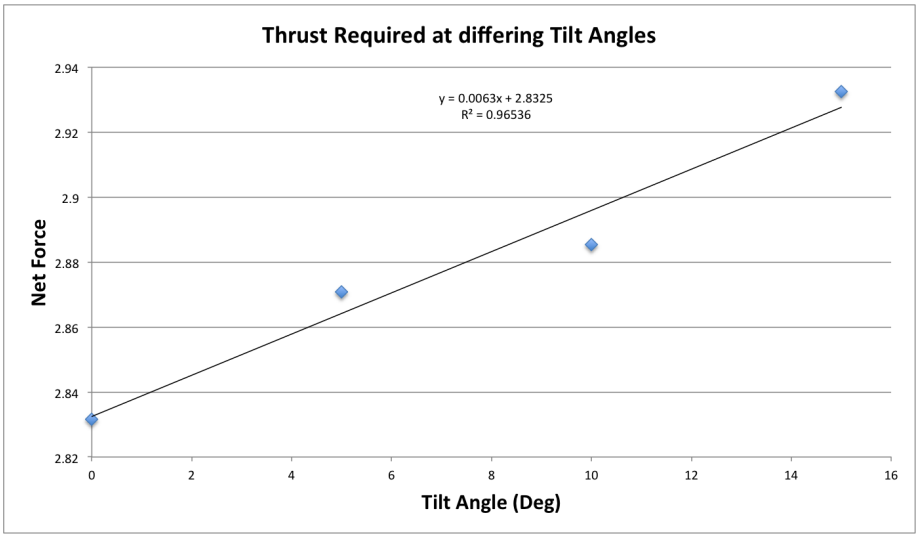


Figure 53. Net force on the body at various tilts

10 Appendix E Rotorcraft: Forward Flight Trim

For all cases:
Boundary Size = 15x10x10
Grid = 30x20x20
Refinement Box Size = 6x4x4
Refinement = 5
Body Refinement = 6

Table 15. Full Configuration Trimmed Studies

	Flight Condition	Tilt (deg)	Front Rotor Tip Speed	Rear Rotor Tip Speed
Case 3A	general (29.333,0,0)	4.62	242.7	242.7
Case 3B	general (29.333,0,0)	4.62	218	257
Case 3C	general (29.333,0,0)	4.62	230	250

Rotor Refinement = 6
Time Grid = 300 time steps in 3 seconds

Table 16. Boundary Conditions for Trimmed Forward Flight

X-Min Type	Velocity
X-Min Velocity	29.333
	0
	0
X-Max Type	Mass Outflow Correction
Y-Min Type	Velocity
Y-Min Velocity	29.333
	0
	0
Y-Max Type	Velocity
Y-Max Velocity	29.333
	0
	0
Z-Min Type	Velocity
Z-Min Velocity	29.333
	0
	0
Z-Max Type	Velocity
Z-Max Velocity	29.333
	0
	0

10.1 Calculation Procedural Overview

In order to determine the forward flight trim values of the quadrotor,a process was used to balance the thrusts generated by the motors to maintain the quad in steady, forward flight. This was completed through the following steps.

1. Determine the mean thrust for each of the four motors

2. Take the average of the thrusts of the two rear motors and the two forward motors to compute an average rear motor thrust and an average forward motor thrust
3. Compare the rear motor thrust value and forward motor thrust value to the desired motor thrust value
4. Compute the absolute value of the difference between the average thrust and the actual thrust, finding the percent correction needed
5. Adjust the tip speed accordingly and repeat

After case 3C had been run, a linear interpolation was used to input a forward and rear rotor speed in RotCFD for an additional forward flight run. This resulted in a rear tip speed of 250 ft/s and a forward tip speed of 230 ft/s.

10.2 Data

The following tables display the data and results of the calculations for the trimming of the forward flight.

Table 17. Data for case 3A of forward flight trim

Rotor	Tip Speed	Mean Thrust Output	Thrust Needed	Percent Correction
Front Right	242.6	0.7974095	0.7154	0.1028449
Front Left	242.6	0.7974095	0.7154	0.1028449
Rear Right	242.6	0.676393	0.7154	0.05765819
Rear Left	242.6	0.676393	0.7154	0.05765819

Table 18. Data for case 3B of forward flight trim

Rotor	Tip Speed	Mean Thrust Output	Thrust Needed	Percent Correction
Front Right	218	0.660264	0.7154	0.1371455
Front Left	218	0.660264	0.7154	0.1371455
Rear Right	257	0.773965	0.7154	0.097565
Rear Left	257	0.775016	0.7154	0.097565

Table 19. Data for case 3C of forward flight trim

Rotor	Tip Speed	Mean Thrust Output	Thrust Needed	Percent Correction
Front Right	229.766	0.726377	0.7154	0.010977
Front Left	229.766	0.726377	0.7154	0.010977
Rear Right	241.682	0.725512	0.7154	0.010112
Rear Left	241.682	0.725512	0.7154	0.010112

Table 20. Full Configuration in Hover Out of Ground Effect

	Flight Condition	Tilt	Tip Speed
Case 4A	hover	0	236.094
Case 4B	hover	0	283.432

11 Appendix F Rotorcraft: Hover Trimming

11.1 Out of Ground Effect

For all cases:
Boundary Size = 10x10x5
Grid = 20x20x10
Refinement Box Size = 4x4x2
Refinement = 5
Body Refinement = 6
Rotor Refinement = 6
Time Grid = 500 time steps in 5 seconds

Table 21. Boundary Conditions for Hover Out of Ground Effect

X-Min Type	Pressure
X-Min Pressure	2118.17
X-Max Type	Pressure
X-Max Pressure	2118.17
Y-Min Type	Pressure
Y-Min Pressure	2118.17
Y-Max Type	Pressure
Y-Max Pressure	2118.17
Z-Min Type	Mass Outflow Correction
Z-Max Type	Pressure
Z-Max Pressure	2118.17

Table 22. Out of Ground Effect Hover at 236 ft/s

Rotor	**Thrust**				
Rear Right	0.58304	lbs	Vertical Force Required for Hover	3.07629884	lbs
Rear Left	0.58347	lbs	Vertical Force Generated by Rotors	2.33162	lbs
Front Right	0.58285	lbs	Disparity in Forces	-0.74467884	lbs
Front Left	0.58226	lbs			

Table 23. Out of Ground Effect Hover at 283 ft/s

Rotor	Thrust				
Rear Right	0.8537	lbs	Vertical Force Required for Hover	3.16992284	lbs
Rear Left	0.85133	lbs	Vertical Force Generated by Rotors	3.40469	lbs
Front Right	0.84703	lbs	Disparity in Forces	0.23476716	lbs
Front Left	0.85263	lbs			

11.2 In Ground Effect

Table 24. Full Configuration in Hover In Ground Effect

	Flight Condition	Tilt	Tip Speed	Time Grid
Case 5A	hover	0	236.094	5.500.
Case 5B	hover	0	283.432	5.500.
Case 5C	hover	0	283.432	10.1000.

Table 25. Full Configuration in Hover In Ground Effect

X-Min Type	Pressure
X-Min Pressure	2118.17
Y-Min Type	Pressure
Y-Min Pressure	2118.17
Y-Max Type	Pressure
Y-Max Pressure	2118.17
Z-Min Type	Velocity
Z-Min Velocity	0
	0
	0
Z-Max Type	Velocity
Z-Max Velocity	0
	0
	-0.5

Table 26. In Ground Effect Hover at 236 ft/s

Rotor	Thrust				
Rear Right	0.623212	lbs	Vertical Force Required for Hover	2.75309964	lbs
Rear Left	0.622323	lbs	Vertical Force Generated by Rotors	2.503954	lbs
Front Right	0.625361	lbs	Disparity in Forces	-0.24914564	lbs
Front Left	0.633058	lbs			

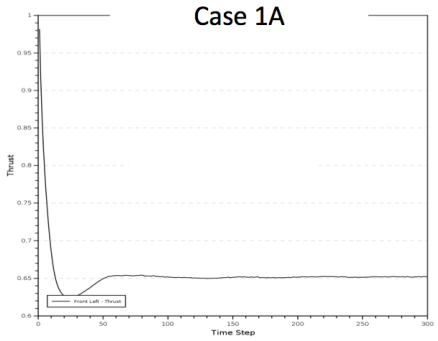
Table 27. In Ground Effect Hover at 283 ft/s

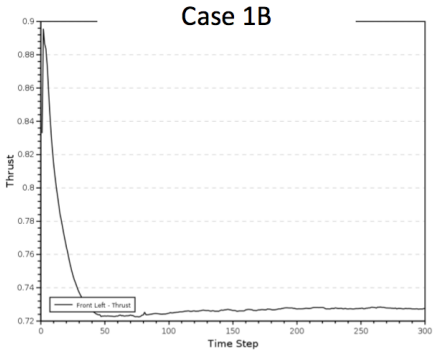
Rotor	Thrust				
Rear Right	0.906603	lbs	Vertical Force Required for Hover	2.75806694	lbs
Rear Left	0.905235	lbs	Vertical Force Generated by Rotors	3.623566	lbs
Front Right	0.89818	lbs	Disparity in Forces	0.86549906	lbs
Front Left	0.913548	lbs			

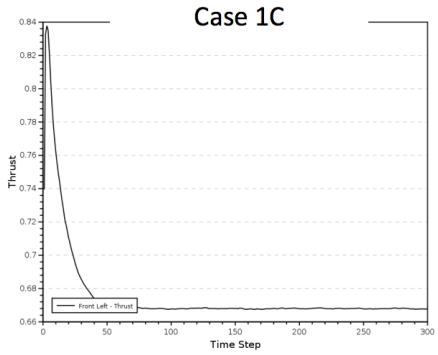
12 Appendix G Rotorcraft: Convergence Data from RotCFD

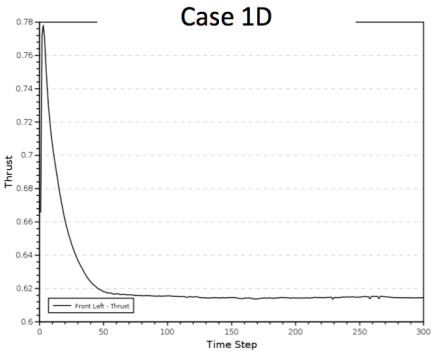
The following plots show the raw output convergence data from RotCFD in terms of either thrust or force versus time step. See the previous appendices for case definition information.

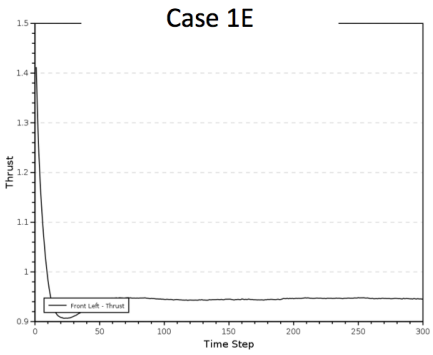
It should be noted that the convergence for the full configuration hover cases (5A, 5B, 5C) do not reach as steady of a final value as the other cases; there is still some oscillation in the force and moment data present. Future work could build upon solving this issue, by running a longer computation with a higher time density.

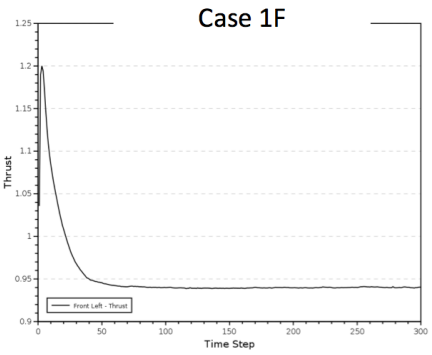

Case 1A
Thrust
Time Step
Front Left - Thrust

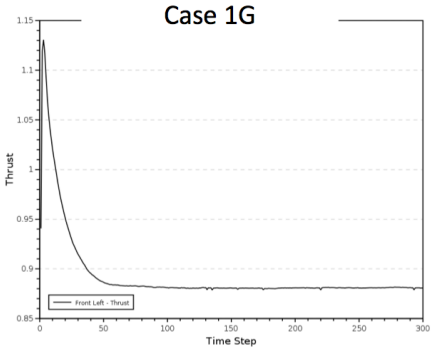

Case 1B
Thrust
Time Step
Front Left - Thrust

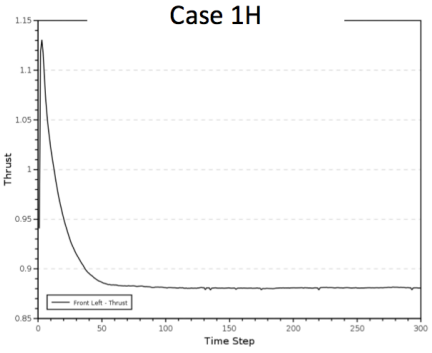

Case 1C
Thrust
Time Step
Front Left - Thrust

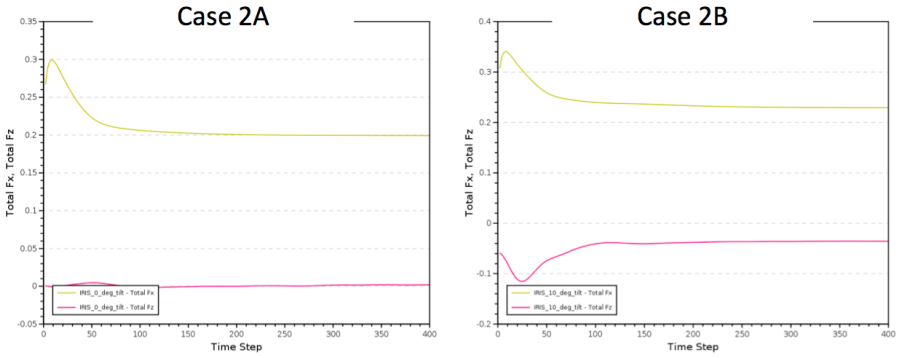

Case 1D
Thrust
Time Step
Front Left - Thrust

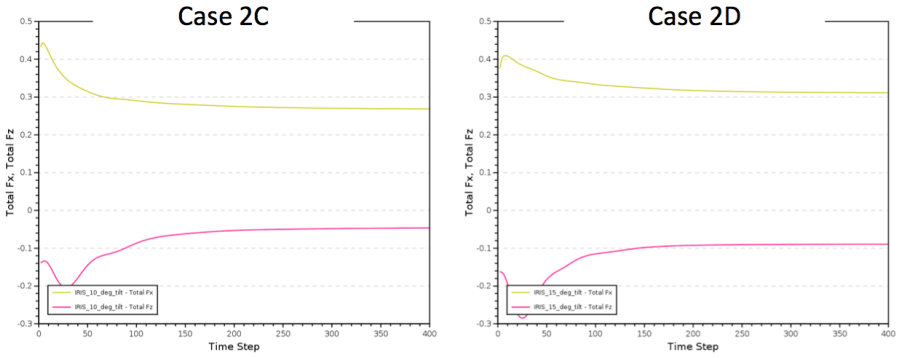

Case 1E
Thrust
Time Step
Front Left - Thrust


Case 1F
Thrust
Time Step
Front Left - Thrust

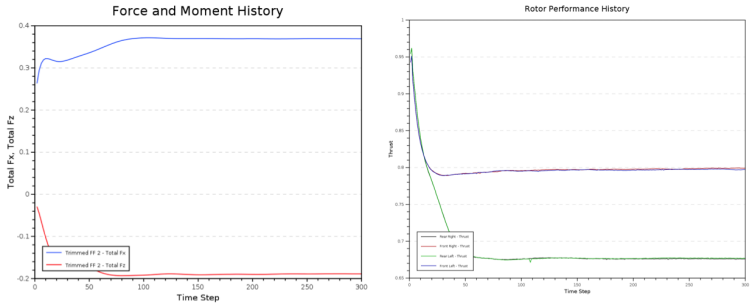

Case 1G
Thrust
Time Step
Front Left - Thrust


Case 1H
Thrust
Time Step
Front Left - Thrust

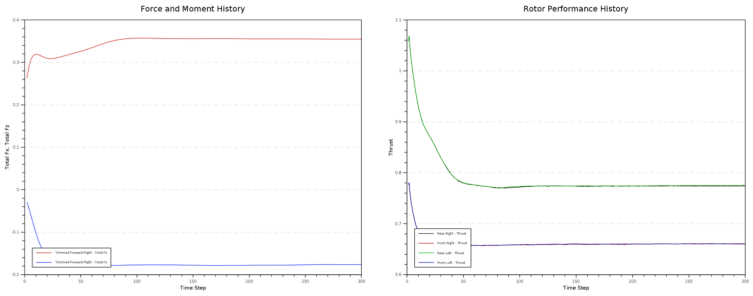




Case 3A

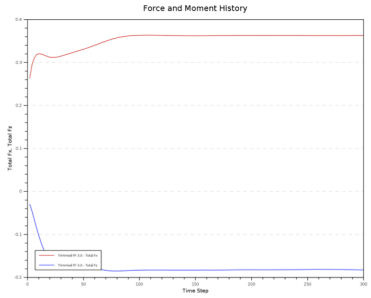


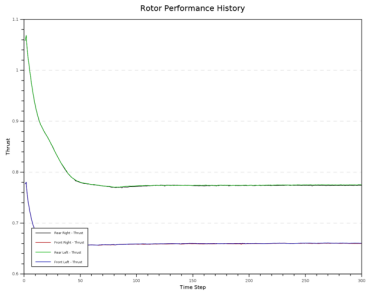
Case 3B



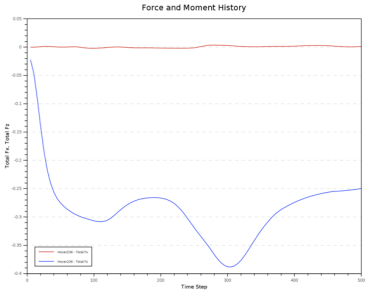
[No Title]

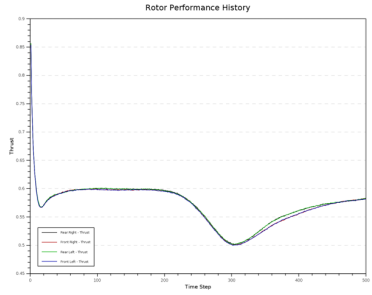
Case 3C



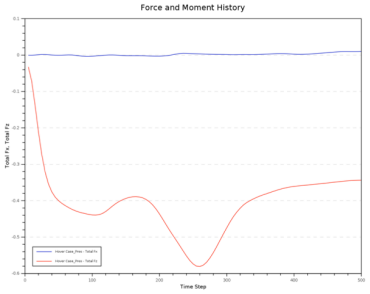


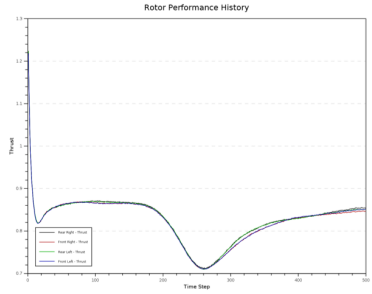
Case 4A



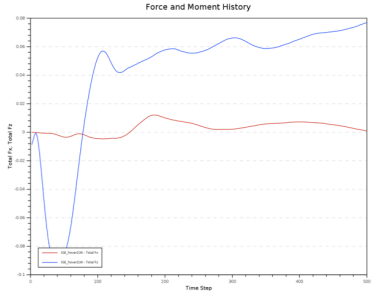


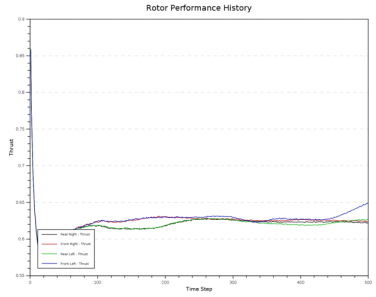
Case 4B



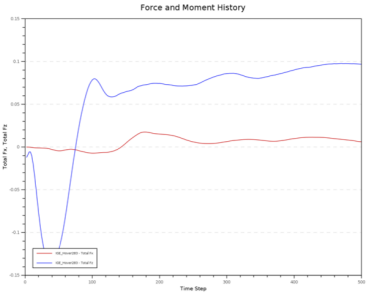


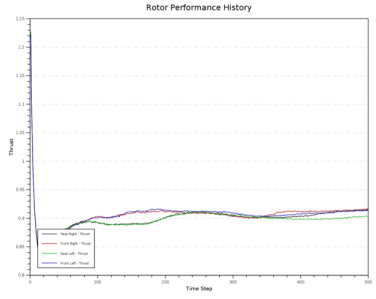
Case 5A



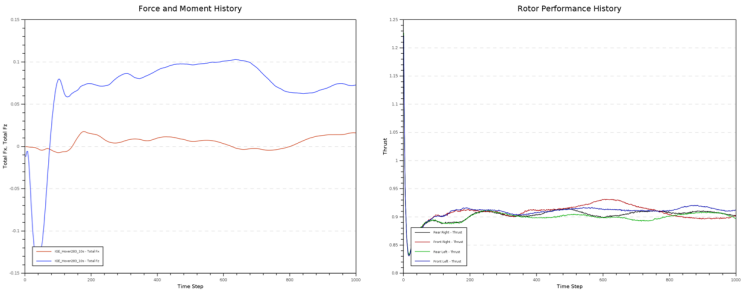


Case 5B





Case 5C



13 Appendix A Urban: CFD results

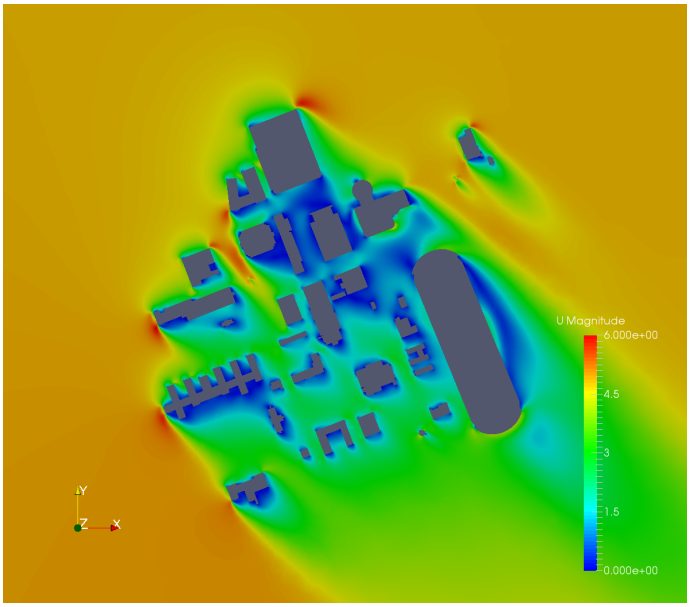


Figure 54. NW15

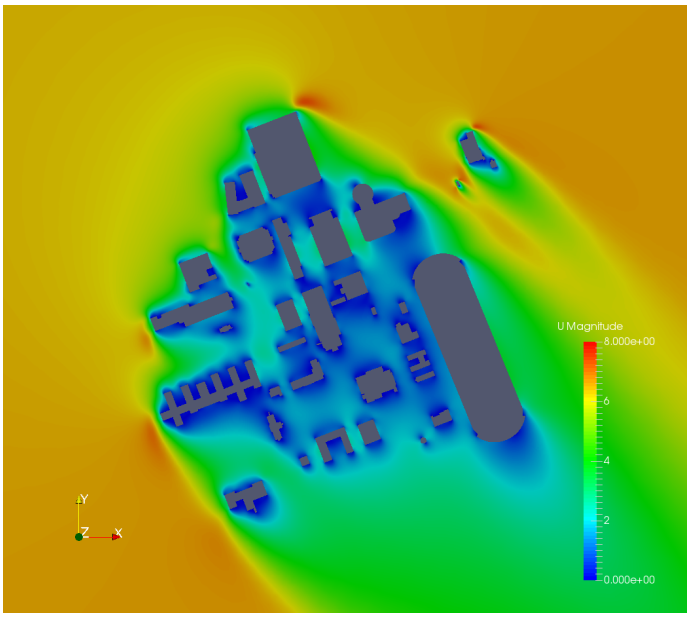


Figure 55. NW20

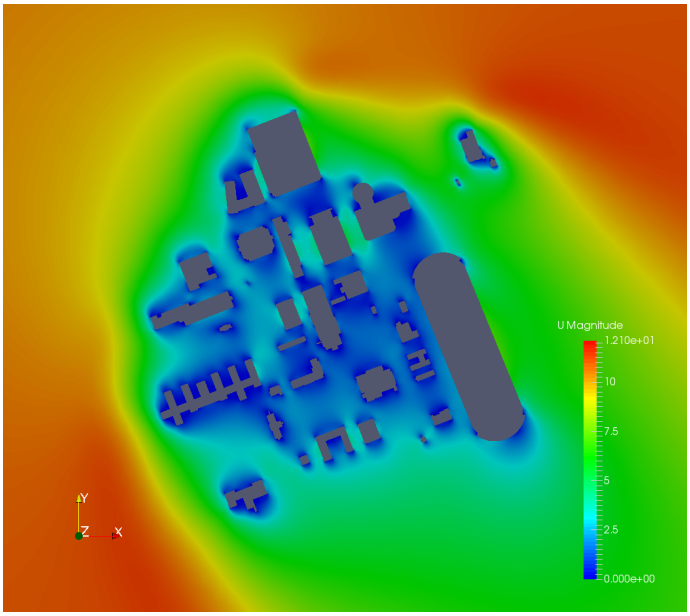


Figure 56. NW25

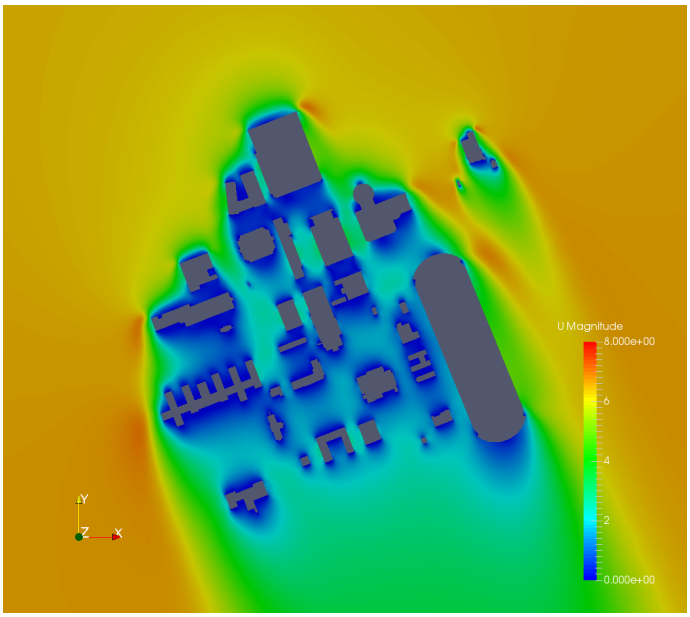


Figure 57. NNW15

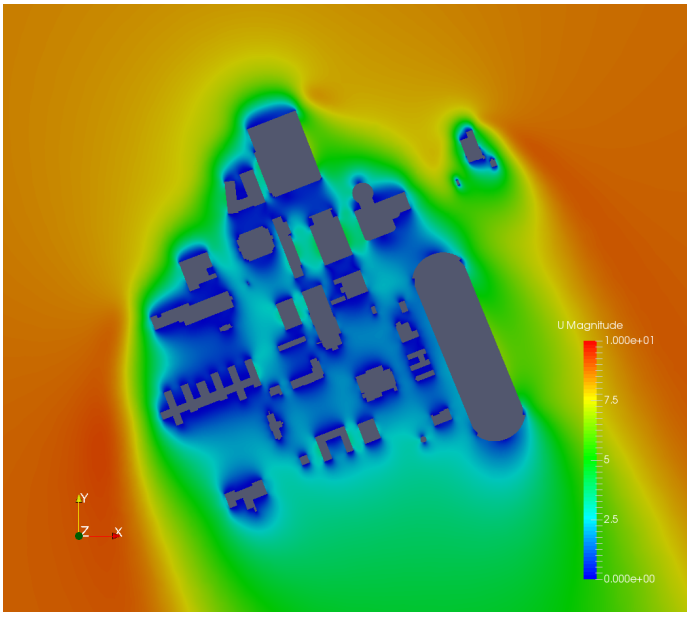


Figure 58. NNW20

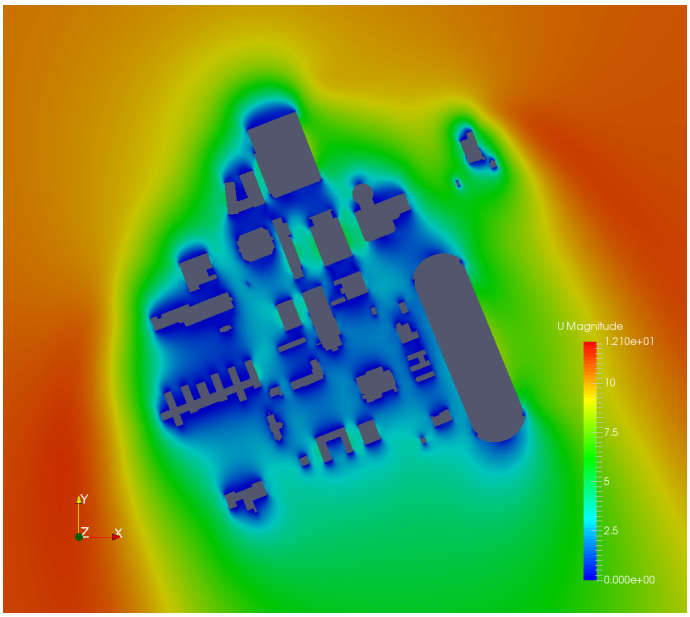


Figure 59. NNW25

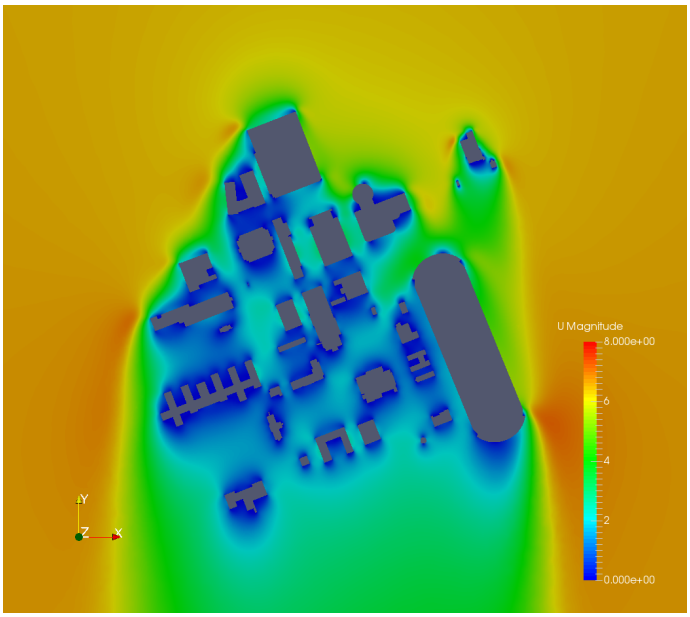


Figure 60. N15

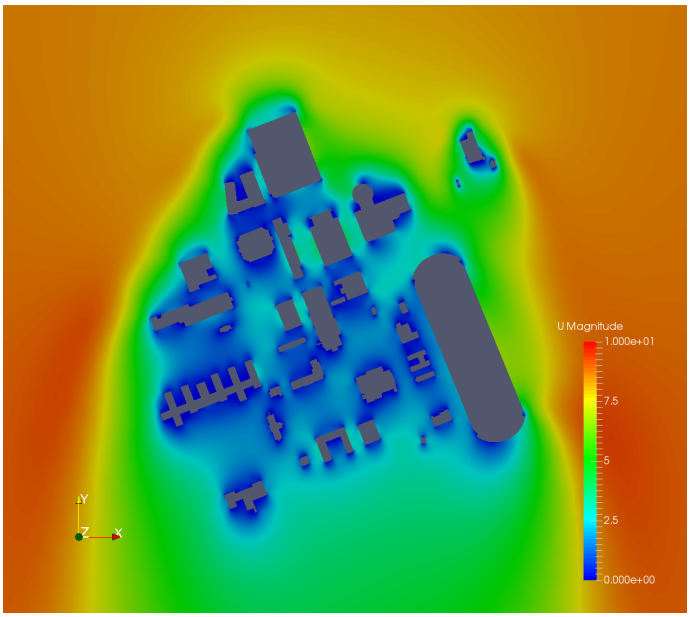


Figure 61. N20

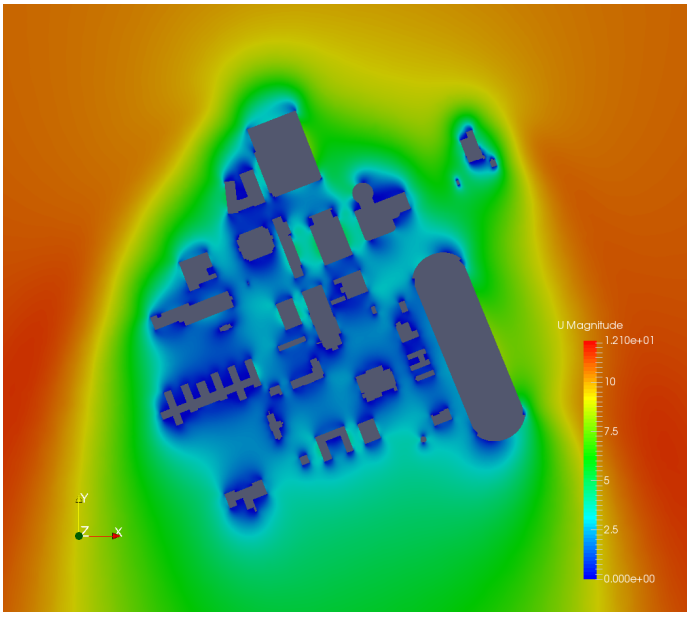


Figure 62. N25

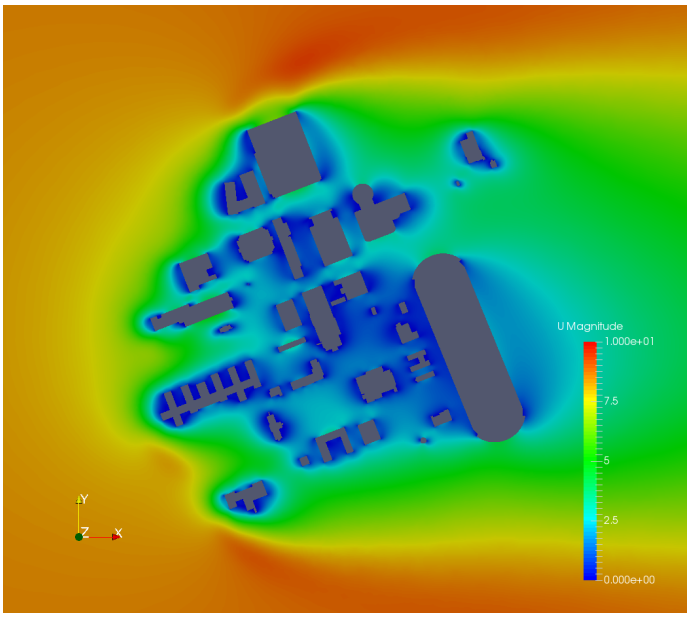


Figure 63. W20

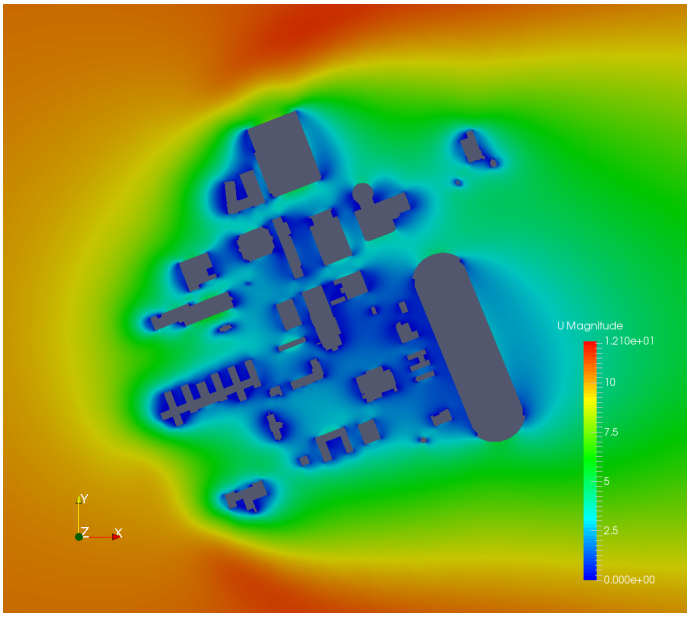


Figure 64. W25

14 Appendix B Urban: Optimization Path

Listing 1. OptimizationPath.m

```
% This code is used to find the optimal path between
    two points given a
```

```
% wind velocity field. The code minimizes the amount
    of time taken and uses
% the Matlab function fmincon

% This program is a modification of a code already
   written by Mathworks.
% Copyright (c) 2012, MathWorks, Inc.
%Distance and time units are in kilometer and hours
   respectively, this was
%done in order to view considerable effects per test



%% set up the velocity field and grid

%This example will use a randomly generated wind
   field
clear all
clc

%Set up speed parameters in km/hr
AirSpeed = 500;
%limits of the grid
sizeX = 50;
sizeY = 25;
%number of waypoints to optimize
numWayPoints = 5;
% set up start and end point
start = [0  0];
finish = [0  0];


i = 1;
while i < 2
    start(1) = input('input the x-coordinate of your
       starting point (must be less than sizeX)\n')
       ;
    if start(1) > 50
        sprintf('Sorry the value exceeds the limit, 
           try again')
        i = 1;
        continue
    else i = i + 1;
    end
end
```

```
i = 1;
while i <2
    start(2) = input('input the y-coordinate of your
       starting point (must be less than sizeX)\n')
       ;
    if start(2) > 25
        sprintf('Sorry the value exceeds the limit , 
           try again')
        i = 1;
        continue
    else i = i + 1;
    end
end
i = 1;
while i < 2
    finish(1) = input('input the x-coordinate of 
       your ending point (must be less than sizeX)\n
       ');
     if finish(1) > 50
        sprintf('Sorry the value exceeds the limit , 
           try again')
        i = 1;
        continue
    else i = i + 1;
     end
end
i = 1;
while i < 2
    finish(2) = input('input the y-coordinate of 
       your ending point (must be less than sizeX)\n
       ');
    if finish(2) > 25
        sprintf('Sorry the value exceeds the limit , 
           try again')
        i = 1;
        continue
    else i = i + 1;
    end
end
i = 1;


rng(50);

%import wind field and generate grid
W_x = makeWindFun(sizeX,sizeY);
```

```
W_y = makeWindFun(sizeX,sizeY);
[X_grid, Y_grid] = meshgrid(0:sizeX,0:sizeY);

%% Add obstacles (Buildings)
numberOfObst = input('Input the number of obstacles
    being added\n');
obstvalue = 300;

for i = 1:numberOfObst

    horizontalHigh = input('Input the y-coordinate
        of the highest side (must be less than sizeY)
        \n');
    verticalHigh = input('Input the x-coordinate of
        the right-most side (must be less than sizeX)
        \n');
    horizontalLow = input('Input the y-coordinate of
        the lowest side(must be less than sizeY)\n')
        ;
    verticalLow = input('Input the x-coordinate of
        the left-most side (must be less than sizeX)\
        n');

    if start(1) < finish(1)
        W_x(horizontalLow:horizontalHigh,verticalLow
            :verticalHigh) = -obstvalue;
    else W_x(horizontalLow:horizontalHigh,
        verticalLow:verticalHigh) = obstvalue;
    end

    if start(2) > finish(2)
        W_y(horizontalLow:horizontalHigh,verticalLow
            :verticalHigh) = obstvalue;
    else W_y(horizontalLow:horizontalHigh,
        verticalLow:verticalHigh) = -obstvalue;
    end
    i = i + 1;
end


%% This section can be used to test a default set of
    way points

%default waypoints
xwaypoints = linspace(0,50,5)';
ywaypoints = linspace(25,0,5)';
```

```
%Generate path points from way points
PathPoints = WayPoints_To_Path([xwaypoints
    ywaypoints], 'linear',sizeX,sizeY,101);

%Calculate the time taken to travel
time = getTimeFromPath(PathPoints,W_x,W_y,AirSpeed);

%display in hours/ minutes
fprintf('Straight line travel time: %d hours, %.1f
    minutes\n',floor(time),rem(time,1)*60);

hold on
plot(xwaypoints,ywaypoints,'color','k','linestyle','
    none','marker','.','markersize','16');
plot(PathPoints(:,1),PathPoints(:,2),'color','k','
    linestyle','none','marker','.','markersize','16')
    ;



%% Find an optimal path using FMINCON
% Define Objective Function
objectiveFun = @(P) getTimeFromPath(P,W_x,W_y,
    AirSpeed,sizeX,sizeY,'pchip',start,finish);

% Set optimization options
opts = optimset('fmincon');
opts.Display = 'iter';
opts.Algorithm = 'active-set';
opts.MaxFunEvals = 5000;
optimset('funValCheck','on');

% Set Initial Conditions
xWayPoints = linspace(0,sizeX,numWayPoints+2)';
yWayPoints = sizeY/2 * ones(numWayPoints+2,1);
ic = [xWayPoints(2:end-1)'; yWayPoints(2:end-1)'];
ic = ic(:);

% Bounds
lb = zeros(size(ic(:)));
ub = reshape([sizeX*ones(1,numWayPoints); sizeY*ones
    (1,numWayPoints)],[],1);

%Perform the optimization
```

```
optimalWayPoints = fmincon(objectiveFun, ic(:),
    [],[],[],[],lb,ub,[],opts)


%% Set up vector plot

hq = quiver(X_grid,Y_grid,W_x,W_y,'k');
hold on;
axis equal tight
plot([start(1) finish(1)],[start(2) finish(2)],'k.',
    'markersize',16)

%% Set up color plot

L = (sqrt((X_grid-sizeX).^2 + (Y_grid-sizeY/2).^2));
Favorability = ((sizeX-X_grid).*W_x + (sizeY/2-
    Y_grid).*W_y)./L;
Favorability(~isfinite(Favorability)) = 0;

% This will be the background for the vector field
hold on;
h_im = imagesc(Favorability);
axis equal tight
set(h_im,'Xdata',[0 sizeX],'Ydata',[0 sizeY]);
uistack(h_im,'bottom');

% Changes the colormap
aa = colormap(interp1([0,1,2],[1 0 0; 1 1 1; 0 1
    0],0:0.01:2));
caxis(max(abs(Favorability(:)))*[-1 1]);

h_colorbar = colorbar;
title(h_colorbar,'Tailwind')
set(h_colorbar,'Ytick',[])
set(gca,'YTick',[],'XTick',[]);



%% Plot the optimal solution:

optimalWayPoints = [start(1) start(2); reshape(
    optimalWayPoints,2,[])'; finish(1) finish(2)];

xWayPoints = optimalWayPoints(:,1);
yWayPoints = optimalWayPoints(:,2);
```

```
h_wp = plot(xWayPoints,yWayPoints,'color','k','
    linestyle','none','marker','.','markersize',16);

PathPoints = WayPoints_To_Path([xWayPoints,
    yWayPoints],'pchip',sizeX,sizeY,101);
h_path = plot(PathPoints(:,1),PathPoints(:,2),'k','
    linewidth',2);
LineTime = getTimeFromPath(PathPoints,W_x,W_y,
    AirSpeed);
%Note that the units being used for this example are
     in km/hr not m/s
fprintf('Optimal Travel Time: %d hours, %.1f minutes
    \n',floor(LineTime),rem(LineTime,1)*60);

h_wp = plot(xWayPoints,yWayPoints,'color','k','
    linestyle','none','marker','.','markersize',16);
```

15 Appendix C Urban: Get Time From Path

Listing 2. getTimeFromPath.m

```
function TravelTime = getTimeFromPath(PathPoints,W_x
    ,W_y,AirSpeed,sizeX,sizeY,METHOD,start,finish)
% This function calculates the line integral
% along the path. It is used to find the time it
    takes to traverse the path.
%
% Copyright (c) 2012, MathWorks, Inc.
%

if isvector(PathPoints)
    PathPoints = [start(1) start(2); reshape(
        PathPoints,2,[])'; finish(1) finish(2)];
    PathPoints = WayPoints_To_Path(PathPoints,METHOD
        ,sizeX,sizeY,101);
end

dP = diff(PathPoints);

% Interpolate the wind vector field at all the
    points in PathPoints.
V_wind = [interp2(W_x,PathPoints(1:end-1,1)+1,
    PathPoints(1:end-1,2)+1,'linear') interp2(W_y,
    PathPoints(1:end-1,1)+1,PathPoints(1:end-1,2)+1,'
    *linear')];

% Dot product the wind (Vwind_path) with the
    direction vector (dP) to get
% the tailwind/headwind contribution
V_add = (sum(V_wind.*dP,2))./sqrt(sum(dP.^2,2));
dx = sqrt(sum(dP.^2,2))*100; %dx is the length of
    each subinterval in PathPoints
dt = dx./(AirSpeed+V_add);  %dT = dP/dV
TravelTime = sum(dt);
```

16 Appendix D Urban: Way Points to Path

Listing 3. WayPointsToPath.m

```
function PathPoints = WayPoints_To_Path(p,METHOD,
    sizeX,sizeY,fineness)
% Interpolate the curve based on the discrete
    waypoints to generate a
% continuous path.
%
% Copyright (c) 2012, MathWorks, Inc.
%

nP = size(p,1);
PathPoints = [interp1(1:nP,p(:,1),linspace(1,nP,
    fineness)',METHOD,'extrap') interp1(1:nP,p(:,2),
    linspace(1,nP,fineness)',METHOD,'extrap')];

% Do not leave the box
PathPoints(:,1) = min(PathPoints(:,1),sizeX);
PathPoints(:,1) = max(PathPoints(:,1),0);
PathPoints(:,2) = min(PathPoints(:,2),sizeY);
PathPoints(:,2) = max(PathPoints(:,2),0);
```

17 Appendix E Urban: Ames Section

Listing 4. AmesSection.m

```
%% Ames model optimized path
%This code reads a small section of the Ames CFD
    results and applies the
%optimization algorithm in order to find the fastest
     path from one point to
%another. In this case the wind field is already
    defined with all of the
%buildings input.
%
%This code is a modification of a code already
    written by
%Mathworks
% Copyright (c) 2012, MathWorks, Inc.

clear all
clc


%%Import data
W_x = xlsread('amesSection.xlsx','W_x');
W_y = xlsread('amesSection.xlsx','W_y');
[X_grid,Y_grid] = meshgrid(0:60,0:54);


%Set up starting parameters
sizeX = 60;
sizeY = 54;
AirSpeed = 20;
numWayPoints = 5;

%%
%Adjust the wind component of the buildings
a = find(W_x == 300);
W_x(a) = -10;
b = find(W_y == 300);
W_y(b) = 10;

%plot figure
figure()
quiver(X_grid,Y_grid,W_x,W_y,'k')
grid on
```

```
axis equal tight

%Set starting and end points
start = [0 sizeY];
finish = [sizeX 0];

%plot it
hold on
plot([start(1) finish(1)],[start(2) finish(2)],'k.',
    'markersize',16)

%% Find an optimal path using FMINCON
% Define Objective Function
objectiveFun = @(P) getTimeFromPath(P,W_x,W_y,
    AirSpeed,sizeX,sizeY,'pchip',start,finish);

% Set optimization options
opts = optimset('fmincon');
opts.Display = 'iter';
opts.Algorithm = 'active-set';
opts.MaxFunEvals = 2000;
optimset('funValCheck','on');

% Set Initial Conditions
xWayPoints = linspace(0,sizeX,numWayPoints+2)';
yWayPoints = sizeY/2 * ones(numWayPoints+2,1);
ic = [xWayPoints(2:end-1)'; yWayPoints(2:end-1)'];
ic = ic(:);

% Bounds
lb = zeros(size(ic(:)));
ub = reshape([sizeX*ones(1,numWayPoints); sizeY*ones
    (1,numWayPoints)],[],1);

%Perform the optimization
optimalWayPoints = fmincon(objectiveFun, ic(:),
    [],[],[],[],lb,ub,[],opts)

%% Plot the optimal solution:

optimalWayPoints = [start(1) start(2); reshape(
    optimalWayPoints,2,[])'; finish(1) finish(2)];

xWayPoints = optimalWayPoints(:,1);
yWayPoints = optimalWayPoints(:,2);
```

```
h_wp = plot(xWayPoints,yWayPoints,'color','k','
    linestyle','none','marker','.','markersize',16);

PathPoints = WayPoints_To_Path([xWayPoints,
    yWayPoints],'pchip',sizeX,sizeY,101);
h_path = plot(PathPoints(:,1),PathPoints(:,2),'k','
    linewidth',2);


%% Make wind field more clear to see

a = find(W_x == -10);
W_x(a) = 0;
b = find(W_y == 10);
W_y(b) = 0;

quiver(X_grid,Y_grid,W_x,W_y,'b')
```

18 Appendix F Urban: Obstacle Generator

Listing 5. buildingAdder.m

```
function [newWindFieldU, newWindFieldV] =
    buildingAdder(wind_u,wind_v, buildingVector,
    spacing, windSpeed)
%% Read in buidling vector
points = length(buildingVector);

%% Initialize start point

    %% Iterate over lines
    for i = 1:points;
        if (i+1) > points
            last = buildingVector(1,:);
        else
            last  = buildingVector(i+1,:);
        end
        start = buildingVector(i,:);

        slope = (last(2)-start(2))/(last(1)-start(1)
            );
        inverseSlope = -slope;
        totalDistance = sqrt((last(2)-start(2))^2+(
            last(1)-start(1))^2);
        distance = 0;
        % u = sqrt(inverseSlope^2*windSpeed^2/(
            inverseSlope^2+1));
        u = sqrt(windSpeed^2/(inverseSlope^2+1));
        delx = sqrt(spacing^2/(1+slope^2));
        if(start(1)>last(1))
            delx = -delx;
        else
            u=-u;
        end

        dely = slope*delx;
        v = u*inverseSlope;
        currentPoint = start;
        while distance < totalDistance
            x = floor(currentPoint(1));
            y = floor(currentPoint(2));
            wind_u(x,y) = u;
            wind_v(x,y) = v;
```

```
            currentPoint(1) = currentPoint(1) + delx
                ;
            currentPoint(2) = currentPoint(2) + dely
                ;
            distance = distance + spacing;
        end
        wind_u(last(1),last(2)) = u;
        wind_v(last(1),last(2)) = v;



    end

    newWindFieldU = wind_u;
    newWindFieldV = wind_v;

%% Output new wind field

end
```

19 Appendix Controls: Code

Listing 6. The initialization file for our Simulink model which inputs inertia data motor parameters controller gains and estimator parameters

```
clear all;
type = '4x';
%% Quadrotor data
m=0.26;
J1=0.0008;
J2=0.00065;
J3=0.0014;
g=9.81;
inertia = [m; J1; J2; J3;m*g];

failtime=15;
delay=0.02;
dist_on=[1; 1; 1; 1; 1; 1]; %Amplitude of sinusoidal
    disturbances
kt=[0.3; 0.3; 0.2];  %drag coefficients
kr=[0.02;0.02;0.01];

%% Motor data

L  = .001;     % motor inductance, H (estimate)
R  = .142;      % motor, resistance, Ohms (from data
    sheet)
```

```
kb = 0.01679618; % Back-EMF constant, V*sec/radians
    (backed out from data sheet)
J  = .001;       % Motor Inertia (without blades) kg *
     m^2
B  = 1e-6;       % Motor friction coefficient, kg * m
    ^2/sec^2
km = kb;         % kg * m^2 * rad/Coloumbs
kf = 1e-6;          % Drag coefficient, kg * m^2 * sec
    /rad
f_max = 5;
f_min = 0;%-5;

%% Control gains
cphi=5;
ctheta=5;
cpsi=1;

cphi_p=2.4;
ctheta_p=3.2;
cpsi_p=1.5;

cp=2.4*7;
cq=3.2*7;
cr=20;

cz2=0.3;
cz1=2*0.9*sqrt(cz2);

cx2=0.2;
cx1=2*0.9*sqrt(cx2);

cy2=0.2;
cy1=2*0.9*sqrt(cy2);

%% Motor Matrix

numb_motors = str2num(type(1));
Bmat = zeros(numb_motors,4,2^numb_motors);
[Bmat,B] = MatrixAggregator(type);


%% Estimator parameters
gammaGen = 100;

gammaX  = gammaGen;
gammaY  = gammaGen;
```

```
gammaZ  = gammaGen;
gammaP  = gammaGen;
gammaQ  = gammaGen;
gammaR  = gammaGen;

lambdaX = 2*sqrt(gammaGen)*m;
lambdaY = 2*sqrt(gammaGen)*m;
lambdaZ = 2*sqrt(gammaGen)*m;
lambdaP = 2*sqrt(gammaGen)*J1;
lambdaQ = 2*sqrt(gammaGen)*J2;
lambdaR = 2*sqrt(gammaGen)*J3;

params = [lambdaX; lambdaY; lambdaZ; lambdaP;
    lambdaQ; lambdaR; gammaX; gammaY; gammaZ; gammaP;
     gammaQ; gammaR];


%% Other stuff
A  = eye(4);
top = zeros(4,4);
bottom = eye(4);
top(1,1) = 1;
bottom (1,1) = 0;
failed_motors = [];
```

Listing 7. This function creates the set of all possible motor allocation matrices in order to save computation time

```
function [ Bmat, B ] = MatrixAggregator(type)
%This function takes as input the multirotor type (4
    x, 6+) and outputs the 3-dimensional array of
    possible motor allocation matrices (Bmat) and the
     B matrix of multirotor type
disp('hello')
numb_motors = str2num(type(1));
Bmat = zeros(numb_motors,4,2^numb_motors-1); %
    Massive matrix that accounts for every possible
    combination (2^numb_motors)of motor failures
%Bmat is a matrix of all the
%needed matrix inverses to
%immediately convert to force
inner = 360/numb_motors;

%% Starting Angle
if type(2) == '+'
     offset = 0;
else%'x'
```

```
    offset = -inner/2;
end

%% Build Matrix
% angles = zeros(1,numb_motors);
% radii= zeros(1,numb_motors);
% z= zeros(1,numb_motors);
% A= zeros(1,numb_motors);
for j = 1:numb_motors;
    angles(j) = pi/180*(offset+inner*(j-1));
    radii(j) = .3;
    z(j) = 0;
    A(j) = 0.03*(-1)^(j);
end
%A = [3, 3,-3,-3, -3, 3];
%radii = [2,4,4,2,4,4];
% angles = [36,108,252,324]*pi/180
% radii = [3,3,3,3];
% z = [0,0,0,0];
% A = [3,-3,-3,3];
B= MatrixBuilder(radii,angles,z,A); %original matrix
    , has dimensions 4 x numb_motors
%Bmat(:,:,2^numb_motors) = B^-1

for ii = 1:2^numb_motors-1
    T = 4;
    Btemp = B;
    failed_motors = zeros(1, numb_motors);
    %% Adjust Control Matrix
    %Determine which motors have failed and adjust
        control matrix accordingly
    status = dec2bin(2^numb_motors + ii); % Need at
        least digits in status as the number of
        motors.
                                          %2^
                                              numb_motors
                                               ensures
                                               there
                                              are
                                          %exactly
                                              num_motors
                                              +1
                                              digits
                                              and
                                          %only
                                              results
```

```
                                                    in a
                                                    digit
                                                    shift

tot_failed = 0;
for i = 1:numb_motors
    if status(i + 1) == '0' %if the i'th motor
       has failed
        disp('cutting out columns')
        Btemp(:,i-tot_failed) = []; %cut out the
             ith column in the matrix
        tot_failed = tot_failed + 1;
        failed_motors(i) = i; %index the new
           failed motor
    end
end
net = numb_motors-length(failed_motors);
CW = sum(Btemp(4,:)<0);
CCW = sum(Btemp(4,:)>0);

%% Decision Making Protocol
%Decide Safemode or not
if CW < 2 || CCW < 2  % enter safemode
    disp('Safe Mode, Sacrificing Yaw')
    T = 3; %get rid of the yaw dimension
    mode = 0;
    Btemp(4,:) = [];
elseif det(Btemp*Btemp') < .0000001%condition
    disp('Singular matrix, Sacrificing Yaw')
    T = 3; %get rid of the yaw dimension
    Btemp(4,:) = [];
    mode = 0;
else
    disp('Nominal Control')
    mode = 1;%mode is 1 if nominal
end
% Decide how to calculate forces, based on
   matrix dimensions
if net == T
    disp('Matched dimensions')
    Binv = Btemp^-1;
else
    disp('Unmatched dimensions')
    Binv = (Btemp'*(Btemp*Btemp')^-1);
end
sz = size(Binv);
```

```
        Binv = [Binv, zeros(sz(1),4-sz(2))];
        Binv = [Binv; zeros(numb_motors-sz(1),4)];
        if (~any(isnan(Binv)) == 1) & (~any(isinf(Binv))
            == 1)
            Bmat(:,:,ii) = Binv;
        end
    end

end
```

Listing 8. This function creates a B matrix for a multirotor given motor positions and spin directions

```
function [ B ] = MatrixBuilder( radii,angles, z,
    thrust2drag)
%Builds Control matrix based on location of motors
%Sample: MatrixBuilder([2,2,2,2,2], pi
    /180*[-36,-36+72,-36+72*2,-36+72*3,-36+72*4],
    [0,0,0,0,0], [3,3,3,3,3]);
len = length(radii);
if len ~= length(angles)
    disp('ERROR: Vector lengths do not match')
elseif len ~= length(z)
    disp('ERROR: Vector lengths do not match')
end

B = ones(4,len);
for ii = 1:len
    B(2,ii) = -sin(angles(ii))*radii(ii);
    B(3,ii) = cos(angles(ii))*radii(ii);
    B(4,ii) = thrust2drag(ii);
end
end
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-08-2015	Technical Memorandum	06/2015–08/2015

4. TITLE AND SUBTITLE
Urban SAFE50: MARTI Project 2015

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Ames Research Center
Moffett Field, Mountain View, CA 94035

8. PERFORMING ORGANIZATION REPORT NUMBER
L–

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)
NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NASA/TM–2015–

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Category
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES
An electronic version can be found at http://ntrs.nasa.gov.

14. ABSTRACT
In NASAs quest to develop Unmanned Aerial Systems Traffic Management (UTM), there are many problems which must be solved. Some of the most interesting challenges come from small UAS which can operate below 50 feet AGL. While these UAS offer many benefits to society, including the potential for same-day package delivery and infrastructure monitoring, they must be thoroughly studied and tested to ensure they do not present a danger to the National Airspace or those on the ground. Enter Urban SAFE50. By performing CFD and trim analysis on a COTS quadcopter, studying wind flow around buildings, performing battery prognostics, and implementing and tuning control algorithms, we have begun the necessary detailed analysis which must be performed on small multirotors to ensure they are safe to fly below 50 feet.
Simulating the airflow around these platforms allows us to predict problems which might arise due to the wake of the aircraft. For example, if one multirotor flies into the wake of another, it may stall and fall onto the ground below, damaging property or hurting people. Using RotCFD, we performed computational fluid dynamics analysis on the effect of a commercially available quadcopter within the fluid domain. In particular, a trimming scheme was developed and implemented which aimed to place 3 different representative quadrotor cases in trim (Hover OGE, Hover IGE, and steady forward flight at 20 mph). The trim coefficients for forward flight at 20 mph were found, as well as the required motor speeds for hover in and out of ground effect. This will allow for robust, representative fluid dynamics modeling so that a better understanding of the behavior of these craft within the fluid domain may be better understood, and may also be used in the development of more accurate and safe autopilot controls.
Studying the wind flow around buildings allows us to identify problem areas in urban cities where UAS should not fly. For example, if all areas within a few feet of buildings present very turbulent winds, it should be regulated that all UAS stay out of that area. After analyzing the CFD of a model of Ames Research Center in OpenFoam, we determined that problem areas can include corners of buildings facing the free stream, wakes caused by architecture, and urban canyon settings. Using the wind field obtained from CFD results, we also developed an optimal route planner, which minimizes time by taking into account distance travelled and wind velocity. By running more thorough CFD studies on larger city scapes, the route planner can be used to calculate the best route around a windy city, potentially saving a UAS battery power and time.

15. SUBJECT TERMS
UAS SAFE50 CFD control prognostics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Information Desk (email: help@sti.nasa.gov)
U	U	U	UU		19b. TELEPHONE NUMBER *(Include area code)* (757) 864-9658