UNIVERSITY OF CALGARY

Quad-Rotor Application of an Introspective Weight Voting Cerebellar Model Articulation Controller

by

Troy Clark

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

September, 2015

Abstract

A quad-rotor is an agile, four rotor aerial vehicle capable of operating in a variety of environments while carrying diverse payloads. This mobile platform has been applied in numerous fields including surveillance, search & rescue, mapping, and most recently, package delivery. However, the quad-rotor is an underdamped and underactuated vehicle that requires a stable control which must be robust to environmental disturbances, such as wind, and adaptive to diverse payloads. The Cerebellar Model Articulation Controller (CMAC) is an adaptive control that has proven itself to be stable in the literature. However, even mathematically robust weight update schemes for the CMAC have been shown to exhibit a detrimental bursting phenomenon especially in practice. This work applies an introspective weight voting scheme to a CMAC to provide a stable, adaptive, robust quad-rotor control scheme in both simulation and practice.

Acknowledgements

I would like to acknowledge Dr. Chris Macnab for his continued guidance and wisdom throughout the course of my research. Sometimes through his words, and often simply by listening, Chris allowed me to focus my efforts on the successful completion of my work. I would also like to acknowledge Dr. Abraham Fapojuwo whose confidence in my potential prompted me to begin my journey at the University of Calgary. And finally, I would like to acknowledge the members of my family, those that have been with me from birth and those that have been added over the years, that gave me strength along the way.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
CES	Consumer Electronics Show
CMAC	Cerebellar Model Articulation Controller
DoF	Degree of Freedom
GAS	Globally Asymptotically Stable
GPS	Global Positioning System
LED	Light Emitting Diode
LQ	Linear Quadratic
LQR	Linear Quadratic Regulator
MAV	Micro Air Vehicle
MLP	Multi-Layer Perceptron
MRAC	Model Reference Adaptive Control
NN	Neural Network
OCTP	Optimized Computed-Torque Program
PID	Proportional Integral Derivative
PD	Proportional Derivative
PWM	Pulse Width Modulation
RBFN	Radial Basis Function Network
RMS	Root Mean Squared
RPM	Revolutions Per Minute
STR	Self-Tuning Regulator
UAS	Uniformly Asymptotically Stable
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol

UUB	Uniformly Ultimately Bounded
VTOL	Vertical Take-off and Land
b	Thrust factor
$\hat{b}_i$	CMAC best weight estimate
C	Trigonometric cosine function
C_T	Thrust coefficient
d	Aerodynamic drag factor
E_i	Average output error
F	Force
F_g	Gravitational force
F_T	Force due to wing thrust
F_{T_L}	Force due to leading wing thrust
F_{T_T}	Force due to trailing wing thrust
g	Gravitational constant
I	Inertia
I_{xx}	Inertial tensor in the x plane
I_{yy}	Inertial tensor in the y plane
I_{zz}	Inertial tensor in the z plane
I_r	Rotor inertia
k_i	Control Gain
l	Distance
m	Mass
p_i	CMAC permanent weight value
$\hat{p}_i$	CMAC permanent weight estimate
$\tilde{p}_i$	CMAC permanent weight error
$\dot{\hat{p}}_i$	CMAC permanent weight update

r	Radius
$R_{\psi,E}$	Body roll to Earth frame of reference matrix
$R_{\theta,E}$	Body pitch to Earth frame of reference matrix
$R_{\phi,E}$	Body yaw to Earth frame of reference matrix
$R_{\psi\theta\phi,E}$	Combined Body to Earth frame of reference matrix
S	Trigonometric sine function
S_i	Smallest output error value
t	Time
t_i	Cell activation time
U_i	Control signal
v	Linear velocity
V	Lyapunov function
$\dot{V}$	Lyapunov derivative
w_i	CMAC adaptive weight value
$\hat{w}_i$	CMAC adaptive weight estimate
$\tilde{w}_i$	CMAC adaptive weight error
$\dot{\hat{w}}_i$	CMAC adaptive weight update
x	Linear position along the x plane
X	Earth reference position along the x plane
$\dot{X}$	Desired earth position along the x plane
y	Linear position along the y plane
Y	Earth reference position along the y plane
$\dot{Y}$	Desired earth position along the y plane
z	Linear position along the z plane
z_d	Desired linear position along the z plane
$\dot{x}$	Linear velocity along the x plane

$\dot{y}$	Linear velocity along the y plane
$\dot{z}$	Linear velocity along the z plane
$\dot{z}_d$	Desired linear velocity along the z plane
$\ddot{x}$	Linear acceleration along the x plane
$\ddot{y}$	Linear acceleration along the y plane
$\ddot{z}$	Linear acceleration along the z plane
α	Control term
β	Adaptation gain
Γ	CMAC activation function output
ζ	Error signal
θ	Angular position along the y plane
θ_d	Desired angular position along the y plane
$\dot{\theta}$	Angular velocity along the y plane
$\dot{\theta}_d$	Desired velocity position along the y plane
$\ddot{\theta}$	Angular acceleration along the y plane
λ	Control gain
μ	Weight leanring gain
ν	Learning gain
ρ	Air density
$\sigma_{E,min}$	Minimum error improvement constant
τ	Torque
τ_b	Reactionary body torque
τ_r	Rotor torque
ϕ	Angular position along the x plane
ϕd	Desired angular position along the x plane
$\dot{\phi}$	Angular velocity along the x plane

$\dot{\phi}_d$	Desired velocity position along the x plane
$\ddot{\phi}$	Angular acceleration along the x plane
ψ	Angular position along the z plane
ψd	Desired angular position along the z plane
$\dot{\psi}$	Angular velocity along the z plane
$\dot{\psi}_d$	Desired velocity position along the z plane
$\ddot{\psi}$	Angular acceleration along the z plane
Ω	Rotor angular velocity, single rotor
Ω_r	Summation of combined rotor angular velocity

Chapter 1

Introduction

1.1 Motivation

Throughout its history, humanity has demonstrated a desire to improve its quality of life. Often, this requires automation of menial or dangerous tasks. In the past, many of these tasks would fall down to those on the bottom rungs of the class ladder, but, with increased technical knowledge and morality, these tasks are now more often delegated to machines. More and more, we predict that robots might sense, articulate, and even think for us in areas where repetitive stress or the risk of life loss lower human quality of life. As a result, stable, robust, autonomous, robotic platforms are becoming more desirable.

Over land, under the sea, and through the air there is no shortage of opportunity for vehicle automation. Real-world testing has already begun to pave the way for automated ground transportation of both passengers and goods. However, where human and automated controllers share the same asphalt autonomous control moves beyond system modelling and actuation towards human decision modelling and reactionary movement. On the open sea, greater focus can be placed on actuation and control in an environment that can be tremendously dangerous to human life. However, vehicle speeds both on and under the waves are greatly reduced and present less of a challenge to a control system than a land or even air based environment. Of all three environments, the air presents vehicles with fast articulation speeds, numerous system disturbances, and overall the most challenging control environment.

From myths of our earliest days, we know that the human desire for flight is ancient. The ability not only to travel with greater ease, but to interact with objects never before reachable has driven inventors from Leonardo DiVinci to Orville & Wilbur Wright, to Dr. George de Bothezat, to Igor Sikorsky to design flying machines [90]. In the search for a stable, articulate

flying platform, a four rotor helicopter, known as a quad-rotor, is often explored due to its gyroscopic balance, ease of control, and six degree of freedom (DoF) manoeuvrability. However, with six input directions and only four output rotor speed controls, the under-actuated quad-rotor is difficult for a human to control, without some level of automation.

In the past, the physical capabilities of a quad-rotor were limited by the complexity of the control system applied and the computing power it required. However, computational power has begun to diverge from Gordon Moore's computational paradigm and significant investment has been made into smaller, more powerful energy storage units [36]. While we have lived with Moore's notion of doubling the number and speed of transistors on a single integrated circuit for some 60 years now, we have unsuspectingly approached an epoch where computational processing power equivalent to that of the human brain may be purchased for a mere $1,000 [67]. As the technology of computing and power plateau at a level that has not been observed in years, and may not increase for some time to come, now is an optimal time to explore new, more advanced methods of robotic control. Such is the motivation behind this work to analyze the stability, robustness, and adaptability of a Cerebellar Model Articulation Controller (CMAC) with introspective voting weight updates for quad-rotor control in theory, simulation, and experiment [6]. A novel robotic control method developed as an analogue to the human motor control function over forty years ago has become viable. A robot designed to sense, move, and react like a human being is possible.

1.2 Background

A quad-rotor is a four rotor, vertical take-off and landing (VTOL) capable, multi-rotor aerial vehicle (MAV) with six DoF movement ability. Each of its rotors spins in a direction that opposes that of its two neighbours and can be varied in speed to provide control. Attitude and altitude are measured using an array of fused measurements taken by sensors including, but not limited to, gyroscopes, accelerometers, sonar, and global positioning systems (GPS).

Its small size, manoeuvrability, and payload carrying capabilities have allowed it to find applications in surveillance, surveying, inspection, and mapping fields. For these applications, it is desirable to have a vehicle that performs predictably in the face of environmental disturbances, mechanical changes, and payload deviations.

1.2.1 Military Applications

The oldest, and most lucrative, applications for Unmanned Aerial Vehicles (UAVs) including the quad-rotor can be found in the military theatre. From humble beginnings in a Los Angeles garage in 1980, the Predator program has become the infamous poster child of the booming UAV presence in the military [37]. From their inception, it was necessary for unmanned craft to prove their reliability before they would be accepted into service. Nonetheless, even with future improvements still to be made to robustness and stability, the value of UAVs has resulted in widespread, immediate demand. Within the next ten years, thousands of drones, including Shrike quad-rotors from AeroVironmental, are expected to find their way into the backpacks of American soldiers [37]. As a result, the revenue of AeroVironmental, who also build a civilian, Qube quad-rotor, has grown from approximately $30M USD in 2001 to nearly $300M USD in just ten years [37].

Even here in Canada, with a fraction of the generous budget afforded to the United States military, VTOL UAVs have been installed in military service since Canadair designed the CL-227 in 1977. Its successor, the CL-327, was at one time the "worlds most advanced VTOL UAV in production today" [105]. Though its primary objective was to provide information, a number of environmental and control challenges for the VTOL class of UAVs were documented and solved by the Canadair and Bombardier teams. The result of the Sentinel, affectionately referred to as the Flying Peanut, and later the Guardian, is that VTOL UAVs have carved out a permanent place within military applications. As retired Major General Fraser Holman puts it, "...their accessibility and responsiveness make them invaluable in the tactical realm" [52].

1.2.2 Civilian Applications

The quad-rotor vehicle has found use as a platform for other, civilian applications as well, including exploration, surveying, inspection, search and rescue, package delivery, and even acting. In dangerous, dynamic areas, such as landslide paths, the quad-rotors has proven a valuable platform for data acquisition tools that aid in map generation and understanding of debris flows [72] [103]. The benefits of a quad-rotor in construction applications, as an intermediary between safety managers and workers has been extensively explored [55]. Here the mobility capability and autonomous potential of the platform provide the frequency and detail of observation required for effective safety inspections to protect workers in dangerous occupations. Furthermore, even in more static locations, the quad-rotor has proven a valuable tool for tasks that carry inherent risk or prohibitive cost such as high voltage power transmission line inspection [75]. According to the Chugoku Electric Power company in Japan, not only do autonomous vehicles perform this task more safely, but "the total cost of power line inspection with an unmanned helicopter is approximately half that of manned helicopter inspection" [97].

In less overtly dangerous applications, such as vegetation monitoring, UAVs fill an important niche between satellite imagery and ground based sensors. In the delicate ecological environment of Robinson Ridge in the Antarctic, a multi-rotor vehicle has been used to monitor mass beds for signs of climate change [74]. In another mapping scheme, a UAV was the platform that allowed a novel civil mapping application to successfully navigate around structures in three dimensional space and accurately obtain real-time mapping feedback [120]. In their conclusion the team, which included Sebastian Thrun, expressed an explicit desire for autonomy. Sebastian went on to work on the Google autonomous car project, so his insight into future applications is tremendously encouraging for the quad-rotor platform [85].

Where search and rescue operations are concerned, UAVs with traditional and infra-

red cameras offer extended mobility and sensing capabilities over land or water based craft. Additionally, their manoeuvrability and payload capacity are ideal to provide monitoring and aid after stage one reconnaissance is successful [7]. The quad-rotor has even been suggested for extraterrestrial exploration. On Saturn's' largest moon, Titan, the quad-rotor has been proposed to fulfill the active role of a multi-vehicle team designed to explore and map surface topography, thereby furthering scientific knowledge of the moon, planet, and the cosmos in general [89].

Back on planet Earth, UAVs are currently used for more humble, yet equally important daily tasks. From traffic pattern recognition and control optimization to effective crop management, UAVs are increasingly bearing the load of often mundane every day tasks [48]. Amazon has created significant excitement in the UAV field by suggesting that a fleet autonomous quad-rotor vehicles could be implemented to deliver physical goods within as little as thirty minutes of an on-line purchase. A number of obstacles from regulation to energy requirements must be overcome, but research into life cycle assessment of drone-based package delivery is ongoing [114]. Massachusetts Institute of Technology (MIT) has noted the need for autonomous vehicle development and created a Real-Time indoor Autonomous Vehicle testing ENvironment (RAVEN) specifically to refine control schemes for UAVs including quad-rotors [53]. With an advanced motion capture system, MIT is developing technologies to organize vehicles to autonomously perform high level tasks. These tasks could include search and rescue capabilities, for which the University of Oxford has developed search techniques specifically for quad-rotor UAVs [125].

Finally, on a cultural scale, UAVs have been used in photogrammetric applications to visually and mathematically preserve historical locations [33]. Even in more dynamic applications, such as sports cinematography, the speed, agility, and autonomous capabilities of the quad-rotor UAV provide the perfect platform for a camera. Using an AR.Drone quad-rotor as a free floating platform, research has been conducted to determine how target tracking

and virtual reality displays can be used within sporting venues. This work provides not only exciting new viewing possibilities for spectators, but more importantly, facilitates greater visualization opportunities which are incredibly important for athletes during training [49] [50]. Once more, the quad-rotor provides a means to increase our quality of life without compromising safety.

Research is increasingly moving beyond simple quad-rotor control and into the realm of human interaction. Taking cues from familiar fields such as falconeering, the development of natural gesture control for a quad-rotor has been successfully explored at the University of Calgary [100]. On a more intimate level, the quad-rotor has even been suggested as a sort of combination companion and coach for jogging applications. In this facility, the AR.Drone based platform used has been lovingly dubbed the joggobot [42]. From motivating to entertaining, practical work has been completed to synchronize oscillating movements of a quad-rotor to the amplitude and frequency of music. Researchers at ETH Zurich envision regular quad-rotor performances in a "Flying Machine Arena" [111]. Also performing in the flying machine arena are juggling quad-rotor robots, which demonstrate the tremendous motion capture and computational power available within the $10m^3$ arena facility. Using a Kalman filter to estimate the trajectory of a ping pong ball, the team applied linear control to successfully calculate an impact trajectory that allows a racket wielding quad-rotor to return ball tosses, juggle alone, and even rally with a second vehicle [98]. Also using a linear Proportional Integral Derivative (PID) control and the motion capture capabilities of the Flying Machine Arena, an iterative parameter learning scheme was used to push a Hummingbird quad-rotor vehicle well beyond its normal modelled parameters and successfully demonstrate a triple backflip [73]. Furthermore, in a more active role an Air Robot 100-b quad-rotor found itself in a company of actors at a Texas A&M production of *A Midsummer Nights Dream* [99]. This project delved into the nature of the increasingly common interaction between robots and humans to determine how we might interface with autonomous

vehicles, such as quad-rotors, in the near future. And it is a future that is coming closer everyday, as other disciplines are already investigating the moral and ethical questions that will arise with more frequent UAV use [126]. Greater control and autonomy is coming for the quad-rotor family of vehicles, and it is coming at great speed.

1.2.3 Quad-rotor Design

The quad-rotor is applicable in so many different environments to perform so many tasks due to its vertical take-off and land ability, manoeuvrability, and stability. While other platforms share some of these characteristics, none can boast all. A traditional plane, for example, may be able to cover greater distances at higher speeds, but requires a specific area of significant size to take-off and land. Its inability to perform vertical take-off and land manoeuvres, coupled with a lack of low speed agility, means that a plane can be passed over for exploration, mapping, and inspection applications. A lighter than air ship, such as a blimp or balloon, may provide extended range over its heavier counterparts, and even provides vertical take-off and land capabilities. However, like the plane, these too suffer from a lack of agility as well as susceptibility to wind. Finally, cousin to the quad-rotor, the helicopter enjoys most of the same benefits, with one exception. A dual rotor or tail stabilizer system must be employed to counter the gyroscopic effect generated by a single rotor. This increases mechanical complexity and results in a motivation scheme that is significantly more difficult for a human operator to control, let alone an autonomous system.

A quad-rotor then provides agility, disturbance rejection, and mechanical stability. However, quad-rotor control is not without its own set of specific problems. A quad-rotor uses just four motor outputs to actuate control over a total of six degrees of freedom in a three dimensional operating environment. This under-actuation means that when one degree of freedom undergoes a control force, all others are affected as well. In addition, the quad-rotor has little opposition or friction to retard its movement, especially in the x and y planes. This under-damped nature compounds the control challenge presented by under-actuation. Sta-

bility can be improved through mechanical means by designing vehicles with large moments of inertia and orientating the craft in an X formation [107] [47]. However, these qualities, coupled with non-linear thrust dynamics such as temperature, altitude, atmospheric pressure, angle of attack, and airspeed have historically made the quad-rotor difficult to control via human intuition alone. Luckily, control systems exist to solve these problems, allowing human intellectual effort to be turned to higher level applications for this platform.

Linear controls, ranging from simple proportional derivative (PD) control [58] [57], to more advanced variations have been proposed with some success [70], but often in a purely theoretical forum. Instead, non-linear systems are often proposed for practical efforts, including fuzzy regulators [24], back steppers [15], feedback linearizers [123], and neural networks (NN) [124]. After performing well in simulated systems with modelled disturbances, some of these systems have been put into practice where more real-world challenges have been uncovered and documented.

1.3 Literature Review

Within the literature a number of quad-rotor projects, past and present, can be found that have attempted to leverage quad-rotor control complexity against available computational power to unlock the potential of this vehicular platform. Some researchers have dedicated significant portions of their lives to the exploration of quad-rotor design and control, others have contributed smaller but tremendously important kernels of knowledge to the field, and still others have made novel advancements in other fields that can be used here. From a veritable mountain of available literature, the significant, unique, and applicable reference that guided this work are detailed below.

1.3.1 Major Projects

Taking a practical approach to the quad-rotor, or multi-rotor aerial vehicle as they call it, the team of R. Lozano and P Castillo have shed significant light on the real-world challenges of this platform. Understanding the difficulties of controlling a six DoF, under actuated system, Lozano and Castillo refined a stable proportional derivative (PD) controlled platform [23] [22]. This type of control showed good responsiveness in hover and tracking application when tested in an experimental, indoor environment. However, the real-world presents a number of additional challenges not found in the lab, including wind. To compensate, Lozano implemented a hybrid sliding mode and adaptive control which exhibited promising results in winds up to 40% of vehicle speed [35].

Another layer was added to the PD scheme by Lozano and a team from the University of Technology of Compiegne, France with the introduction of a gain switching scheme [21]. This control allowed the implementation of an experimental line following application for a quad-rotor which was robust to single perturbations. However, hints to PD limitations are found in the project conclusion where the desire for switching not just between gains by different control schemes is mentioned.

Additionally, Lozano contributed to work with teams from CINESTAV in Mexico to develop more advanced control for quad-rotor motors themselves [110] [41]. By implementing reflective sensors and advanced current monitoring, these teams were able to greatly improve the performance of a simple PD control. This work shows that with careful attention to detail, PD control can offer quad-rotor stabilization solutions. However, the effectiveness of this approach in more advanced applications, such as trajectory tracking, remains to be seen. This experimental work underscores the challenges of flying a quad-rotor in the real-world and the need for adaptive control.

Further work by Lozano with Alfredo Guerrero provides a definitive mathematical measure of robustness for a simple PD control. They found that stability of a simple PD control

could only be guaranteed if the actual system parameters differed by no more than 5% from the modelled values, and defined a maximum system delay of only 23ms [68]. Fortunately, an impressive amount of computational power and a specialized environment allowed Lozano and Guerrero to demonstrate the theoretical capabilities of the quad-rotor in experimental multi-vehicle formation tracking applications [45]. However, the constraints of a purely linear control system cannot be overlooked.

Also working in an experimental environment, the team of Samir Bouabdallah and Roland Siegwart at the Autonomous Systems Lab in Zurich, Switzerland found control of a quad-rotor difficult without adaptation. Using PID, linear-quadratic (LQ) [12], and even back stepping[14], Bouabdallah and Siegwart found the success of a non-adaptive control scheme depends heavily on the accuracy of the measured control parameters. Understanding the challenges of quad-rotor control, this team worked to develop minimally invasive safety measures for their vehicle [13]. In his thesis, Bouabdallah investigated a number of different control techniques in simulation before settling on a robust method combining the integral action of a traditional PID scheme and a backstepping method. Using the safety measures he developed with Siegwart, Bouabdallah was able to achieve stable hovering within approximately 5 degrees and position tracking accurate to 20 centimeters [10]. While these measures provide important design guidance, they also highlight the dependency of a non-adaptive system on a specific model.

Taking a page from Bouabdallahs thesis, a team from the University of Rio Grand do Norte in Brazil built a quad-rotor from scratch and tested a simple PID control scheme [46]. The findings of this project were presented at the Brazilian Robotics Symposium in 2012, but the linear control showed overshoot to step impulses and disturbances. Unfortunately, no experimental performance data is provided for this impressive piece of hardware, perhaps due to the limitations of the linear software control employed.

Taking the quad-rotor to a "meso-scopic" scale, and perhaps even out of our own sub-

orbital space, the Mesicopter team at Stanford University, led by Ilan Kroo, has uncovered inconsistencies in a number of accepted aerodynamic modelling principles [65]. At smaller scales, were quad-rotors shine, classical airfoil and finite wing analysis was difficult for the team, which found that the propellers they produced were only able to output 80% of the thrust mathematically predicted. Propeller deformation, due to stress from repetitive flights and payload changes, resulted in non-linear dynamics that challenged control system design. In their Phase II final report of the Mesicopter project, Kroo and his team overcame a number of practical design challenges to build an optimized 40 gram, four rotor, four blade, battery powered rotor-craft capable of stabilizing itself in two degrees of freedom. However, the control design is limited by the sensing capability of the platform, and perhaps the use of a linear feedback control in the presence of the practical non-linear thrust dynamics noted in the report [66].

So too, the STARMAC team, also at Stanford University in California, found a large discrepancy between simulated and experimental results [54]. Investigation into quad-rotor dynamics by the researchers uncovered system non-linearities present due to blade flap, aerodynamic drag, and rotor vortices, especially during speed changes. As a result, the desire for an adaptive integral term within the basic PID control employed was explicitly expressed [51]. However, the team has developed a mathematically derived control strategy using known stable sets with some practical success [39]. By transitioning a quad-rotor between areas of known stability, the team was able to implement an experimental, repeatable backflip algorithm. This result is impressive indeed, and highlights the agility of the quad-rotor platform. Nonetheless, additional work is required to ensure vehicle stability in a wider range of scenarios and different quad-rotor configurations.

In 2012, Robert Mahony and Peter Corke allied with Vijay Kumar to provide a state of quad-rotor technology article to IEEE [84]. Kumar went on to work with another team on the application of an adaptive linear control to achieve practical aggressive manoeuvring

[92]. Switching between five different linear controls, the team was able to iteratively tune simple gains to manoeuvre through and onto test spaces. This method relies upon off-line learning and external observation systems, but nonetheless shows the capability of the quad-rotor under a sufficiently tuned control. Kumar has also used a pathing scheme reliant upon frames or windows rather than specific way points to extend the operational envelope of linear techniques [91]. Finally, to overcome the classic lifting limitations of the quad-rotor vehicle, Kumar worked with Daniel Mellinger and a team at the GRASP Laboratory within the University of Pennsylvania to develop a cooperative grasping control to harness the power of multiple vehicles. Using four quad-rotors with special impaling grabbers and an extensive Vicon motion capture system operating at 100 Hz, the team was able to successfully demonstrate a PD control to coordinate individual PID controlled vehicles in the lifting and transporting of large objects in various configurations. Through this work, the practicality of the quad-rotor platform was expanded once more [93]. Mahony and Corke, known for their work with the robust X-4 flyer in Canberra, Australia, have done extensive work to model and predict a disturbance phenomenon known as blade-flapping [106]. Mahony also worked with Paul Pounds to investigate quad-rotor micro air vehicles at larger scales for industrial applications. Here, the team focused on mechanical stability to aid their simple linear controller and demonstrated a need for a controlled test approach over a less scientific try and see method common in applications of smaller vehicles [107]. This approach allowed Pounds to further test the mechanical stability of the quad-rotor platform by successfully designing a less conventional, but more efficient, Y configuration [31]. And again, both X-4 and Y-4 projects presented significant theoretical knowledge of the operation of a quad-rotor vehicle in a practical environment, but were limited by linear control techniques.

1.3.2 Vehicle Stability

The interest of the military in the quad-rotor vehicle and the necessity for stable flight is concisely outlined by Anthony Calise, who brought his Raytheon defence contracting experience

to a paper co-authored by stability and control specialist Rolf Rysdyk [19]. Here the need for a stable platform to operate safely under both human and autonomous control in varying conditions, including failure modes, has led to a program investigating robust adaptive neural control. An impressive list of applicable platforms has been provided, including the X-33 suborbital space plane, the X-36 tail-less ground control jet, the F-18 Hornet fighter jet, and the C-17 Globemaster heavy transport, as well as undisclosed UAV platforms. Their work shows the effectiveness of robust, adaptive neural network control in real world conditions to compensate for disturbances such as turbulence and unmodeled dynamics.

Attempting to replace the need for non-linear control with a purely mechanical stabilization system has resulted in two novel quad-rotor designs from a team at the University of Tehran [108]. Through the use of active pendulum and dynamic cross stabilization design methods, the team was able to improve a simple PID control in simulation.

The mechanical stability of the quad-rotor platform, coupled with a sufficiently advanced control, has allowed a team at the Seoul National University, South Korea, to extend vehicle functionality by adding a 2 DoF manipulator [117]. In an important step towards quad-rotor autonomy, Kim, Choi, and Kim used a sliding mode control to not only successfully manipulate the vehicle itself, but also allow it to manipulate other objects in the real world. At Utah State University, another team has equipped their quad-rotor with a more economical custom gripping mechanism with the same goal of environmental interaction [38]. This project used an advanced Vicon motion capture system with a simple PID control to orientate a quad-rotor vehicle and guide it to objects of interest. The control was able to recover from the effects of singular model dynamic changes due to environmental interaction, but its robustness to persistent, especially oscillating disturbances, such as wind, has not been shown. These projects highlight a new paradigm of the quad-rotor as a versatile tool that can be applied to human tasks in ways never thought possible.

1.3.3 Sensory Input

Using an array of sensors and pathing algorithms normally reserved for ground vehicle navigation, a team at the Autonomous Systems Lab within the University of Freiburg, Germany, demonstrated a practical PID control for a quad-rotor [44]. By augmenting this simple control scheme with increased data from a laser scanner, decent autonomous results were achieved to yaw disturbances. While position tracking and persistent disturbance problems remain to be tackled, this project further demonstrates the versatility of the quad-rotor platform, especially when outfitted with state of the art sensors.

Taking the opposite approach, a team from Ecole Polytechnique Federale de Lausanne, in Switzerland, used minimal on-board sensors and PID control to implement a stabilization and obstacle avoidance system on a custom built quad-rotor [109]. Despite the challenges of sensor drift and limitation, the team was able to demonstrate that minimal investment is required to get a quad-rotor project off the ground. This type of conventional PID control scheme is often used as a baseline to measure control performance of novel systems. At the Jamia Millia Islamia University in New Dehli, a Linear Quadratic Regulator (LQR) scheme was found to be faster to respond and easier to tune than a PID, which are two common weaknesses of this simple linear control [60]. While PID control is more stable on paper, when poles and zeros are considered, in practice, speed and robustness are often more important. This design consideration demonstrates the challenge of moving from a simulated environment to practical implementation.

1.3.4 Payload Support

Introducing a different wrinkle to the quad-rotor stabilization problem, work on payload suspension has been undertaken in a number of different studies. Potential solutions to this type of slung load issue have a number of practical, especially industrial, applications and the team at the MARHES lab in Albuquerque, New Mexico has demonstrated some success

using a dynamic programming approach [104]. This solution has yet to be proven outside the lab, but it shows exciting possibilities for the quad-rotor platform in real-world applications.

1.3.5 Quaternion Notation

At Lakehead University in Thunder Bay, Canada, the duo of Abdelhamid Tayebi and Stephen McGilvray has offered an improvement upon the rotation matrix traditionally used to relate movements from the quad-rotor vehicle to an Earth, or Inertial, frame of reference. Using Quaternion vector notation, Tayebi and McGilvray initially simulated a simple PD^2 control which could, in theory, provide Lyapunov stable attitude control at much larger angles than Euler notation, which is limited by geometric singularities in its cosine terms [119]. This work was further developed into practice using a Dragonflyer quad-rotor frame, but unfortunately, the simple PD^2 control was not subjected to large angles [118]. Using a more advanced backstepping control, a team from the University of California also used quaternion notation to extend the theoretical quad-rotor angular window of operation [112]. Using quaternion filters the team was able to successfully implement a basic backstepping control in simulation. Unfortunately, no practical trials are noted at this time, bu as the quad-rotor is placed in more demanding applications this research will likely only increase in value.

1.3.6 Non-Linear Control

At the University of Moratowa, in Sri Lanka, a team made significant progress in understanding the way in which a quad-rotor is controlled by a human operator [2]. After interviewing pilots, the researchers re-prioritized control and used a fuzzy approach to aid human intuition in control. As a result, they were able to show significant benefits of non-linear control for human control of a quad-rotor UAV. Dr. Holger Voos, working at the University of applied sciences in Weingarten Germany, successfully incorporated a non-linear adaptive scheme into his Riccati control system as he began his investigation into applied non-linear techniques

[122]. By linearizing vehicle dynamics at specific points through adaptive gains, position stabilization was achieved in simulation. Voos later went on to explore additional non-linear quad-rotor control applications in both neural network and feedback linearization systems [124] [123].

Also employing a non-linear control scheme, a team at the Toyohashi University of Technology in Japan demonstrated an improved sliding mode control in a practical application [115]. By creating non-linear sliding surfaces, an exponentially stable control was created and then applied using a stand that allowed application of a 5 to 12 km/hr wind disturbance. This type of robust non-linear control showed good performance, even with the disturbance of a small breeze, but additional work will be required to take the vehicle off of the stand and prove real-world system stability.

In France, at the Laboratoire de Robotique in Versailles, Abdelaziz Benallegue began his work in quad-rotor control in 2001 by defining a basic linearization control for a simulated vehicle model [95] . This initial work demonstrated a need for a more advanced Non-Linear Observer to measure the feedback of all system states and preserve useful system non-linearities. In 2004, he worked with Abdellah Mokhtari on a simple, linear feedback control [96]. Benellegue quickly found that "feedback linearization control is not robust enough towards wind disturbances" and endeavoured to investigate other methods. He then partnered with Tarek Madani to apply a back-stepping control to a quad-rotor, first in simulation then in experimentation [82] [83]. By using an advanced system model and breaking control into three subsystems, the pair was able to extensively prove the stability of their system in theory and practice [81]. However, without the presence of disturbances, such as wind, and relatively simple trials, the control system was not challenged. Both Benallegue and Madani have since moved on to investigate non-linear, neural network control by way of a multilayer preceptron (MLP) scheme for robotic control [28].

While more advanced controls, like neural networks, have tremendous computational

efficiency, the team of Dunfield, Tarbouchi, and Labonte, at the Royal Military College of Canada, found that a non-adaptive system has limits. Using a Dragonflyer platform with MATLAB software, they were able to meticulously teach a neural network to successfully control a quad-rotor within a specific window of operation [32]. Unfortunately, operation outside the training points they collected was unstable. This work shows the need not only for non-linear control, but also the limitations of non-adaptive schemes.

A number of projects, especially [65] and [51], have documented that it is extremely difficult to predict and model all unknown dynamics of a quad-rotor vehicle. As a result, many successful quad-rotor control research projects have embraced an adaptive approach. Work in the field of non-linear control has shown that back stepping can be slow, fuzzy regulators comparatively complex, and CMACs prone to weight drift. All previously proposed control systems show room for improvement, and significant work has been done to optimize the weight update scheme used for the CMAC. At the University of Calgary, Dr. Chris Macnab has supervised a significant body of work that harnesses the power of the CMAC for control, and noted promising theoretical results for quad-rotor control.

1.3.7 Cerebellar Model Articulation Controller

The CMAC was originally proposed by James Albus in 1970 as an analogue to the simple motor control functions performed by the human brain [4]. Albus expanded his theories in subsequent publications to provide a refined and thoroughly investigated control system [3] [5]. This control design has proven itself over the years and has returned to favour as electronic memory has grown more compact, and more capable of performing the theoretical function of Albus proposed memory cells.

The CMAC has enjoyed periods of popularity in academia at times where computations advancements have outpaced control requirements. In 1990, L. Gordon Kraft and David Compagna compared a CMAC to two adaptive techniques of the time: a self-tuning regulator (STR) and a model reference adaptive control (MRAC) [63]. Kraft and Compagna concluded

that each control had strengths and weaknesses, but that the future lay in modifying current adaptive techniques to supplement CMAC systems. A year later Kraft, with Glanz and Miller, outlined three simple applications to test CMAC control: pattern recognition, robot control, and signal processing. In this work, it was noted that even in simple tasks, the CMAC was a viable alternative to a backpropagated multilayer preceptron for learning, especially in real time situations. [40]

Frank Lewis began work on more advanced CMAC system control in 1995, by analyzing structure, stability, and passivity in non-linear systems lacking *a priori* knowledge with S. Cummuri [25]. The success of this work led Lewis to propose a feedback linearization scheme using two CMACs with individual weight updates with Sarangapani Jagannatan [56]. The resulting linearization was mathematically proven to be uniformly ultimately bounded (UUB) and led to further work with Young Kim to compare the CMAC to traditional control methods using a more advanced, two-link robot manipulators in a trajectory tracking exercise [61]. While they showed a baseline optimized computed-torque program (OCTP) control method to be globally asymptotically stable (GAS), it is noted that this conclusion requires knowledge of all non-linearities of the model. Greater attention is paid to their UUB proof of the CMAC system as it does not require this *a priori* knowledge and carries more weight in a practical application. Jagannathan also continued neural network research, applying it specifically to a quad-rotor vehicle [30]. His work proposed an advanced control using not just one, but three neural networks, to compliment a backstepping scheme with a system observer. Extensive proofs are provided to show theoretical system stability, but unfortunately this system was not put into experimental application. In 2011, Lewis worked with Emanuel Stingu on a quad-rotor application of a Radial Basis Function Network (RBFN) NN [113]. While some adaptive techniques, specifically the monitoring and pruning of network nodes based on value metrics, prove similar in their success to work by Dr. Chris Macnab at the University of Calgary, the neural implementation chosen has limitations for

quad-rotor application. Lewis and Stingu are aware of this and specifically cite the "curse of dimensionality" that limits the practical implementation of an RBFN for a system with many inputs, such as a quad-rotor, in their work [113].

At the University of Calgary, Dr. Chris Macnab has guided an adaptive approach to control, in both neural and fuzzy disciplines, with a focus on stability. Dr. Macnab has done significant work to investigate the limitations of adaptive control, identifying bursting as an area requiring improvement. When introduced to sinusoidal disturbances, present as wind in normal quad-rotor operation, an adaptive control may suffer weight drift, leading to values that either cannot be stored within physical memory or quickly approach system bounds. Ultimately, this weight growth results in a burst phenomenon within the system [26]. The early work of Dr. Macnab focused on getting weights to behave themselves, in underdamped systems with persistent oscillation [77]. This effort has resulted in a computationally efficient, robust update scheme that is not prone to bursting in the presence of persistently oscillating disturbances, without sacrificing performance, applicable to both fuzzy and neural applications. Naturally, this work lends itself to quad-rotor control. In the resulting years, Dr. Macnab has assembled a body of work using a system of alternate weights for CMAC updates in simulated and experimental quad-rotor control [80] [101]. Additionally, exploratory work has also been done to theoretically investigate a system of weight voting, or introspective weight updates [87] [88] [27]. This approach has been tested in simulation using a quad-rotor model and in experiments using a two-link flexible-joint robotic arm [86]. In simulation, this method provided performance similar to that of an e-modification weight update scheme, while halting weight drift that has been shown to lead to bursting. However, experimental data for a quad-rotor vehicle has not been obtained, to date.

The body of CMAC research to which Dr. Macnab has lent not only his name but extensive expertise shows theoretical promise and some experimental results for stable control of a quad-rotor vehicle. The work accomplished by Dr. Macnab and the entire University

of Calgary team has focused on the most transferable aspects of quad-rotor control, namely self-contained, robust platforms designed for real-world tasks. This paradigm is shared by a number of the most successful quad-rotor research projects found in the literature. Clearly there is an immediate and urgent need for the quad-rotor in a multitude of varied applications, so if stable control of a self-contained quad-rotor can be achieved in real-world scenarios, the technology will have far reaching applications. Certainly an investigation into the theory of quad-rotor dynamics and stable, robust control is well warranted.

1.4 Scientific Intention

Throughout the literature, there exists an underlying desire for a quad-rotor platform which is stable in flight, robust to disturbances, and adaptive to variable payloads [7] [114]. Less predominant, but no less important, is the avoidance of an adaptive weight update phenomenon known as bursting that can lead to unpredictable system behaviour [26] [77]. Finally, the focus of current research points towards a desire for greater quad-rotor vehicle autonomy in the future [120] [125].

Within the literature, linear PID style controls are often used by a human operator to analyze the performance of hardware or provide a basic control for advanced hierarchical systems [75] [55]. More complex non-linear controls are often explored only within a simulation environment, and even when applied to a physical vehicle, are often only viable within the lab [53] [98] [73].

This work aims to apply a proven, adaptive CMAC not only to a new simulation environment, but also to experimental trials with a commercially available quad-rotor vehicle within an accessible environment. In this way, the stability, robustness, and adaptability of a CMAC with an introspective voting weight update scheme can be proven using practical hardware in experimental applications designed to showcase potential in this control to improve quality of life in a variety of diverse fields by facilitating more advanced autonomy.

Chapter 2

Theory

2.1 Flight

The theory of any flying vehicle begins by defining the force by which gravity is overcome. For a heavier than air vehicle, such as a quad-rotor, this requires a definition for the lift or thrust generated by a wing. The dynamics of this quantity were first explored in a practical setting within a wind tunnel used by the Wright Brothers prior to their first manned flight. The Brothers' findings corroborate the combined work of Newton in the field of rigid body motion and Bernoulli's principles of fluid motion.

While the kinematic effect of a wing is well known, the root of its dynamics has been less clearly defined over the years. Theories speculating differing airflow speeds over the top and bottom edges of a wing are prevalent but easily refuted in practice, as shown in Figure 2.1 [9]. Essentially, the basis of winged flight lies in forcing a fluid stream, consisting of an innumerable number of infinitesimal particles, to travel in an arc. Just as a sailing yacht can generate a side force by tacking into a head wind by simply curving the incoming particles around a sail, a wing can produce lift by vertically curving an onrushing stream of air.

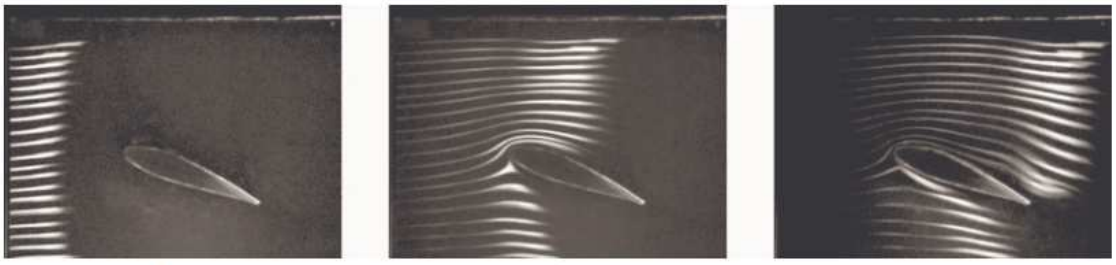


Figure 2.1: Aerofoil Fluid Stream Demonstrating Lift
[9]

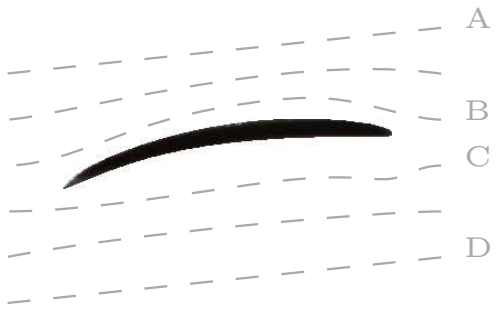


Figure 2.2: Fluid Stream Around a Stationary Wing

Figure 2.3: Rotary Wing Lift Dynamics

Newtonian mechanics postulate that when the path of an inert moving particle is forced to follow a curved trajectory it is under the influence of an external force, in this case centripetal force. As the wing exerts a force on the inert stream of air particles that pass over and under it, so to the particles exert an equal and opposite force. For a fluid stream, typically this force is deemed to be the result of pressure, as surface friction is small due to the limited contact between the stream and wing surface area. Bernoulli's principles then indicate that a pressure difference between the face of the particle closest to the wing and that farthest is created as a result of this centripetal force.

$$F_T = C_T \rho \pi r^2 \Omega^2 = b\Omega^2 \tag{2.1}$$

The stream of fluid particles over the outer edge of the wing curve create an area of low pressure, the particles over the inner edge create high pressure, as shown in Figure 2.2. In short, applying a centripetal force to a fluid stream by curving the path of the inert particles within the stream generates a resultant, and useful force, which for aeronautical purposes is called thrust.

Where a fixed wing vehicle generates thrust by propelling a stationary wing through the

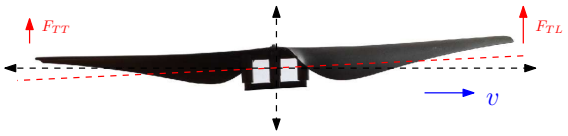


Figure 2.4: Blade Flapping Disturbance During Flight

air, a rotary wing vehicle, such as a quad-rotor, uses a rotational motion, spinning its wings through the air. Nonetheless, the basic principles of thrust remain unchanged. For a rotor craft, the size of the area of curvature creating thrust, or wing, is estimated as the entire surface area it carves through the air, πr^2, as shown in Figure 2.3. The larger the wing, the greater the area available for a thrust generating pressure differential. The linear velocity of the wing is calculated using the angular velocity, Ω, at the very outer edge, or tip, of the wing multiplied by the distance from the centre of the wing to the tip, r. Thrust increases proportionally to the square of the speed of the wing through the air. The number and size of particles in the fluid through which the wing is travelling also have a proportional role to play. Collectively the quantity and size of air particles is termed density and given the symbol ρ. Finally, a constant parameter, termed the thrust coefficient, C_T, is added to take into account the aerodynamic properties of the wing such as curvature and surface friction. Constant, or slowly varying, terms are simplified by defining a thrust factor, b. Combined, these terms produce an accurate estimate of the thrust created by a single rotating wing, or rotor, defined by Equation 2.1 [90].

For most systems, this estimate is often more than sufficient to provide stable control within a large operational window. However, it is important to mention the non-linear and unmodeled dynamics at the edges of quad-rotor operational capability to understand how non-linear and adaptive controls can push the practical envelope of the vehicle.

$$F_{T_L} = b(\Omega^2 + 2\Omega v + v^2) \tag{2.2}$$

$$F_{T_T} = b(\Omega^2 - 2\Omega v + v^2)$$

$$\Delta F_T = 4\Omega v$$

First, the propeller blades used by many small rotor-craft, including the quad-rotor used in this work, are often flexible. This results in shape changes that can cause non-linear variations in the lift coefficient at different rotor speeds. Additionally, flexibility contributes to a phenomenon known as blade-flapping, whereby the horizontal translation of the quad-rotor induces a faster airspeed on the leading blade of a multi-bladed propeller than on those trailing, as defined by Equation 2.1 [90]. This causes a vertical deflection in the blades as they transition from a faster airspeed region where they do more work to a slower region where they do less, and back, as shown in Figure 2.4. The flapping motion generated by these deflections results in slight variations in the thrust coefficient at different air or translational speeds and is difficult to model.

Second, thrust is proportional to air density which varies with elevation, temperature, humidity, and barometric pressure. As a result, very different thrust coefficients can be expected in Calgary on a cold, dry winter day than in Vancouver on a temperate, humid spring day. Temperature and air density have a significant and documented effect on VTOL aircraft, as shown by the data compiled in Figure 2.5. Again, these changes are difficult to model for all locations and conditions.

Finally, the dimensionality of the rotor blades themselves will change over time as repeated stress causes warping and occasional crashes reshape the topography. All of these factors lead to non-linearities and unmodeled dynamics that although small, do affect the control and overall stability of a quad-rotor.

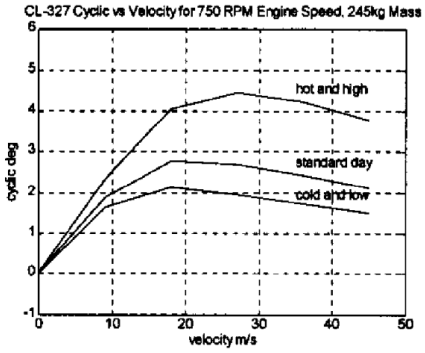


Figure 5-4: CL-327 Cyclic vs Velocity for 750 RPM Conditions (Theoretical)

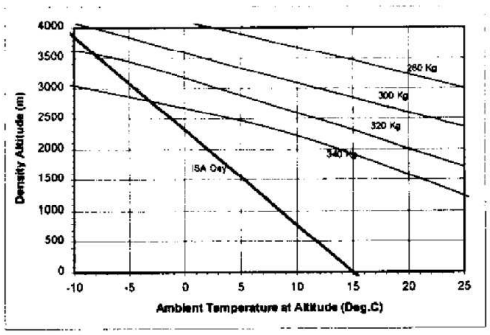


Figure 6-1: CL-327 Take-off and Landing Envelope (Preliminary)

Figure 2.5: CL-327 Pitch vs Velocity and Payload Capability Changes in Variable Environmental Conditions

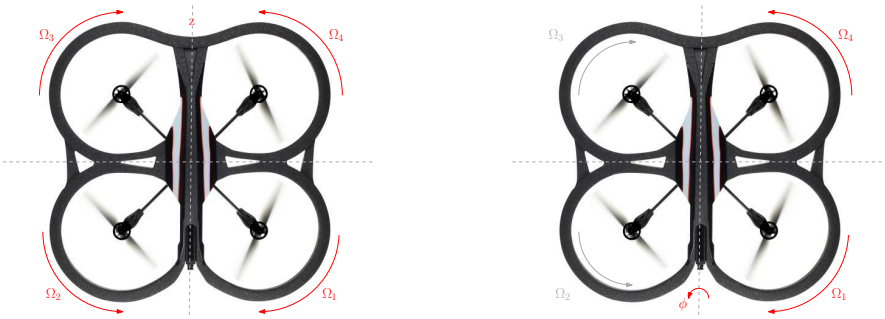
[105]

2.2 Quad-rotor Basics

A quad-rotor incorporates four rotors laid out in a box formation. Any two blades on opposing corners of the formation will spin in the same direction. The other two blades will also spin in the same direction, but opposite that of the first pair. In this way, the reactionary torque created by the drag of the blades is balanced as their movement is resisted by the air through which they turn. This makes the quad-rotor more efficient than a traditional, single rotor helicopter which requires a power sapping stabilization tail rotor [90]. As a result, a balanced quad-rotor will hover in place with no change in rotational or translational position. Furthermore, by simply unbalancing the speeds of these four rotors, control within six DoF can be obtained.

As demonstrated in the previous section, a rotor produces a force equal to its speed, Ω, squared multiplied by a thrust coefficient, b. For the purposes of mathematically modelling the quad-rotor used for this work, each of the four rotor speeds is given a unique Ω variable. Rotor speeds Ω_1 and Ω_3, are positive in a clockwise direction, while Ω_2 and Ω_4 are positive in an anti-clockwise direction, as shown in Figure 2.6. When mounted to a rigid frame, other variables such as the rotor placement from the body center of mass, l, vehicle aerodynamic

Figure 2.6: Quad-rotor Position and Attitude

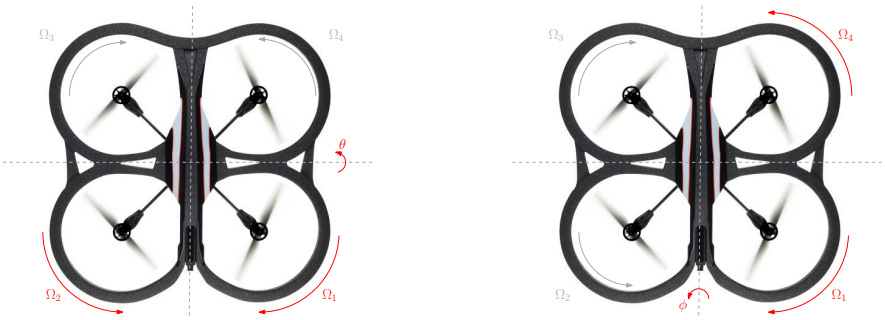


(a) Z Translation

(b) Roll (ϕ) Rotation

drag, d, and current vehicle movement also affect thrust. These parameters will be explored later, but for now a different variable, C, will hold their place in each of the four control equations.

The height of the vehicle in three dimensional space is described by the variable z, where a positive direction is upwards. Its position in the remaining two dimensions of space is described by the variables x and y, where positive directions are right and backward from the perspective of the vehicle. The yaw, or rotational position of the quad-rotor in the z plane is defined by the variable ψ. This value is positive in a clockwise direction, from the perspective of the vehicle. The tilt of the quad-rotor along the x axis, or roll, is described by ψ and the along the y axis it is given the variable θ with designates pitch. These quantities are linked to the x and y positions described earlier, and are positive when the vehicle tilts toward its right and backwards respectively.



(c) Pitch (θ) Rotation (d) Yaw (ψ) Rotation

Figure 2.7: Quad-rotor Motivating Dynamics

$$\ddot{z} = C_1 U_1 = C_z(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \tag{2.3}$$

$$\ddot{\phi} = C_2 U_2 = C_2(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \tag{2.4}$$

$$\ddot{\theta} = C_3 U_3 = C_3(\Omega_1^2 + \Omega_2^2 - \Omega_3^2 - \Omega_4^2) \tag{2.5}$$

$$\ddot{\psi} = C_4 U_4 = C_4(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \tag{2.6}$$

The first, and most basic, movement is simply a change in height, or z position. This motion is accomplished by varying the speeds of all four rotors in unison, producing a change in z position that is proportional to the change in speed, as shown in Figure 2.7a and defined by Equation 2.3 [20]. Second, increasing the speed of two blades closest to one another, while decreasing that of the pair across the quad-rotor, will produce a change in the roll, ϕ, or pitch, θ, rotation of the vehicle, as shown in Figures 2.7b and 2.7c and defined by Equations 2.4 and 2.5 respectively [20]. As a by-product of this change in angle, a portion of the thrust that was initially directed straight down will be directed along one or both of the horizontal planes of the quad-rotor, thereby affecting its position and velocity. Finally, increasing the speed of the two blades rotating in the same direction, while decreasing the speed of the counter rotating pair, will introduce an unbalance in the net torque of the vehicle, resulting in a change in yaw rotation, ψ, as shown in Figure 2.7d and defined by Equation 2.6. As shown in [20], these differences of torque provide the basis of quad-rotor control.

(a) Roll Frame

(b) Pitch Frame

(c) Yaw Frame

Figure 2.8: Quad-rotor Body to Earth Reference Frame Translations

These four basic control signals, defined as a collective by U, represent the entire gamut of control input available to articulate the quad-rotor through six degrees of output freedom. By relating these controls from the quad-rotor to an Earth from of reference, and expanding applicable dynamics, an accurate kinematic model of the vehicle can be obtained. This accurate model is the first step in the process of designing a successful quad-rotor control system.

2.2.1 Reference Frames

$$R_{\phi,E} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \tag{2.7}$$

$$R_{\theta,E} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.8}$$

$$R_{\psi,E} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

The forces and torques described previously must be translated from the quad-rotor frame

of reference to the view of the controller in order to enact the desired control on the vehicle from the ground. For this purpose, the frame of reference of the quad-rotor is referred to as the Body, and the controller frame as the Earth view. Simply put, the dynamics of the body frame are subjected to three successive rotation matrices, defined by Equations 2.7, 2.8, and 2.9, in the x, y, and z planes, as shown in Figures 2.8a,2.8b, and 2.8c.

$$R_{\psi\theta\phi,E} = R_{\psi,E}R_{\theta,E}R_{\phi,E} = \begin{bmatrix} C\phi C\theta & -S\psi C\phi - C\psi S\theta S\phi & -S\psi S\phi + C\psi S\theta C\phi \\ S\phi C\theta & C\psi C\phi - S\psi S\theta S\phi & C\psi S\phi + S\psi S\theta C\phi \\ -S\theta & -C\theta S\phi & C\theta C\phi \end{bmatrix} \tag{2.10}$$

These matrices align the movement of the quad-rotor to the Earth frame using the combined matrix defined by Equation 2.10. Where C and S represent cos and sin, respectively. Once the movement of the vehicle is aligned to the same frame of reference as the control system, motivating dynamics become clear and appropriate control signals can be defined.

2.2.2 Dynamics

First, the control force of total thrust, U_1, may be applied not only to the z plane of motion, but the x and y as well to translate the quad-rotor through three degrees of freedom. While the total thrust directly affects the z plane alone, a portion of that thrust may be indirectly applied to the x and y planes via roll, pitch, and a lesser extent, yaw rotations.

$$F_c = b \begin{bmatrix} \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\phi\sin\theta\sin\psi + \sin\phi\cos\psi \\ \cos\theta\cos\phi \end{bmatrix} U_1 \tag{2.11}$$

Collectively, the control force applied motivate to the vehicle in a linear fashion is designated F_c, as defined by Equation 2.11. The attitude of the quad-rotor is translated to an Earth frame of reference using portions of the matrix defined in Equation 2.10. In [20] this is described as converting Newton's equations of angular position to Lagrange's equations

of movement. A constant term, b, is a single coefficient that represents all of the thrust constants and slowly varying terms found in the wing thrust equation.

$$\tau_c = \begin{bmatrix} lb \\ lb \\ d \end{bmatrix} \begin{bmatrix} U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{2.12}$$

Second, the remaining three control forces, U_2, U_3, and U_4 are applied to the quad-rotor to generate roll, pitch, and yaw torques, collectively designated τ_c, that rotate the vehicle in three additional degrees of freedom. This general torque relationship is defined by Equation 2.12 [20]. In the roll and pitch planes, the thrust factor, b, again makes an appearance, alongside a new term, l, which represents the distance from the centre of the quad-rotor body to the centre of any given rotor. The vehicle is symmetrical so this distance is the same for all rotors. Finally, in the yaw plane a variable, d, is introduced to represent the aerodynamic drag coefficient of the quad-rotor as it rotates.

$$F_g = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} mg \tag{2.13}$$

Once in motion, a quad-rotor feels the effects of a number of smaller forces and torques other than the powerful thrust generated by its rotors. First, while a quad-rotor is in close proximity to Earth, it will experience a force due to gravity, F_g, along its vertical, z axis, proportional to its mass, m, and our gravitational constant, g, as defined by Equation 2.13 [20].

$$\tau_b = \begin{bmatrix} I_{yy} - I_{xx} \\ I_{zz} - I_{xx} \\ I_{xx} - I_{yy} \end{bmatrix} \begin{bmatrix} \dot{\theta}\dot{\psi} \\ \dot{\phi}\dot{\psi} \\ \dot{\phi}\dot{\theta} \end{bmatrix} \tag{2.14}$$

Next, when rotating a quad-rotor is subject to Newton's first law. Specifically, when a quad-rotor attempts to force a change of one angular quantity and a velocity is present in any other angular direction of motion, the control force will be affected, as defined by Equation 2.14. The torque present in the body of the vehicle, τ_b, is proportional to the difference of the concerned moments of inertia, designated I_{xx}, I_{yy}, and I_{zz} in each plane, and the angular velocities, $\dot{\phi}$, $\dot{\theta}$, and $\dot{\psi}$, in the roll pitch and yaw planes.

$$\Omega_r = (\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \tag{2.15}$$

$$\tau_r = \begin{bmatrix} -I_r \\ I_r \\ 0 \end{bmatrix} \Omega_r \tag{2.16}$$

Finally, the torque generated by the gyroscopic effect of the spinning rotors themselves, τ_r, must be taken into consideration. This dynamic is proportional to the combined moment of inertia of the rotors, I_r, and the combined velocity of the rotors, defined by Equation 2.15 [20]. It affects the roll and pitch planes only as these are the only quad-rotor motions that are perpendicular to the axis of rotor rotation, as defined by Equation 2.16.

$$F = m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = b \begin{bmatrix} \cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi \\ \cos\phi \sin\theta \sin\psi + \sin\phi \cos\psi \\ \cos\theta \cos\phi \end{bmatrix} U_1 + \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} mg \tag{2.17}$$

$$\tau = \begin{bmatrix} I_{xx}\ddot{\phi} \\ I_{yy}\ddot{\theta} \\ I_{zz}\ddot{\psi} \end{bmatrix} = \begin{bmatrix} lb \\ lb \\ d \end{bmatrix} \begin{bmatrix} U_2 \\ U_3 \\ U_4 \end{bmatrix} + \begin{bmatrix} I_{yy} - I_{xx} \\ I_{zz} - I_{xx} \\ I_{xx} - I_{yy} \end{bmatrix} \begin{bmatrix} \dot{\theta}\dot{\psi} \\ \dot{\phi}\dot{\psi} \\ \dot{\phi}\dot{\theta} \end{bmatrix} + \begin{bmatrix} -I_r \\ I_r \\ 0 \end{bmatrix} \Omega_r \tag{2.18}$$

Ignoring any non-linearities present in the thrust coefficient due to environmental considerations, overlooking trivial unmodeled dynamics such as blade-flapping, and neglecting rotor acceleration by slowly varying speeds, these are all of the motivating forces and torques

present in the quad-rotor system. All forces are represented by a single vector, defined by Equation 2.17, and likewise all of the torques are combined into Equation 2.18. It may be noted that the quad-rotor model offers little resistance to dynamics, especially in the x and y translational planes. This means that the system is under damped, therefore, control input will play a strong and lasting role in vehicle kinematics.

2.2.3 Kinematics

The vehicle dynamics, based on Newton's laws, are combined with Euler's laws of rigid body motion to produce a Newton-Euler equation for the kinematics of the quad-rotor system. The force and torque equations previously defined are simply transformed into recognizable kinematic models of the vehicle by dividing the right sides by the quad-rotor mass and moment of inertia respectively.

$$\ddot{x} = U_1 \frac{b}{m}(\sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi)) \tag{2.19}$$

$$\ddot{y} = U_1 \frac{b}{m}(-\sin(\phi)\cos(\psi) + \sin(\phi)\sin(\theta)\sin(\psi)) \tag{2.20}$$

$$\ddot{z} = U_1 \frac{b}{m}(\cos(\phi)\cos(\theta)) - g \tag{2.21}$$

$$\ddot{\phi} = U_2 \frac{lb}{I_{xx}} + \frac{I_{yy} - I_{zz}}{I_{xx}}\dot{\theta}\dot{\psi} - \frac{I_r \Omega_r}{I_{xx}}\dot{\theta} \tag{2.22}$$

$$\ddot{\theta} = U_3 \frac{lb}{I_{yy}} + \frac{I_{zz} - I_{xx}}{I_{yy}}\dot{\phi}\dot{\psi} + \frac{I_r \Omega_r}{I_{yy}}\dot{\phi} \tag{2.23}$$

$$\ddot{\psi} = U_4 \frac{d}{I_{zz}} + \frac{I_{xx} - I_{yy}}{I_{zz}}\dot{\phi}\dot{\theta} \tag{2.24}$$

Together, the translation Equations 2.19, 2.20, and 2.21, as well as rotation equations 2.22, 2.23, and 2.24 form the basis of complete position and attitude control for the quad-rotor vehicle. By monitoring and adjusting vehicle acceleration, desired positions and rotations can be achieved to accurately move the quad-rotor through a variety of different trajectories, given the right control scheme is applied.

2.3 Control

Having defined an accurate mathematical model of the quad-rotor, any one of a multitude of different control schemes may be used to affect desired changes in the attitude and position of the vehicle. By understanding the dynamics of the quad-rotor platform, it is possible to estimate the strengths and weaknesses of different control schemes to select the best potential candidate.

The first, and most important, attribute of the quad-rotor model with respect to control, is the non-linear nature of the system. Given the variable and unpredictable nature of the thrust response due to blade and air dynamics described in the flight section, the quad-rotor is extremely difficult to accurately model and control in a purely linear fashion, especially in the presence of disturbances. A proportional, integral, derivative (PID) control applies three constant gains to terms representing the current error, predicted future error, and past error signals.

$$f(x) = k_p\zeta + k_i \int \zeta(t)dt + k_d \frac{d\zeta(t)}{dt} \tag{2.25}$$

The simple PID system defined by Equation 2.25 can produce reasonable results when time is taken to carefully tune each gain value, especially within simplistically modelled simulation environments. While various combinations of proportional, integral, and derivative feedback controls have been proposed in the current literature, they are often successful within only a small window of operation and rely upon a human operator for an adaptive control component. This reliance on human input is not ideal for a fully autonomous system. It is more desirable for a control system to not only stabilize the quad-rotor, but move it through desired trajectories as well, freeing human operators to manage higher level functions. For this reason, a non-linear control has been chosen for this project.

Within the world of non-linear control systems, there are still many potential candidates from which to choose a type of function to act as a non-linear approximator. Where linear

control may not provide accurate results in non-linear portions of the operational control window, an approximator function adapts weights associated with different output vectors to learn the true nature of the system over time. A non-linear approximator can be constructed in a number of different ways.

First, a Radial Basis Function Network (RBFN) can be used to approximate a function by summing a series of simple functions to build a more complex, non-linear function. Known, simple functions such as triangles, Gaussian curves, or more commonly, splines are summed in a weighted manner to quickly approximate more complex or unknown functions.

$$f(x) = \sum w_i \Gamma_i(x) \tag{2.26}$$

While RBFNs, like the one described in Equation 2.26, work very well for a small number of inputs, they suffer from what is called the curse of dimensionality, whereby the number of networks required to estimate a function grows exponentially with the number of inputs desired. With approximately a dozen inputs, an RBFN based system is not ideal for quadrotor control.

$$f(x) = \sum A_i(x) c_i \tag{2.27}$$

Alternatively, an approximator can be built using a series of fuzzy logic membership functions, as defined in Equation 2.27. In this method, input values are assigned a truth value rather than a simple Boolean designation. The fuzzy rules governing the input truth values are chosen based on some knowledge or statistical data of the system.

Given the variety of inputs to the system, not only from the vehicle itself but external disturbances, fuzzy descriptions for all of the differing environmental models and vehicular configurations required for this work are not easy to choose. Therefore, a fuzzy logic approximation system is less than ideal for control.

A different approach to the non-linear approximator can be found in using a series of

simple nodal activation functions to model a more complex one. In one method, these activation nodes are termed perceptrons and when many are used together, they form a Multi Layer Perceptron (MLP) neural network. An MLP uses a series of visible as well as hidden weights in conjunction with layers of perceptron cells to approximate a function.

$$f(x) = \sum w_i \Gamma_i(x)(q_i H_i(x)) \tag{2.28}$$

An MLP, like the one defined in Equation 2.28 can associate a large number of inputs to activation functions and ultimately generate a multitude of unique, non-linear outputs. However, MLP networks require significant training to effectively tune all weights and estimate a function. Given the under damped, under actuated nature of the quad-rotor vehicle, this method could result in initial instability detrimental to the integrity of the vehicle and future control development.

$$f(x) = \sum w_i \Gamma_i(x) \tag{2.29}$$

Sharing the same design philosophy as the MLP network, a Cerebellar Model Articulation Controller neural network uses a few hypercube activation cells rather than many simple perceptrons. Inputs are applied to multiple layers of the CMAC, where on each they activate a single, unique cell. The value of the activated cell in each memory layer of the CMAC is then combined to produce an approximation output, as defined in Equation 2.29. A CMAC neural network learns faster than an MLP structure, and is able to effectively compute more inputs than an RBFN. For these reasons, the CMAC has been selected as the non-linear approximator for this project.

2.3.1 CMAC

The CMAC is an associative memory device often used as an adaptive approximator. It receives an input vector, x, that is applied to a series of, m, offset memory layers, each divided

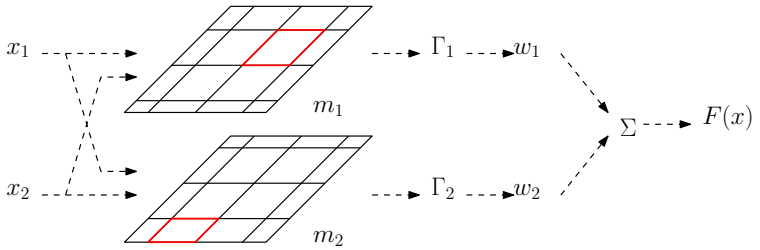


Figure 2.9: CMAC Cell Matrix Structure

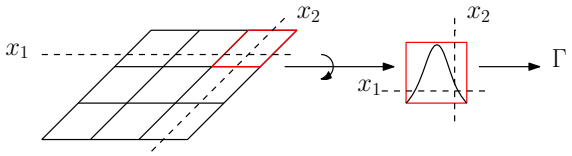


Figure 2.10: Spline Basis Function within a Cell

into Q quantizations. The layer offsets decrease the likelihood of the same cell location being activated on more than one layer. On each layer the inputs activate a single N dimensional hypercube cell and within each cell an activation function, Γ, produces a single value, as shown in Figure 2.9. Common activation functions include binary, triangular, Gaussian, and spline distributions. The more complex distributions typically have greater accuracy when estimating a function, but, computational complexity must be balanced with accuracy and additional layers can often increase the later with minimal effect on the former. For this reason, a less mathematically complex Gaussian-esque function called a spline is often used, as shown in Figure 2.10.

On each layer, the unique value produced by the activation function is multiplied by an

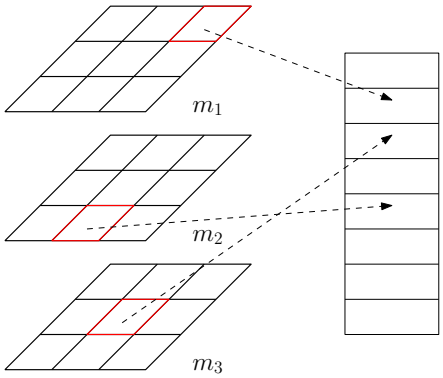


Figure 2.11: CMAC Hash Coding Scheme

adaptive weight estimate value, $\hat{w}$. The products produced on each individual layer are then summed and normalized to create the CMAC output. Weights begin with an initial value of zero and are updated with a robust scheme. When a weight is updated, its value is updated in physical memory and it is accessed by address using a random hash coding scheme, shown in Figure 2.11. In this way, only activated weight values are saved, dramatically reducing memory requirements. As it is possible that only a small region of all available cells may be active for a given operational window, this method removes the need to store numerous extraneous zeros in memory. Memory is arranged such that the same cell will always activate the same weight, producing a similar output vector, which provides a stable output for an unlearned input, if it is near a previously activated cell. The weights can be trained to adjust the final CMAC output function and reduce estimation error. By slowly expanding the input signal envelope to a CMAC, every desired output response can eventually be learned.

A CMAC can be trained off-line, then the resulting memory structure inserted into a system, or it can learn in real-time and grow as an adaptive control. In either case, successful learning depends on the way in which updates to memory location weights are executed. This determines not only the speed with which the CMAC learns a system, but also the accuracy of its approximation, as well as its margin of robustness to disturbances. When using an adaptive scheme such as a CMAC, it is important to ensure the weight updates do not affect total weight values in such a way that the system becomes unstable. For this purpose, a number of robust weight updated schemes have been developed and presented in the literature, each with their own strengths and weaknesses. Traditional weight update techniques include dead-zone, leakage, e-modification (e-mod), and projection.

$$\dot{\hat{w}} = \begin{cases} \beta \Gamma^T(x) x & \text{if } \|x\| > \frac{D_{max}}{K_{min}} \\ 0 & \text{Otherwise} \end{cases} \tag{2.30}$$

Dead-zone relies upon knowledge of the maximum disturbance found in the input signal, d_{max}, as well as the control gain matrix K, as defined by Equation 2.30. This method halts

the weight update, $\dot{\hat{w}}$, when the estimation error in the signals is no longer greater than the maximum expected disturbance.

$$\dot{\hat{w}} = \begin{cases} 0 & \text{if } \Gamma^T(x)x > 0, \hat{w} \geq w_{max} \\ 0 & \text{if } \Gamma^T(x)x < 0, \hat{w} \leq w_{min} \\ \beta\Gamma^T(x)x & \text{Otherwise} \end{cases} \tag{2.31}$$

Projection requires knowledge of the maximum and minimum weight values for the system, represented by w_{max} and w_{min} respectively. Weight updates are halted when these bounds are reached, as defined by Equation 2.31.

$$\dot{\hat{w}} = \beta(\Gamma^T(x)x - \nu\hat{w}) \tag{2.32}$$

Leakage, defined by Equation 2.32, and its cousin E-Mod, both try to force weight estimates to zero by including them in the weight estimate update calculation.

$$\dot{\hat{w}} = \beta(\Gamma^T(x)x - \nu\|x\|\hat{w}) \tag{2.33}$$

Where leakage introduces an adaptation constant, ν, E-Mod is a little more forceful in this task by incorporating the norm of the input error vector, x, as defined by Equation 2.33. If the weight errors move towards null, the non-linear estimator can be said to be replicating the actual signal exactly.

However, when an otherwise stable adaptive system lacks persistent excitation, or especially when it is subjected to a destructive excitation such as an oscillation, it may exhibit sudden and unexpected periods of dramatic error change. In 1985, Brian Anderson at the Australian National University, Canberra, found that when an adaptive system is stagnant, weights can drift to large magnitudes, especially if the system adapts aggressively [8]. Eventually, the weight values can no longer be represented in physical electronic memory and the system experiences a sudden spike of instability before weights are relearned. In more robust

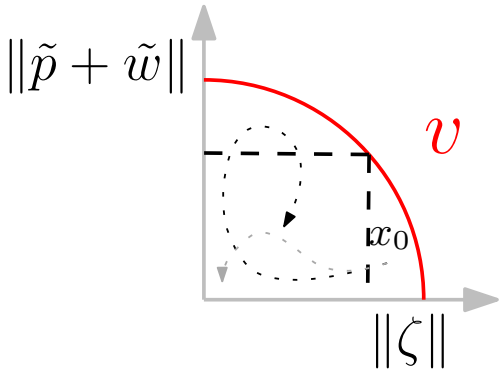


Figure 2.12: Bursting Behaviour within a Stable Bound (Black)

weight update schemes, such as deadzone or leakage, weight updates will be stopped when weight values reach a calculated bound. At this point, the control error may dramatically increase over a short period of time, before weight values move away from the bound and updates begin again.

Figure 2.12 shows the relationship between system error and weight error on the x and y axis respectively, with a stable system bound in red. Starting from an initial point, x_0, the grey path moves in a stable manner to minimize both weight and system errors. However, if oscillations or disturbances cause weights, and thereby their associated errors, to grow, the system will hit the stable bound, weight updates will stop, and the system error will grow unexpectedly. The behaviour is demonstrated by the path outlined in black. Working with an intimate knowledge of this bursting phenomenon and the weight update laws that contribute to it, Dr. Macnab has proposed an introspective weight voting scheme that will be explored in this work [87].

Unlike deadzone schemes, the introspective weight voting system requires no *a priori* knowledge of model parameters such as maximum disturbance, minimum weights, or maximum weights. Furthermore, it analyzes the effect of a weight update before permanently storing it to minimize bursting found in leakage style schemes. In this way, weight drift is managed and weight updates are stopped before values reach calculated bounds or exceed memory capacity, significantly reducing the likelihood of bursting.

The voting algorithm stores the last two weight values for all active cells, as well as the system error. It can determine if weights are having a significant and positive impact on the overall system error, or if they are simply in an escalating squabble over a petty error difference. Using this information, the algorithm obtains a vote from every cell as to whether the current active weight should be stored replace the current permanent weight, or if the permanent weight should simply be moved towards the best value found so far. In this way, the introspective voting weight update scheme seeks to improve upon existing techniques, especially when applied to a system exposed to oscillatory disturbances, such as the effect of wind on a quad-rotor.

2.3.2 Stability

Given an accurate mathematical model of the quad-rotor system, and a potential control candidate, the final step in system design is the development of a stability proof. In this respect, stability is not only the confinement of system trajectories to a desired point or region, but also a measure of the ability of a system to maintain a trajectory in the presence of disturbances. The measure of stability for a control system is in many ways tantamount to its value.

Aleksandr Lyapunov first proposed two theories regarding "The General Problem of Stability of Motion" in 1892, and his work is used to this day to estimate the stability of linear and non-linear systems alike [76]. Lyapunov began with the simple notion that if the energy in a mechanical system is dissipated via heat, sound, or some other exoenergetic process, then that system must settle to a specific point. He applied that principle to a more general class of systems to describe the ways in which they settle, or become stable. Lyapunov was able to see the relativity of stability, and the specific language of his definitions is key to understanding how a system can be expected to behave.

At the center of Lyapunov's work is the concept of an equilibrium point or region. In a linear control system, the point at which the system is in equilibrium, where the value of

the mathematical function describing the system is zero, is defined as the equilibrium point. A non-linear system may have a number of equilibrium points, so the area of interest is expended from a point to a region, or area of stability. The outer limit of this stable region has been termed a bound and the value of a control system is often inversely proportional to the size of its stable bound.

When analyzing a linear system, Lyapunov's linearization method can be used to determine stability. For a non-linear system, such as the adaptive CMAC used in this work, Lyapunov's direct method is used to determine stability. Lyapunov's direct method describes a system in terms of the energy contained therein, denoted v. The rate of change of energy in the system, which for a stable system is dissipating overtime, is the derivative of the energy function, termed $\dot{v}$. Typically, a Lyapunov stability analysis is preformed to determine under which conditions a system is stable, and control parameters are then adjusted accordingly. As a result, the function v can be called a Lyapunov candidate, as Lyapunov stability can only be proven at the end of the analysis.

As defined by Lyapunov, a Stable function is one whose value, $x(t)$ is guaranteed to stay within a bound, B_R, over all time, given that it starts within a region, $B_r(B_R, t_0)$, inside that bound. [116]. To fulfill the most basic stability requirements, a Lyapunov candidate function describing the energy of a system, $v(x)$, must be continuously differentiable and positive definite, its derivative must be at least negative semi-definite . That is to say, the rate of change of energy in the system must be calculable at all points in time, the energy should be positive, and that energy should dissipate over time.

$$\|x(t_0)\| \leq B_r(B_R, t_0) \Rightarrow \|x(t)\| \leq B_R \forall t \geq t_0 \tag{2.34}$$

In mathematical terms, this concept can be written much more concisely using some short hand notations, as defined by Equation 2.34. Here, $\|\|$ represents a magnitude calculation, $\Rightarrow$ describes an implied relationship, and $\forall$ stands in place of the phrase "for all".

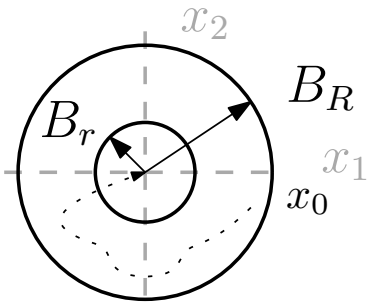


Figure 2.13: Lyapunov Uniformly Asymptotically Stable Bound

To expand on stability using Lyapunov's definitions, a definite function is one whose value is zero when its variables are zero, and must not be equal to zero at any other time. A semi-definite function relaxes the later rule, allowing a value of zero even the variables of the function have a value that is not. A positive function is one whose value is greater than zero at all times, save when all variables are zero. A negative function is just the opposite.

Lyapunov defines a uniformly stable function as one that is not only stable, but decrescent. A decrescent function is one whose value remains within the bounds of a positive definite function. That is to say, a uniformly stable function requires that the inner ball defined by B_r is not a function of initial time, t_0. If the value of a uniformly stable function enters the inner ball defined by B_r at any time, it will remain within B_R for all time.

$$\|x(t_0)\| \leq B_r(t_0) \Rightarrow \lim_{t \to \inf} x(t) = 0 \tag{2.35}$$

Lyapunov also defined theorems for convergence. A convergent function is one whose value trends towards an equilibrium point over time, as defined by Equation 2.35. Mathematically, a convergent functions is one whose limit approaches an equilibrium point as time approaches infinity, given the initial value of the function starts within a ball of radius δ. Expanding convergence from an equilibrium point to a region provides a definition for a Uniformly Bounded (UB) function [59].

$$\|\|x(t_0)\|\| \leq B_r(B_R) \Rightarrow \|\|\|x\|\|\| \in B_R \forall t \geq t_0 + t(B_R, \delta) \tag{2.36}$$

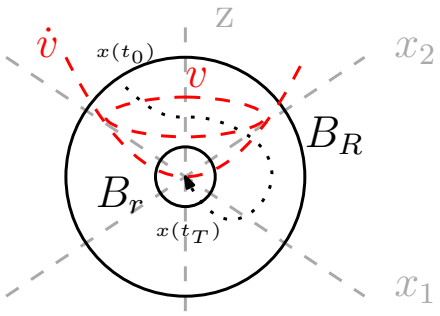


Figure 2.14: Lyapunov Uniformly Ultimately Bound

Combining the definitions of uniformly stable and uniformly convergent functions yields a description of a Uniformly Asymptotically Stable (UAS) function. This most rigid stability definition states that a function will trend toward an equilibrium point as time approaches infinity, once it enters a ball of radius B_r, staying within a ball of radius B_R, as shown in Figure 2.13. The regional analogue for the UAS function is termed a Uniformly Ultimately Bounded (UUB) function. A UUB function is one whose value remains within a bound, B_R, for all time, and given that it enters an inner ball of radius B_r, will re-enter that ball at a specific, or ultimate, time, T. A UUB function is defined by Equation 2.36, and shown in Figure 2.14 [59].

The outer ball of radius B_R is the bound on the Lyapunov function and the inner ball of radius B_r represents an expanded equilibrium point for a bounded system. The proposed control system will be mathematically proven to fit within this ultimate non-linear stability definition. Proving the mathematical stability of a control system using Lyapunov's methods is the first step in implementing it in practice. A stability result will confirm that the control chosen, based on the derived dynamic and kinematic model of the system will be successful, in theory. The methodology of proving this theoretical result in practice will introduce new challenges to the problem. Nonetheless, an understanding of the basic principles at work will greatly improve the chances of successfully implementing the proposed control system in practical applications.

2.4 Simulation & Experimentation

Through this section, the theories that have formed current quad-rotor modelling techniques, control schemes, and stability measures have been explored. Using basic, established rotor-craft flight dynamics and a simple rotation matrix, a kinematic model of the quad-rotor has been presented. To control this vehicle, a number of existing schemes have been explored, specifically the CMAC and its many weight update schemes. Finally, mathematical proofs for control stability were outlined using tried Lyapunov methods. In the next section, novel methods for testing the described model and controls in simulation and experimentation will be detailed. This moment represents the start of an original work which aims to explore concepts of stability, adaptation, robustness, bursting, and ultimately autonomy as relevant to the development of a more viable experimental quad-rotor platform than what is currently found in the literature.

Chapter 3

Methodology

3.1 Overview

In order to test the viability of introspective voting weight updates for CMAC non-linear control using a quad-rotor vehicle, a software environment, control scheme, and test platform are all required. In this endeavour, it was useful to explore the paths already taken to avoid previous pitfalls and navigate towards a successful testing process. Therefore, concepts such as real-time plotting, incremental testing progression, environment safety and controllability as well as hardware confidence were modelled upon previous work exploring an alternate weight voting system. The thoughts shared by Chris Nicol in his thesis and in person provided invaluable guidance for this work [102]. Based on a review of projects found in the literature, as well as the theoretical concepts presented here, the most appropriate choices for control, software, and hardware have been selected from a number of available options.

At the heart of this work lies a control scheme which, given the applications found in the literature, should be stable, adaptive, robust, resilient to bursting, and offer confidence in autonomous applications. The overview of the complete quad-rotor system, and specifically the place of the control scheme therein is highlighted in red in Figure 3.1. The first, baseline control analyzed for this purpose is a linear PID scheme. This method is straight forward

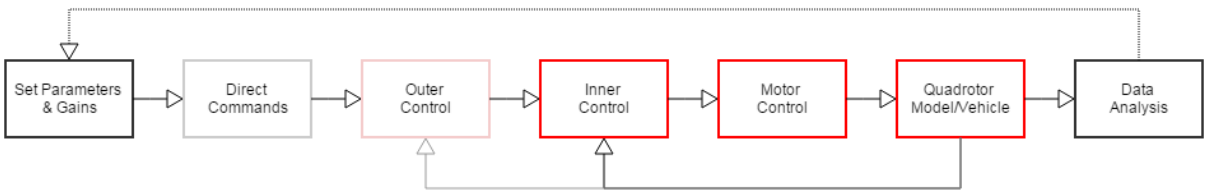


Figure 3.1: System Overview for Simulation and Experimentation

to implement as it requires only an error signal, as well as its derivative and integral forms. Three manually adjustable gains can be used to tune the control for a specific vehicle, application, and environment. This type of control is found often in the literature because it can easily be tuned to be stable in specific experiments. However, the PID is not adaptive and must rely upon human input to adjust gains, and especially to alter control input to overcome disturbances in flight. As a result, the potential for autonomy for the PID control is limited.

The second control explored in this work is a non-linear, adaptive Cerebellar Model Articulation Controller. Using Lyapunov's direct method this control has also been proven to be stable, but unlike PID schemes, a CMAC can adapt automatically to disturbances and changes to both model and environment using a unique method of on-line weight estimate control. Given the non-linearities of the quad-rotor control problem, and the disturbances noted in the literature, it is hypothesised that the CMAC will outperform the baseline PID in adaptive tasks. The number of inputs and learning rate required for the vehicle meant that a CMAC would theoretically perform better than an RBFN or MLP scheme. With resiliency to disturbances a key attribute in the challenging application environments researched, a robust update algorithm was added. This update scheme seeks to eliminate bursting behaviour found in the literature by introspectively monitoring weights to ensure updates significantly improve performance. In short, the update scheme will remove weight drift that has been shown to lead to bursting. The introspective weight voting scheme is viewed as the most advanced robust update algorithm found in the literature, but, it has yet to be proven in practical experimentation. The simulation environment, and eventually, a commercially available quad-rotor platform, will validate the viability of this control scheme.

The software environment chosen for both simulation and practical application is MATLAB, shown in black in Figure 3.1. This program facilitates a simple transition from matrix mathematical models to implementable code, an expansive feature library, and even offers

tools to profile code for efficiency. Furthermore, integration of file input/output (I/O) and user datagram protocol (UDP) tools allow for simple data storage and retrieval as well as communication to and from the selected quad-rotor platform. This project makes use of these tools to present a real-time, graphical control interface capable not only of direct, user control, but playback from previous simulations or purpose-built trajectory tracking programs. The flexibility of the MATLAB environment allows a number of different control schemes to be compared using the same model and input vectors, making it easy to evaluate different control schemes.

The quad-rotor platform chosen to experimentally demonstrate the viability of the introspective voting CMAC scheme investigated in this project is an AR.Drone 2.0 model from Parrot, whose place in the system is shown in grey in Figure 3.1. The AR.Drone 2.0 arrives assembled, with gyroscopic, accelerometer, sonar, and two camera sensors to calculate an accurate position and orientation reading, as well as integrated motor controls to actuate the vehicle. Additionally, an optional Global Positioning System (GPS) sensor is available to more accurately track the vehicle in 3D space. Communication to and from the platform requires a simple 802.11 wifi connection and the AR.Drone 2.0 can interface with multiple devices for added redundancy in emergency situations. Should the numerous redundancy systems implemented fail, replacement parts are easy to procure and the entire vehicle is extremely economical, with a replacement cost of approximately $300 CAD. With the addition of a custom enclosure and multiple batteries, the quad-rotor platform offered by Parrot is invaluable for investigating the practical performance of the proposed control system.

However, before the AR.Drone 2.0 platform and enclosure are tested, even prior to constructing a simulation environment, a viable control system, the heart of this entire work, must be crafted. A number of control schemes were considered for this application, and many were tested in simulation, but the focus of this work is the derivation and stability of an introspective weight voting update scheme for a Cerebellar Model Articulation Controller.

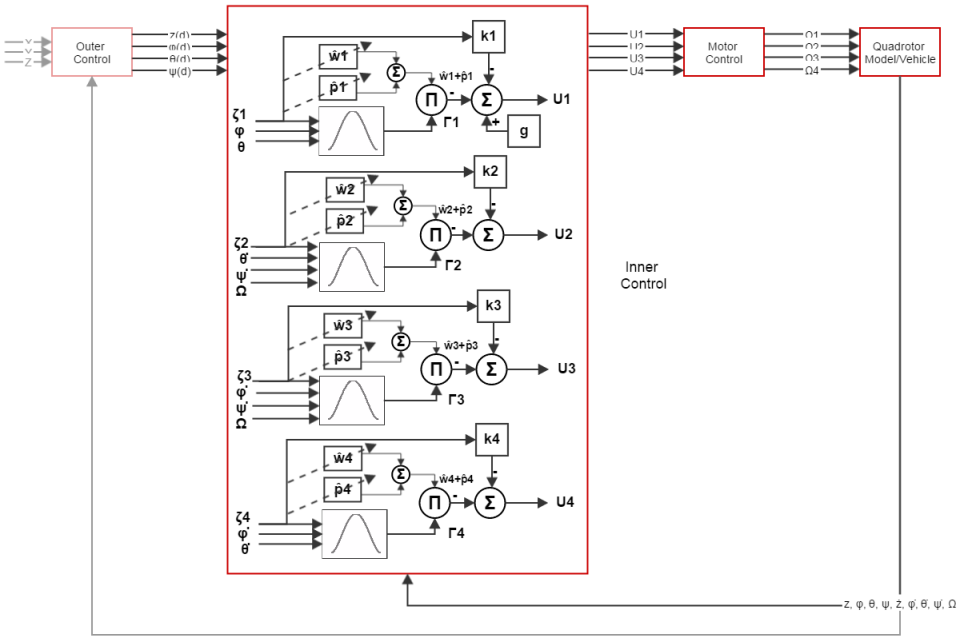


Figure 3.2: Control System Block Diagram

3.2 Control Scheme

The basic control design of the system employed consists of an inner loop responsible for stabilizing the orientation and rotation of the vehicle and an outer loop responsible for directing the quad-rotor to specific locations. At one end of the loop is user input from the interface environment, either from a direct control signal or a translated trajectory path. At the other end is the quad-rotor vehicle, specifically the motor control signals that actuate the platform. Finally, at the heart of the control scheme is the method of control itself.

As shown in Figure 3.2, direct input to the control system includes desired roll, pitch, yaw, and z values. These desired parameters may be provided from the inner loop when the vehicle is controlled directly, or translated from desired x, y, and z positions provided from the outer loop when trajectory tracking is active. The trajectory tracking algorithm employed for this control is a simple linear scheme, as defined by Equation **??**. The mathematical model of aquad-rotor can be described using accelerations in the x, y, and z directions as well as roll, pitch, and yaw rotations. However only the later four variables are directly

controlled. The four desired values are coupled with a number of actual values provided by the model in order to derive a control signal. These feedback signals from the model include not only x, y, and z positions as well as roll, pitch, and yaw translations but their first derivatives and motor speed. When in simulation mode, these values are merely calculated from the mathematical model. However, when in flight mode they are derived from the AR.Drone 2.0 sensor output. With actual and desired values provided, the control is able to calculate required rotor speed changes to drive the vehicle toward the desired orientation. The accuracy of these four signals determines the success of the control method.

$$\zeta_l = \begin{vmatrix} z - z_d \\ \phi - \phi_d \\ \theta - \theta_d \\ \psi - \psi_d \end{vmatrix} \quad \zeta_h = \dot{\zeta}_l = \begin{vmatrix} \dot{z} - \dot{z}_d \\ \dot{\phi} - \dot{\phi}_d \\ \dot{\theta} - \dot{\theta}_d \\ \dot{\psi} - \dot{\psi}_d \end{vmatrix} \tag{3.1}$$

The difference between the measured and desired values for these four variables, as well as the rate of error change, are combined to form the state space of the system, ζ, as defined by Equation 3.1. Actual values are listed without a subscript, as in z, their desired counterparts are designated with a subscript, z_d. Finally, an adjustable weight, λ, is multiplied by the position error. This auxiliary error is the focus of the control design.

3.2.1 PID

The most basic controls found in the literature are PID schemes, or variations thereof. This type of control is typically described using its most recognizable application; ship steering. A PID control sums three signals representing current error in course, previous errors, and the rate of change in error.

$$U = K_p\zeta + K_i \int \zeta + K_d \frac{\Delta\zeta}{\Delta t} \tag{3.2}$$

As defined by Equation 3.2, each error is multiplied by and adjustable gain, K, to form a

control signal. Given the error signal, ζ, previously presented in Equation 3.1, the derivation and implementation of the PID control scheme used as a baseline for this work is quite straight forward.

3.2.2 CMAC

The control scheme which is compared to the baseline PID, and which is the primary focus of this work is a Cerebellar Model Articulation Controller. This scheme uses the same basic error signal as the PID, and also breaks control into four parts. Each control has a separate, dedicated CMAC structure. While the layout and initialization of each of the four CMACs is the same, each executes its own individual weight updates. Each CMAC contains ten offset layers, each layer is split into twelve quantizations, or cells. Each cell contains a local activation function, which in the design implemented is a spline function. Between all four CMACs, there is a total of twelve unique input signals, but any given CMAC will use only three or four of these signals.

$$\begin{aligned} \Gamma(x) &= 16(h^2 - 2h^3 + h^4) \\ \text{where } h &= \frac{x - (c_i - a)}{2a} \end{aligned} \tag{3.3}$$

When an input signal is provided to a CMAC, a single cell will be activated on each layer. The location of each cell on the ten layers is often unique due to the offset of each layer from the others. The input is then normalized and applied to the spline function of the activated cell on each layer, as defined by Equation 3.3. Only active values are computed on each layer, saving computing time for the other eleven cells.

$$\Gamma(x)(\hat{p} + \hat{w}) + D = \alpha^{-1}(\zeta_h - \dot{\zeta}_h) \tag{3.4}$$

Each cell output is multiplied by the sum of the current permanent and active weight estimates, $\hat{p}$ and $\hat{w}$ respectively, which together with a disturbance value, D, approximate the non-linear dynamics of the quad-rotor, as defined by Equation 3.4.

3.2.3 Leakage Weight Update

A leakage style weight update scheme is a useful adaptation technique for a CMAC because it requires no *a priori* information about the system. It is a less aggressive update scheme than its cousin, e-mod, making it less prone to bursting. As a result, it is used to analyze the performance of the introspective weight voting scheme, specifically with respect to bursting resiliency.

$$\dot{\hat{w}} = \beta(\Gamma^T(x)\zeta - \nu\hat{w}) \tag{3.5}$$

The leakage weight update defined by Equation 3.5 has been shown to provide Lyapunov stable weight and error values, given the gains are chosen appropriately. Increasing gains can improve system performance, but also increase the likelihood of bursting. Given that the quad-rotor vehicle already exhibits two major factors for bursting; an underdamped system and oscillatory environmental disturbances, bursting is a real concern.

Practically, bursting can occur when an adaptive system is subjected to an oscillatory disturbance that consistently excites two or more nearby cells repeatedly, causing their weights to drift in opposing directions. As weight values drift closer towards the Lyapunov bound defined by the chosen gains, updates are stopped providing an opportunity for the system error to grow quickly and unexpectedly. If the gains are adjusted to increase the Lyapunov stability bound on the system, bursting events may be delayed to a trivial frequency, but performance will certainly be reduced. For this reason, an introspective voting weight update scheme has been explored.

3.2.4 Introspective Voting Weight Update

Adaptation of active CMAC weights is controlled using a robust e-mod technique, with a deadzone component to ensure active weight updates are stopped before such time as bursting would typically appear. A second set of permanent weights is updated according

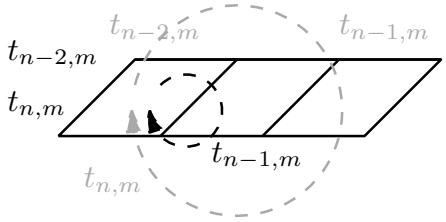


Figure 3.3: Oscillating CMAC Cell Cycle Leading to Bursting

to an introspective weight voting scheme.

$$\dot{\hat{w}} = \begin{cases} \beta(\Gamma^T(x)\zeta - \nu\|\zeta\|\hat{w}) & \text{if } t - t_{n-m} < T_{max} \\ 0 & \text{Otherwise} \end{cases} \tag{3.6}$$

Weight updates are calculated continuously, but only analyzed and permanently applied when cells are deactivated. To avoid the possibility of an active weight estimate bursting while waiting for cell deactivation, weight updates are stopped using a deadzone style control at a time period, T_{max}, which is less than the typical observed bursting period. Weight estimate updates are defined by Equation 3.6. The weight update rule includes a number of predefined values, as well as an adaptation constant, β, and a learning constant, ν.

$$\dot{\hat{p}} = \begin{cases} \hat{w}_{n-2,m-1} & \text{if } vote_{n-2} > 0 \\ \mu(\hat{b}_{n-2} - \hat{p}_{n-2}) & \text{Otherwise} \end{cases} \tag{3.7}$$

Permanent weights are updated on-line when a cell has been deactivated using a voting algorithm which takes into account the effect of recent weight updates, as defined by Equation 3.7 [87]. Current, last, and second last weight values, $\hat{w}_{n,m}$, $\hat{w}_{n-1,m}$, and $\hat{w}_{n-2,m}$ respectively, are all stored in memory. Sample activation trajectories for a two cell and a multi-cell oscillatory behaviour are shown in Figure 3.3. Voting occurs at the current time, $t_{n,m}$, for the second last layer only to ensure the system has had a sufficient amount of time to accurately measure the effect of the active weight. If a vote goes in favour of the on-line weight update, it will become the permanent weight, if not the permanent weight will instead

be nudged toward the best weight found thus far by a small learning factor, μ.

$$vote_{n-2} = \begin{cases} & \frac{E_{n-2}}{\hat{w}_{n-2,m-1}} < 0 \\ \|\hat{w}_{n-2,m-1}\| & \text{if } E_{n-2,m}(E_{n-2,m}) > 0 \\ & |\Delta E_{n-2}| < \sigma_{E,min} \\ -\|\hat{w}_{n-2,m-1}\| & \text{otherwise} \end{cases} \tag{3.8}$$

A vote will go in favour of a weight update if three conditions are met, as shown in Figure 3.8. First, the error and weight update must be of the same sign to signify that the update is aiding the system. Second, the sign of the error must be the same as it was in the last calculation to show that the weight update is still moving in the correct direction. And most importantly, the weight update must significantly decrease the output error of the system, the variable of significance being represented by $\sigma_{E,min}$.

$$\hat{b} = \begin{cases} \hat{p}_{n-2} & \text{if } E_{n-2,m} < S_{n-2,m-1} \\ \hat{b}_{n-2,m-1} & \text{Otherwise} \end{cases} \tag{3.9}$$

$$S_{n-2,m} = \begin{cases} |E_{n-2,m}| & \text{if } |E_{n-2,m}| < S_{n-2,m-1} \\ S_{n-2,m-1} & \text{Otherwise} \end{cases} \tag{3.10}$$

Finally the best weight for each cell is updated based on the error noted in the system output while it was activated, as defined by Equation 3.9. If a permanent weight results in a lower system error during its activation, it will become the new best weight for that cell. The lowest system errors are saved for each cell using the system defined by Equation 3.10.

While a traditional e-mod update will simply try to force errors to null values, this new technique evaluates the effect of a new update against the size of the new weight proposed. This simple form of democratic governance eliminates weight overgrowth to prevent bursting of the CMAC system. While traditional weight representation and update rules are shown

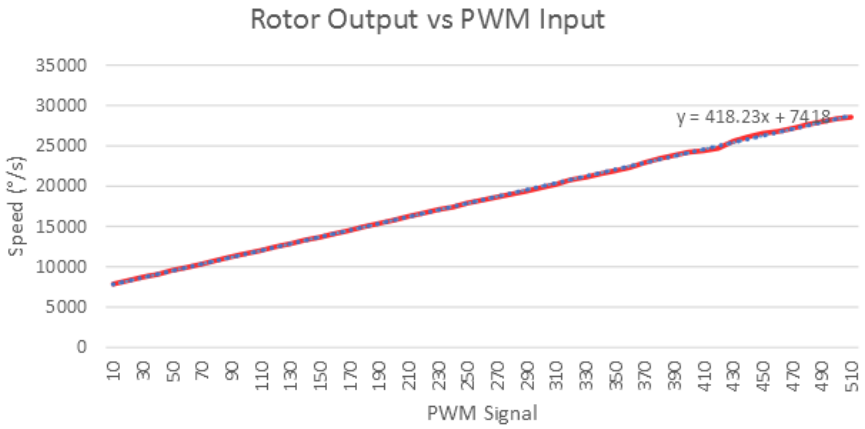


Figure 3.4: AR.Drone 2.0 Non-linear PWM Input versus Rotor Speed Output

in this work, the complexity of the introspective weight voting mechanism at work behind the scenes should always be remembered.

Finally, the CMAC output is calculated by multiplying the vector of values produced by all activated cells by a vector of weight estimates. Each of the four outputs provided is a control signal for the quad-rotor vehicle. In this way a non-linear estimation of the system is formed and refined.

3.2.5 Motor Control

Regardless of the control scheme chosen, the output is a control signal, U, representing desired changes to z, roll, pitch, and yaw accelerations. This control signal must be converted into four rotor speeds that are then sent to the quad-rotor vehicle.

$$\begin{vmatrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \\ \Omega_4 \end{vmatrix} = \begin{vmatrix} 0.25 & 0.25 & 0.25 & -0.25 \\ 0.25 & -0.25 & 0.25 & 0.25 \\ 0.25 & -0.25 & -0.25 & -0.25 \\ 0.25 & 0.25 & -0.25 & 0.25 \end{vmatrix} \begin{vmatrix} U_1 & U_2 & U_3 & U_4 \end{vmatrix}^{\frac{1}{2}} \tag{3.11}$$

As defined by Equation 3.11, rotor speeds can be obtained from control signals using a simple matrix and square root function. In simulation, these speeds are used directly to actuate the quad-rotor model. However, when the physical vehicle is employed, these

values must be converted to single byte PWM values. For this purpose, an experimental conversion formula was obtained by measuring rotor speeds at varying PWM values and applying a linear best fit function, as shown in Figure 3.4. In this way, the quad-rotor vehicle is actuated by the control signal calculated in the inner control loop.

3.2.6 Outer Loop Control

$$\ddot{X} = -sin(\theta)cos(\phi)\frac{b}{m}U_1 \tag{3.12}$$

$$\ddot{Y} = sin(\phi)\frac{b}{m}U_1 \tag{3.13}$$

$$\phi_d = sin^{-1}(K_1(Y_d - Y) - K_2\dot{Y}) \tag{3.14}$$

$$\theta_d = -sin^{-1}((K_1(X_d - X) - K_2\dot{X})/cos(\phi)) \tag{3.15}$$

The outer control loop passes desired roll and pitch angles into the inner loop when position tracking experiments are performed. The outer control operates under the assumption that the vehicle has been moved to the correct yaw orientation and that the desired height has been set. As described in Equations **??** and 3.13, desired roll and pitch angles are derived from the kinematics previously presented in Equations 2.19 and 2.20, by substituting $\psi_d = 0$.

Position acceleration values are simplified analytically into position and velocity components, supplemented by control gains K_1 and K_2. Finally, desired roll and pitch angles are defined by the outer loop control using Equations 3.14 and 3.15. These values are passed into the inner loop to actuate the vehicle in a stable manner towards desired coordinates during tracking applications.

3.2.7 Stability

Using Lyapunov's theorems outlined in the introduction, the stability of the introspective weight voting CMAC system is analyzed to demonstrate that it is a successful candidate for further exploration via simulation. The derivation that follows is found commonly within the literature, including [79], and is adapted here for the specific control equation and weight update scheme used in this work. The first step of determining a uniform ultimate bound for the system is to establish a positive definite, decrescent function, or Lyapunov candidate, v.

$$V = \frac{1}{2}\zeta^T \alpha^{-1} \zeta + \frac{1}{2}\tilde{w}^T \beta \tilde{w} \tag{3.16}$$

The proposed function, defined by Equation 3.16, sums a normalized, weighted auxiliary error with a weighted, normalized permanent weight error. The choice of this seemingly arbitrary Lyapunov surface will become clear when its derivative is calculated in the course of the stability analysis.

$$U = -\Gamma(x)(\hat{p} + \hat{w}) - k\zeta \tag{3.17}$$

During the Lyapunov stability analysis of this proposed function, key substitutions will be made. First, the control signal for the system, U, is defined by Equation 3.17 using CMAC non-linear estimation and system error functions. Second, the weight update error of the system will be defined as the difference between the actual weight update and the estimated weight update calculated by the introspective weight voting system, $\dot{\tilde{w}} = \dot{w} - \dot{\hat{w}}$.

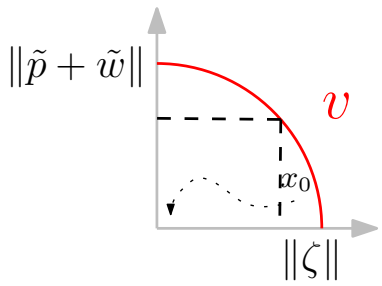


Figure 3.5: Lyapunov CMAC Weight and System Error Bounds

$$\dot{V} = \zeta^T \alpha^{-1} \dot{\zeta}^T + (\tilde{p}+\tilde{w})^T \beta^{-1} (\dot{\tilde{p}} + \dot{\tilde{w}}) \tag{3.18}$$

$$\dot{V} = \zeta^T \alpha^{-1} \dot{\zeta}^T - (\tilde{p}+\tilde{w})^T \beta^{-1} (\dot{\hat{p}} + \dot{\hat{w}})$$

$$\dot{V} = \zeta^T (\alpha^{-1} (\lambda x_2 + \dot{x}_2)) - (\tilde{p}+\tilde{w})^T \beta^{-1} (\dot{\hat{p}} + \dot{\hat{w}})$$

$$\dot{V} = \zeta^T (\Gamma(\hat{p}+\hat{w}) + D + U) - (\tilde{p}+\tilde{w})^T \beta^{-1} (\dot{\hat{p}} + \dot{\hat{w}})$$

$$\dot{V} = \zeta^T (\Gamma(\hat{p}+\hat{w}) + D - \Gamma\hat{w} - k\zeta) - (\tilde{p}+\tilde{w})^T \beta^{-1} (\dot{\hat{p}} + \dot{\hat{w}})$$

$$\dot{V} = \zeta^T (D - k\zeta) + (\tilde{p}+\tilde{w})^T (\Gamma\zeta(\hat{p}+\hat{w}) - \beta^{-1} (\dot{\hat{p}} + \dot{\hat{w}}))$$

$$\dot{V} = \zeta^T (D - k\zeta) + \nu(\tilde{p}+\tilde{w})^T \|\zeta\| (\hat{p}+\hat{w})$$

$$\dot{V} = \zeta^T D - \zeta^T k\zeta + \nu(\tilde{p}+\tilde{w})^T \|\zeta\| (p+w) - \nu(\tilde{p}+\tilde{w})^T \|\zeta\| (\tilde{p}+\tilde{w})^T$$

These equations allow the derivative of the Lyapunov surface candidate, $\dot{V}$, defined by Equation 3.2.7, to be investigated with the goal of funding a negative semi-definite region.

$$\dot{V} \leq 0 \; when \; |\zeta| > \frac{D_{max}}{4} + \frac{\nu\|p+w\|^2}{4k} \tag{3.19}$$

$$\dot{V} \leq 0 \; when \; |\tilde{p}+\tilde{w}| > \frac{\|p+w\|}{2} + \sqrt{\frac{D_{max}}{\nu} + \frac{\|p+w\|^2}{4}}$$

The final Lyapunov result indicates that the system is mathematically bounded within a region inside the proposed surface, V, when the derivative of that function, $\dot{V}$ is negative semi-definite. The negative semi-definite definition of the Lyapunov surface requires that the norm of the auxiliary error and the norm of the weight error fall within a specific region defined by Equation 3.2.7. Error signals are normalized to allow a clearer view of the bound using only one graphical quadrant. The Lyapunov bound is shown in red, and a sample, stable trajectory in black. Both regions are defined by the maximum system disturbance, D_{max}, the learning constant ν, current weight values w, and the control constant, k. This result mirrors that described in [79], and is similar to that found using a different, describing function anaylsis in [78] Therefore, the proposed control system is proven to be uniformly ultimately bounded as shown in Figure 3.5.

3.3 Software

The first step in proving the value of any control system is to build a simulation environment. The software environment includes not only the control system heart, but a number of important extremities. Before the control system is reached, a number of inputs from vehicle parameters, to the control scheme, to control gains, and even input commands must be set. And, once the simulation is complete, data output for analysis and improvement is key.

For this purpose, the MATLAB software environment was used. MATLAB not only offers a number of mathematical, communication, and optimization tools behind the scenes, but also fosters the creation of user interfaces to present data in an immediately impactful manner. The software environment created for this application provides dynamic user control input, comprehensive data retention, and a seamless switch between simulated and practical trials within the same interface environment.

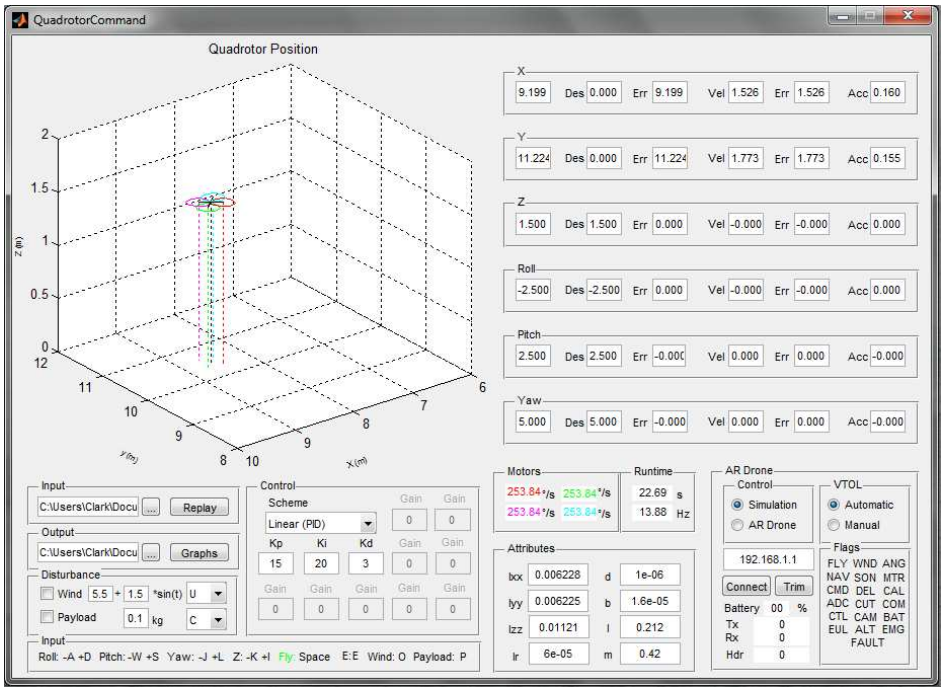


Figure 3.6: Matlab Simulation Environment

3.3.1 Interface

The interface environment created to test the control schemes of interest presents a significant amount of data to the user in a well-defined and easy to interpret sections, as showcased in Figure 3.6. At the top left, a three dimensional representation of vehicle orientation and position provides real-time feedback. At the top right, numerical outputs of all kinematic values and errors are shown to give a more detailed overview of performance. Finally, rotor speeds and runtime complete the user feedback provided in the top portion of the interface.

The bottom third of the environment contains user input, data management, and vehicle specific information. At the left of this portion, folder selection for input and output data files is available above parameter information for simulation disturbances. At the centre left position is a section for control scheme selection and associated gain adjustment. Under the entire bottom left portion of the interface lies a section that not only defines valid keyboard input, but also highlights keystrokes in real-time. Moving to the bottom right of the environment, a tracking functionality switch and associated gains can be found above a

model parameter section for the vehicle. Finally, at the far right, hardware specific input and output information can be found for the AR.Drone 2.0 quad-rotor vehicle. This section includes communication control, key flight status flags, and a control switch that allows the interface to easily apply user input to the simulation environment or the AR.Drone 2.0 hardware.

A typical interaction with the interface will begin with user input. This process consists of setting the attributes of the specific quad-rotor model to be used, and if necessary, disturbance values. Next, a control scheme can be selected and gains appropriately specified. If desired, a tracking scheme can be activated and gains selected. The AR.Drone 2.0 can also be selected as the target for control signals and data acquisition then configured if an experimental trial is desired. Finally, to conclude the user input process, the space bar can be struck to command the quad-rotor to a heigh of one meter and await real-time instructions, or the replay button pressed to start playback of a previously saved input routine. Once a trial is complete, the results can be viewed immediately and the data analyzed to guide the implementation of control improvements. Through its design, this software system enables the rapid development of any control scheme and clear performance comparisons between contrasting control paradigms.

3.3.2 User Input

User input to the simulation interface occurs in two distinct ways. First, while the simulation is at rest, parameters for model dynamics, disturbances, control gains, and even the control scheme itself may be adjusted while the simulation is at rest. Second, while the simulation is running, control input is available from the user via the keyboard. In this way both the vehicle model and control scheme can be adjusted and quickly tested with real-time input to simulate practical application as closely as possible.

Model parameter adjustment provides a means to simulate control using the published dynamics of an existing vehicle, or develop a model of a physical vehicle with unknown

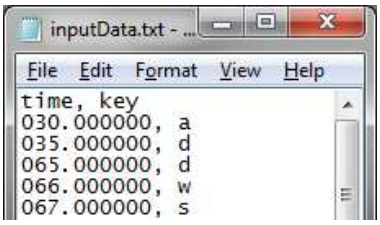
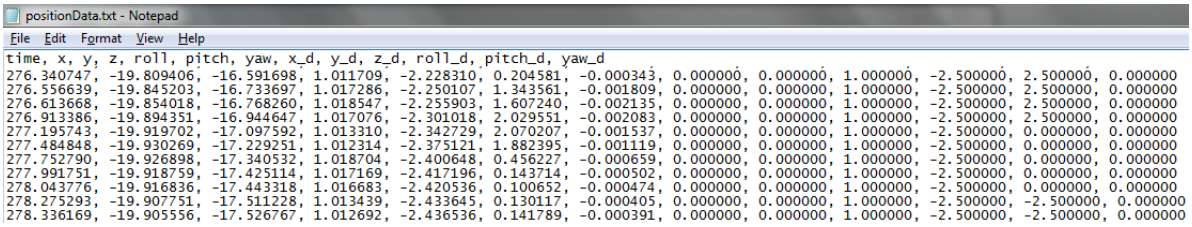


Figure 3.7: Sample Control Input Simulation File

dynamics. Any one of a number of control schemes may be selected from a drop down menu in the interface and gain parameters adjusted between simulations to allow tuning. A selection of basic schemes found in the existing literature is provided, alongside the introspective weight voting scheme.

Disturbance inputs include wind, via constant and oscillating velocity parameters plus direction, as well as payload, by way of mass and position parameters. Control selection allows the same model parameters and environmental disturbances to be easily applied to different schemes to quickly compare their strengths and weaknesses. Keyboard input allows real-time control input through simple sequential roll and pitch rotation or x and y translations, plus height and yaw commands. This feature permits direct control of the quad-rotor by a human operator, as well as outer loop position tracking input. Combined, the variety of user inputs allow a multitude of different quad-rotor configurations to be simulated under varying environmental conditions with easily tuned control schemes.

Finally, the simulation can be run completely autonomously, using input commands from a previously created file rather than real-time input. Input files can be generated specifically for specific tasks, such as search pattern execution, or saved from previous human input, as shown in Figure 3.7. In this way control schemes can be developed and tested with either partial or full autonomy in challenging simulated environments and the resulting data analyzed to compare and contrast differing styles. Once satisfactory performance is achieved, control can be transferred to the quad-rotor platform via a simple switch in the interface.


positionData.txt - Notepad

File Edit Format View Help

```
time, x, y, z, roll, pitch, yaw, x_d, y_d, z_d, roll_d, pitch_d, yaw_d
276.340747, -19.809406, -16.591698, 1.011709, -2.228310, 0.204581, -0.000343, 0.000000, 0.000000, 1.000000, -2.500000, 2.500000, 0.000000
276.556639, -19.845203, -16.733697, 1.017286, -2.250107, 1.343561, -0.001809, 0.000000, 0.000000, 1.000000, -2.500000, 2.500000, 0.000000
276.613668, -19.854018, -16.768260, 1.018547, -2.255903, 1.607240, -0.002135, 0.000000, 0.000000, 1.000000, -2.500000, 2.500000, 0.000000
276.913386, -19.894351, -16.944647, 1.017076, -2.301018, 2.029551, -0.002083, 0.000000, 0.000000, 1.000000, -2.500000, 2.500000, 0.000000
277.195743, -19.919702, -17.097592, 1.013310, -2.342729, 2.070207, -0.001537, 0.000000, 0.000000, 1.000000, -2.500000, 0.000000, 0.000000
277.484848, -19.930269, -17.229251, 1.012314, -2.375121, 1.882395, -0.001119, 0.000000, 0.000000, 1.000000, -2.500000, 0.000000, 0.000000
277.752790, -19.926898, -17.340532, 1.018704, -2.400648, 0.456227, -0.000659, 0.000000, 0.000000, 1.000000, -2.500000, 0.000000, 0.000000
277.991751, -19.918759, -17.425114, 1.017169, -2.417196, 0.143714, -0.000502, 0.000000, 0.000000, 1.000000, -2.500000, 0.000000, 0.000000
278.043776, -19.916836, -17.443318, 1.016683, -2.420536, 0.100652, -0.000474, 0.000000, 0.000000, 1.000000, -2.500000, 0.000000, 0.000000
278.275293, -19.907751, -17.511228, 1.013439, -2.433645, 0.130117, -0.000405, 0.000000, 0.000000, 1.000000, -2.500000, -2.500000, 0.000000
278.336169, -19.905556, -17.526767, 1.012692, -2.436536, 0.141789, -0.000391, 0.000000, 0.000000, 1.000000, -2.500000, -2.500000, 0.000000
```

Figure 3.8: Sample Simulation Data File

3.3.3 Data Analysis

Whether simulating a control scheme on a model, or applying it to the AR.Drone 2.0 quad-rotor platform, data acquisition and retention is key to analysing it effectiveness. For this purpose, the file I/O capabilities of MATLAB are tremendously valuable. For each flight, a unique folder is created to hold text files that store all data, either simulated or measured by the test platform.

Individual files for all data, from kinematics, to disturbances, motor speeds, even the type of control used are saved for each flight. An example is shown in Figure 3.8. Furthermore, input data such as model parameters, control gains, and even operator keystrokes, is also saved to comma delimited text files. This process not only allows simulated input programs to be created, but permits the same user input to be applied to various control schemes in real-time. This input retention capability, along with the output data acquisition techniques described, allows a detailed graphical analysis of control within an interactive environment.

3.4 Hardware

Once a control system has been proven through simulation, the next logical step is to test it in practice using a specific vehicle. In the early literature explored, quad-rotor vehicles were largely custom built for a specific research application. With the recent popularity of the vehicle for a number of commercial applications, industry has begun to develop versatile platforms for consumer use. Given the delicate balance of the quad-rotor vehicle and advice from those who have embarked on work similar to this project before, a commercially

available AR.Drone 2.0 vehicle was chosen for experimental trials. This choice provides a capable, reliable hardware platform upon which software control can be accurately tested.

3.4.1 AR.Drone 2.0 Quad-Rotor

A small Paris based engineering firm called Parrot began an ambitious project to design and market an affordable, easy to use quad-rotor vehicle to the general public in 2004 [18]. The Engineering team took the time to observe and model advanced rotor vehicle dynamics including blade flapping, ground effects, and aero drag in an effort to understand how best to fuse measurements from low-cost sensors to obtain accurate control input in a closed loop system [121] [17]. The Parrot team applied this knowledge from an existing single rotor vehicle to their quad-rotor design, and then explored the effect of mass placement to complete their design [16]. The extensive work compiled for this project was eventually packed into a quad-rotor design dubbed the AR.Drone and presented to consumers at the Consumer Electronics Show (CES) in 2010. From the outset, this project endeavoured to combine the unique strengths of an arsenal of low cost sensors to provide an accessible quad-rotor platform. By exposing and documenting not only a command structure, but navigation data and video streams, Parrot ensured their quad-rotor platform could be used for research applications. As demonstrated by Thomas Krajnk and a team at the Czech Technical University in Prague, the interface structure of the AR.Drone allows for simple control of the quad-rotor using roll, pitch, yaw, and height commands designed on top of built-in stabilization algorithms [64]. Additionally, more direct control can be achieved by sending pulse width modulation (PWM) command values directly to each motor, or even replacing the built-in Linux based stabilization algorithms altogether. This design paradigm has resulted in the use of Parrot products in various projects from camera based navigation at the University of Munich [34], to human interaction at the University of Texas [71]. The success of the AR.Drone project led to the release of the AR.Drone 2.0 vehicle in 2012, among many others.

Figure 3.9: Parrot AR.Drone 2.0 Hardware

The specific quad-rotor design used for this work is the improved AR.Drone 2.0, as shown in Figure 3.9. In the configuration employed for all tests, it weighs approximately 420 grams and measures 517 millimeters squared. It runs Linux 2.6.32 on a 1 gigahertz, 32 bit instruction length ARM Cortex A8 processor with 1 gigabit DDR2 RAM running at 200 megahertz. Among the impressive array of sensors it can boast are a 3-axis gyroscope with 2000 degree per second precision, a 3-axis accelerometer with a 50 mg tolerance, a 3-axis magnetometer with 6 degree precision, a pressure sensor accurate to a 10 Pascal tolerance, an ultrasonic sensor, and a 60 frames per second QVGA resolution downward facing camera. Thrust is generated from four, 14.5 watt, 28,500 revolutions per minute motors, each geared at 1:8.75 reduction and controlled by a dedicated 8 million instructions per second, central processing unit. Power to the vehicle is supplied by a single 1000 milliamp hour lithium polymer battery offering approximately 12 minutes of flight time. Communication to this powerful system is offered via a wifi connection over b, g, or n standards.

3.4.2 AR.Drone 2.0 Parameters

While the promotional material provided by Parrot contains an extensive list of specifications, there are a few necessary parameters that have been omitted. To obtain an accurate model of the system for simulation and PID control, some attributes were measured. First, the mass of the vehicle and linear distances were measured using a simple scale and rule respectively.

$$I = \frac{mgT^2D^2}{4\pi^2L} \tag{3.21}$$

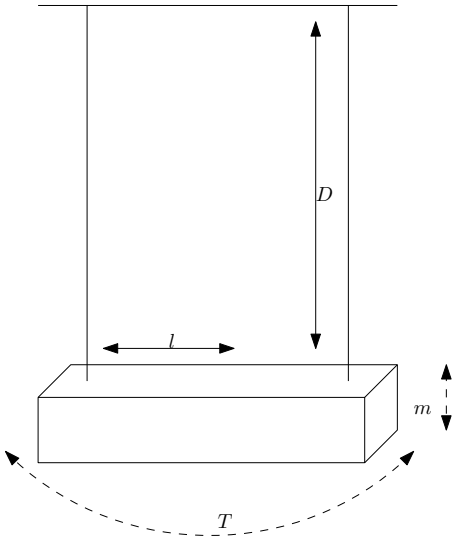


Figure 3.10: Bifilar Body Inertia Measurement Experiment Layout

$$I_r = \frac{mr^2}{2} \tag{3.20}$$

Next, the drag coefficient and rotor inertia values were estimated through mathematical modelling, as defined by Equation 3.20. Vehicle bode moment of inertia values in the x, y, and z planes were estimated experimentally using the bifilar pendulum built into the enclosure design. This type of bifilar measurement has been used for inertia measurement in much larger aircraft since 1927 with less than 1% error, improved in 1930, and even used as recently in 2014 to model current UAV dynamics [43] [94] [62]. The basic principle of the apparatus is rooted in simple Lagrangian mechanics which dictate that the kinetic energy of the system transfers to potential energy and back again as the attached mass is raised and lowered by the rotating motion of the twisting filars.

Ignoring damping effects, this results in a moment of inertia calculation that is proportional to the force on the suspended mass due to gravity, the period squared of the oscillation induced in the mass, the distance squared from the center of mass to the filars, and inversely

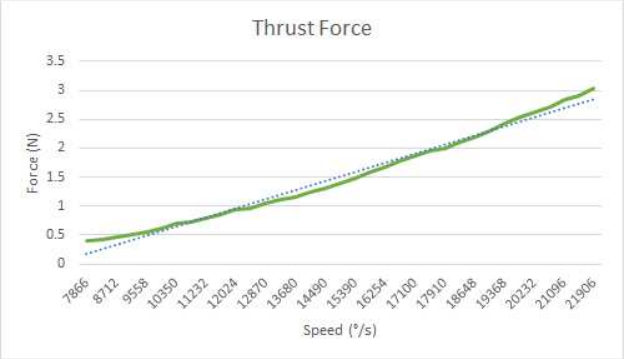


Figure 3.11: AR.Drone 2.0 Non-Linear Rotor Thrust Dynamics

PWM	Force (N)	Thrust Coefficient
20	0.42	1.54×10^{-9}
50	0.56	1.53×10^{-9}
80	0.74	1.58×10^{-9}
100	0.85	1.57×10^{-9}
120	0.96	1.54×10^{-9}
150	1.15	1.53×10^{-9}
180	1.40	1.56×10^{-9}
200	1.58	1.59×10^{-9}
220	1.77	1.60×10^{-9}
250	1.99	1.55×10^{-9}
280	2.30	1.59×10^{-9}
300	2.53	1.61×10^{-9}
320	2.70	1.56×10^{-9}
350	3.04	1.58×10^{-9}

Table 3.1: AR.Drone 2.0 Thrust Coefficient Characteristics

proportional to the length of the filars, as defined by Equation 3.21. By measuring the distance and mass parameters and timing the period of the induced oscillation, the bifilar pendulum allows moments of inertia to be estimated in the x, y, and z planes.

Finally, to understand the thrust characteristics of the rotors, the force generated at a series of speeds was measured to generate a linearized thrust coefficient, as shown in Table 3.1. Figure 3.11 shows the force generated by the combined effort of all four rotors at various speeds obtained through practical testing.

All parameter values, provided and measured, are listed in Table 3.2 which permits the formulation of an accurate mathematical model of the quad-rotor for control purposes.

Symbol	Quantity	Measure	Unit
m	Mass	0.420	Kg
l	Length	0.325	m
b	Thrust Coefficient	$1.573 * 10^{-9}$	$Kg(m)/\deg^2$
d	Drag Coefficient	$1.000 * 10^{-6}$	$Kg(m)/\deg^2$
I_r	Rotor Inertia	$3.880 * 10^{-5}$	$N(m)$
I_x	Roll Inertia	$1.096 * 10^{-2}$	$N(m)$
I_y	Pitch Inertia	$8.260 * 10^{-3}$	$N(m)$
I_z	Yaw Inertia	$2.946 * 10^{-2}$	$N(m)$

Table 3.2: AR.Drone 2.0 Vehicle Parameters

Experimental application of a proposed control system on the quad-rotor is performed by simply retrieving sensor data from the vehicle and transmitting the appropriate motor speeds to the craft. Each motor is controlled using an individual pulse width modulated (PWM) signal that can be varied among a variety of discrete levels resulting in rotor speeds ranging from 0 to 28,500 revolutions per minute (RPM). Communication to and from the aerial test platform flows wirelessly via user datagram protocol (UDP) which affords a modest 50m range, and permits the AR.Drone 2.0 the be operated not only indoors, but in more harsh outdoor environments as well.

The AR.Drone 2.0 makes one UDP port available to receive commands from the controlling interface, while a second separate port transmits orientation and velocity data approximately fifteen times per second. Data from the AR.Drone 2.0 is transmitted using IEEE-754 floating point standards and is saved to six decimal places to retain precision [1]. Commands to the AR.Drone 2.0 are sent in a string format with a unique, incrementally increasing index to ensure sequential execution. A variety of commands have been made available by the powerful Linux operating system onboard the AR.Drone 2.0, including simple automated take-off and land algorithms, orientation controls, direct rotor speed orders, and a variety of redundancy and safety features. The literature provides a list of commands as well as C language examples of many features that have been translated into the MATLAB environment for this application. This means that the applied control scheme employs the same structure

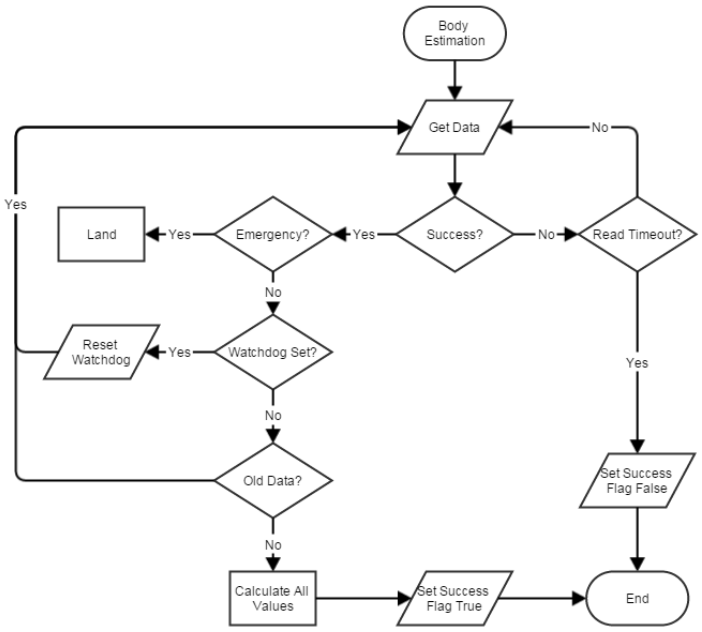


Figure 3.12: AR.Drone 2.0 Body Orientation Calculation

as that of the simulation, and by measuring the physical parameters of the quad-rotor model the applied outcome can be expected to be very similar to the simulation.

With the power of a Linux operating system, a variety of sensors, and a minimal UDP communication scheme the AR.Drone 2.0 is able to accurately measure its orientation in three dimensional space and quickly stream that data to the host controller. At rest the orientation data provided by the AR.Drone 2.0 does exhibit some drift, especially in the yaw plane, but the accuracy of its measurements is consistent with or better than other vehicles documented in the existing literature. However, streaming and synchronizing this data with the host controlling computer results in some unused measurements. On the host side, with minimal multi-tasking ability and complex control computations the interface environment requests only 15 packets of data per second, rather than the full 200 that the AR.Drone 2.0 can provide. In addition, some advanced variables important to control are not reported by the vehicle, so they must be inferred.

Therefore, a running time is kept for each control cycle to permit rate of change calculations while optimizing control speed. This allows the roll, pitch, yaw, and z positions

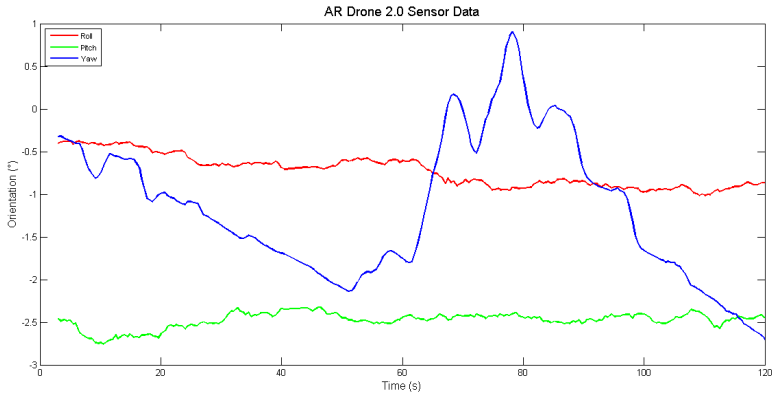


Figure 3.13: AR.Drone 2.0 Sensor Accuracy and Drift at Rest

as well as x and y velocities to be expanded into the complete range of position, velocity, and acceleration values for all parameters that are required for accurate control, as shown in Figure 3.12. All data is saved by the MATLAB interface for off-line analysis, as well as used by the selected control scheme to calculate output values on-line, which are sent to the flight control system.

The variety of sensors incorporated into the AR.Drone 2.0 design means that a fusion of readings from different measurement units can be created to form a more accurate value. But in practice, the measured values provided by the quad-rotor, as shown in Figure 3.13, vary by as much as one degree in the roll and pitch planes and four degrees in the yaw plane, even at rest, adding a certain non-linearity to the system. So too, the output of the AR.Drone 2.0 with respect to direct rotor speed control exhibits a non-linear thrust dynamic, especially at operational extremes, as shown in Figure 3.11. A linear function of best fit has been chosen for modelling and simulation purposes, but, the underlying quadratic nature of the rotor thrust dynamics observed through measurement is evident. In practice, the non-linear output behaviour of the system will affect control accuracy.

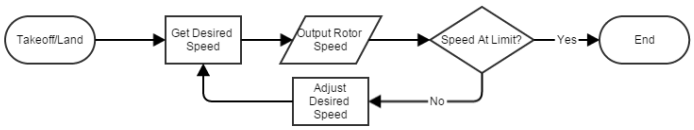


Figure 3.14: AR.Drone 2.0 Takeoff/Land Algorithm

3.4.3 AR.Drone 2.0 Flight

Control of the AR.Drone 2.0 begins with take-off, and ends with landing. Both are non-trivial operations that require dedicated algorithms, as defined in Figure 3.14. Once the AR.Drone 2.0 has been lifted to a safe height, the control system is implemented to stabilize, then move the quad-rotor as desired. The control loop operates at approximately 20 to 30 Hz, depending on the complexity of the chosen control scheme, and continuously checks for landing commands initiated by user input or a vehicle emergency condition. In this way, safe flight control can be assured.

When manually controlling rotor speeds of the AR.Drone 2.0, as intended by the control system, all rotor speeds are increased gradually and simultaneously to raise the vehicle to a hover before the selected control scheme takes over to stabilize the quad-rotor. A linearized function best fitting the rotor speeds observed through practical testing and measurement is used to produce raw PWM signals for each of the four motors. In this direct mode, the flight systems designed by Parrot operates at a reduced capacity and the quad-rotor reports limited orientation data. Furthermore, height sensors are not activated. Nonetheless, this mode of flight is extremely useful for tuning control systems for experimental application and demonstrating the effect of direct hover stabilization control of the quad-rotor.

3.4.4 Practical Redundancy

In practice, non-linearities and unmodeled dynamics can cause unexpected behaviour. First, within the quad-rotor platform itself, sensor readings vary within a certain margin of error.

While visual analysis of the quad-rotor orientation provides a coarse understanding of

(a) AR.Drone 2.0 Stability under PID Control (b) AR.Drone 2.0 at Rest with LEDs (c) AR.Drone 2.0 Stability under CMAC Control

Figure 3.15: AR.Drone 2.0 Stability shown with long exposure LED output

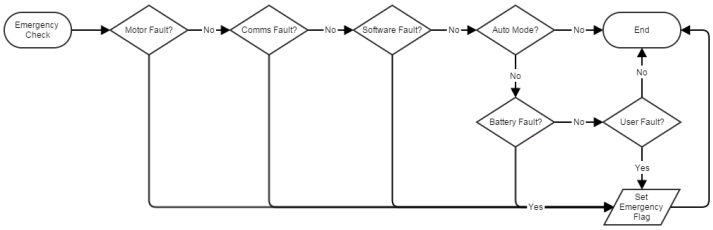


Figure 3.16: AR.Drone 2.0 Emergency Algorithm

the success of the applied control system, finer movements can be magnified by capturing the light emitted from the four perimeter light emitting diodes (LEDs) in a long exposure image, as shown in Figure 3.15. The colour of these LEDs can also be directly controlled to allow communication not only of outward system stability, but the otherwise hidden internal status of the working control scheme.

Additionally, the AR.Drone 2.0 provides status data with every piece of information it returns to the MATLAB interface. The environment is able to request an automatic landing from the vehicle when an error is reported in the returned data, when the battery level drops below twenty percent, or at the direct request of the user, as shown in Figure 3.16. An emergency landing feature has also been implemented which stops all rotors immediately at the request of the user. Finally, the AR.Drone 2.0 supports a connection to multiple devices, so a smart phone is always able to take control of the vehicle should MATLAB stop responding.

Outside the vehicle, environmental variables can introduce non-linear disturbances to the system. From the literature it is known that air density, which varies with elevation and temperature, has a significant impact on VTOL UAV translation speed and payload capabil-

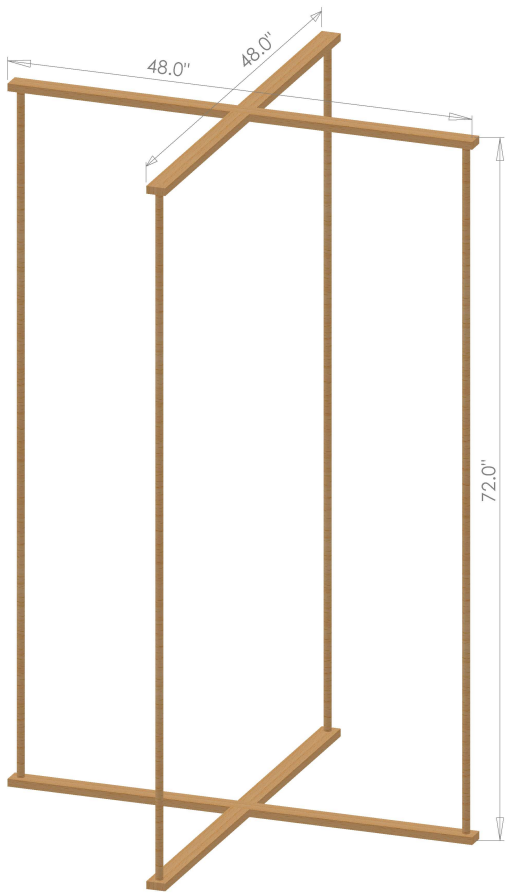


Figure 3.17: AR.Drone 2.0 Experimental Enclosure Design

ities [105]. For this reason, the simulation environment has been designed to accommodate the effects of wind and payload which have also been noted in the literature.

As a result of the non-linearities and disturbances expected in an experimental environment and given the kinetic energy of the blades under normal operation as well as the overall agility of the vehicle, the first trials of the proposed control scheme were confined to a flexibly bounded environment. Rather than employ a rigid jig, restricting one or more of the intricately connected degrees of vehicle freedom, the test platform places the AR.Drone 2.0 on tensile tracks within a rigid, yet flexibly bounded enclosure. This allows all six degrees of freedom to remain coupled in order to accurately test and refine the controlling algorithm within a safe environment.

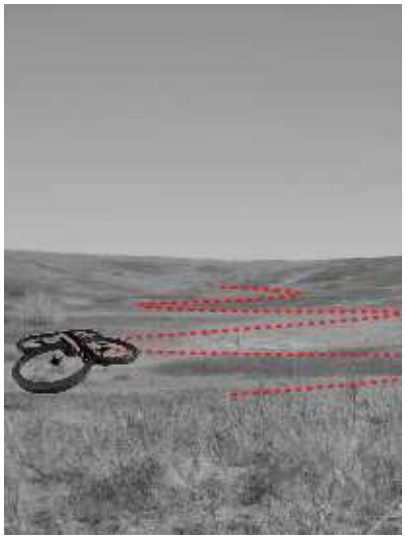
The enclosure shown in the images of Figure 3.17 provides a volume for experiments of

approximately one meter square by two meters high. Two rigid cross members and four columns provide the shape of the environment while criss-crossing cables bind the structure together. Inside the enclosure, two fillars run the height of the apparatus to contain the quad-rotor vehicle and facilitate experimental body inertia estimation. The enclosure allows freedom in measured steps to provide a safe observation environment and preserve the integrity of the AR.Drone 2.0 vehicle throughout testing.

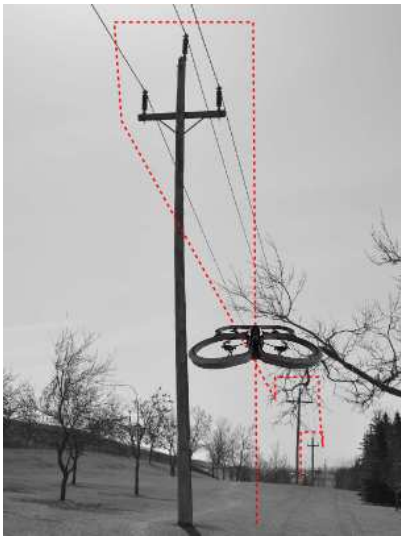
First, the enclosure can accommodate a scale for thrust measurements and a bifillar pendulum to experimentally determine moment of inertia values allowing an accurate model to be created for simulation. Once a control system is tuned in simulation, the AR.Drone 2.0 quad-rotor can be placed on a spherical surface to test stabilization without the high rotor speeds required for hovering operations. Finally, the vehicle may be allowed to travel on flexible cables and even freely within the enclosure as control is refined. As performance is improved, redundancies tested, and safety confirmed, the tracks, bound, and eventually the entire enclosure can be removed from the environment. At this point, the methodology of this work, the theory upon which it is based, and the literature from which it was born, can all be tested and the effectiveness of the introspective weight voting CMAC scheme analyzed.

3.5 Test Cases

Based on a review of the literature, three distinct test cases have been developed to evaluate the performance of the introspective CMAC scheme against traditional alternatives. The baseline control will be a simple PID scheme, a CMAC scheme with leakage weight updates will also be used to demonstrate the robustness of the introspective voting weight update to bursting. In all test cases, disturbances can be added to simulate applications found in the literature. The effect of wind is noted in a number of papers, including [115], [26], and [35]. Additionally, payload is discussed explicitly in [93], and [104]. Implementation will begin with simulation to confirm safe operation in the presence of disturbances and facilitate gain



(a) Search & Rescue Application



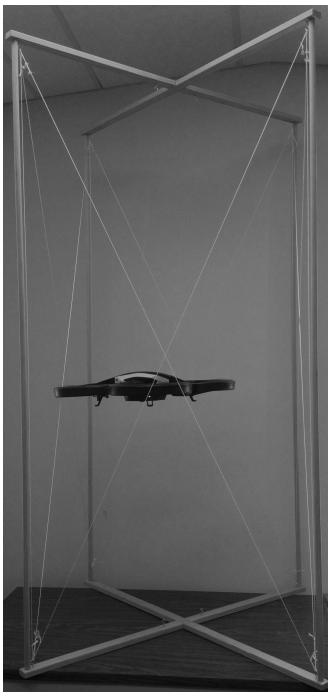
(b) Power-line Inspection Application

Figure 3.18: Quad-rotor Experimental Application Examples from the Literature

tuning.

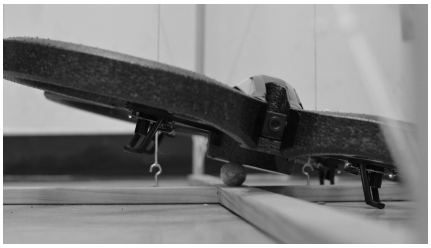
In the first simulation, the ability of a control scheme to maintain the quad-rotor in a stable hovering state will be evaluated. This application is an obvious starting point for a VTOL vehicle, as shown in references [122], [119], [69], and [23]. Next, the control schemes will be asked to alter the orientation of the vehicle in flight to specific angles using the inner loop control. This is a more specific example reminiscent of human piloted missions in mapping, such as [72], [89], and [103], and search missions including those found in [125], and [7]. A sample search pattern over terrain to be replicated in simulation is shown in Figure 3.18a for reference. The ultimate simulation will require that the control schemes accurately manoeuvre the quad-rotor vehicle through various angles provided by the outer loop control to stably track desired positions in three dimensional space. References to this rather advanced application can be found specifically in [55], [75], and [97]. A visual representation of the desired simulation behaviour is shown in Figure 3.18b.

After simulation, a similar group of tests will be performed in the experimental environment shown in the panels of Figure 3.19. The first experiment will be even less strenuous than the initial simulation; the quad-rotor will be placed on a spherical surface and com-



(a) AR.Drone 2.0 in Flight within Enclosure

(b) AR.Drone 2.0 Orientation within Enclosure



(c) AR.Drone 2.0 on Spherical Surface within Enclosure

Figure 3.19: Quad-Rotor Testing Enclosure

Symbol	Value
k_p	3
k_i	4
k_d	2

Table 3.3: PID Simulated Control Gains

manded to stabilize roll and pitch angles with limited rotor speeds. In this way, practical stability and redundancy systems can be assessed with lower kinetic energy in the system than normal flight testing to permit gain tuning and environment adjustments with less dramatic consequences than a full hovering experiment. As stability is confirmed, maximum rotor speeds can be increased to allow the vehicle to hover and eventually alter orientation using inner loop control. An outer loop tracking experiment will not be undertaken in this work due to the lack of a controllable environment of sufficient size. Still, these test cases will serve well to analyze the control schemes described.

During simulation, a number of assumptions were made about the quad-rotor vehicle to

Symbol	Value
λ	1.0
ν	1.0×10^{-2}
β	1.0×10^{3}
k_1	6.0×10^{8}
k_2	7.5×10^{7}
k_3	7.5×10^{7}
k_4	5.0×10^{4}

Table 3.4: Leakage CMAC Simulated Control Gains

Symbol	Value
λ	1.2
ν	1.0×10^{-2}
β_1	5.0×10^{4}
β_2	2.0×10^{3}
β_3	2.0×10^{3}
β_4	1.0×10^{3}
k_1	6.0×10^{8}
k_2	7.5×10^{7}
k_3	7.5×10^{7}
k_4	5.0×10^{4}

Table 3.5: Introspective CMAC Simulated Control Gains

Symbol	Value
k_p	6.0×10^{-1}
k_i	7.0×10^{-1}
k_d	3.0×10^{-2}

Table 3.6: PID AR.Drone 2.0 Experimental Control Gains

Symbol	Value
λ	15
ν	1.0×10^{-3}
β_1	5.0×10^{3}
β_2	1.0×10^{4}
β_3	1.5×10^{4}
β_4	1.0×10^{3}
k_1	9.0×10^{6}
k_2	4.5×10^{4}
k_3	5.25×10^{4}
k_4	5.0×10^{2}

Table 3.7: Introspective CMAC AR.Drone 2.0 Experimental Control Gains

simplify the model used. In the interest of facilitating implementation and testing of the control systems used in this work, the specific values used for the variables present in their respective controls are listed in Tables 3.3, 3.4, and 3.5. Inevitably, when transitioning from simulation to experimentation, control gains will require some tuning.

As a result, the experimental gains used for both linear and non-linear schemes have been adjusted, as shown in Tables 3.6 and 3.7 respectively. While gain selection was aided by the test case incorporating simulated disturbances, the AR.Drone 2.0 thrust dynamics and especially the margin of error and frequency of the sensor readings, were difficult to reproduce prior to practical testing. For both schemes, the experimental gains implemented were significantly less aggressive than those used in simulation in order to delay vehicle responses and decrease any sudden or erratic behaviour that might be detrimental to stable flight and the AR.Drone 2.0 itself.

Firstly, due to the fixed reporting rate of the AR.Drone 2.0 navigation data stream, the inner control loop runs at a lower rate than in simulation. As a result, all gains were lowered

for experimental testing. Additionally, the under-damped nature of the quad-rotor coupled with the lower control speed, required a further reduction of the derivative portion of the PID gains and a larger λ value for the CMAC to move the focus of the error signal towards the position component and away from the velocity term. These steps resolved stability issues that led to the occasional disturbingly increasing oscillatory roll and pitch movements that initially tested the redundancy components of the system and enclosure rather strenuously.

As experimental tests moved from stability, to hovering, to orientation tracking, PID gains were tuned specifically for each task. As greater freedom was granted to the quad-rotor, settling time was sacrificed for to reduce overshoot by reducing and rebalancing gains. In the case of the CMAC, the learning rate, β, and adaptation gain, μ, were used to balance the system. As these values were tuned for more advanced experiments, performance in previous, more basic tests, also improved. In this respect, the adaptive nature of the CMAC was preferable in practical tuning to the linear PID scheme.

At the conclusion of each trial, data analysis was performed by graphing the recorded actual data points against time in all six degrees of freedom. Performance was analyzed by plotting the root mean squared error between the recorded and desired positions. To further evaluate all schemes, control error in the form of the values of all four control signals and all four resulting rotor speeds has also been provided. Finally, the weight values used by the CMAC schemes were plotted to visually confirm the underlying stabilizing effect of the proposed introspective voting weight update scheme, specifically to bursting phenomenon.

In this way, the performance of the introspective weight voting scheme was evaluated against traditional methods in experiments relevant to real-world applications. The resulting data has been clearly presented to allow the formation of meaningful conclusions regarding the effect of the CMAC introspective weight voting scheme in the stable, adaptive, and robust control of a quad-rotor without the presence of bursting in experiments designed to explore increasing levels of vehicle autonomy.

Chapter 4

Results

The performance of the PID and CMAC controls described in theory will now be analyzed using the methodology previously presented to obtain results from which conclusions about stability, adaptability, robustness, and autonomy can be drawn. Testing begins with simulation, using three controls: a PID, a CMAC with leakage weight updates, and a CMAC with introspective voting weight updates. It the performance of the introspective voting weight updates is proven to be as good as, or better than, the leakage scheme and is shown to eliminate bursting, the leakage CMAC will be dropped from further simulations. Disturbances such as payload and wind will be simulated to replicate real-world scenarios found in the literature. If the controls show stable, robust performance in simulation, even with added disturbances, they will be used to control an AR.Drone 2.0 quad-rotor vehicle in experimental trials.

4.1 Simulated Trials

After describing the theory of PID and CMAC control schemes, the next step in implementation is to simulate performance in a MATLAB environment using established quad-rotor model parameters. Initially, three controls will be used in simulation. A traditional PID method will serve as a baseline, a leakage update CMAC scheme will illustrate bursting, and an introspective weight voting update will seek to prove that it is more stable, adaptive, robust, and suited to autonomy than the others.

Three distinct simulations were designed based on real-world tasks found in the literature. First, a take-off and land simulation will test the VTOL and basic stabilization of the controls. It is during this simulation that robustness and adaptability will be tested using simulated

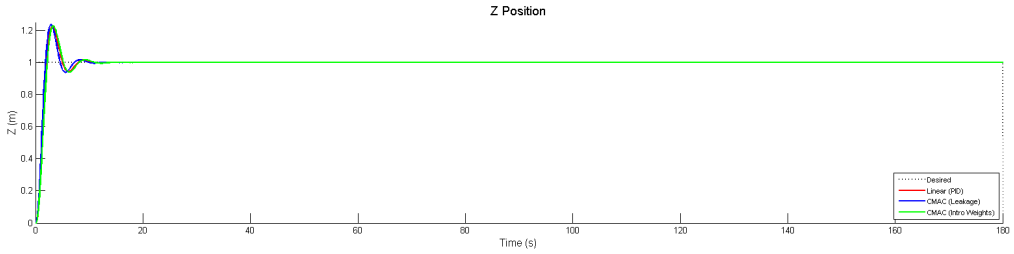


Figure 4.1: Height During Take-off

disturbances such as payload and wind. Next, an orientation simulation will be undertaken to show the ability of a control scheme to actuate the quad-rotor vehicle to specific desired rotation angles. Finally, a position tracking simulation will implement the outer loop control in a test designed to analyze the potential autonomy of each control method.

4.1.1 Stabilization

In the first simulation, the system receives a step input command of one meter in magnitude for the z position variable. This command sequence is designed to simulate vertical take-off, one of the most common tasks for a quad-rotor found in many places within the existing literature. The goal of all three control systems is to quickly, yet steadily, lift the quad-rotor to a hovering position at one meter in altitude with minimal overshoot and steady state oscillatory error.

In the first set of graphs, shown in Figure 4.1, actual and desired positions are displayed over time, for PID, leakage weight update CMAC, and introspective voting weight update schemes in red, blue and green respectively. Additionally, the minimum and maximum values of the four control signals for each of the three schemes and resulting average rotor speeds are shown in Figure 4.2, thus permitting straight forward comparisons of the three controls.

After adequate tuning, all three control schemes perform very well in a simple take-off simulation. The leakage control exhibits slightly more overshoot than the others, raising the quad-rotor model approximately 24 cm above the desired 1 m mark, compared to 23

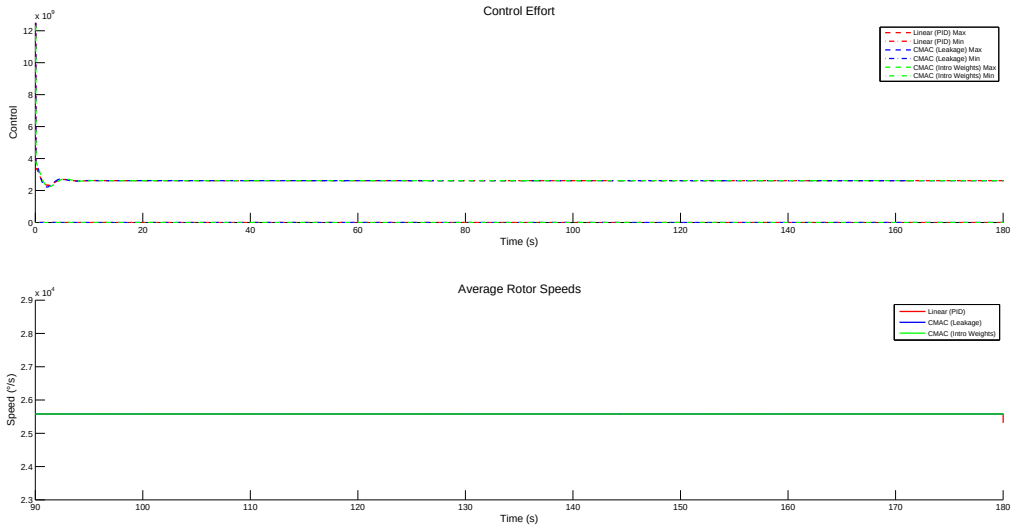


Figure 4.2: Control Effort and Rotor Speeds During Take-off

cm for the other two controls. The linear control takes approximately 5 seconds to settle and exhibits an RMS error of 10 cm over the entire 180 second simulation. All three control schemes also show similar settling times and overall RMS. The leakage weight update scheme marginally out performs the others with regard to settling time and overall error. In terms of control effort, all three controls are very similar in performance. The Leakage weight update scheme momentarily expends the highest control effort to initially lift the vehicle, the Introspective control reaches the lowest control value, but all three schemes perform nearly identically over the three minute test. The PID control requests an average rotor speed of 25,570 RPM over the course of the simulation, Leakage asks for 25,558 RPM, and the Introspective scheme splits the two with an average speed of 25,565 RPM. Given this most basic task and sufficient tuning of all three controls, especially PID, there is little difference in performance. However, environmental disturbances will show the robustness of each control scheme.

In the next trial, a payload is added to the model, as defined in Table 4.1, to simulate a sensory or material package that may be applied to the quad-rotor to perform a specific task. From the literature, it is known that there exists a desire for a quad-rotor to carry

Disturbance	Value	Units
Payload	0.042	kg
Wind	15 ± 5	km/hr

Table 4.1: Simulated Disturbance Inputs

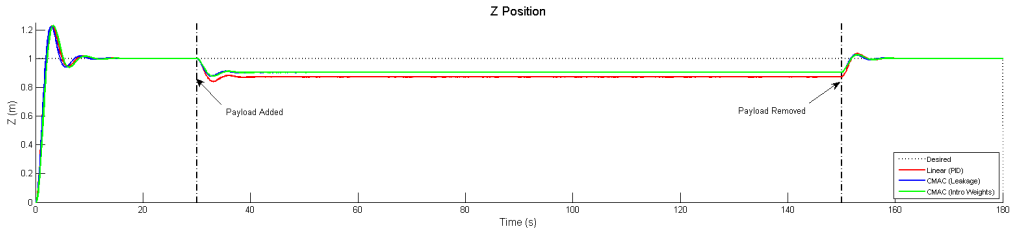


Figure 4.3: Height During Loaded Take-off

a variety of different items from sophisticated cameras and LIDAR, to manipulators and grippers, even first aid supplies for search and rescue missions [120] [117] [7]. These payloads may differ from one application or task to the next, and downtime due to calibration will likely be costly, so it is important that the control is able to adapt quickly.

With the introduction of a disturbance to the system, the ability of the three controls to adapt to unmodelled dynamics is more evident in the error measurements, as shown in Figure 4.3. After the application of a simulated 42 gram payload, representing 10% of the total vehicle weight, the PID control could only manage to lift the quad-rotor to 0.87 m rather than the requested height of 1 m. This resulted in an increased overall RMS error of 14 cm. The non-linear controls were able to better adapt to the payload disturbances, falling to only 0.90 m, and providing an RMS error of just 13 and 12 cm over the entire simulation.

The control effort and average rotor speed graphs shown in Figure 4.4 demonstrate that the PID control is taxed slightly more than the CMAC schemes. Initially, it exerts the most force on the vehicle, and retrains a strong influence. The PID scheme requires an average rotor speed of 25,931 RPM compared to identical averages of 25,898 RPM for both the Leakage and Introspective weight update schemes. Again, with careful tuning the PID may be able to perform well in an operation window that lies well away from any non-linear

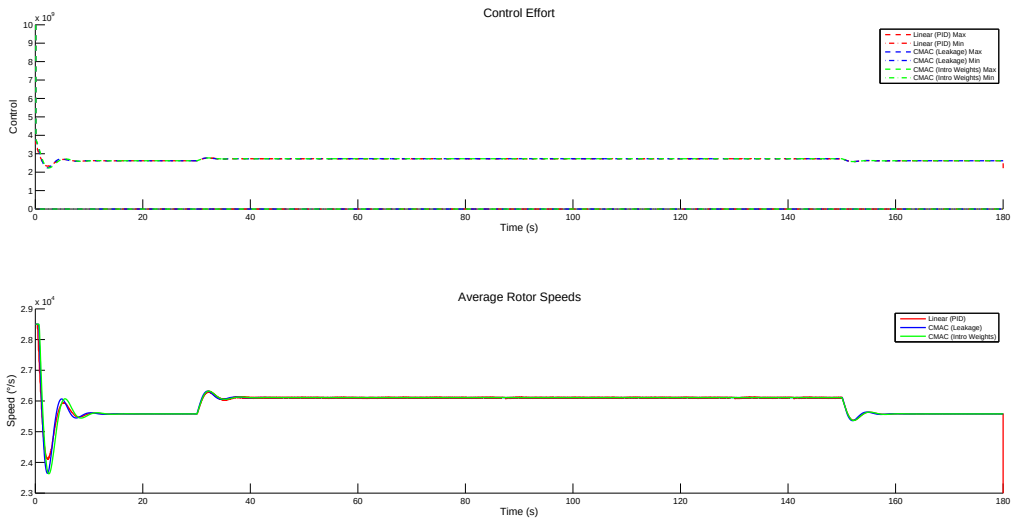


Figure 4.4: Control Effort and Rotor Speeds During Loaded Take-off

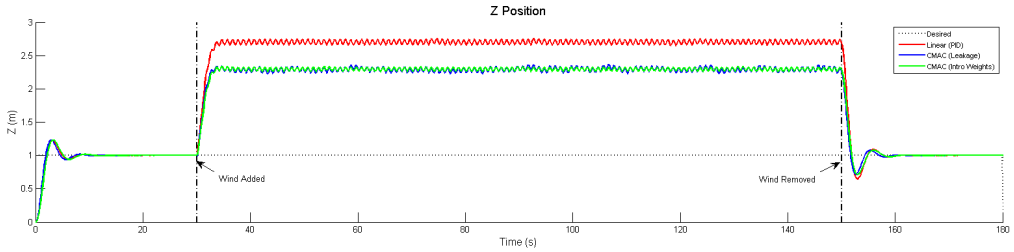


Figure 4.5: Height During Windy Take-off

regions, but only the CMAC schemes can be taught to adapt to changes in model dynamics and function effectively over the entire operational window.

Finally, a last take-off simulation was performed with the same control schemes, without a payload, but with the addition of a sinusoidal wind disturbance, as defined in Table 4.1. This simulates an up-draft condition which may be expected in a real-world scenario. The sinusoidal disturbance is a worst case scenario for testing the robustness as it can excite underlying unmodeled dynamics in the system. And, as shown in the PID control, these disturbances can lead to unexpected and sometimes unstable behaviour. In this simulation, both non-linear control schemes begin to markedly outperform the linear PID.

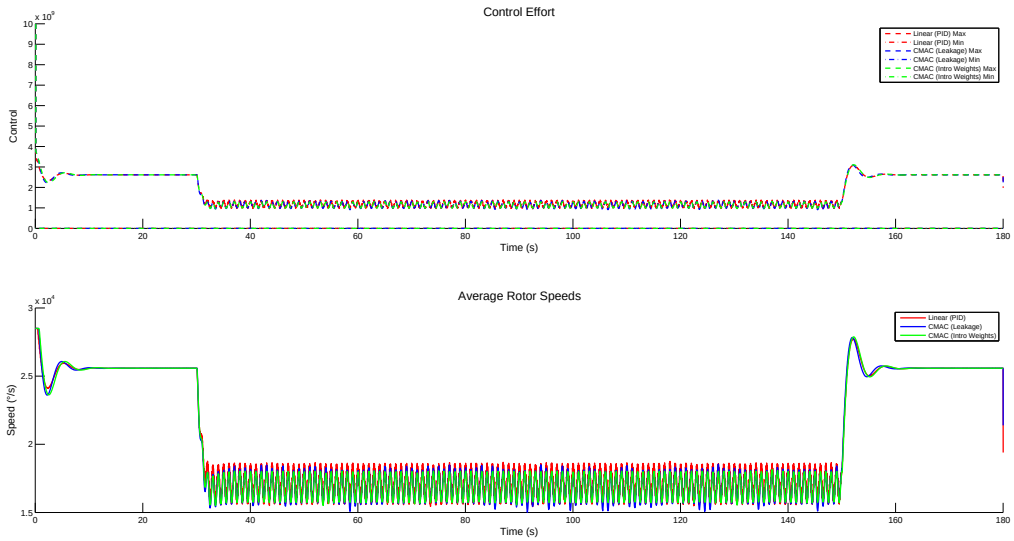


Figure 4.6: Control Effort and Rotor Speeds During Windy Take-off

In slightly windy conditions, approximately 15 km/hr with a sinusoidal gust of 5 km/hr, all three control schemes were significantly challenged, as evident in the difference between actual and desired positions shown in Figure 4.5. The inability of the PID control to adapt to these changing conditions resulted in an RMS error of 1.42 m over the course of the simulation, with a maximum height of 1.7 m above the goal height. The CMAC schemes also could not keep the quad-rotor from rising, but in their case only 1.3 m above the 1 m goal. As a result, these adaptive schemes demonstrate an RMS error of just 1.07 m under the most difficult take-off simulation conditions.

The ability and speed with which the control schemes adapt is further evident in the control effort required for all three, as shown in Figure 4.6. The Introspective scheme showed the greatest variation between lowest and highest control forces over the test, while the PID demonstrated the least variation and an inability to adapt. As a result, the PID required the highest average rotor speed at 19,693 RPM, the Introspective scheme next at 19,683 RPM, and finally the Leakage scheme at just 19,603 RPM.

Additionally, the maximum single weight and maximum average weight values for the leakage and introspective weight voting schemes are shown in Figure 4.7 to illustrate robust-

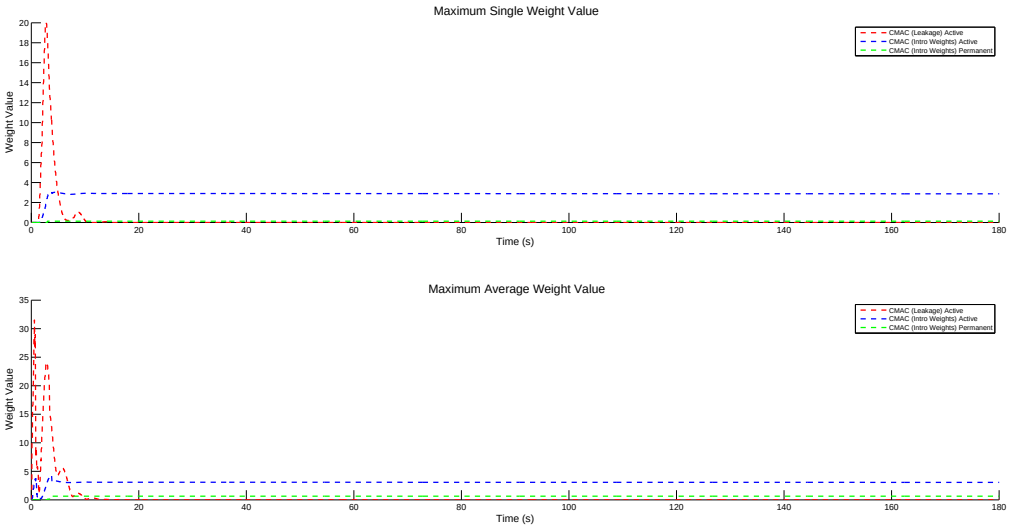


Figure 4.7: Introspective Weights During Take-off (Active and Permanent)

ness bursting phenomenon described earlier. During the first, take-off simulation, leakage and active introspective weights change significantly as the vehicle first leaves the ground, but stabilize for the remainder of the flight. The leakage scheme exhibits the highest initial weight values, with single weight values reaching a maximum value of 31.5. The introspective scheme exhibits maximum single weight values of just 5.3 and 4.2 for active and permanent weights respectively. For the introspective scheme, active weights are always slightly larger than permanent ones as voting stops updates once changes no longer have a significant impact on control.

When unmodeled dynamics, such as payload weight are added to the system during the second simulation, weight changes are more pronounced, especially in the active weights, as shown in Figure 4.8. Weights grow to account for unknown dynamics present in the system, especially the introspective active weight. The maximum single weight value for the leakage update scheme was only 18 for this trial, with maximum single values of 3.9 and 3.1 recorded for the active and permanent portions of the introspective voting weight respectively. The maximum average weight plot shows that this trial results in the activation of a multiple hypercube cells in the CMAC, and as a result, more weights are updated from their initial nil

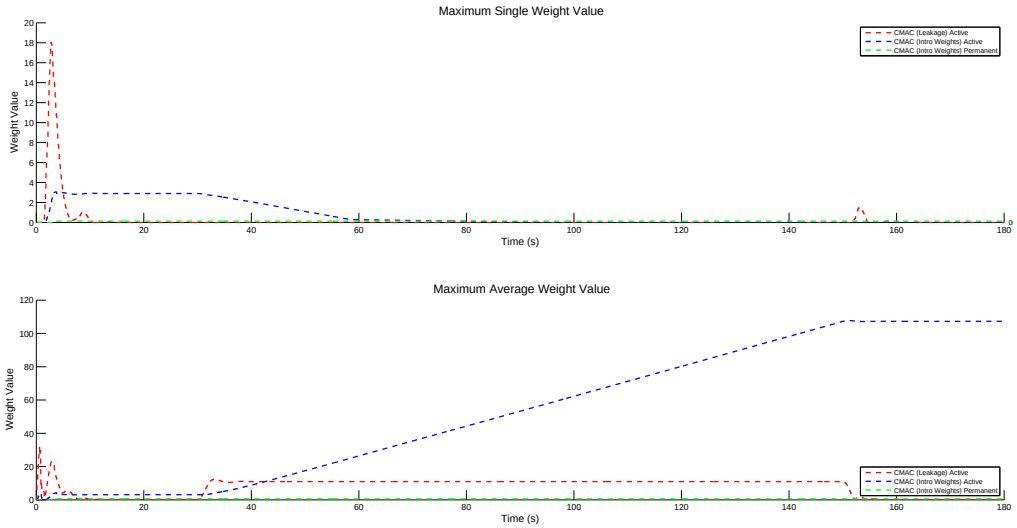


Figure 4.8: Introspective Weights During Loaded Take-off (Active and Permanent)

value. Activity in the maximum average weight plot begins in earnest at 30 seconds, when the payload is added, and continues until it is removed at the 150 second mark. However, the voting algorithm ensures that permanent weights are only updated when a marked increase in performance is observed in active weights. As a result, permanent weight growth is shown to be bounded and stable.

The most dramatic weight changes are observed when the sinusoidal wind disturbance is added to the simulation, as shown in Figure 4.9. For this trial, single maximum values of 141, 1226, and 268 were observed for the leakage, active, and permanent weights respectively. Interestingly, both single value and average value maximums increased over the course of the simulation. This behaviour points to the growth of single weight values as an oscillatory wind disturbance buffets the system from the 30 second mark to the 150 second point. In this case, active weights of the introspective scheme continue to grow as neighbouring cells are activated and begin to fight for dominance. Without persistent excitation, there is little change in the input vector resulting in the activation of only a small portion of available CMAC cells. However, voting ensures that permanent weights are far more stable in times of instability, yet flexible during changes such as the introduction of the wind disturbance.

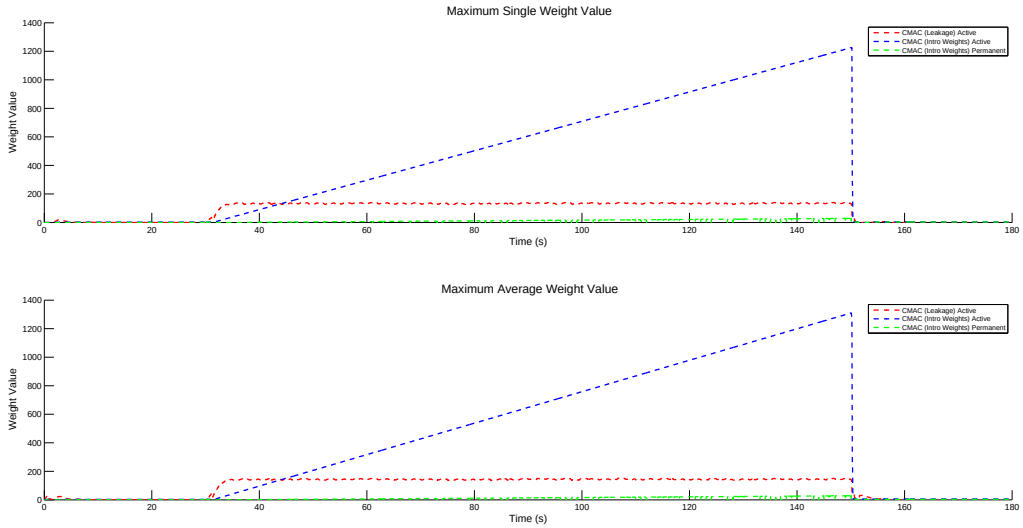


Figure 4.9: Introspective Weights During Windy Take-off (Active and Permanent)

Symbol	Value
λ	1
ν	1.0×10^{-5}
β_1	1.0×10^{4}
β_2	2.0×10^{3}
β_3	2.0×10^{3}
β_4	1.0×10^{3}
k_1	2.5×10^{10}
k_2	7.5×10^{7}
k_3	7.5×10^{7}
k_4	5.0×10^{4}

Table 4.2: Aggressive CMAC Control Gains Used for Bursting Simulation

4.1.2 Bursting

When the leakage update CMAC system is tuned aggressively, as described in Table 4.2, weight values grow dramatically towards the stable system bound, before being stopped by the update law. As a result, the system error decreases over this period, before suddenly increasing as weight updates are brought to an abrupt halt. As the error increases, the weight values move away from their bound and updates can commence to decrease the error once more, as shown in Figure 4.10. This burst cycle may also be excited by oscillatory

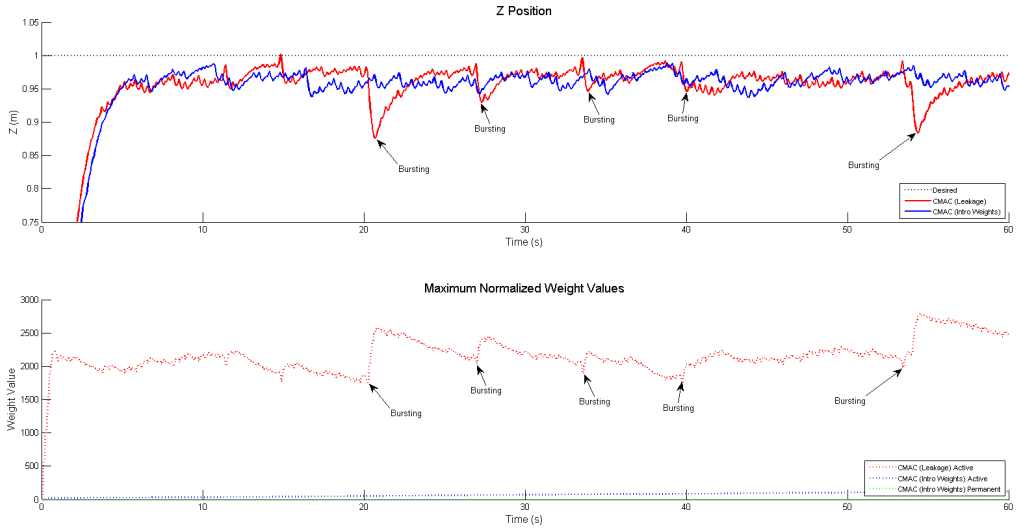


Figure 4.10: Z Position and Maximum Weight Values Showing Bursting Phenomenon in Simulation

disturbances, such as wind, and is detrimental to overall system performance. While this leakage update for the CMAC is more adaptive than the PID and offered greater autonomy through less human control input, bursting does negatively affect the performance of this traditional weight update scheme.

4.1.3 Controlled Flight

In the next simulation, desired roll and pitch rotations are varied in incremental steps of 2.5 degrees. These inputs are combined to generate a sweeping search pattern that a pilot operator might use to survey and map a region or locate a target of interest in a search and rescue mission [33] [125]. For this trial, the baseline PID control is contrasted against only the introspective voting CMAC. The introspective voting weight update scheme has been shown to perform as well or better than a standard leakage scheme, with the exception of improved robustness to bursting, as shown in Figure 4.10. The x and y positions resulting from the same input to both the PID control and Introspective Weight Voting scheme are shown in Figure 4.11.

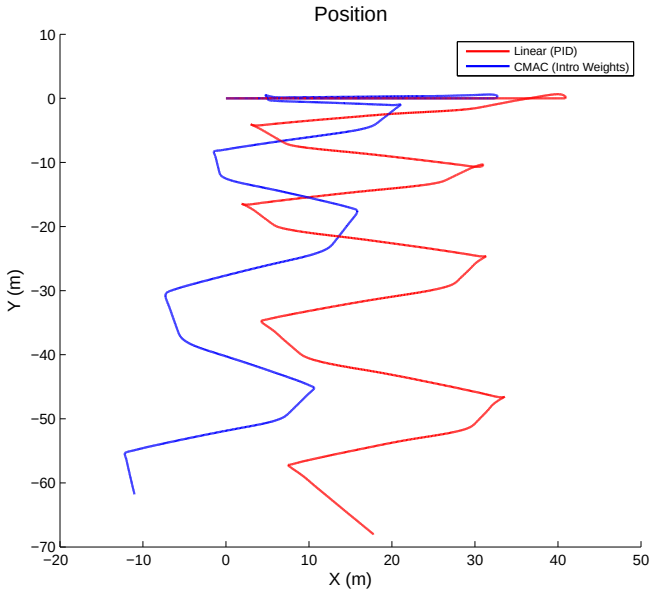


Figure 4.11: Position During Search

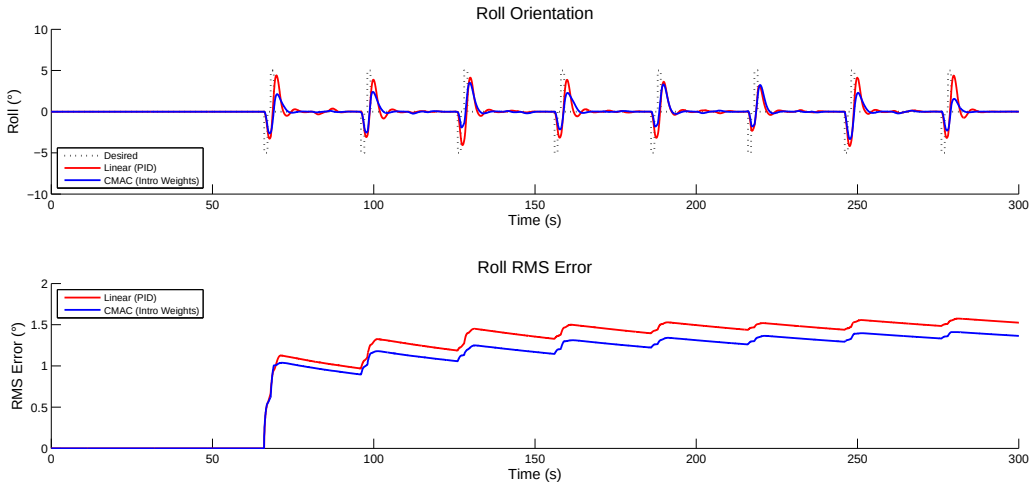


Figure 4.12: Roll During Search

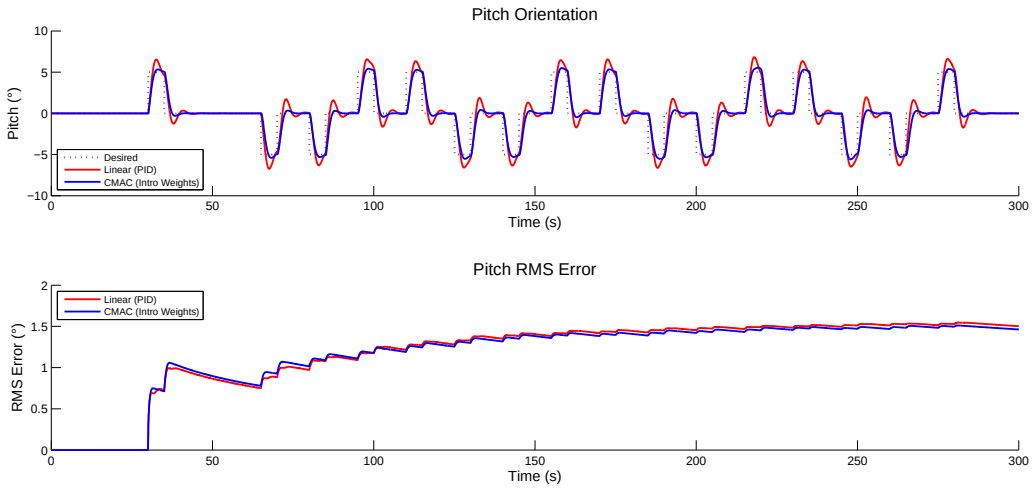


Figure 4.13: Pitch During Search

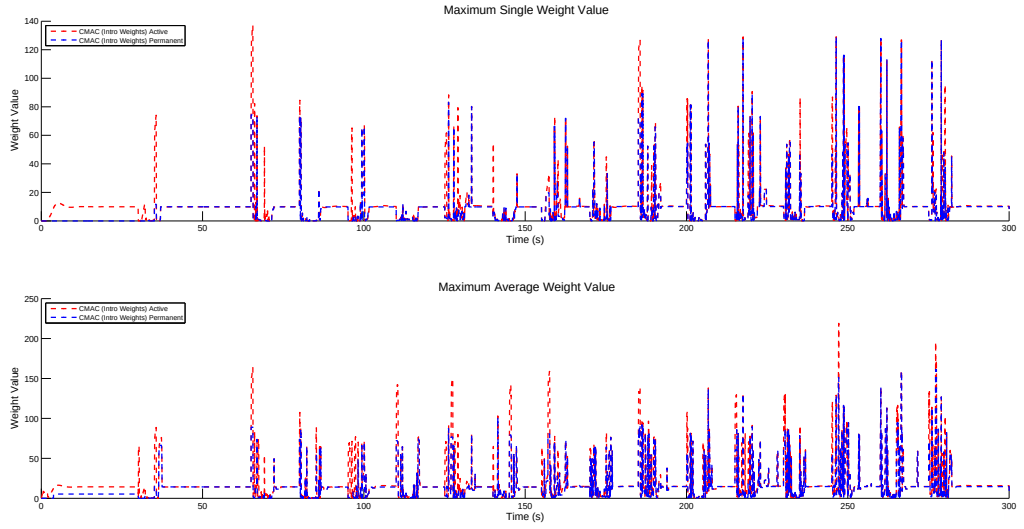


Figure 4.14: Active and Permanent Introspective Weight Stability During Search Simulation

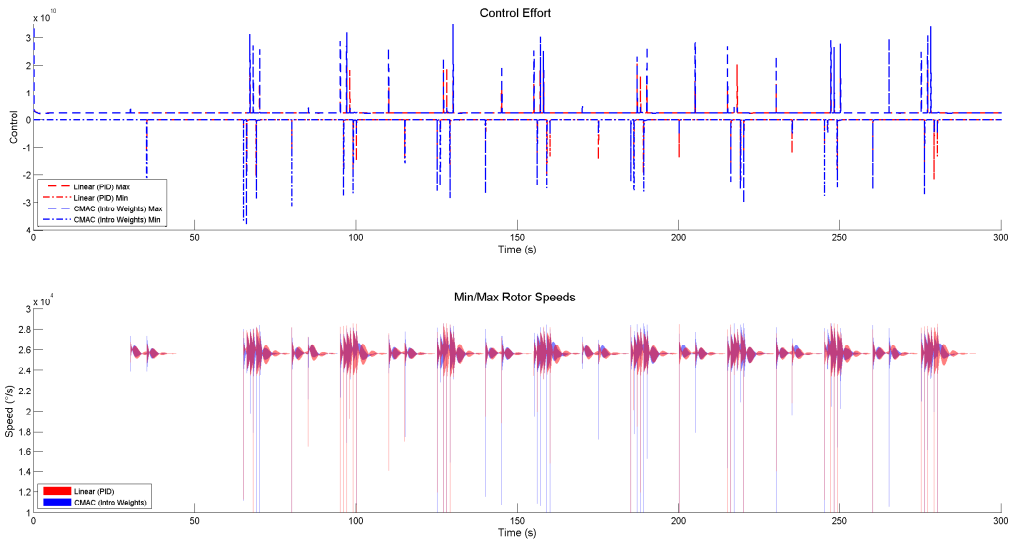


Figure 4.15: Control Effort and Rotor Speed During Search

Furthermore, roll and pitch angles are investigated to ensure the control scheme is stabilizing the quad-rotor to the desired attitude. For this simulation, signal error is not as straight forward, so a running RMS error plot is provided for each input. Response to roll commands are shown for PID and Introspective controls in Figure 4.12, with pitch responsiveness for the same controls shown in Figure 4.13. As a baseline, the PID control exhibits cumulative RMS errors of 1.52 and 1.50 degrees for roll and pitch angles respectively. However, the introspective CMAC scheme provided total roll and pitch RMS errors of only 1.36 and 1.46 degrees respectively. This slight error discrepancy is likely a result of the slow response time of the PID control. However, the introspective voting scheme has held its own against the traditional PID control. And, the weight values shown in Figure 4.14 confirm that there is no update behaviour indicative of bursting in the CMAC system. Within the relatively narrow flight window provided by this practical search test, neither control system is extensively taxed, but the performance of the voting CMAC has been shown to be as good as, even slightly better than, that of the PID.

Finally, the control effort required for each scheme is shown in Figure 4.15, using the minimum and maximum control signal values as well as the average rotor speed. Again, the

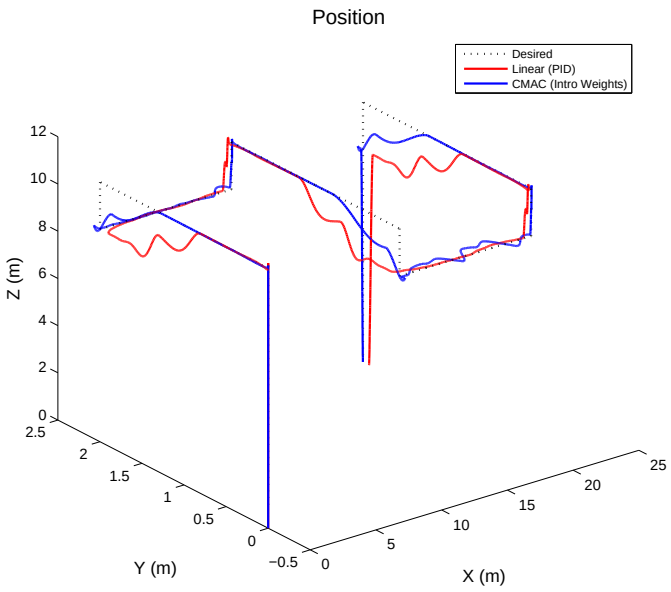


Figure 4.16: Position During Inspection

Introspective scheme preforms very similarly to the baseline PID. The PID again exhibits a higher average rotor speed of 25,503 RPM compared to 25,487 RPM for the Introspective scheme. The tuning of the introspective voting CMAC and its ability to learn input to output associations over time allow it to slightly outperform the PID control.

4.1.4 Position Tracking

In the last simulation, a position tracking loop is added outside the previously tested stabilization loop. This higher level control adds autonomy to the quad-rotor by allowing desired locations to be specified in three dimensional space for the vehicle to track. The outer loop runs at a slower rate than the inner, and plans a path which is then passed into the main control loop. This application is modelled after a more complex autonomous tasks found in the literature, such as power-line inspection and target tracking [75] [42]. Rather than step inputs from a pilot, a series of way-points have been selected for closer inspection. The outer loop requests attitude and position adjustments from the inner control to achieve these points and accomplish a rather complex autonomous task.

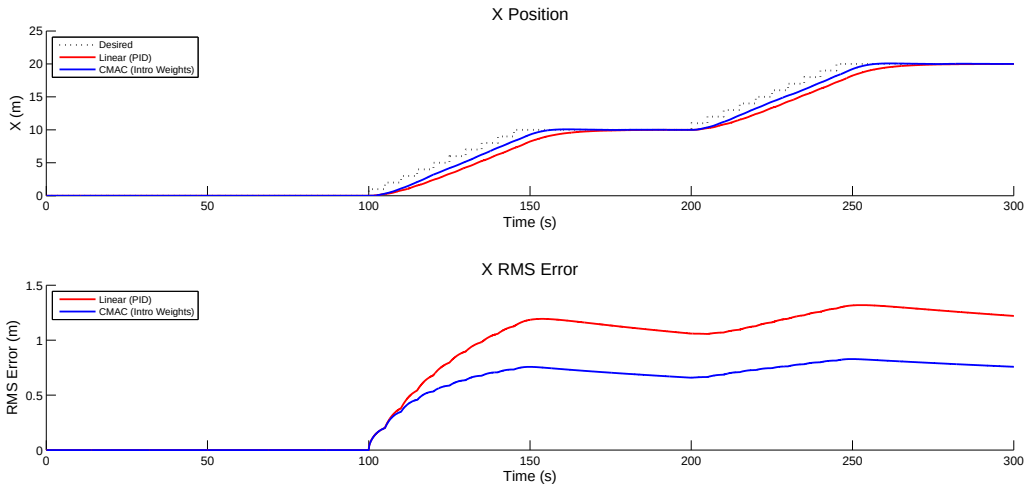


Figure 4.17: X Position During Inspection

Position response of the PID and introspective CMAC to the tracking control along x, y, and z axes is shown in Figure 4.16. Yaw movements are not required directly for this task, but may be implemented in the real-world by direct operator control. An RMS error measure is used once more to highlight the effectiveness of proposed control scheme compared to a more traditional PID solution.

This simulation showcases the x and y virtual controls, as well as the direct altitude control. The x, y, and z response of both the baseline PID control and introspective voting CMAC are overlaid in Figures 4.17, 4.18, and 4.19 respectively. In the x direction, the PID exhibited an overall RMS error of 1.21m, while the introspective scheme showed only a 0.75m error. In the y plane, the two controls were closer in performance, with the PID providing an error of 0.47m to the 0.36 of the introspective scheme. Finally, in the z direction there was little to choose between the PID and introspective schemes with errors of 0.60m and 0.63m respectively.

The control effort and rotor speed graphs for this simulation, shown in Figure 4.20, expand upon the story presented by the x, y, and z response plots. The introspective voting scheme adapts faster and with greater conviction than the PID, which becomes clear when

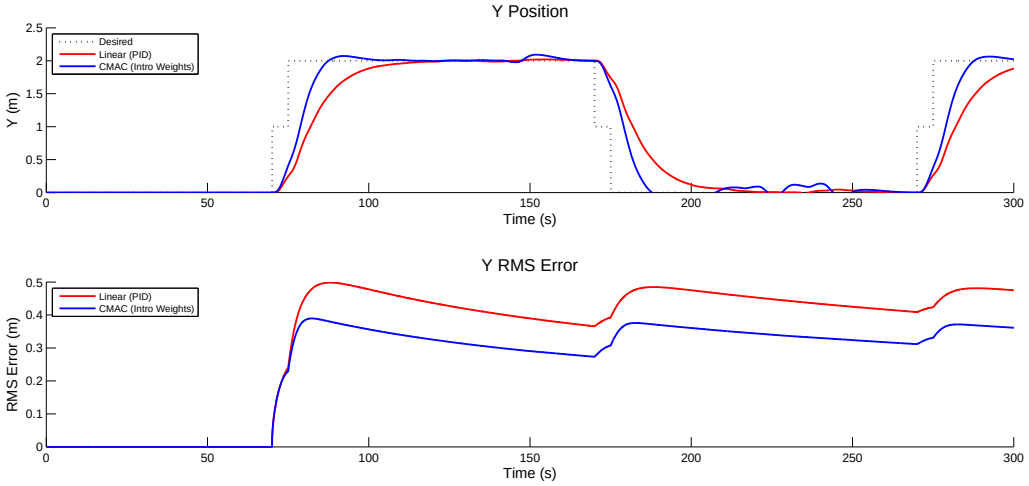


Figure 4.18: Y Position During Inspection

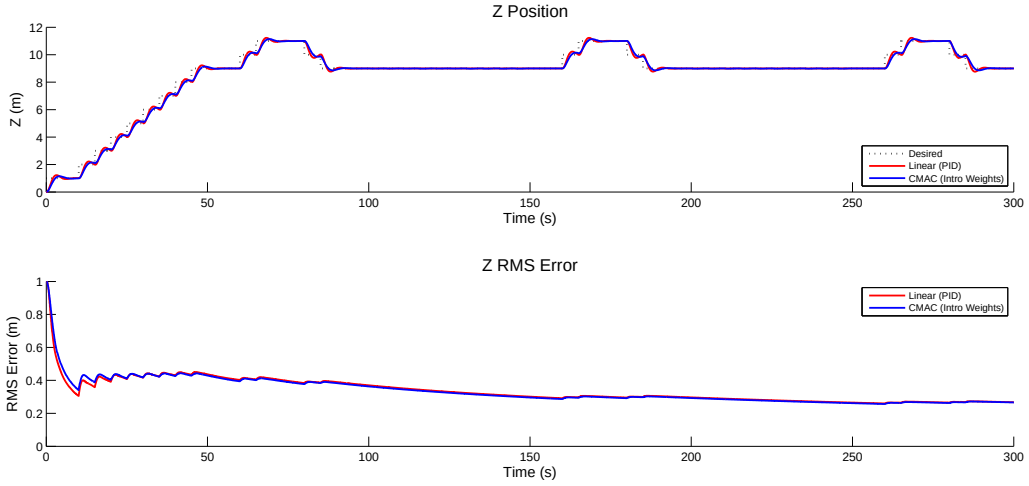


Figure 4.19: Z Position During Inspection

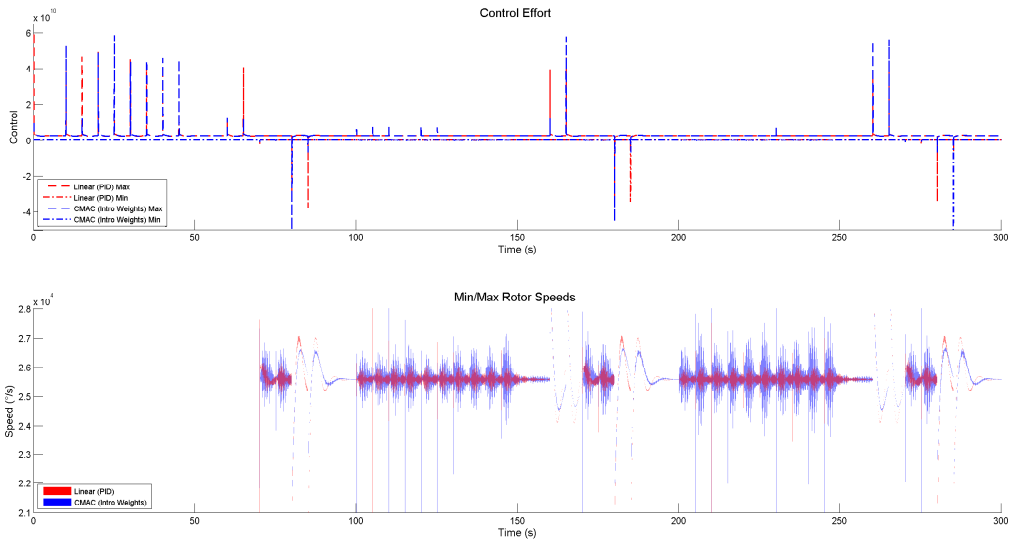


Figure 4.20: Control Effort and Rotor Speed During Inspection

rotor speeds are investigated. The average rotor speed of the introspective CMAC is higher, at 25,458 RPM compared to a slightly lower 25,450 RPM. In this case, a lower speed is not necessarily the result of a lower control effort on the part of the PID scheme, but an inability to keep up with the rapid position requests of the tracking application.

Again, single weight and average weight maximum values for both active and permanent weights are shown in Figure 4.21 to confirm robustness to bursting. Overall, the introspective weight voting scheme out performs the PID control in this more advanced simulated tracking application by being able to adapt to unmodeled vehicle dynamics and external disturbances.

4.2 Experimental Trials

While both the baseline linear PID control and introspective CMAC scheme provided good performance in simulation, experimental application presents a significantly greater array of challenges. Disturbances such as wind and payload were modelled during simulation, but advanced dynamics such as ground effects, blade flap, and variable environmental conditions were not taken into consideration. Furthermore, practical considerations such as commu-

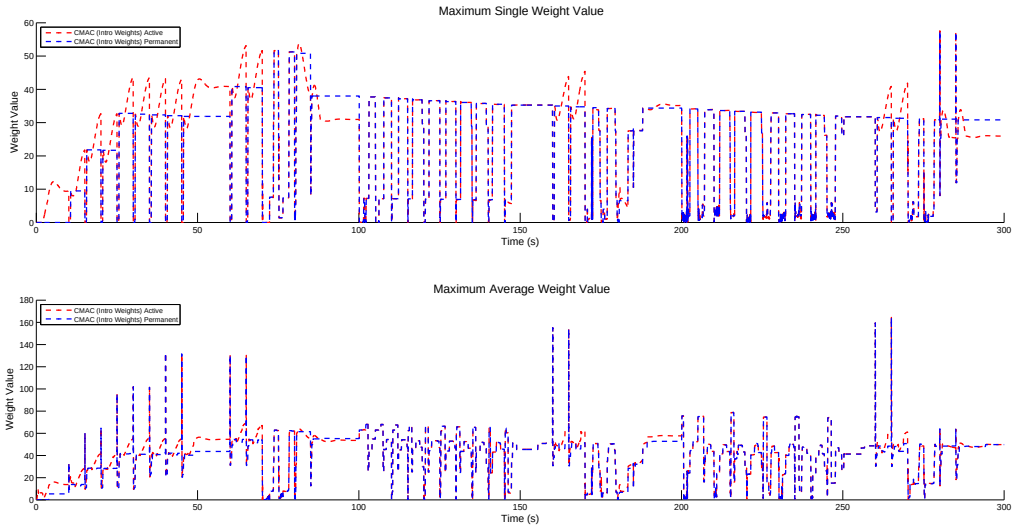


Figure 4.21: Introspective Weights During Inspection (Active and Permanent)

nication speed and sensor drift, which were not modelled will certainly affect experimental results. For example, simply switching from a laptop system, first used for simulation, to an Intel quad-core i5 desktop machine running at 3.6GHz with 8GB of RAM and dual GTX-650 Ti graphic cards for real-time model rendering made a huge difference in experimental control performance. This desktop machine was eventually used for all simulations and experiments, more than doubling the control loop frequency to over 30Hz. Nonetheless, the unmodelled dynamics of the quad-rotor, and environmental factors such as surface friction of the spherical surface used to test stabilization and the bifilar restraints used to safely confine the vehicle add disturbances to the system. As a result, errors noted in the experimental section are often higher, but by comparing the introspective voting scheme against a PID control a fair evaluation can be performed.

Although the tasks asked of the control systems are very similar to those undertaken in simulation, experimental application adds some wrinkles. First, the AR.Drone 2.0 has two operational modes, direct and indirect rotor control. The former allows raw PWM values to be sent to each of its four rotors for very intimate control. However, height data is not available. The later, indirect, mode provides all measurements, but accepts only vertical

translation and yaw rotational speeds as well as roll and pitch angles as input. Second, care must be taken when tuning any control applied to the AR.Drone 2.0 as, unlike in simulation, mistakes can lead to a very real crash. As a result, control testing began by resting the quadrotor on a sphere and tuning roll and pitch response in manual control mode. Only then was the AR.Drone 2.0 commanded to hover.

During experimental trials, data was recorded in real-time and analyzed off-line in much the same manner as described in the simulation process. However, this data does not provide details of system non-linearities and unmodeled environmental disturbances, rather the magnitude of their effect is evident in the more widely differing results of each control system. Testing followed the same pattern previously discussed: first take-off, then inner loop control. Again, weight updates are observed to confirm the absence of dramatic growth leading to bursting, using both maximum single weight and average weight values. Understanding the performance of each control system, inside and out, ultimate leads to a clearer indication of the potential impact of each.

4.2.1 Stabilization

In the first experimental trial, the stabilizing effect of a simple PID and the introspective CMAC scheme is analyzed using direct rotor control of the AR Drone 2.0 vehicle. To understand initial performance while minimizing the negative effects of an untuned control scheme, rotor speeds were manually limited to restrain the AR.Drone 2.0 to an earthly bound. The vehicle was placed upon a spherical surface with wedges supporting the vehicle at rest, resulting in initial roll and pitch angles of approximately ten degrees. For this trial, the AR.Drone 2.0 was commanded to stabilize itself to roll and pitch angles of zero degrees while height and yaw control inputs were fixed. This allowed the PID baseline and introspective CMAC schemes to not only be tuned, but compared before the vehicle even took proper flight.

In this first experimental task, the baseline PID and introspective CMAC scheme perform

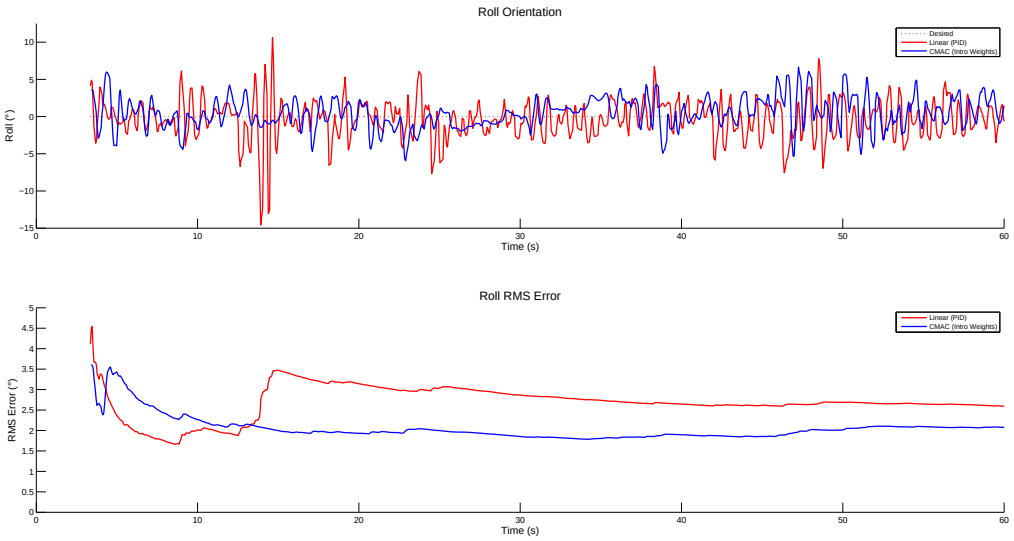


Figure 4.22: AR.Drone 2.0 Roll Angle During Stabilization Experiment

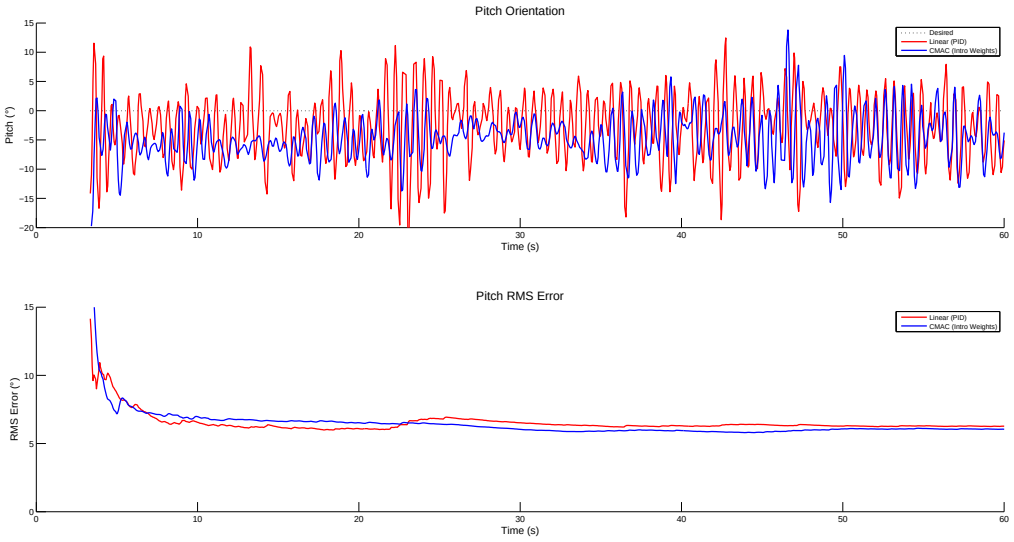


Figure 4.23: AR.Drone 2.0 Pitch Angle During Stabilization Experiment

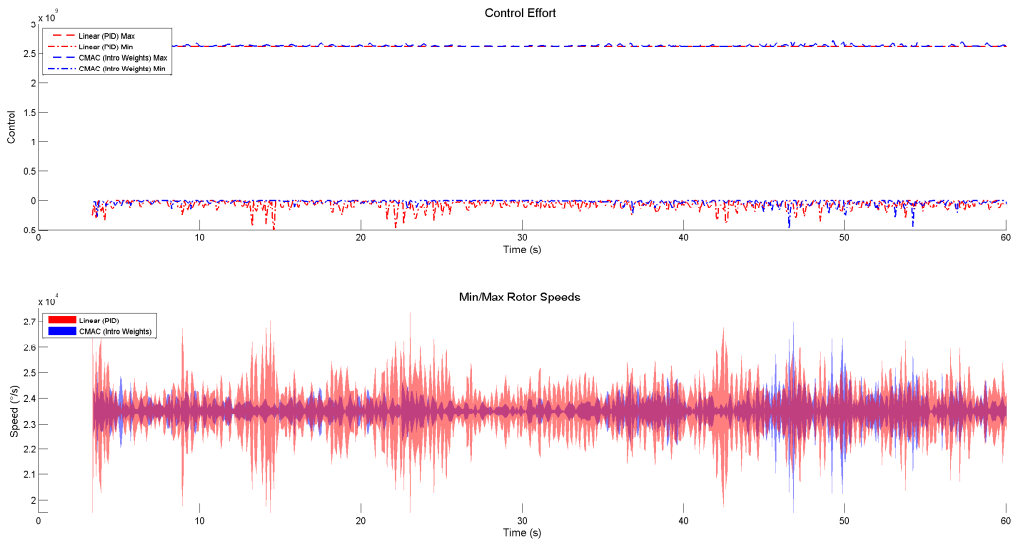


Figure 4.24: Control Effort and Rotor Speed During Stabilization Experiment

with greater error than in simulation. For the PID test, the initial roll angle was 9.8 degrees, for the CMAC it was 9.3 degrees. The PID scheme exhibited an RMS error of 0.9 degrees over the entire simulation, while the CMAC provides a deviation of 0.8 degrees, as shown in Figure 4.22. The results in the pitch plane are similar, as shown in Figure 4.23. The initial pitch angle was measured at 10.0 degrees for the PID control and 10.1 degrees for the introspective scheme. The CMAC provided an overall error of 4.7 degrees and the PID varied by an RMS value of 4.8 degrees. Part of this decreased performance is the result of drift errors in roll and pitch measurements which were observed to be as high as 1 degree, as shown in Figure 3.13. Yet, stable performance is achieved by both control schemes, within 5 seconds for the PID scheme and 10 seconds for the introspective voting CMAC.

The PID scheme immediately exerted a significant force, while the introspective voting scheme took some time to learn the dynamics of the system before settling. As a result, while the introspective CMAC scheme exhibited a much smaller range of rotor speeds, between 19,606 and 21,356 RPM, whereas the PID scheme requested a greater range of speeds from the AR.Drone 2.0, from 14,522 to 25,091 RPM. The speeds required for the introspective CMAC are then less taxing on the quad-rotor, while the overall error result is very similar

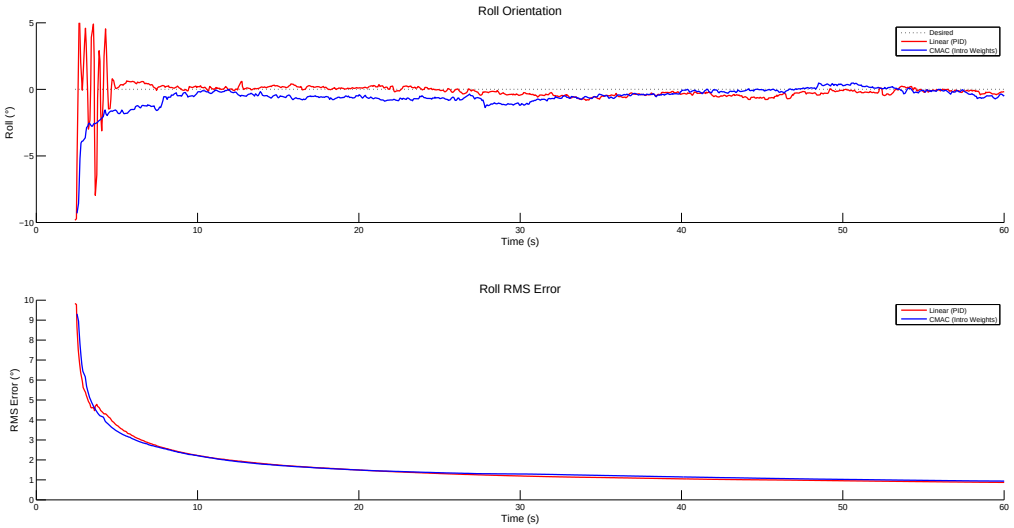


Figure 4.25: AR.Drone 2.0 Roll Angle During Hover Experiment

to that of the PID. Furthermore, the success of this test for both controls provides a basis upon which more advanced applications can be explored.

4.2.2 Hover

In the next experimental application, the AR.Drone 2.0 was manually accelerated to a hovering position before a control system took control to stabilize the roll and pitch angles of the vehicle. Again, the height control was fixed and the yaw control disabled. Only the ability of the control to stabilize roll and pitch angles was investigated. In addition to the measurement drift noted in the previous stabilization trials, this test was performed within the enclosure described earlier, with the most rigid elastic restraints in place.

Large variations in orientation were retarded by the filars of the enclosure through elastic forces which were not modelled in control. Regardless, during the experimental take-off and hover task the PID baseline control exhibited a combined roll RMS error of just 2.6 degrees, as shown in Figure 4.25. The introspective CMAC scheme achieved a slightly better result, with an error of only 2.1 degrees. After additional tuning, the PID control did exhibit some stabilizing effect on the AR Drone 2.0 while hovering. However, all of the work done to

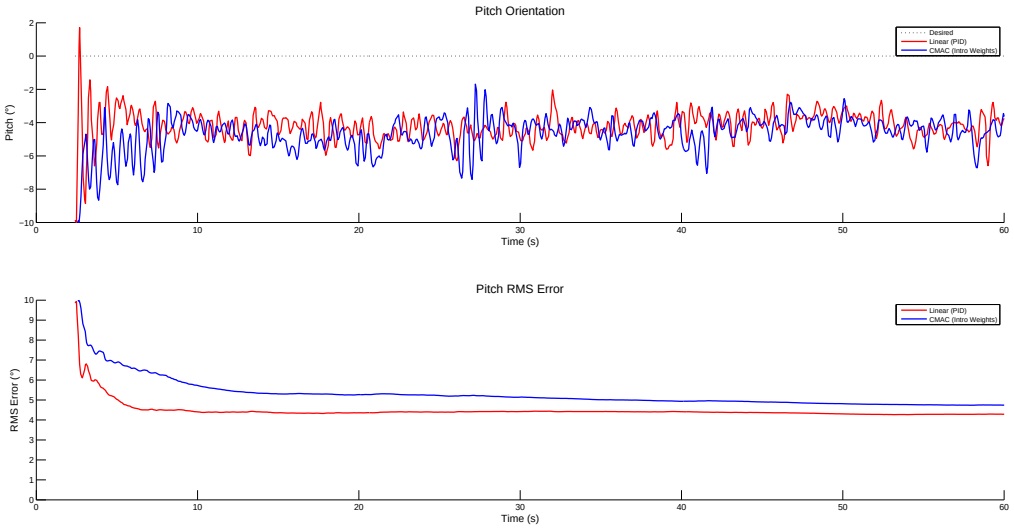


Figure 4.26: AR.Drone 2.0 Pitch Angle During Hover Experiment

calculate the correct gains to achieve this effect will be for naught if vehicle or environmental variables change. This effort may be better placed in tuning an adaptive system which should provide a consistent response over a variety of different conditions.

The introspective weight voting scheme performed with similar results when stabilizing pitch as well with an RMS error of only 6.1 degrees compared to 6.2 degrees for the PID control, as shown in Figure 4.26. The introspective CMAC scheme exhibits performance that is as good as, if not slightly better than, that the PID in this simple stabilization experiment. This result mirrors that of the performance noted in the first, most basic, simulation trials, before the adaptive introspective CMAC began to show marked performance improvements over the linear PID in more advanced tasks.

The PID scheme shows a much higher control effort at the beginning of the test then seems to rest. The introspective weight voting scheme starts slowly, but begins to take greater control in the later portion of the test as it learns the control dynamics of the vehicle. The PID scheme again requests a wider range of rotor speeds, from 17,945 to 27972 RPM, as opposed to the introspective voting scheme which required just 19,137 to 27,171 RPM, as shown in Figure 4.27. The PID scheme shows significant control force fluctuations

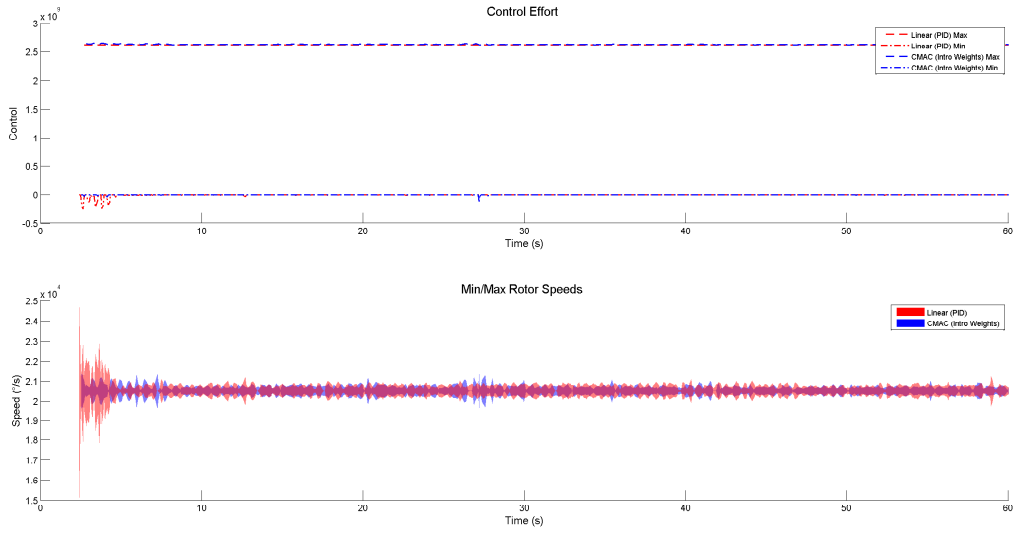


Figure 4.27: Control Effort and Rotor Speed During Hover Experiment

contributing to this wider range of speed and more erratic initial vehicle movement. Again, the PID is stressing the vehicle more during operation, while providing the same, if not slightly worse performance than the introspective CMAC.

As in simulation, the adaptive weights used by this scheme have been tracked in real time and featured in Figure 4.28 to show the underlying stability of the system and ensure there is no danger of bursting. Having achieved satisfactory performance in stabilizing the AR Drone 2.0 roll and pitch angles via direct rotor control, the PID and adaptive systems can be tasked with the same controlled flight challenge presented in the simulation section.

4.2.3 Controlled Flight

In the next experiment, the AR.Drone 2.0 was commanded to follow roll and pitch angles designed in simulation to carve out a search pattern. This application requires that the control not only provides stable flight, but also the ability to quickly and accurately alter the orientation of the vehicle. As a result, the gains from both the PID scheme and the proposed CMAC required additional tuning. However, where tuning the PID scheme merely shifts the linear control to better fit the current non-linear region of the quad-rotor operating

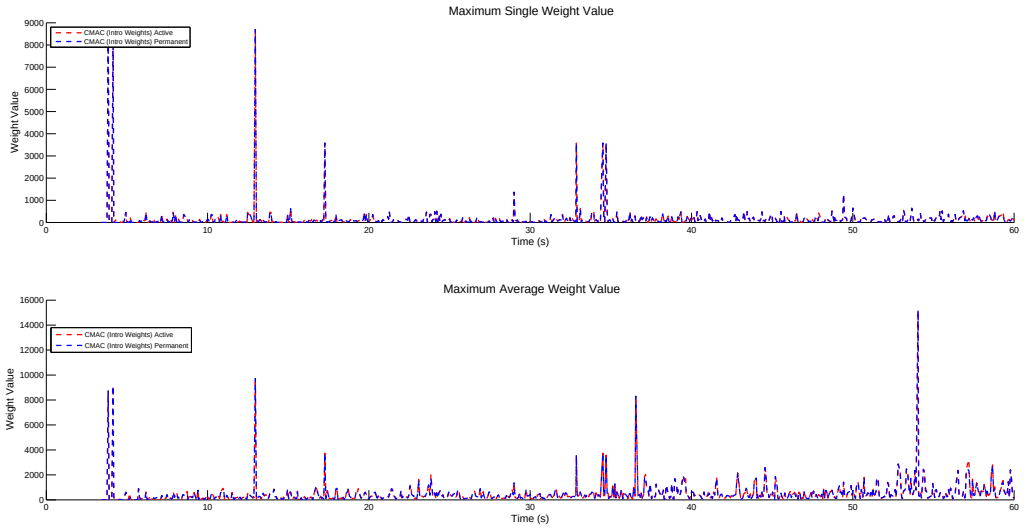


Figure 4.28: Introspective Weights During Practical Hover

window, tuning the CMAC improves performance for all regions of operation. That is to say, meticulously adjusting the PID gains to suit this application may adversely affect previous trials, but refining the CMAC gains improved system performance in all applications. Adaptation to this task, and those that follow, strengthens the introspective weight voting scheme, compared to the baseline PID control.

For this application, roll and pitch commands were issued to orientate the AR.Drone 2.0 to ten, zero, and negative ten degrees over the course of two and a half minutes. As shown in Figure 4.29, the introspective CMAC control provides an RMS error of 4.7 degrees during a series of simple roll commands, whereas the baseline PID exhibits an error of 3.4 degrees.

During a pitch input sequence, shown in Figure 4.30, the RMS error between commanded and PID controlled angles is 5.7 degrees. For the same input commands under introspective CMAC control, the RMS error is 5.6 degrees. These results begin to show the adaptive power of the non-linear CMAC, and the stability exhibited by both controls permits the exploration of more advanced applications.

The control effort put forth by the PID scheme is greater and more erratic, whereas the Introspective scheme is more stable, as shown in Figure 4.31. As a result, the introspective

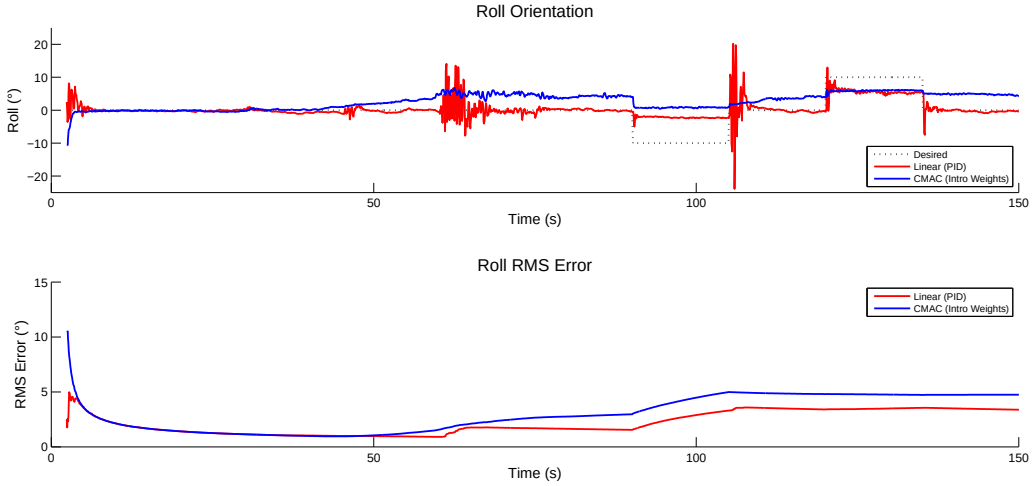


Figure 4.29: Roll During Practical Search

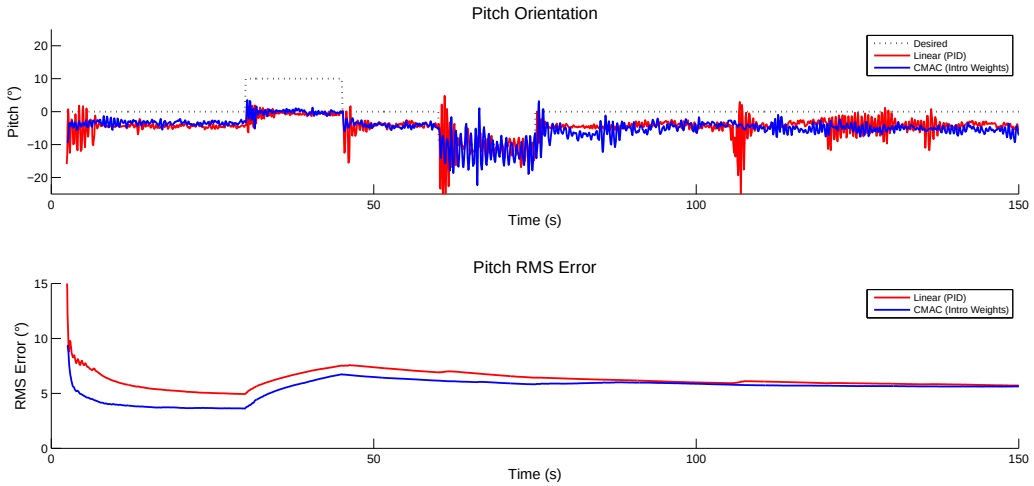


Figure 4.30: Pitch During Practical Search

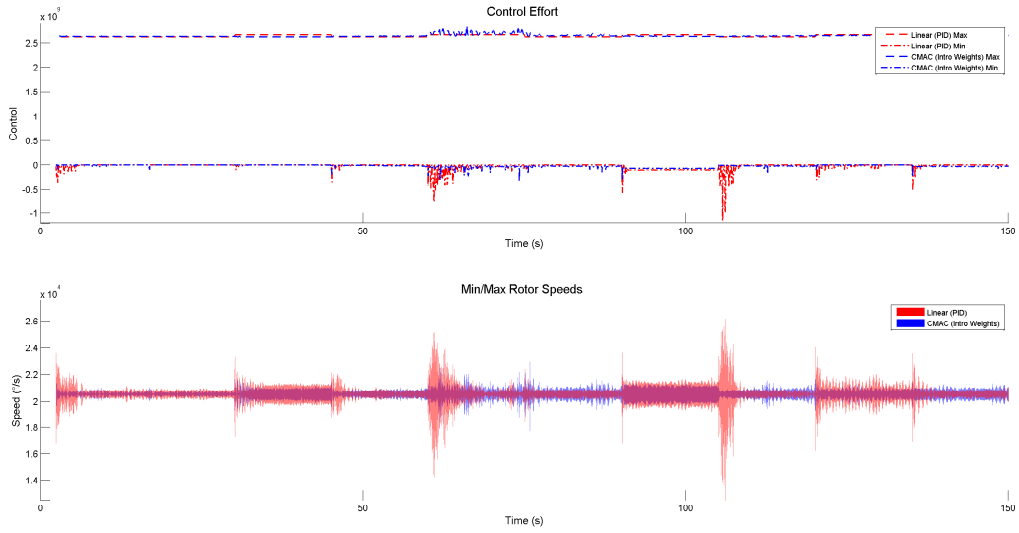


Figure 4.31: Control Effort and Rotor Speed During Practical Search

scheme again has a smaller range of rotor speeds. Nonetheless, the average rotor speeds for both schemes are very similar, 19,994 RPM for the PID and 19,998 RPM for the CMAC.

Again, the introspectively elected weights of the CMAC are shown in Figure 4.32 to demonstrate the underlying stability of the update scheme. While the applied weights may grow, the permanent elected weights are clearly bounded, ensuring the stability of the system to the bursting phenomenon over time.

4.3 Discussion

Given the data gathered, a number of useful conclusions can be made regarding the stability, adaptability, robustness, bursting susceptibility, and autonomous potential of all control system employed, including the CMAC introspective weight voting scheme.

4.3.1 PID Control

First, linear control, by way of a PID scheme, has proven to be effective in simulation, given sufficient tuning and minimal disturbances. This simple system may also have some success

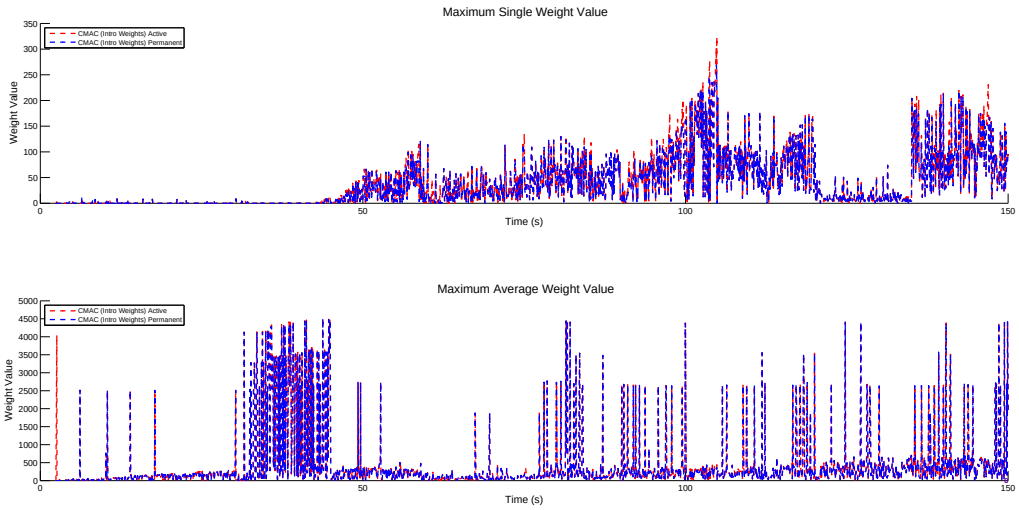


Figure 4.32: Introspective Weights During Practical Search

in specific application where tuning time and disturbance bounds permit. This system is easy to implement, stable, and in the hands of a human operator, robust in a variety of conditions. However, a PID control will always require tuning for different quad-rotor designs, environmental conditions, and operators.

Throughout the course of the simulations and experiments undertaken in this work, the window of operation for the quad-rotor system was constantly expanded, with the goal of greater autonomy in mind. Given the complexity of the quad-rotor system, the model used did not take into account all dynamics, and each time the operation window was expanded through more demanding tasks, the response of the control changed due to these unmodelled dynamics. As a result, the PID control had to be repeatedly returned. This system relies extensively on adaptive input from a human operator, and as a result PID is not an ideal candidate for autonomous control.

4.3.2 CMAC with Leakage Updates

Second, an adaptive CMAC with a traditional leakage style weight update scheme has also provided adequate control of a quad-rotor system in simulation, even in the presence of intro-

duced disturbances such as payload and wind. This system was able to adapt to unmodelled dynamics in the quad-rotor model as its operational window was expanded and required significantly less tuning than the PID as a result. Mathematically, and when tuned correctly in simulation, the traditional leakage style update was proven stable and robust. However, this scheme sacrifices performance for stability in order to reduce bursting.

As shown in Figure 2.12, an aggressively tuned leakage weight update scheme is susceptible to bursting, where unexpected, sudden, and dramatic error signal changes may plague the CMAC control scheme. Less aggressive update gains may decrease the frequency of bursting phenomena, but performance will be sacrificed. A better weight update scheme for the CMAC is achievable.

4.3.3 CMAC with Introspective Voting Updates

Finally, the CMAC control with introspective voting weight updates was also proven to be stable and robust mathematically, in simulation, and throughout experimentation. Like the CMAC with leakage weight updates, it proved more adaptive than the PID control and required less tuning. Unlike the leakage weight update, the introspective voting scheme could be tuned more aggressively without sacrificing performance to mitigate bursting phenomena, as shown in Figure 4.10. In the most complex autonomous simulation, the introspective voting CMAC outperformed the PID in roll and pitch tracking as shown in Figures 4.22 and 4.23. This relatively new weight voting system held its own against the established PID control in simple experimental applications, just as simulation performance predicted. Using the simulations as an indication of experimental performance, the introspective voting CMAC is expected to outperform the PID in more autonomous real-world tasks such as position tracking.

This new introspective weight voting update scheme for the tried and tested CMAC system, as applied to a quad-rotor vehicle is then stable and robust to disturbances, able to adapt to the full operational window of the system with less human control input than PID

control, and is less susceptible to bursting than traditional weight update methods such as leakage. Of the three control techniques explored in earnest, the CMAC with introspective voting weight updates is the best suited for the future autonomous applications predicted within the current literature.

4.4 Impact

Through careful analysis of simulated and applied data, the positive impact of the CMAC control scheme with introspective voting weight updates is clear. Given an accurate system model with no disturbances this adaptive control performs similarly to a simple PID control. However, when introduced to disturbances such as wind and payloads, system performance is superior to that of non-adaptive, linear control schemes such as PID. Additionally, an accurate system model is not required for stability, so unlike PID, this scheme may be applied directly to other quad-rotor platforms without re-tuning. Finally, the weight bursting problem plaguing traditional weight update schemes, such as leakage, has been solved by the governance of introspective weight voting.

The adaptive nature of the proposed control provides the benefit of being scalable to different quad-rotor platforms: from the meso scale up to reconnaissance style drones. It also ensures that the same quad-rotor can be used in a variety of different environments without additional tuning. Whether performing geological surveys at 1,000 meters above sea level in the plains of Alberta, documenting motor vehicle crash sites high in the Rockies, or getting the perfect shots on location at sea level in Vancouver, the same quad-rotor and control can be used out of the box. Furthermore, this scheme is even more stable than the traditional leakage weight update technique shown, meaning unexpected bursting is no longer a concern.

Chapter 5

Conclusion

5.1 Conclusion

The quad-rotor vehicle was identified as an agile, adaptive, aerial platform suitable for a large and growing number of applications in numerous and diverse fields. Its VTOL ability and in-flight manoeuvrability have made the quad-rotor the vehicle of choice for work in surveillance, mapping, inspection, and even artistic applications.

The control challenges presented by the underactuated and underdamped quad-rotor vehicle in a practical environment were identified and overcome using an introspective weight voting CMAC scheme that was proven to be UUB stable. This system provided the benefit of robust, adaptive control while solving a bursting issue present in leakage style weight update schemes.

In simulation, a MATLAB environment was built to test the hypothetical benefits of an introspective weight voting scheme over a traditional update scheme and a linear PID control. The introspective weight voting scheme was shown to eliminate a bursting phenomenon present in traditional update schemes. In simple simulation tasks, such as take-off and direct human controlled flight, the CMAC showed little improvement over the less complex PID control. However, the CMAC demonstrated a superior adaptive ability in the presence of disturbances, and significantly less error in applications requiring greater autonomy.

To analyze experimental applications, a communication scheme and redundancy system were developed to ensure safe flight of a commercially available AR Drone 2.0 quad-rotor vehicle in any environment. Experimental data for the CMAC and PID schemes provided similar performance. However, the CMAC does exhibit a tendency towards lower control forces, thereby minimizing mechanical strain and providing greater predictability in control.

Additionally, with regard to practical implementation, the linear control performs best when tuned specifically for the operational window of an individual task, whereas any improvements made by tuning the CMAC provided gains for all tasks.

This work confirmed the mathematical stability, adaptability, and robustness of an existing CMAC control scheme in simulation and experimental application through the creation of simulated and physical testing environments. The ability of the introspective weight voting scheme to eliminate bursting found in traditional schemes was confirmed in simulation. Finally, this work proved the experimental applicability of a CMAC with introspective weight voting updates implemented on a commercially available quad-rotor platform.

5.2 Contributions

This work ultimately resulted in a new experimental stability proof for an introspective voting weight update scheme for a CMAC applied to quad-rotor control. On the journey, a method for body estimation and control of an AR.Drone 2.0 quad-rotor was developed, along with a safe testing environment. And at the heart of this work, a MATLAB simulation environment was developed to not only provide an interface for simulation and experimental testing, but also record and present all data obtained. The simulation interface, data management and analysis tools, flight testing enclosure and AR Drone 2.0 specific communication blocks can all be applied to future research.

5.3 Future Work

While the work undertaken for this project was extensive, there are a number of areas of interest outside its scope still left to explore. From a control standpoint, a CMAC tracking algorithm might be substituted for the current PID scheme to demonstrate Albus' vision for the CMAC as a hierarchical control. Additionally, the deadzone style weight update boundary used to prevent bursting during long periods of cell activation might be replaced with a

simulated annealing approach to provide a more wholly organic control scheme. With regard to the hardware, using the AR.Drone 2.0 platform within the existing software environment, additional experimental testing might be undertaken to further prove the stability of the introspective voting scheme in more advanced hovering and tracking applications.

Furthermore, while the MATLAB environment used provided an excellent interface for data acquisition, storage, and analysis, the introspective CMAC scheme could be either installed directly into the AR Drone 2.0, or another quad-rotor platform. In this way, further improvements in performance would be likely due to the removal of UDP communication latency. Furthermore, the ultimate goal of providing fully autonomous control in a variety of real-world applications could be explored. This leads to additional potential areas of future work for the introspective CMAC scheme in a specific real-world applications. Given the documented success of the introspective voting control in simulated real-world applications, a partner in industry may be interested in applying the control to a purpose built quad-rotor for a specific task. This would truly prove the success of this research.

Finally, by undertaking not only one real-world application, but by implementing this introspective weight voting CMAC on a variety of different platforms, the benefits of this adaptive scheme can be realized and the autonomous envelope of quad-rotor operation expanded.

Bibliography

[1] Anonymous, "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008, pp. 1-70*, 2008.

[2] D. M. W. Abeywardena, L. A. K. Amaratunga, S. A. A. Shakoor and S. R. Munasinghe, "A velocity feedback fuzzy logic controller for stable hovering of a quad rotor UAV," *2009 International Conference on Industrial and Information Systems (ICIIS 2009), pp. 558-62*, 2009.

[3] J. S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller," *(Cmac), Trans. Asme, Series g. Journal of Dynamic Systems, Measurement and Control*, 1975.

[4] J. S. Albus, "A Model for Memory in the Brain." *National Aeronautics and Space Administration, vol. 6456*, 1971.

[5] J. S. Albus, "Data Storage in the Cerebellar Model Articulation Controller (CMAC)," *Journal of Dynamic Systems, Measurement, and Control, vol. 97, pp. 228-233*, September 1, 1975.

[6] J. S. Albus, "A theory of cerebellar function," *Math. Biosci., vol. 10, pp. 25-61, 2*, 1971.

[7] H. A. F. Almurib, P. T. Nathan and T. N. Kumar, "Control and path planning of quadrotor aerial vehicles for search and rescue," *SICE Annual Conference (SICE), 2011 Proceedings of, pp. 700-705*, 2011.

[8] B. D. O. Anderson, "Adaptive systems, lack of persistency of excitation and bursting phenomena," *Automatica, vol. 21, pp. 247-258, 5*, 1985.

[9] H. Babinsky, "How do wings work?" *Physics Education, vol. 38, pp. 497*, 2003.

[10] S. Bouabdallah, "Design and Control of Quadrotors with Application to Autonomous Flying", 2007.

[11] S. Bouabdallah, P. Murrieri and R. Siegwart, "Design and control of an indoor micro quadrotor," *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on, pp. 4393-4398*, 2004.

[12] S. Bouabdallah, A. Noth and R. Siegwart, "PID vs LQ control techniques applied to an indoor micro quadrotor," *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, pp. 2451-2456*, 2004.

[13] S. Bouabdallah and R. Siegwart, "Full control of a quadrotor," *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on, pp. 153-158*, 2007.

[14] S. Bouabdallah and R. Siegwart, "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor," *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, pp. 2247-2252*, 2005.

[15] H. Bouadi, M. Bouchoucha and M. Tadjine, "Sliding mode control based on backstepping approach for an UAV type-quadrotor," *World Academy of Science, Engineering and Technology, vol. 26, pp. 22-27*, 2007.

[16] P. Bristeau, P. Martin, E. Salan and N. Petit, "The role of propeller aerodynamics in the model of a quadrotor UAV," *European Control Conference*, 2009.

[17] P. Bristeau, E. Dorveaux, D. Vissire and N. Petit, "Hardware and software architecture for state estimation on an experimental low-cost small-scaled helicopter," *Control Eng. Pract., vol. 18, pp. 733-746*, 2010.

[18] P. Bristeau, F. Callou, D. Vissiere and N. Petit, "The navigation and control technology inside the ar. drone micro uav," *18th IFAC World Congress, pp. 1477-1484*, 2011.

[19] A. J. Calise and R. T. Rysdyk, "Nonlinear adaptive flight control using neural networks," *Control Systems, IEEE, vol. 18, pp. 14-25*, 1998.

[20] L. R. G. Carrillo, A. E. D. Lpez, R. Lozano and C. Pgard, *Quad Rotorcraft Control: Vision-Based Hovering and Navigation*. Springer Science & Business Media, 2012.

[21] L. R. G. Carrillo, G. Flores, G. Sanahuja and R. Lozano, "Quad-rotor switching control: An application for the task of path following," *American Control Conference (ACC), 2012, pp. 4637-4642*, 2012.

[22] P. Castillo, R. Lozano and A. Dzul, "Stabilization of a mini rotorcraft with four rotors," *IEEE Control Syst. Mag., vol. 25, pp. 45-55*, 2005.

[23] P. Castillo, A. Dzul and R. Lozano, "Real-time stabilization and tracking of a four-rotor mini rotorcraft," *Control Systems Technology, IEEE Transactions on, vol. 12, pp. 510-516*, 2004.

[24] B. Castillo-Toledo, S. Di Gennaro and F. Jurado, "Trajectory tracking for a quadrotor via fuzzy regulation," *World Automation Congress (WAC), pp. 1-6*, 2012.

[25] S. Commuri and F. L. Lewis, "CMAC neural networks for control of nonlinear dynamical systems: structure, stability and passivity," *Automatica, vol. 33, pp. 635-641*, 1997.

[26] C. Coza, C. Nicol, C. Macnab and A. Ramirez-Serrano, "Adaptive fuzzy control for a quadrotor helicopter robust to wind buffeting," *Journal of Intelligent and Fuzzy Systems, vol. 22, pp. 267-283*, 2011.

[27] C. Coza and C. J. B. Macnab, "A new robust adaptive-fuzzy control method applied to quadrotor helicopter stabilization," *Fuzzy Information Processing Society, 2006. NAFIPS 2006. Annual Meeting of the North American, pp. 454-458*, 2006.

[28] B. Daachi, T. Madani and A. Benallegue, "Adaptive neural controller for redundant robot manipulators and collision avoidance with mobile obstacles," *Neurocomputing, vol. 79, pp. 50-60*, 2012.

[29] A. Das, K. Subbarao and F. Lewis, "Dynamic inversion with zero-dynamics stabilisation for quadrotor control," *Control Theory & Applications, IET, vol. 3, pp. 303-314*, 2009.

[30] T. Dierks and S. Jagannathan, "Output feedback control of a quadrotor UAV using neural networks," *Neural Networks, IEEE Transactions on, vol. 21, pp. 50-66*, 2010.

[31] S. Driessens and P. E. I. Pounds, "Towards a more efficient quadrotor configuration," *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, pp. 1386-1392*, 2013.

[32] J. Dunfied, M. Tarbouchi and G. Labonte, "Neural network based control of a four rotor helicopter," *Industrial Technology, 2004. IEEE ICIT'04. 2004 IEEE International Conference on, pp. 1543-1548*, 2004.

[33] H. Eisenbeiss, "A mini unmanned aerial vehicle (UAV): system overview and image acquisition," *International Archives of Photogrammetry.Remote Sensing and Spatial Information Sciences, vol. 36*, 2004.

[34] J. Engel, J. Sturm and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 2815-2821*, 2012.

[35] J. Escareno, S. Salazar, H. Romero and R. Lozano, "Trajectory control of a quadrotor subject to 2D wind disturbances: Robust-adaptive approach," *Journal of Intelligent and Robotic Systems: Theory and Applications, vol. 70, pp. 51-63*, 2013.

[36] H. Esmaeilzadeh, E. Blem, R. St.Amant, K. Sankaralingam and D. Burger, "Dark silicon and the end of multicore scaling," *Computer Architecture (ISCA), 2011 38th Annual*

International Symposium on, pp. 365-376, 2011.

[37] P. Finn, "Rise of the drone: From Calif. garage to multibillion-dollar defense industry," *Washington Post, vol. 23*, 2011.

[38] V. Ghadiok, J. Goldin and Wei Ren, "Autonomous indoor aerial gripping using a quadrotor," *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pp. 4645-4651*, 2011.

[39] J. H. Gillula, Haomiao Huang, M. P. Vitus and C. J. Tomlin, "Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice," *Robotics and Automation (ICRA), 2010 IEEE International Conference on, pp. 1649-1654*, 2010.

[40] F. H. Glanz, W. T. Miller and L. G. Kraft, "An overview of the CMAC neural network," *Neural Networks for Ocean Engineering, 1991., IEEE Conference on, pp. 301-308*, 1991.

[41] I. Gonzalez, S. Salazar, J. Torres, R. Lozano and H. Romero, "Real-Time Attitude Stabilization of a Mini-UAV Quad-rotor Using Motor Speed Feedback," *J. Intell. Robot. Syst., vol. 70, pp. 93-106, APR*, 2013.

[42] E. Graether and F. Mueller, "Joggobot: A flying robot as jogging companion," *CHI'12 Extended Abstracts on Human Factors in Computing Systems, pp. 1063-1066*, 2012.

[43] M. Green, "Measurement of the moments of inertia of full scale airplanes," 1927.

[44] S. Grzonka, G. Grisetti and W. Burgard, "Towards a navigation system for autonomous indoor flying," *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, pp. 2878-2883*, 2009.

[45] J. A. Guerrero and R. Lozano, "Flight formation of multiple mini rotorcraft based on nested saturations," *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pp. 634-639*, 2010.

[46] J. P. F. Guimaraes, T. L. Laura, A. S. Sanca, A. N. Schildt, M. S. de-Deus, P. J. Alsina, A. T. da-Silva and A. A. D. Medeiros, "Fully autonomous quadrotor: A testbed platform for aerial robotics tasks," *Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian, pp. 68-73*, 2012.

[47] S. Gupte, P. I. T. Mohandas and J. M. Conrad, "A survey of quadrotor unmanned aerial vehicles," *Southeastcon, 2012 Proceedings of IEEE, pp. 1-6*, 2012.

[48] F. Heintz, P. Rudol and P. Doherty, "From images to traffic behavior - A UAV tracking and monitoring application," *Information Fusion, 2007 10th International Conference on, pp. 1-8*, 2007.

[49] K. Higuchi, Y. Ishiguro and J. Rekimoto, "Flying eyes: Free-space content creation using autonomous aerial vehicles," *CHI'11 Extended Abstracts on Human Factors in Computing Systems, pp. 561-570*, 2011.

[50] K. Higuchi, T. Shimada and J. Rekimoto, "Flying sports assistant: External visual imagery representation for sports training," *Proceedings of the 2nd Augmented Human International Conference, pp. 7*, 2011.

[51] G. M. Hoffmann, H. Huang, S. L. Waslander and C. J. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," *Proc. of the AIAA Guidance, Navigation, and Control Conference, pp. 1-20*, 2007.

[52] D. F. Holman, The Future of Drones in Canada: Perspectives from a Former RCAF Fighter Pilot. *Canadian Defence & Foreign Affairs Institute (CDFAI), Strategic Studies Working Group Papers*, 2013.

[53] J. P. How, B. Bethke, A. Frank, D. Dale and J. Vian, "Real-time indoor autonomous vehicle test environment," *Control Systems, IEEE, vol. 28, pp. 51-64*, 2008.

[54] H. Huang, G. M. Hoffmann, S. L. Waslander and C. J. Tomlin, "Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering," *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pp. 3277-3282*, 2009.

[55] J. Irizarry, M. Gheisari and B. N. Walker, "Usability assessment of drone technology as safety inspection tools," *Journal of Information Technology in Construction (ITcon), vol. 17, pp. 194-212*, 2012.

[56] S. JAGANNATHAN, S. COMMURI and F. L. LEWIS, "Feedback Linearization using CMAC Neural Networks," *Automatica, vol. 34, pp. 547-557, 5*, 1998.

[57] S. H. Jeong and S. Jung, "Design and control of a small quad-rotor system under practical limitations," *11th International Conference on Control, Automation and Systems, ICCAS 2011, October 26, 2011 - October 29, pp. 1163-1167*, 2011.

[58] Jun Li and Yuntang Li, "Dynamic analysis and PID control for a quadrotor," *Mechatronics and Automation (ICMA), 2011 International Conference on, pp. 573-578*, 2011.

[59] H. K. Khalil and J. Grizzle, "Nonlinear Systems". *Prentice hall Upper Saddle River*, 2002.

[60] S. Khatoon, D. Gupta and L. K. Das, "PID & LQR control for a quadrotor: Modeling and simulation," *Advances in Computing, Communications and Informatics (ICACCI), 2014 International Conference on, pp. 796-802*, 2014.

[61] Y. H. Kim and F. L. Lewis, "Optimal design of CMAC neural-network controller for robot manipulators," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, vol. 30, pp. 22-31*, 2000.

[62] A. Kotikalpudi, C. Moreno, B. Taylor, H. Pfifer and G. Balas, "Low cost development of a nonlinear simulation for a flexible uninhabited air vehicle," *American Control Conference (ACC), 2014, pp. 2029-2034*, 2014.

[63] L. G. Kraft and D. P. Campagna, "A comparison between CMAC neural network control and two traditional adaptive control systems," *Control Systems Magazine, IEEE, vol. 10, pp. 36-43*, 1990.

[64] T. Krajnk, V. Vonsek, D. Fi?er and J. Faigl, "AR-drone as a platform for robotic research and education," *Research and Education in Robotics-EUROBOT 2011, Anonymous Springer, pp. 172-186*, 2011.

[65] I. Kroo and P. Kunz, "Development of the mesicopter: A miniature autonomous rotorcraft." *American Helicopter Society (AHS) Vertical Lift Aircraft Design Conference, San Francisco, CA*, 2000, .

[66] I. Kroo, F. Prinz, M. Shantz, P. Kunz, G. Fay, S. Cheng, T. Fabian and C. Partridge, "The Mesicopter: A miniature rotorcraft concept phase ii interim report," 2000.

[67] R. Kurzweil, *The Age of Spiritual Machines: When Computers Exceed Human Intelligence*. Penguin, 2000.

[68] D. Lara, G. Romero, A. Sanchez, R. Lozano and A. Guerrero, "Robustness margin for attitude control of a four rotor mini-rotorcraft: Case of study," *Mechatronics, vol. 20, pp. 143-152, 2*, 2010.

[69] D. Lara, A. Sanchez, R. Lozano and P. Castillo, "Real-time embedded control system for VTOL aircrafts: Application to stabilize a quad-rotor helicopter," *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE, pp. 2553-2558*, 2006.

[70] Li Jin-song, Yang Lian and Wang Le-tian, "Aaptive control-optimization of a small scale quadrotor helicopter," *Mechanika, pp. 559-566*, 2013.

[71] A. Lioulemes, G. Galatas, V. Metsis, G. L. Mariottini and F. Makedon, "Safety challenges in using AR. drone to collaborate with humans in indoor environments," *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments, pp. 33*, 2014.

[72] A. Lucieer, S. de Jong and D. Turner, "Mapping landslide displacements using Structure from Motion (SfM) and image correlation of multi-temporal UAV photography," *Progress in Physical Geography, vol. 38(i), pp. 97-116*, 2013.

[73] S. Lupashin, A. Schollig, M. Sherback and R. D'Andrea, "A simple learning strategy for high-speed quadrocopter multi-flips," *Robotics and Automation (ICRA), 2010 IEEE International Conference on, pp. 1642-1648*, 2010.

[74] A. Lucieer, S. Robinson, D. Turner, S. Harwin and J. Kelcey, "USING A MICRO-UAV FOR ULTRA-HIGH RESOLUTION MULTI-SENSOR OBSERVATIONS OF ANTARCTIC MOSS BEDS," *International Archives of the Photogrammetry. Remote Sensing and Spatial Information Sciences, vol. XXXIX-B1, pp. 429-433*, 2012.

[75] L. F. Luque-Vega, B. Castillo-Toledo, A. Loukianov and L. E. Gonzalez-Jimenez, "Power line inspection via an unmanned aerial system based on the quadrotor helicopter," *Mediterranean Electrotechnical Conference (MELECON), 2014 17th IEEE, pp. 393-397*, 2014.

[76] A. M. Lyapunov, "The general problem of the stability of motion," *Int J Control, vol. 55, pp. 531-534*, 1992.

[77] C. J. B. Macnab, "Getting weights to behave themselves: Achieving stability and performance in neural-adaptive control when inputs oscillate," *American Control Conference, pp. 3192-3197*, 2005.

[78] C. J. B. Macnab, "An introspective algorithm for achieving low-gain high-performance robust neural-adaptive control," *American Control Conference (ACC), pp. 2893-2899*,

2014.

[79] C. J. B. Macnab, "Stable neural-adaptive control of activated sludge bioreactors," *American Control Conference (ACC), pp. 2869-2874*, 2014.

[80] C. J. B. Macnab, "Neural-adaptive control using alternate weights," *Neural Comput. Appl., vol. 20, pp. 211-221, MAR*, 2011.

[81] T. Madani and A. Benallegue, "Sliding mode observer and backstepping control for a quadrotor unmanned aerial vehicles," *American Control Conference, 2007. ACC'07, pp. 5887-5892*, 2007.

[82] T. Madani and A. Benallegue, "Backstepping control for a quadrotor helicopter," *in Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pp. 3255-3260*, 2006.

[83] T. Madani and A. Benallegue, "Control of a quadrotor mini-helicopter via full state backstepping technique," *in Decision and Control, 2006 45th IEEE Conference on, pp. 1515-1520*, 2006.

[84] R. Mahony, V. Kumar and P. Corke, "Multirotor Aerial Vehicles: Modelling, Estimation, and Control of Quadrotor," *Robotics & Automation Magazine, IEEE, vol. 19, pp. 20-32*, 2012.

[85] J. Markoff, "Google Cars Drive Themselves, in Traffic," *New York Times, vol. 9*, 2010.

[86] K. Masaud and C. J. B. Macnab, "Preventing bursting in adaptive control using an introspective neural network algorithm," *Neurocomputing, vol. 136, pp. 300-314, 7/20*, 2014.

[87] K. Masaud and C. Macnab, "An introspective learning algorithm that achieves robust adaptive control of a quadrotor helicopter," *Anonymous American Institute of Aeronautics and Astronautics*, 2012.

[88] K. Masuad and C. J. B. Macnab, "Stable Fuzzy-Adaptive Control Using an Introspective Algorithm," *2012 American Control Conference (Acc), pp. 5622-5627*, 2012.

[89] L. Matthies, P. Tokumaru, S. Sherrit and P. Beauchamp, "Titan Aerial Daughtercraft (TAD) for Surface Studies from a Lander or Balloon," *LPI Contributions, vol. 1795, pp. 8083*, 2014.

[90] B. W. McCormick, *Aerodynamics, Aeronautics, and Flight Mechanics, Wiley, New York*, 1995.

[91] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," *Robotics and Automation (ICRA), 2011 IEEE International Conference on, pp. 2520-2525*, 2011.

[92] D. Mellinger, N. Michael and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research, vol. 31, pp. 664-674*, 2012.

[93] D. Mellinger, M. Shomin, N. Michael and V. Kumar, "Cooperative grasping and transport using multiple quadrotors," *Distributed Autonomous Robotic Systems Anonymous Springer, pp. 545-558*, 2013.

[94] M. P. Miller, "An accurate method of measuring the moments of inertia of airplanes," 1930.

[95] V. Mistler, A. Benallegue and N. K. M'Sirdi, "Exact linearization and noninteracting control of a 4 rotors helicopter via dynamic feedback," *Robot and Human Interactive Communication, 2001. Proceedings. 10th IEEE International Workshop on, pp. 586-593*, 2001.

[96] A. Mokhtari and A. Benallegue, "Dynamic feedback controller of euler angles and wind parameters estimation for a quadrotor unmanned aerial vehicle," *Robotics and Automa-*

tion, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on, pp. 2359-2366 Vol.3, 2004.

[97] S. Montambault, J. Beaudry, K. Toussaint and N. Pouliot, "On the application of VTOL UAVs to the inspection of power utility assets," *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on, pp. 1-7*, 2010.

[98] M. Muller, S. Lupashin and R. D'Andrea, "Quadrocopter ball juggling," *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pp. 5113-5120*, 2011.

[99] R. Murphy, D. Shell, A. Guerin, B. Duncan, B. Fine, K. Pratt and T. Zourntos, "A Midsummer Night's Dream (with flying robots)," *Autonomous Robots, vol. 30, pp. 143-156*, 2011.

[100] W. S. Ng and E. Sharlin, "Collocated interaction with flying robots," *Ro-Man, 2011 Ieee, pp. 143-149*, 2011.

[101] C. Nicol, C. Macnab and A. Ramirez-Serrano, "Robust adaptive control of a quadrotor helicopter," *Mechatronics, vol. 21, pp. 927-938*, 2011.

[102] C. E. Nicol, *A Robust Adaptive Neural Network Control for a Quadrotor Helicopter*. 2011.

[103] U. Niethammer, M. R. James, S. Rothmund, J. Travelletti and M. Joswig, "UAV-based remote sensing of the Super-Sauze landslide: Evaluation and results," *Eng. Geol., vol. 128, pp. 2-11, 3/9*, 2012.

[104] I. Palunko, R. Fierro and P. Cruz, "Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach," *Robotics and Automation (ICRA), 2012 IEEE International Conference on, pp. 2691-2697*, 2012.

[105] M. Pelletier, A. Sakamoto, C. Tessier and G. Saintonge, "Autonomous navigation and control functions of the CL-327 VTOL UAV," *Agard Conference Proceedings Agard Cp, pp. 18-18*, 1998.

[106] P. Pounds, R. Mahony and P. Corke, "Modelling and control of a quad-rotor robot," *Proceedings Australasian Conference on Robotics and Automation 2006*, 2006.

[107] P. Pounds and R. Mahony, "Design principles of large quadrotors for practical applications," *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, pp. 3265-3270*, 2009.

[108] R. Pourbafarani, S. A. A. Moosavian, S. Sadr and P. Zarafshan, "Trajectory tracking of a quadrotor using stabilizing mechanism," *Robotics and Mechatronics (ICRoM), 2014 Second RSI/ISM International Conference on, pp. 410-415*, 2014.

[109] J. Roberts, T. Stirling, J. Zufferey and D. Floreano, "Quadrotor using minimal sensing for autonomous indoor flight," *Proceedings of the European Micro Air Vehicle Conference and Flight Competition (EMAV2007)*, 2007, .

[110] A. Sanchez, L. G. Carrillo, E. Rondon, R. Lozano and O. Garcia, "Hovering flight improvement of a quad-rotor mini UAV using brushless DC motors," *Journal of Intelligent & Robotic Systems, vol. 61, pp. 85-10*1, 2011.

[111] A. Schollig, F. Augugliaro, S. Lupashin and R. D'Andrea, "Synchronizing the motion of a quadrocopter to music," *Robotics and Automation (ICRA), 2010 IEEE International Conference on, pp. 3355-3360*, 2010.

[112] Sheng Zhao, Wenjie Dong and J. A. Farrell, "Quaternion-based trajectory tracking control of VTOL-UAVs using command filtered backstepping," *American Control Conference (ACC), pp. 1018-1023*, 2013.

[113] E. Stingu and F. L. Lewis, "An approximate dynamic programming based controller for an underactuated 6DoF quadrotor," *Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2011 IEEE Symposium on, pp. 271-278*, 2011.

[114] J. Stolaroff, "The Need for a Life Cycle Assessment of Drone-Based Commercial Package Delivery", 2014.

[115] B. Sumantri, N. Uchiyama and S. Sano, "Second order sliding mode control for a quad-rotor helicopter with a nonlinear sliding surface," *Control Applications (CCA), 2014 IEEE Conference on, pp. 742-746*, 2014.

[116] Jean-Jacques E. Slotine and Weiping Li, *Applied Nonlinear Control, Prentice Hall, Inc., Upper Saddle River, New Jersey*, 1991

[117] Suseong Kim, Seungwon Choi and H. J. Kim, "Aerial manipulation using a quadrotor with a two DOF robotic arm," *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, pp. 4990-4995*, 2013.

[118] A. Tayebi and S. McGilvray, "Attitude stabilization of a VTOL quadrotor aircraft," *Control Systems Technology, IEEE Transactions on, vol. 14, pp. 562-571*, 2006.

[119] A. Tayebi and S. McGilvray, "Attitude stabilization of a four-rotor aerial robot," *Decision and Control, 2004. CDC. 43rd IEEE Conference on, pp. 1216-1221 Vol.2*, 2004.

[120] S. Thrun, M. Diel and D. Hhnel, "Scan alignment and 3-D surface modeling with a helicopter platform," *The 4 Th International Conference on Field and Service Robotics*, 2003.

[121] D. Vissiere, P. Bristeau, A. P. Martin and N. Petit, "Experimental autonomous flight of a small-scaled helicopter using accurate dynamics model and low-cost sensors," *Proc. of the 17th IFAC World Congress, pp. 14*, 2008.

[122] H. Voos, "Nonlinear state-dependent riccati equation control of a quadrotor UAV," *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE, pp. 2547-2552*, 2006.

[123] H. Voos, "Nonlinear control of a quadrotor micro-UAV using feedback-linearization," *Mechatronics, 2009. ICM 2009. IEEE International Conference on, pp. 1-6*, 2009.

[124] H. Voos, "Nonlinear and neural network-based control of a small four-rotor aerial robot," *Advanced Intelligent Mechatronics, 2007 IEEE/ASME International Conference on, pp. 1-6*, 2007.

[125] S. Waharte and N. Trigoni, "Supporting search and rescue operations with UAVs," *International Conference on Emerging Security Technologies (EST), pp. 142-147*, 2010.

[126] R. L. Wilson, "Ethical issues with use of drone aircraft," *Ethics in Science, Technology and Engineering, 2014 IEEE International Symposium on, pp. 1-4*, 2014.

Appendix A

Control Code

A.1 PIDControl.m

```
1 function u = pidControl(handles)
2 % --- Function to return a PID control signal
3 t = getValues(handles, 't');    % Get current and elapsed time
4 [kp, ki, kd] = getValues(handles, 'pid_Kp_gain', 'pid_Ki_gain', 'pid_Kd_gain')
      ;
5 % Kp to decrease rise, Ki to reduce ss error, Kd to reduce overshoot
6 [z_d, roll_d, pitch_d, yaw_d] = getValues(handles, 'z_d', 'roll_d', 'pitch_d',
       'yaw_d'); % Get current desired values
7 [z, roll, pitch, yaw] = getValues(handles, 'z', 'roll', 'pitch', 'yaw');  %
      Get current actual values
8 [z_err, roll_err, pitch_err, yaw_err] = getValues(handles, 'z_err', 'roll_err'
      , 'pitch_err', 'yaw_err');  % Get previous error values
9 % Calculate simulated values assuming ideal parameters
10 [Ixx, Iyy, Izz, d, b, l, mass] = getValues(handles, 'Ixx', 'Iyy', 'Izz', 'd',
      'b', 'l', 'mass');
11 g = 9.8;    % Estimated Acceleration due to gravity
12 pid(1) = kp * (z_d - z) + ki * (z_d - z) * t(2) + kd * ((z_d - z) - z_err) / t
      (2);
13 pid(2) = kp * (roll_d - roll) + ki * (roll_d - roll) * t(2) + kd * ((roll_d -
      roll) - roll_err) / t(2);
14 pid(3) = kp * (pitch_d - pitch) + ki * (pitch_d - pitch) * t(2) + kd * ((
      pitch_d - pitch) - pitch_err) / t(2);
15 pid(4) = kp * (yaw_d - yaw) + ki * (yaw_d - yaw) * t(2) + kd * ((yaw_d - yaw)
      - yaw_err) / t(2);
16 % Save current error values
```

```
17 setValues(handles, 'z_err', z_d - z, 'roll_err', roll_d - roll, 'pitch_err',
       pitch_d - pitch, 'yaw_err', yaw_d - yaw);
18 % Calculate control signals
19 u(1) = (pid(1) + g) * mass / (cosd(roll_d)*cosd(pitch_d)*b);
20 u(2) = pid(2) * Ixx / (l*b);
21 u(3) = pid(3) * Iyy / (l*b);
22 u(4) = pid(4) * Izz / d;    % -> Use first derivative of yaw speed for error
23 end
```

A.2 CmacLeakageControl.m

```
1 function u = CmacLeakageControl(handles)
2 % --- Function to return a leakage adaptive, learning control signal
3 m = getValues(handles, 'm');
4 [w_hat_1, w_hat_2, w_hat_3, w_hat_4] = getValues(handles, 'w_hat_1', 'w_hat_2'
      , 'w_hat_3', 'w_hat_4');
5 [loc1, loc2, loc3, loc4] = getValues(handles, 'loc1', 'loc2', 'loc3', 'loc4');
6 [mem1, mem2, mem3, mem4] = getValues(handles, 'mem1', 'mem2', 'mem3', 'mem4');
7 [minstate1, minstate2, minstate3, minstate4] = getValues(handles, 'minstate1',
       'minstate2', 'minstate3', 'minstate4');
8 [maxstate1, maxstate2, maxstate3, maxstate4] = getValues(handles, 'maxstate1',
       'maxstate2', 'maxstate3', 'maxstate4');
9 t = getValues(handles, 't');    % Get current and elapsed time
10 speed = getMotorSpeeds(handles);
11 omega = speed * [1; -1; 1; -1];
12 [z_d, roll_d, pitch_d, yaw_d] = getValues(handles, 'z_d', 'roll_d', 'pitch_d',
       'yaw_d'); % Get current desired values
13 [z, roll, pitch, yaw] = getValues(handles, 'z', 'roll', 'pitch', 'yaw'); % Get
       current actual values
14 [z_d_dot, roll_d_dot, pitch_d_dot, yaw_d_dot] = getValues(handles, 'z_d_dot',
      'roll_d_dot', 'pitch_d_dot', 'yaw_d_dot'); % Get current desired
      velocities
```

```
15 [z_dot, roll_dot, pitch_dot, yaw_dot] = getValues(handles, 'z_dot', 'roll_dot'
       , 'pitch_dot', 'yaw_dot');
16 % Get auxiliary error and control gains
17 [lambda, beta, nu] = getValues(handles, 'cmac_leakage_L_gain', '
       cmac_leakage_Beta_gain', 'cmac_leakage_Nu_gain');
18 K = [getValues(handles, 'cmac_leakage_K1_gain') getValues(handles, '
       cmac_leakage_K2_gain')...
19      getValues(handles, 'cmac_leakage_K3_gain') getValues(handles, '
       cmac_leakage_K4_gain')];
20 % Calculate position and velocity errors
21 x_l = [z-z_d roll-roll_d pitch-pitch_d yaw-yaw_d];
22 x_h = [z_dot-z_d_dot roll_dot-roll_d_dot pitch_dot-pitch_d_dot yaw_dot-
       yaw_d_dot];
23 zeta = lambda * x_l + x_h;  % Calculate auxiliary error
24 inputs1 = [zeta(1) roll pitch roll_d pitch_d];
25 inputs2 = [zeta(2) pitch_dot yaw_dot omega];
26 inputs3 = [zeta(3) roll_dot yaw_dot omega];
27 inputs4 = [zeta(4) roll_dot pitch_dot];
28 % Calculate linear functions via CMAC
29 [G1, loc1] = calculateCmac(handles, inputs1, 5, minstate1, maxstate1, loc1);
30 [G2, loc2] = calculateCmac(handles, inputs2, 4, minstate2, maxstate2, loc2);
31 [G3, loc3] = calculateCmac(handles, inputs3, 4, minstate3, maxstate3, loc3);
32 [G4, loc4] = calculateCmac(handles, inputs4, 3, minstate4, maxstate4, loc4);
33 % Normalize Gamma values
34 G1 = G1./sum(G1);
35 G2 = G2./sum(G2);
36 G3 = G3./sum(G3);
37 G4 = G4./sum(G4);
38 % Calculate weights
39 for j = 1:m
40      w_hat_1(j) = mem1(loc1(j));
41      w_hat_2(j) = mem2(loc2(j));
```

```
42      w_hat_3(j) = mem3(loc3(j));
43      w_hat_4(j) = mem4(loc4(j));
44  end
45  setValues(handles, 'w_hat_1', w_hat_1, 'w_hat_2', w_hat_2, 'w_hat_3', w_hat_3,
        'w_hat_4', w_hat_4);
46  % Calculate weight updates
47  w1_hat_dot = beta * (G1' * zeta(1) - nu * w_hat_1);
48  w2_hat_dot = beta * (G2' * zeta(2) - nu * w_hat_2);
49  w3_hat_dot = beta * (G3' * zeta(3) - nu * w_hat_3);
50  w4_hat_dot = beta * (G4' * zeta(4) - nu * w_hat_4);
51  for j = 1:m
52      mem1(loc1(j)) = mem1(loc1(j)) + w1_hat_dot(j) * t(2);
53      mem2(loc2(j)) = mem2(loc2(j)) + w2_hat_dot(j) * t(2);
54      mem3(loc3(j)) = mem3(loc3(j)) + w3_hat_dot(j) * t(2);
55      mem4(loc4(j)) = mem4(loc4(j)) + w4_hat_dot(j) * t(2);
56  end
57  setValues(handles, 'mem1', mem1, 'mem2', mem2, 'mem3', mem3, 'mem4', mem4);
58  %Calculate control signals
59  [mass, thrust] = getValues(handles, 'mass' ,'b');
60  g = 9.8;
61  u(1) = -G1 * w_hat_1 - K(1) * zeta(1) + g/(thrust * cosd(roll) * cosd(pitch) /
        mass);
62  u(2) = -G2 * w_hat_2 - K(2) * zeta(2);
63  u(3) = -G3 * w_hat_3 - K(3) * zeta(3);
64  u(4) = -G4 * w_hat_4 - K(4) * zeta(4);
65  end
```

A.3 CalculateCmac.m

```
1  function [gamma, locations] = calculateCmac(handles, inputs, num_inputs,
       minstate, maxstate, locations)
2  % --- Function to return normalized CMAC gamma and values memory locations
```

```
[m, q, size] = getValues(handles, 'm', 'q', 'mem_size');
[hashtable, offset] = getValues(handles, 'hashtable', 'offset');
norm_inputs = (inputs(1:num_inputs)-minstate(1:num_inputs))./(maxstate(1:
    num_inputs)-minstate(1:num_inputs));
norm_inputs = min(norm_inputs, ones(1, num_inputs));
norm_inputs = max(norm_inputs, zeros(1, num_inputs));
gamma = zeros(1, m);      cell = zeros(1, num_inputs);
for j = 1:m
    gamma(j) = 1;
    locations_total = 0;
    for i = 1:num_inputs
        place = norm_inputs(i) * (q - 1) + offset(i, j);
        cell(i) = floor(place);
        h = place - cell(i);
        func = 16 * (h*h  -2.0*h*h*h + h*h*h*h);
        gamma(j) = gamma(j) * func;
        locations_total = locations_total + hashtable(cell(i) + 1 + q*(j-1) +
    q*m*(i-1));
    end
    locations(j) = int32(floor(mod(locations_total, size)));
end
end
```

A.4 CmacIntroWeightControl.m

```
function u = CmacIntroWeightControl(handles, speed)
% --- Function to return an introspective learning control signal
gain = 0.01;     % Learning Deadzone
m = getValues(handles, 'm');
[w1_hat, w2_hat, w3_hat, w4_hat] = getValues(handles, 'w_hat_1', 'w_hat_2', '
    w_hat_3', 'w_hat_4');
[p1_hat, p2_hat, p3_hat, p4_hat] = getValues(handles, 'p_hat_1', 'p_hat_2', '
```

```
    p_hat_3', 'p_hat_4');
[loc1, loc2, loc3, loc4] = getValues(handles, 'loc1', 'loc2', 'loc3', 'loc4');
[mem1, mem2, mem3, mem4] = getValues(handles, 'mem1', 'mem2', 'mem3', 'mem4');
[memperm1, memperm2, memperm3, memperm4] = getValues(handles, 'alt_mem1', '
    alt_mem2', 'alt_mem3', 'alt_mem4');
[loc1_last, loc2_last, loc3_last, loc4_last] = getValues(handles, 'loc1_last',
     'loc2_last', 'loc3_last', 'loc4_last');
[loc1_secLast, loc2_secLast, loc3_secLast, loc4_secLast] = getValues(handles,
    'loc1_secLast', 'loc2_secLast', 'loc3_secLast', 'loc4_secLast');
[loc1_changed, loc2_changed, loc3_changed, loc4_changed] = getValues(handles,
    'loc1_changed', 'loc2_changed', 'loc3_changed', 'loc4_changed');
[z1_num, z2_num, z3_num, z4_num] = getValues(handles, 'z1_num', 'z2_num', '
    z3_num', 'z4_num');
[z1_num_last, z2_num_last, z3_num_last, z4_num_last] = getValues(handles, '
    z1_num_last', 'z2_num_last', 'z3_num_last', 'z4_num_last');
[z1_net, z2_net, z3_net, z4_net] = getValues(handles, 'z1_net', 'z2_net', '
    z3_net', 'z4_net');
[z1_net_last, z2_net_last, z3_net_last, z4_net_last] = getValues(handles, '
    z1_net_last', 'z2_net_last', 'z3_net_last', 'z4_net_last');
[z1_avg, z2_avg, z3_avg, z4_avg] = getValues(handles, 'z1_avg', 'z2_avg', '
    z3_avg', 'z4_avg');
[z1_avg_last, z2_avg_last, z3_avg_last, z4_avg_last] = getValues(handles, '
    z1_avg_last', 'z2_avg_last', 'z3_avg_last', 'z4_avg_last');
[cost1_best, cost2_best, cost3_best, cost4_best] = getValues(handles, '
    cost1_best', 'cost2_best', 'cost3_best', 'cost4_best');
[votes1_good_w, votes2_good_w, votes3_good_w, votes4_good_w] = getValues(
    handles, ...
     'votes1_good_w', 'votes2_good_w', 'votes3_good_w', 'votes4_good_w');
[votes1_bad_w, votes2_bad_w, votes3_bad_w, votes4_bad_w] = getValues(handles,
    ...
      'votes1_bad_w', 'votes2_bad_w', 'votes3_bad_w', 'votes4_bad_w');
[votes1_good_tot, votes2_good_tot, votes3_good_tot, votes4_good_tot] =
```

```
    getValues(handles, ...
25      'votes1_good_tot','votes2_good_tot', 'votes3_good_tot', 'votes4_good_tot'
    );
26 [votes1_bad_tot, votes2_bad_tot, votes3_bad_tot, votes4_bad_tot] = getValues(
    handles, ...
27      'votes1_bad_tot','votes2_bad_tot', 'votes3_bad_tot', 'votes4_bad_tot');
28 [prevUpdate1, prevUpdate2, prevUpdate3, prevUpdate4] = getValues(handles, '
    prevUpdate1', 'prevUpdate2', 'prevUpdate3', 'prevUpdate4');
29 [prevError1, prevError2, prevError3, prevError4] = getValues(handles, '
    prevError1', 'prevError2', 'prevError3', 'prevError4');
30 [membest1, membest2, membest3, membest4] = getValues(handles, 'membest1', '
    membest2', 'membest3', 'membest4');
31 [cost1, cost2, cost3, cost4] = getValues(handles, 'cost1', 'cost2', 'cost3', '
    cost4');
32 [w1_hat_dot, w2_hat_dot, w3_hat_dot, w4_hat_dot] = getValues(handles, '
    w1_hat_dot', 'w2_hat_dot', 'w3_hat_dot', 'w4_hat_dot');
33 [w1_hat_dot_last, w2_hat_dot_last, w3_hat_dot_last, w4_hat_dot_last] =
    getValues(handles, ...
34      'w1_hat_dot_last','w2_hat_dot_last', 'w3_hat_dot_last', 'w4_hat_dot_last'
    );
35 % Get specified min and max state variables
36 [minstate1, minstate2, minstate3, minstate4] = getValues(handles, 'minstate1',
     'minstate2', 'minstate3', 'minstate4');
37 [maxstate1, maxstate2, maxstate3, maxstate4] = getValues(handles, 'maxstate1',
     'maxstate2', 'maxstate3', 'maxstate4');
38 omega = speed * [1; -1; 1; -1];
39 [z_d, roll_d, pitch_d, yaw_d] = getValues(handles, 'z_d', 'roll_d', 'pitch_d',
     'yaw_d'); % Get current desired values
40 [z, roll, pitch, yaw] = getValues(handles, 'z', 'roll', 'pitch', 'yaw'); % Get
     current actual values
41 [z_d_dot, roll_d_dot, pitch_d_dot, yaw_d_dot] = getValues(handles, 'z_d_dot',
    'roll_d_dot', 'pitch_d_dot', 'yaw_d_dot'); % Get current desired
```

```
    velocities
42 [z_dot, roll_dot, pitch_dot, yaw_dot] = getValues(handles, 'z_dot', 'roll_dot'
    , 'pitch_dot', 'yaw_dot');
43 % Calculate simulated values assuming ideal parameters
44 [b, mass] = getValues(handles, 'b', 'mass');
45 % Get auxiliary error and control gains
46 lambda = getValues(handles, 'cmac_introWeight_L_gain');
47 K = [getValues(handles, 'cmac_introWeight_K1_gain') getValues(handles, '
    cmac_introWeight_K2_gain')...
48      getValues(handles, 'cmac_introWeight_K3_gain') getValues(handles, '
    cmac_introWeight_K4_gain')];
49 [beta1, beta2, beta3, beta4] = getValues(handles, 'cmac_introWeight_Beta1_gain
    ',...
50      'cmac_introWeight_Beta2_gain', 'cmac_introWeight_Beta3_gain', '
    cmac_introWeight_Beta4_gain');
51 % Calculate position and velocity errors
52 x_l = [z-z_d roll-roll_d pitch-pitch_d yaw-yaw_d];
53 x_h = [z_dot-z_d_dot roll_dot-roll_d_dot pitch_dot-pitch_d_dot yaw_dot-
    yaw_d_dot];
54 zeta = lambda * x_l + x_h;  % Calculate auxiliary error
55 % Create CMAC input signals
56 inputs1 = [zeta(1) roll pitch roll_d pitch_d];
57 inputs2 = [zeta(2) pitch_dot yaw_dot omega];
58 inputs3 = [zeta(3) roll_dot yaw_dot omega];
59 inputs4 = [zeta(4) roll_dot pitch_dot];
60 % Calculate linear functions via CMAC
61 [G1, loc1, loc1_changed, loc1_last, loc1_secLast] = calculateVoteCmac(handles,
     inputs1, 5, minstate1, maxstate1, loc1, loc1_last, loc1_secLast,
    loc1_changed);
62 [G2, loc2, loc2_changed, loc2_last, loc2_secLast] = calculateVoteCmac(handles,
     inputs2, 4, minstate2, maxstate2, loc2, loc2_last, loc2_secLast,
    loc2_changed);
```

```
63 [G3, loc3, loc3_changed, loc3_last, loc3_secLast] = calculateVoteCmac(handles,
       inputs3, 4, minstate3, maxstate3, loc3, loc3_last, loc3_secLast,
      loc3_changed);
64 [G4, loc4, loc4_changed, loc4_last, loc4_secLast] = calculateVoteCmac(handles,
       inputs4, 3, minstate4, maxstate4, loc4, loc4_last, loc4_secLast,
      loc4_changed);
65 % Save locations
66 setValues(handles, 'loc1', loc1, 'loc2', loc2, 'loc3', loc3, 'loc4', loc4);
67 setValues(handles, 'loc1_last', loc1_last, 'loc2_last', loc2_last, 'loc3_last'
      , loc3_last, 'loc4_last', loc4_last);
68 setValues(handles, 'loc1_secLast', loc1_secLast, 'loc2_secLast', loc2_secLast,
       'loc3_secLast', loc3_secLast, 'loc4_secLast', loc4_secLast);
69 setValues(handles, 'loc1_changed', loc1_changed, 'loc2_changed', loc2_changed,
       'loc3_changed', loc3_changed, 'loc4_changed', loc4_changed);
70 % Normalize Gamma values
71 G1 = G1./sum(G1);
72 G2 = G2./sum(G2);
73 G3 = G3./sum(G3);
74 G4 = G4./sum(G4);
75 setValues(handles, 'gamma1', G1, 'gamma2', G2, 'gamma3', G3, 'gamma4', G4);
76 % Calculate flexible weights
77 for j = 1:m
78    w1_hat(j) = mem1(loc1(j));
79    w2_hat(j) = mem2(loc2(j));
80    w3_hat(j) = mem3(loc3(j));
81    w4_hat(j) = mem4(loc4(j));
82 end;
83 setValues(handles, 'w_hat_1', w1_hat, 'w_hat_2', w2_hat, 'w_hat_3', w3_hat, '
      w_hat_4', w4_hat);
84 % Calculate permanent weights
85 for j = 1:m
86    p1_hat(j) = memperm1(loc1(j));
```

```
87    p2_hat(j) = memperm2(loc2(j));
88    p3_hat(j) = memperm3(loc3(j));
89    p4_hat(j) = memperm4(loc4(j));
90 end;
91 setValues(handles, 'p_hat_1', p1_hat, 'p_hat_2', p2_hat, 'p_hat_3', p3_hat, '
       p_hat_4', p4_hat);
92 % Calculate voting weight updates
93 [zeta(1), z1_num, z1_num_last, z1_net, z1_net_last, z1_avg, z1_avg_last,
       cost1_best,...
94      votes1_good_tot, votes1_bad_tot, votes1_good_w, votes1_bad_w, mem1,
       memperm1,...
95      prevUpdate1, membest1, cost1, prevError1, w1_hat_dot, w1_hat_dot_last]...
96      = updateVoteWeights(handles, zeta(1), loc1_changed, z1_num, z1_num_last,
       z1_net,...
97      z1_net_last, z1_avg, z1_avg_last, cost1_best,...
98      votes1_good_tot, votes1_bad_tot, votes1_good_w, votes1_bad_w,...
99      mem1, memperm1, prevUpdate1, membest1, cost1, prevError1,...
100     loc1, loc1_last, loc1_secLast, w1_hat_dot, w1_hat_dot_last, G1, beta1,
       gain);
101 [zeta(2), z2_num, z2_num_last, z2_net, z2_net_last, z2_avg, z2_avg_last,
       cost2_best,...
102     votes2_good_tot, votes2_bad_tot, votes2_good_w, votes2_bad_w, mem2,
       memperm2,...
103     prevUpdate2, membest2, cost2, prevError2, w2_hat_dot, w2_hat_dot_last]...
104     = updateVoteWeights(handles, zeta(2), loc2_changed, z2_num, z2_num_last,
       z2_net,...
105     z2_net_last, z2_avg, z2_avg_last, cost2_best,...
106     votes2_good_tot, votes2_bad_tot, votes2_good_w, votes2_bad_w,...
107     mem2, memperm2, prevUpdate2, membest2, cost2, prevError2,...
108     loc2, loc2_last, loc2_secLast, w2_hat_dot, w2_hat_dot_last, G2, beta2,
       gain);
109 [zeta(3), z3_num, z3_num_last, z3_net, z3_net_last, z3_avg, z3_avg_last,
```

```
    cost3_best,...
110     votes3_good_tot, votes3_bad_tot, votes3_good_w, votes3_bad_w, mem3,
    memperm3,...
111     prevUpdate3, membest3, cost3, prevError3, w3_hat_dot, w3_hat_dot_last]...
112     = updateVoteWeights(handles, zeta(3), loc3_changed, z3_num, z3_num_last,
    z3_net,...
113     z3_net_last, z3_avg, z3_avg_last, cost3_best,...
114     votes3_good_tot, votes3_bad_tot, votes3_good_w, votes3_bad_w,...
115     mem3, memperm3, prevUpdate3, membest3, cost3, prevError3,...
116     loc3, loc3_last, loc3_secLast, w3_hat_dot, w3_hat_dot_last, G3, beta3,
    gain);
117 [zeta(4), z4_num, z4_num_last, z4_net, z4_net_last, z4_avg, z4_avg_last,
    cost4_best,...
118     votes4_good_tot, votes4_bad_tot, votes4_good_w, votes4_bad_w, mem4,
    memperm4,...
119     prevUpdate4, membest4, cost4, prevError4, w4_hat_dot, w4_hat_dot_last]...
120     = updateVoteWeights(handles, zeta(4), loc4_changed, z4_num, z4_num_last,
    z4_net,...
121     z4_net_last, z4_avg, z4_avg_last, cost4_best,...
122     votes4_good_tot, votes4_bad_tot, votes4_good_w, votes4_bad_w,...
123     mem4, memperm4, prevUpdate4, membest4, cost4, prevError4,...
124     loc4, loc4_last, loc4_secLast, w4_hat_dot, w4_hat_dot_last, G4, beta4,
    gain);
125 % Save calculated values
126 setValues(handles, 'mem1', mem1, 'mem2', mem2, 'mem3', mem3, 'mem4', mem4);
127 setValues(handles, 'alt_mem1', memperm1, 'alt_mem2', memperm2, 'alt_mem3',
    memperm3, 'alt_mem4', memperm4);
128 setValues(handles, 'z1_num', z1_num, 'z2_num', z2_num, 'z3_num', z3_num, '
    z4_num', z4_num);
129 setValues(handles, 'z1_num_last', z1_num_last, 'z2_num_last', z2_num_last, '
    z3_num_last', z3_num_last, 'z4_num_last', z4_num_last);
130 setValues(handles, 'z1_net', z1_net, 'z2_net', z2_net, 'z3_net', z3_net, '
```

```
    z4_net', z4_net);
131 setValues(handles, 'z1_net_last', z1_net_last, 'z2_net_last', z2_net_last, '
    z3_net_last', z3_net_last, 'z4_net_last', z4_net_last);
132 setValues(handles, 'z1_avg', z1_avg, 'z2_avg', z2_avg, 'z3_avg', z3_avg, '
    z4_avg', z4_avg);
133 setValues(handles, 'z1_avg_last', z1_avg_last, 'z2_avg_last', z2_avg_last, '
    z3_avg_last', z3_avg_last, 'z4_avg_last', z4_avg_last);
134 setValues(handles, 'cost1_best', cost1_best, 'cost2_best', cost2_best, '
    cost3_best', cost3_best, 'cost4_best', cost4_best);
135 setValues(handles, 'votes1_good_w', votes1_good_w, 'votes2_good_w',
    votes2_good_w, 'votes3_good_w', votes3_good_w, 'votes4_good_w',
    votes4_good_w);
136 setValues(handles, 'votes1_bad_w', votes1_bad_w, 'votes2_bad_w', votes2_bad_w,
     'votes3_bad_w', votes3_bad_w, 'votes4_bad_w', votes4_bad_w);
137 setValues(handles, 'votes1_good_tot', votes1_good_tot, 'votes2_good_tot',
    votes2_good_tot, 'votes3_good_tot', votes3_good_tot, 'votes4_good_tot',
    votes4_good_tot);
138 setValues(handles, 'votes1_bad_tot', votes1_bad_tot, 'votes2_bad_tot',
    votes2_bad_tot, 'votes3_bad_tot', votes3_bad_tot, 'votes4_bad_tot',
    votes4_bad_tot);
139 setValues(handles, 'prevUpdate1', prevUpdate1, 'prevUpdate2', prevUpdate2, '
    prevUpdate3', prevUpdate3, 'prevUpdate4', prevUpdate4);
140 setValues(handles, 'prevError1', prevError1, 'prevError2', prevError2, '
    prevError3', prevError3, 'prevError4', prevError4);
141 setValues(handles, 'membest1', membest1, 'membest2', membest2, 'membest3',
    membest3, 'membest4', membest4);
142 setValues(handles, 'cost1', cost1, 'cost2', cost2, 'cost3', cost3, 'cost4',
    cost4);
143 setValues(handles, 'w1_hat_dot', w1_hat_dot, 'w2_hat_dot', w2_hat_dot, '
    w3_hat_dot', w3_hat_dot, 'w4_hat_dot', w4_hat_dot);
144 setValues(handles, 'w1_hat_dot_last', w1_hat_dot_last, 'w2_hat_dot_last',
    w2_hat_dot_last, 'w3_hat_dot_last', w3_hat_dot_last, 'w4_hat_dot_last',
```

```
    w4_hat_dot_last);
% Calculate control signals
g = 9.8;
u(1) = -G1 * (p1_hat + w1_hat) - K(1) * zeta(1) + g * mass/(b * cosd(roll) *
    cosd(pitch));
u(2) = -G2 * (p2_hat + w2_hat) - K(2) * zeta(2);
u(3) = -G3 * (p3_hat + w3_hat) - K(3) * zeta(3);
u(4) = -G4 * (p4_hat + w4_hat) - K(4) * zeta(4);
end
```

A.5 CalculateVoteCmac.m

```
function [gamma, loc, changed, loc_last, loc_secLast] = calculateVoteCmac(
    handles, inputs, num_inputs, minstate, maxstate, loc, loc_last,
    loc_secLast, changed)
% --- Function to return normalized CMAC gamma values and memory locations
[m, q, size] = getValues(handles, 'm', 'q', 'mem_size');
[hashtable, offset] = getValues(handles, 'hashtable', 'offset');
gamma = zeros(1, m);
norm_inputs = (inputs(1:num_inputs)-minstate(1:num_inputs))./(maxstate(1:
    num_inputs)-minstate(1:num_inputs));
norm_inputs = min(norm_inputs, ones(1, num_inputs));
norm_inputs = max(norm_inputs, zeros(1, num_inputs));
cell = zeros(1, num_inputs);
for j = 1:m
    gamma(j) = 1;
    locations_total = 0;
    for i = 1:num_inputs
        place = norm_inputs(i) * (q - 1) + offset(i, j);
        cell(i) = floor(place);
        h = place - cell(i);
        func = 16 * (h*h  -2.0*h*h*h + h*h*h*h);
```

```
18          gamma(j) = gamma(j) * func;
19          locations_total = locations_total + hashtable(cell(i) + 1 + q*(j-1) +
    q*m*(i-1));
20      end
21      oldlocations = loc(j);
22      loc(j) = int32(floor(mod(locations_total, size) + 1 ));  % add 1 for
    matlab 1 based cells;
23      if (loc(j) ~= oldlocations)
24          loc_secLast(j) = loc_last(j);
25          loc_last(j) = oldlocations;
26          changed(j) = 1;
27      else
28          changed(j) = 0;
29      end;
30  end
31  end
```

Appendix B

Data Logging Code

B.1 updateSaveFiles.m

```
1 function updateSaveFiles(handles, u)
2 % --- Function to save variable data to files
3 t = getValues(handles, 't');   % Get current and elapsed time
4 [x_d, y_d, z_d, roll_d, pitch_d, yaw_d] =...
5     getValues(handles, 'x_d', 'y_d', 'z_d', 'roll_d', 'pitch_d', 'yaw_d'); %
    Get current desired values
6 [x, y, z, roll, pitch, yaw] =...
7     getValues(handles, 'x', 'y', 'z', 'roll', 'pitch', 'yaw'); % Get current
    actual values
8 [x_d_dot, y_d_dot, z_d_dot, roll_d_dot, pitch_d_dot, yaw_d_dot] =...
9     getValues(handles, 'x_d_dot', 'y_d_dot', 'z_d_dot', 'roll_d_dot', '
    pitch_d_dot', 'yaw_d_dot'); % Get current desired values
10 [x_dot, y_dot, z_dot, roll_dot, pitch_dot, yaw_dot] =...
11     getValues(handles, 'x_dot', 'y_dot', 'z_dot', 'roll_dot', 'pitch_dot', '
    yaw_dot'); % Get current actual values
12 [x_ddot, y_ddot, z_ddot, roll_ddot, pitch_ddot, yaw_ddot] =...
13     getValues(handles, 'x_ddot', 'y_ddot', 'z_ddot', 'roll_ddot', 'pitch_ddot'
    , 'yaw_ddot'); % Get current actual values
14 [x_dist, y_dist, z_dist, roll_dist, pitch_dist, yaw_dist] =...
15     getValues(handles, 'x_dist', 'y_dist', 'z_dist', 'roll_dist', 'pitch_dist'
    , 'yaw_dist'); % Get disturbance values
16 [x_dist_dot, y_dist_dot, z_dist_dot, roll_dist_dot, pitch_dist_dot,
    yaw_dist_dot] =...
17     getValues(handles, 'x_dist_dot', 'y_dist_dot', 'z_dist_dot', '
    roll_dist_dot', 'pitch_dist_dot', 'yaw_dist_dot'); % Get disturbance
```

```
    values
18 [x_dist_ddot, y_dist_ddot, z_dist_ddot, roll_dist_ddot, pitch_dist_ddot,
    yaw_dist_ddot] =...
19     getValues(handles, 'x_dist_ddot', 'y_dist_ddot', 'z_dist_ddot', '
    roll_dist_ddot', 'pitch_dist_ddot', 'yaw_dist_ddot'); % Get disturbance
    values
20 speed = getMotorSpeeds(handles);
21 [position_file, velocity_file, acceleration_file, motor_file, disturbance_file
    , control_file] =...
22     getValues(handles, 'position_file_id', 'velocity_file_id', '
    acceleration_file_id',...
23     'motor_file_id', 'disturbance_file_id', 'control_file_id');
24 fprintf(control_file, '%3.3f, %d, %d, %d, %d\r\n', t(1), u(1), u(2), u(3), u
    (4));
25 fprintf(motor_file, '%3.3f, %d, %d, %d, %d\r\n', t(1), speed(1), speed(2),
    speed(3), speed(4));
26 fprintf(position_file, '%3.3f, %2.4f, %2.4f, %2.4f, %2.4f, %2.4f, %2.4f
    , %2.4f, %2.4f, %2.4f, %2.4f, %2.4f\r\n',...
27     t(1), x, y, z, roll, pitch, yaw, x_d, y_d, z_d, roll_d, pitch_d, yaw_d);
28 fprintf(velocity_file, '%3.3f, %4.2f, %4.2f, %4.2f, %4.2f, %4.2f, %4.2f
    , %4.2f, %4.2f, %4.2f, %4.2f, %4.2f\r\n',...
29     t(1), x_dot, y_dot, z_dot, roll_dot, pitch_dot, yaw_dot, x_d_dot, y_d_dot,
     z_d_dot, roll_d_dot, pitch_d_dot, yaw_d_dot);
30 fprintf(acceleration_file, '%3.3f, %5.1f, %5.1f, %5.1f, %5.1f, %5.1f, %5.1f\r\
    n',...
31     t(1), x_ddot, y_ddot, z_ddot, roll_ddot, pitch_ddot, yaw_ddot);
32 fprintf(disturbance_file, '%3.3f, %2.4f, %2.4f, %2.4f, %2.4f, %2.4f, %2.4f ',
    ...
33     t(1), x_dist, y_dist, z_dist, roll_dist, pitch_dist, yaw_dist);
34 fprintf(disturbance_file, '%4.2f, %4.2f, %4.2f, %4.2f, %4.2f, %4.2f ', ...
35     x_dist_dot, y_dist_dot, z_dist_dot, roll_dist_dot, pitch_dist_dot,
    yaw_dist_dot);
```

```
36 fprintf(disturbance_file, '%5.1f, %5.1f, %5.1f, %5.1f, %5.1f, %5.1f\r\n',...
37     x_dist_ddot, y_dist_ddot, z_dist_ddot, roll_dist_ddot, pitch_dist_ddot,
     yaw_dist_ddot);
38 control_scheme = getValues(handles, 'control_scheme');
39 if(~isempty(strfind(control_scheme, 'CMAC')))
40     w_hat_1 = getValues(handles, 'w_hat_1');
41     w_hat_2 = getValues(handles, 'w_hat_2');
42     w_hat_3 = getValues(handles, 'w_hat_3');
43     w_hat_4 = getValues(handles, 'w_hat_4');
44     max_weight_file = getValues(handles, 'max_weight_file_id');
45     fprintf(max_weight_file, '%3.3f, %f, %f, %f, %f\r\n', t(1), max(w_hat_1),
     max(w_hat_2), max(w_hat_3), max(w_hat_4));
46     norm_weight_file = getValues(handles, 'norm_weight_file_id');
47     fprintf(norm_weight_file, '%3.3f, %f, %f, %f, %f\r\n', t(1), norm(w_hat_1)
     , norm(w_hat_2), norm(w_hat_3), norm(w_hat_4));
48     if(strcmp(control_scheme, 'CMAC (Alt Weights)'))
49         p_hat_1 = getValues(handles, 'p_hat_1');
50         p_hat_2 = getValues(handles, 'p_hat_2');
51         p_hat_3 = getValues(handles, 'p_hat_3');
52         p_hat_4 = getValues(handles, 'p_hat_4');
53         max_alt_weight_file = getValues(handles, 'max_alt_weight_file_id');
54         fprintf(max_alt_weight_file, '%3.3f, %f, %f, %f, %f\r\n', t(1), max(
     p_hat_1), max(p_hat_2), max(p_hat_3), max(p_hat_4));
55         norm_alt_weight_file = getValues(handles, 'norm_alt_weight_file_id');
56         fprintf(norm_alt_weight_file, '%3.3f, %f, %f, %f, %f\r\n', t(1), norm(
     p_hat_1), norm(p_hat_2), norm(p_hat_3), norm(p_hat_4));
57     elseif(strcmp(control_scheme, 'CMAC (Intro Weights)'))
58         p_hat_1 = getValues(handles, 'p_hat_1');
59         p_hat_2 = getValues(handles, 'p_hat_2');
60         p_hat_3 = getValues(handles, 'p_hat_3');
61         p_hat_4 = getValues(handles, 'p_hat_4');
62         max_perm_weight_file = getValues(handles, 'max_perm_weight_file_id');
```

```
63          fprintf(max_perm_weight_file, '%3.3f, %f, %f, %f, %f\r\n', t(1), max(
    p_hat_1), max(p_hat_2), max(p_hat_3), max(p_hat_4));
64          norm_perm_weight_file = getValues(handles, 'norm_perm_weight_file_id')
    ;
65          fprintf(norm_perm_weight_file, '%3.3f, %f, %f, %f, %f\r\n', t(1), norm
    (p_hat_1), norm(p_hat_2), norm(p_hat_3), norm(p_hat_4));
66      end
67 end
68 end
```

B.2 QuadrotorFigures.m

```
1 function QuadrotorFigures(file_directory)
2 % --- Function to plot saved data files
3 simulation_info = '';   time = 0;   line_width = 1.5;
4 x = 0;      y = 0;      z = 0;      roll = 0;   pitch = 0;      yaw = 0;
5 x_d = 0;    y_d = 0;    z_d = 0;    roll_d = 0; pitch_d = 0;    yaw_d = 0;
6 u1 = 0;          u2 = 0;          u3 = 0;          u4 = 0;
7 speed1 = 0;      speed2 = 0;      speed3 = 0;      speed4 = 0;
8 w1_max = 0;      w2_max = 0;      w3_max = 0;      w4_max = 0;
9 w1_norm = 0;     w2_norm = 0;     w3_norm = 0;     w4_norm = 0;
10 p1_max = 0;     p2_max = 0;      p3_max = 0;      p4_max = 0;
11 p1_norm = 0;    p2_norm = 0;     p3_norm = 0;     p4_norm = 0;
12 simulation_file = strcat(file_directory, '/simulationData.txt');
13 position_file = strcat(file_directory, '/positionData.txt');
14 control_file = strcat(file_directory, '/controlData.txt');
15 motor_file = strcat(file_directory, '/motorData.txt');
16 max_weight_file = strcat(file_directory, '/maxWeightData.txt');
17 max_alt_weight_file = strcat(file_directory, '/maxAltWeightData.txt');
18 max_perm_weight_file = strcat(file_directory, '/maxPermWeightData.txt');
19 norm_weight_file = strcat(file_directory, '/normWeightData.txt');
20 norm_alt_weight_file = strcat(file_directory, '/normAltWeightData.txt');
```

```
21 norm_perm_weight_file = strcat(file_directory, '/normPermWeightData.txt');
22 input_file = strcat(file_directory, '/inputData.txt');
23
24 if(exist(simulation_file, 'file') == 2)
25     simulation_id = fopen(simulation_file);
26     data = textscan(simulation_id, '%s %s %s', 'delimiter', ',', 'headerLines'
       , 1);
27     simulation_info = strcat(' - ', data{1}(1), ' (', data{2}(1), ')', ' Track
       : ', data{3}(1));
28     fclose(simulation_id);
29 end
30
31 if(exist(position_file, 'file') == 2)
32     position_id = fopen(position_file);
33     data = textscan(position_id, '%f %f %f %f %f %f %f %f %f %f %f %f %f %f', '
       delimiter', ',', 'headerLines', 1);
34     time = data{1};
35     x = data{2}; y = data{3}; z = data{4}; roll = data{5}; pitch = data{6};
       yaw = data{7};
36     x_d = data{8}; y_d = data{9}; z_d = data{10}; roll_d = data{11}; pitch_d =
       data{12}; yaw_d = data{13};
37     x_e = x_d - x;       roll_e = roll_d - roll;
38     y_e = y_d - y;       pitch_e = pitch_d - pitch;
39     z_e = z_d - z;       yaw_e = yaw_d - yaw;
40     x_r = zeros(size(x));    roll_r = zeros(size(roll));
41     y_r = zeros(size(y));    pitch_r = zeros(size(pitch));
42     z_r = zeros(size(z));    yaw_r = zeros(size(yaw));
43     for i = 1:length(time)
44         x_r(i) = sqrt(mean(x_e(1:i).^2));
45         y_r(i) = sqrt(mean(y_e(1:i).^2));
46         z_r(i) = sqrt(mean(z_e(1:i).^2));
47         roll_r(i) = sqrt(mean(roll_e(1:i).^2));
```

```
48          pitch_r(i) = sqrt(mean(pitch_e(1:i).^2));
49          yaw_r(i) = sqrt(mean(yaw_e(1:i).^2));
50      end
51      fclose(position_id);
52  end
53
54  dist = zeros(size(time));
55  if(exist(input_file, 'file') == 2)
56      input_id = fopen(input_file);
57      data = textscan(input_id, '%f %s', 'delimiter', ',', 'headerLines', 1);
58      for i = 1:length(data{1})
59          for j = 1:length(time)
60              if(time(j) >= data{1}(i))
61                  if(strcmp(char(data{2}(i)), 'o') || strcmp(char(data{2}(i)), '
    p'))
62                      dist(j) = 1;
63                      break;
64                  end
65              end
66          end
67      end
68      fclose(input_id);
69  end
70
71  if(exist(control_file, 'file') == 2)
72      control_id = fopen(control_file);
73      data = textscan(control_id, '%f %f %f %f %f', 'delimiter', ',', '
    headerLines', 1);
74      time = data{1};
75      u1 = data{2}; u2 = data{3}; u3 = data{4}; u4 = data{5};
76      fclose(control_id);
77  end
```

```
78
79 if(exist(motor_file, 'file') == 2)
80     motor_id = fopen(motor_file);
81     data = textscan(motor_id, '%f %f %f %f %f', 'delimiter', ',', 'headerLines
       ', 1);
82     time = data{1};
83     speed1 = data{2}; speed2 = data{3}; speed3 = data{4}; speed4 = data{5};
84     fclose(motor_id);
85 end
86
87 if(exist(max_weight_file, 'file') == 2)
88     max_weight_id = fopen(max_weight_file);
89     data = textscan(max_weight_id, '%f %f %f %f %f', 'delimiter', ',', '
       headerLines', 1);
90     time = data{1};
91     w1_max = data{2}; w2_max = data{3}; w3_max = data{4}; w4_max = data{5};
92     fclose(max_weight_id);
93 end
94
95 if(exist(norm_weight_file, 'file') == 2)
96     norm_weight_id = fopen(norm_weight_file);
97     data = textscan(norm_weight_id, '%f %f %f %f %f', 'delimiter', ',', '
       headerLines', 1);
98     time = data{1};
99     w1_norm = data{2}; w2_norm = data{3}; w3_norm = data{4}; w4_norm = data
       {5};
100    fclose(norm_weight_id);
101 end
102
103 if(exist(max_alt_weight_file, 'file') == 2)
104    max_alt_weight_id = fopen(max_alt_weight_file);
105    data = textscan(max_alt_weight_id, '%f %f %f %f %f', 'delimiter', ',', '
```

```
    headerLines', 1);
106     time = data{1};
107     p1_max = data{2}; p2_max = data{3}; p3_max = data{4}; p4_max = data{5};
108     fclose(max_alt_weight_id);
109 elseif(exist(max_perm_weight_file, 'file') == 2)
110     max_perm_weight_id = fopen(max_perm_weight_file);
111     data = textscan(max_perm_weight_id, '%f %f %f %f %f', 'delimiter', ',', '
    headerLines', 1);
112     time = data{1};
113     p1_max = data{2}; p2_max = data{3}; p3_max = data{4}; p4_max = data{5};
114     fclose(max_perm_weight_id);
115 end
116
117 if(exist(norm_alt_weight_file, 'file') == 2)
118     norm_alt_weight_id = fopen(norm_alt_weight_file);
119     data = textscan(norm_alt_weight_id, '%f %f %f %f %f', 'delimiter', ',', '
    headerLines', 1);
120     time = data{1};
121     p1_norm = data{2}; p2_norm = data{3}; p3_norm = data{4}; p4_norm = data
    {5};
122     fclose(norm_alt_weight_id);
123 elseif(exist(norm_perm_weight_file, 'file') == 2)
124     norm_perm_weight_id = fopen(norm_perm_weight_file);
125     data = textscan(norm_perm_weight_id, '%f %f %f %f %f', 'delimiter', ',', '
    headerLines', 1);
126     time = data{1};
127     p1_norm = data{2}; p2_norm = data{3}; p3_norm = data{4}; p4_norm = data
    {5};
128     fclose(norm_perm_weight_id);
129 end
130
131 figure('Name', strcat(file_directory, ' X', simulation_info{1}), 'NumberTitle'
```

```
    , 'off');
132 subplot(2, 1, 1);   % X Desired and Actual vs Time
133 plot(time, x, 'b-', time, x_d, 'r--', 'LineWidth', line_width);
134 for i = 1:length(time)
135     if(dist(i) == 1)
136         line([time(i), time(i)], get(gca,'YLim'), 'Color', 'k', 'LineStyle', '
    -.', 'LineWidth', line_width);
137     end
138 end
139 [max_value, max_index] = max(abs(x));
140 text(time(max_index), x(max_index), strcat(num2str(max_value, '%1.3f'), 'm'),
    'Fontsize', 8, 'HorizontalAlignment', 'Right');
141 title('X Position', 'fontsize', 14);
142 xlabel('Time (s)', 'fontsize', 12);
143 ylabel('X (m)', 'fontsize', 12);
144 h_lgd = legend('Actual', 'Desired', 'Location', 'NorthEast');
145 set(h_lgd, 'Fontsize', 8);
146 subplot(2, 1, 2);   % X RMS Error vs Time
147 plot(time, x_r, 'g-', 'LineWidth', line_width);
148 text(time(end), x_r(end), strcat(num2str(x_r(end), '%1.3f'), 'm'), 'Fontsize',
     8, 'HorizontalAlignment', 'Right');
149 title('X Error', 'fontsize', 14);
150 xlabel('Time (s)', 'fontsize', 12);
151 ylabel('Error (m)', 'fontsize', 12);
152 h_lgd = legend('RMS Error', 'Location', 'NorthEast');
153 set(h_lgd, 'Fontsize', 8);
154
155 figure('Name', strcat(file_directory, ' Y', simulation_info{1}), 'NumberTitle'
    , 'off');
156 subplot(2, 1, 1);   % Y Desired and Actual vs Time
157 plot(time, y, 'b-', time, y_d, 'r--', 'LineWidth', line_width);
158 for i = 1:length(time)
```

```
159     if(dist(i) == 1)
160         line([time(i), time(i)], get(gca,'YLim'), 'Color', 'k', 'LineStyle', '
        -.', 'LineWidth', line_width);
161     end
162 end
163 [max_value, max_index] = max(abs(y));
164 text(time(max_index), y(max_index), strcat(num2str(max_value, '%1.3f'), 'm'),
        'Fontsize', 8, 'HorizontalAlignment', 'Right');
165 title('Y Position', 'fontsize', 14);
166 xlabel('Time (s)', 'fontsize', 12);
167 ylabel('Y (m)', 'fontsize', 12);
168 h_lgd = legend('Actual', 'Desired', 'Location', 'NorthEast');
169 set(h_lgd, 'Fontsize', 8);
170 subplot(2, 1, 2);   % Y RMS Error vs Time
171 plot(time, y_r, 'g-', 'LineWidth', line_width);
172 text(time(end), y_r(end), strcat(num2str(y_r(end), '%1.3f'), 'm'), 'Fontsize',
        8, 'HorizontalAlignment', 'Right');
173 title('Y Error', 'fontsize', 14);
174 xlabel('Time (s)', 'fontsize', 12);
175 ylabel('Error (m)', 'fontsize', 12);
176 h_lgd = legend('RMS Error', 'Location', 'NorthEast');
177 set(h_lgd, 'Fontsize', 8);
178
179 figure('Name', strcat(file_directory, ' Z', simulation_info{1}), 'NumberTitle'
        , 'off');
180 subplot(2, 1, 1);   % Z Desired and Actual vs Time
181 plot(time, z, 'b-', time, z_d, 'r--', 'LineWidth', line_width);
182 for i = 1:length(time)
183     if(dist(i) == 1)
184         line([time(i), time(i)], get(gca,'YLim'), 'Color', 'k', 'LineStyle', '
        -.', 'LineWidth', line_width);
185     end
```

```
186 end
187 [max_value, max_index] = max(z);
188 text(time(max_index), z(max_index), strcat(num2str(max_value, '%1.3f'), 'm'),
      'Fontsize', 8, 'HorizontalAlignment', 'Right');
189 title('Z Position', 'fontsize', 14);
190 xlabel('Time (s)', 'fontsize', 12);
191 ylabel('Z (m)', 'fontsize', 12);
192 h_lgd = legend('Actual', 'Desired', 'Location', 'NorthEast');
193 set(h_lgd, 'Fontsize', 8);
194 subplot(2, 1, 2);   % Z RMS Error vs Time
195 plot(time, z_r, 'g-', 'LineWidth', line_width);
196 text(time(end), z_r(end), strcat(num2str(z_r(end), '%1.3f'), 'm'), 'Fontsize',
      8, 'HorizontalAlignment', 'Right');
197 title('Z Error', 'fontsize', 14);
198 xlabel('Time (s)', 'fontsize', 12);
199 ylabel('Error (m)', 'fontsize', 12);
200 h_lgd = legend('RMS Error', 'Location', 'NorthEast');
201 set(h_lgd, 'Fontsize', 8);
202
203 figure('Name', strcat(file_directory, ' Roll', simulation_info{1}), '
      NumberTitle', 'off');
204 subplot(2, 1, 1);   % Roll Desired and Actual vs Time
205 plot(time, roll, 'b-', time, roll_d, 'r--', 'LineWidth', line_width);
206 for i = 1:length(time)
207     if(dist(i) == 1)
208         line([time(i), time(i)], get(gca,'YLim'), 'Color', 'k', 'LineStyle', '
      -.', 'LineWidth', line_width);
209     end
210 end
211 [max_value, max_index] = max(abs(roll));
212 text(time(max_index), roll(max_index), strcat(num2str(max_value, '%1.3f'), ' '
      ), 'Fontsize', 8, 'HorizontalAlignment', 'Right');
```

```
213 title('Roll Orientation', 'fontsize', 14);
214 xlabel('Time (s)', 'fontsize', 12);
215 ylabel('Roll ( )', 'fontsize', 12);
216 h_lgd = legend('Actual', 'Desired', 'Location', 'NorthEast');
217 set(h_lgd, 'Fontsize', 8);
218 subplot(2, 1, 2);   % Roll RMS Error vs Time
219 plot(time, roll_r, 'g-', 'LineWidth', line_width);
220 text(time(end), roll_r(end), strcat(num2str(roll_r(end), '%1.3f'), ' '), '
      Fontsize', 8, 'HorizontalAlignment', 'Right');
221 title('Roll Error', 'fontsize', 14);
222 xlabel('Time (s)', 'fontsize', 12);
223 ylabel('Error ( )', 'fontsize', 12);
224 h_lgd = legend('RMS Error', 'Location', 'NorthEast');
225 set(h_lgd, 'Fontsize', 8);
226
227 figure('Name', strcat(file_directory, ' Pitch', simulation_info{1}), '
      NumberTitle', 'off');
228 subplot(2, 1, 1);   % Pitch Desired and Actual vs Time
229 plot(time, pitch, 'b-', time, pitch_d, 'r--', 'LineWidth', line_width);
230 for i = 1:length(time)
231     if(dist(i) == 1)
232         line([time(i), time(i)], get(gca,'YLim'), 'Color', 'k', 'LineStyle', '
      -.', 'LineWidth', line_width);
233     end
234 end
235 [max_value, max_index] = max(abs(pitch));
236 text(time(max_index), pitch(max_index), strcat(num2str(max_value, '%1.3f'), '
      '), 'Fontsize', 8, 'HorizontalAlignment', 'Right');
237 title('Pitch Orientation', 'fontsize', 14);
238 xlabel('Time (s)', 'fontsize', 12);
239 ylabel('Pitch ( )', 'fontsize', 12);
240 h_lgd = legend('Actual', 'Desired', 'Location', 'NorthEast');
```

```
241 set(h_lgd, 'Fontsize', 8);
242 subplot(2, 1, 2);    % Pitch RMS Error vs Time
243 plot(time, pitch_r, 'g-', 'LineWidth', line_width);
244 text(time(end), pitch_r(end), strcat(num2str(pitch_r(end), '%1.3f'), ' '), '
      Fontsize', 8, 'HorizontalAlignment', 'Right');
245 title('Pitch Error', 'fontsize', 14);
246 xlabel('Time (s)', 'fontsize', 12);
247 ylabel('Error ( )', 'fontsize', 12);
248 h_lgd = legend('RMS Error', 'Location', 'NorthEast');
249 set(h_lgd, 'Fontsize', 8);
250
251 figure('Name', strcat(file_directory, ' Yaw Orientation', simulation_info{1}),
       'NumberTitle', 'off');
252 subplot(2, 1, 1);    % Yaw Desired and Actual vs Time
253 plot(time, yaw, 'b-', time, yaw_d, 'r--', 'LineWidth', line_width);
254 for i = 1:length(time)
255     if(dist(i) == 1)
256         line([time(i), time(i)], get(gca,'YLim'), 'Color', 'k', 'LineStyle', '
      -.', 'LineWidth', line_width);
257     end
258 end
259 [max_value, max_index] = max(abs(yaw));
260 text(time(max_index), yaw(max_index), strcat(num2str(max_value, '%1.3f'), ' ')
      , 'Fontsize', 8, 'HorizontalAlignment', 'Right');
261 title('Yaw Orientation', 'fontsize', 14);
262 xlabel('Time (s)', 'fontsize', 12);
263 ylabel('Yaw ( )', 'fontsize', 12);
264 h_lgd = legend('Actual', 'Desired', 'Location', 'NorthEast');
265 set(h_lgd, 'Fontsize', 8);
266 subplot(2, 1, 2);    % Yaw RMS Error vs Time
267 plot(time, yaw_r, 'g-', 'LineWidth', line_width);
268 text(time(end), yaw_r(end), strcat(num2str(yaw_r(end), '%1.3f'), ' '), '
```

```
    Fontsize', 8, 'HorizontalAlignment', 'Right');
title('Yaw Error', 'fontsize', 14);
xlabel('Time (s)', 'fontsize', 12);
ylabel('Error ( )', 'fontsize', 12);
h_lgd = legend('RMS Error', 'Location', 'NorthEast');
set(h_lgd, 'Fontsize', 8);

% Figure 4.5: 3D Position
figure('Name', strcat(file_directory, ' Position', simulation_info{1}), '
    NumberTitle', 'off');
plot3(x, y, z, 'b-', x_d, y_d, z_d, 'r:', 'LineWidth', line_width);
title('Position', 'fontsize', 14);
xlabel('X (m)', 'fontsize', 12);
ylabel('Y (m)', 'fontsize', 12);
zlabel('Z (m)', 'fontsize', 12);
h_lgd = legend('Actual', 'Desired', 'Location', 'NorthWest');
set(h_lgd, 'Fontsize', 8);

% Figure 5: control (u/motor 1-4)
figure('Name', strcat(file_directory, ' Control', simulation_info{1}), '
    NumberTitle', 'off');
subplot(2, 1, 1);
plot(time, u1, 'r-', time, u2, 'g-', time, u3, 'm-', time, u4, 'c-', '
    LineWidth', line_width);
title('Control Signals', 'fontsize', 14);
xlabel('Time (s)', 'fontsize', 12);
ylabel('Control', 'fontsize', 12);
h_lgd = legend('U1', 'U2', 'U3', 'U4', 'Location', 'NorthWest');
set(h_lgd, 'Fontsize', 8);
subplot(2, 1, 2);
plot(time, speed1, 'r-', time, speed2, 'g-', time, speed3, 'm-', time, speed4,
    'c-',...
```

```
296     time, mean([speed1, speed2, speed3, speed4], 2), 'b:', 'LineWidth',
        line_width);
297 text(time(end), mean([speed1(end), speed2(end), speed3(end), speed4(end)]),...
298     strcat(num2str(mean(mean([speed1, speed2, speed3, speed4], 2)), '%1.0f'),
        ' /s'), 'Fontsize', 8, 'HorizontalAlignment', 'Right');
299 title('Rotor Speeds', 'fontsize', 14);
300 xlabel('Time (s)', 'fontsize', 12);
301 ylabel('Speed ( /s)', 'fontsize', 12);
302 h_lgd = legend('Rotor 1', 'Rotor 2', 'Rotor 3', 'Rotor 4', 'Mean', 'Location',
        'NorthWest');
303 set(h_lgd, 'Fontsize', 8);
304
305 if(~isempty(strfind(simulation_info{1}, 'CMAC')))
306     figure('Name', strcat(file_directory, ' Active Weights', simulation_info
        {1}), 'NumberTitle', 'off');
307     subplot(2, 1, 1);   % Active Weights (max/norm) vs Time
308     hold on;
309     plot(time, w1_max, 'r:', time, w2_max, 'g:', time, w3_max, 'm:', time,
        w4_max, 'c:', 'LineWidth', line_width);
310     plot(time, w1_norm, 'r--', time, w2_norm, 'g--', time, w3_norm, 'm--',
        time, w4_norm, 'c--', 'LineWidth', line_width);
311     hold off;
312     for i = 1:length(time)
313         if(dist(i) == 1)
314             line([time(i), time(i)], get(gca,'YLim'), 'Color', 'k', 'LineStyle
        ', '-.', 'LineWidth', line_width);
315         end
316     end
317     title('Active Weight Values', 'fontsize', 14);
318     xlabel('Time (s)', 'fontsize', 12);
319     ylabel('Value', 'fontsize', 12);
320     h_lgd = legend('Max W1', 'Max W2', 'Max W3', 'Max W4',...
```

```
321         'Norm W1', 'Norm W2', 'Norm W3', 'Norm W4', 'Location', 'NorthWest');
322     set(h_lgd, 'Fontsize', 8);
323     subplot(2, 1, 2);   % Permanent Weights (max/norm) vs Time
324     if(~isempty(strfind(simulation_info{1}, 'Weights')))
325         hold on;
326         plot(time, p1_max, 'r:', time, p2_max, 'g:', time, p3_max, 'm:', time,
        p4_max, 'c:', 'LineWidth', line_width);
327         plot(time, p1_norm, 'r--', time, p2_norm, 'g--', time, p3_norm, 'm--',
        time, p4_norm, 'c--', 'LineWidth', line_width);
328         hold off;
329         for i = 1:length(time)
330             if(dist(i) == 1)
331                 line([time(i), time(i)], get(gca,'YLim'), 'Color', 'k', '
      LineStyle', '-.', 'LineWidth', line_width);
332             end
333         end
334         title('Permanent Weight Values', 'fontsize', 14);
335         xlabel('Time (s)', 'fontsize', 12);
336         ylabel('Value', 'fontsize', 12);
337         h_lgd = legend('Max W1', 'Max W2', 'Max W3', 'Max W4',...
338         'Norm W1', 'Norm W2', 'Norm W3', 'Norm W4', 'Location', 'NorthWest');
339         set(h_lgd, 'Fontsize', 8);
340     end
341 end
342 end
```

Appendix C

AR.Drone 2.0 Code

C.1 QuadrotorCommand.m Excerpts

```
function droneTakeOff(handles)
% --- Function to take off using built-in Drone algorithm
tx_port = getappdata(handles.qrCommandGui, 'tx_port');
% Send takeoff command until drone flying flag has been set, provided flying
    command still set
while((bitget(getappdata(handles.qrCommandGui, 'drone_state'), 1) == 0))
    cmd_num = getCommandNumber(handles.qrCommandGui);  % Get command number
    TAKEOFF_CODE = 2^9 + 2^18 + 2^20 + 2^22 + 2^24 + 2^28;  % Set bits as per
    manual
    AR_TAKEOFF = sprintf('AT*REF=%d,%d\r', cmd_num, TAKEOFF_CODE);
    fprintf(tx_port, AR_TAKEOFF);
    if(~calculateDroneValues(handles.qrCommandGui))
        setValues(handles.qrCommandGui, 'flying', false);
        return;
    end
    updateDroneData(handles);
    updateGuiData(handles);
    if(getValues(handles.qrCommandGui, 'flying') == true)
        break;
    end
end

function droneLand(handles)
% --- Function to land using built-in Drone algorithm
tx_port = getappdata(handles.qrCommandGui, 'tx_port');
```

```
24 while(bitget(getappdata(handles.qrCommandGui, 'drone_state'), 1) == 1)
25     cmd_num = getCommandNumber(handles.qrCommandGui);  % Get command number
26     LAND_CODE = 2^18 + 2^20 + 2^22 + 2^24 + 2^28;  % Send land command
27     AR_LAND = sprintf('AT*REF=%d,%d\r', cmd_num, LAND_CODE);
28     fprintf(tx_port, AR_LAND);
29     if(~calculateDroneValues(handles.qrCommandGui))
30         setValues(handles.qrCommandGui, 'flying', false);
31         return;
32     end
33     updateDroneData(handles);
34     updateGuiData(handles);
35 end
36
37 function droneToggleEmergency(handles)
38 % --- Function to toggle emergency command to drone
39 tx_port = getappdata(handles.qrCommandGui, 'tx_port');
40 calculateDroneValues(handles.qrCommandGui);
41 state = getValues(handles.qrCommandGui, 'drone_state');
42 if(getValues(handles.qrCommandGui, 'emergency_input'))
43     set(handles.eTxt, 'Foreground', [1 0 0]);
44     if(bitget(state, 32) == 0)  % Toggle drone emergency state
45         cmd_num = getCommandNumber(handles.qrCommandGui);  % Get command
    number
46         E_TOGGLE_CODE = 2^8 + 2^18 + 2^20 + 2^22 + 2^24 + 2^28;  % Toggle
    emergency flag
47         E_TOGGLE = sprintf('AT*REF=%d,%d\r', cmd_num, E_TOGGLE_CODE);
48         fprintf(tx_port, E_TOGGLE);
49     end
50 else
51     set(handles.eTxt, 'Foreground', [0 1 0]);
52     if(bitget(state, 32) == 1)  % Toggle drone emergency state
53         cmd_num = getCommandNumber(handles.qrCommandGui);  % Get command
```

```
    number
54          E_TOGGLE_CODE = 2^8 + 2^18 + 2^20 + 2^22 + 2^24 + 2^28;  % Toggle
    emergency flag
55          E_TOGGLE = sprintf('AT*REF=%d,%d\r', cmd_num, E_TOGGLE_CODE);
56          fprintf(tx_port, E_TOGGLE);
57      end
58      cmd_num = getCommandNumber(handles.qrCommandGui);  % Get command number
59      E_CLEAR_CODE = 2^18 + 2^20 + 2^22 + 2^24 + 2^28;     % Clear emergency flag
60      E_CLEAR = sprintf('AT*REF=%d,%d\r', cmd_num, E_CLEAR_CODE);
61      fprintf(tx_port, E_CLEAR);
62  end
63  calculateDroneValues(handles.qrCommandGui);
64  updateDroneData(handles);
65
66  function droneSetTrim(handles)
67  % --- Function to set trim and calibrate yaw offset while landed
68  tx_port = getappdata(handles.qrCommandGui, 'tx_port');
69  ResetCommandNumber(handles.qrCommandGui);
70  cmd_num = getCommandNumber(handles.qrCommandGui);
71  AR_TRIM = sprintf('AT*FTRIM=%d,\r', cmd_num);
72  fprintf(tx_port, AR_TRIM);
73  fprintf('Trim Set.\n');
74
75  function droneManualTakeoff(handles)
76  % --- Function to manual increase rotor speeds to lift Drone
77  tx_port = getappdata(handles.qrCommandGui, 'tx_port');
78  max_speed = getValues(handles.qrCommandGui, 'drone_max_speed');
79  max_pwm = floor((max_speed - 7500)/41);
80  for pwm = 0:25:min(max_pwm, 300)
81      cmd_num = getCommandNumber(handles.qrCommandGui);
82      MOTOR_CMD = sprintf('AT*PWM=%d,%d,%d,%d,%d\r', cmd_num, pwm, pwm, pwm, pwm
    );
```

```
83      fprintf(tx_port, MOTOR_CMD);
84      if(~calculateDroneValues(handles.qrCommandGui))
85          setValues(handles.qrCommandGui, 'flying', false);
86          return;
87      end
88      if(bitget(getValues(handles.qrCommandGui, 'drone_state'), 31))
89          ResetWatchdog(tx_port, getCommandNumber(handles.qrCommandGui));
90      end
91      updateDroneData(handles);
92      updateGuiData(handles);
93  end
94  if(max_pwm > 300)
95      for pwm = 300:5:max_pwm
96          cmd_num = getCommandNumber(handles.qrCommandGui);
97          MOTOR_CMD = sprintf('AT*PWM=%d,%d,%d,%d,%d\r', cmd_num, pwm, pwm, pwm,
        pwm);
98          fprintf(tx_port, MOTOR_CMD);
99          if(~calculateDroneValues(handles.qrCommandGui))
100             setValues(handles.qrCommandGui, 'flying', false);
101             return;
102         end
103         if(bitget(getValues(handles.qrCommandGui, 'drone_state'), 31))
104             ResetWatchdog(tx_port, getCommandNumber(handles.qrCommandGui));
105         end
106         updateDroneData(handles);
107         updateGuiData(handles);
108     end
109 end
110
111 function droneManualLand(handles)
112 % --- Function to manually reduce rotor speeds to land Drone
113 tx_port = getappdata(handles.qrCommandGui, 'tx_port');
```

```
114 max_speed = getValues(handles.qrCommandGui, 'drone_max_speed');
115 max_pwm = (max_speed - 7500)/41;
116 if(max_pwm > 300)
117     for pwm = max_pwm:-5:300
118         cmd_num = getCommandNumber(handles.qrCommandGui);
119         MOTOR_CMD = sprintf('AT*PWM=%d,%d,%d,%d,%d\r', cmd_num, pwm, pwm, pwm,
        pwm);
120         fprintf(tx_port, MOTOR_CMD);
121         pause(0.1);
122     end
123 end
124 for pwm = min(max_pwm, 300):-25:0
125     cmd_num = getCommandNumber(handles.qrCommandGui);
126     MOTOR_CMD = sprintf('AT*PWM=%d,%d,%d,%d,%d\r', cmd_num, pwm, pwm, pwm, pwm
      );
127     fprintf(tx_port, MOTOR_CMD);
128     pause(0.1);
129 end
```

C.2 StartNavData.m

```
1 function StartNavData(tx_port, rx_port)
2 % --- Function to request transmission of data from Drone
3 ClearDroneBuffer(rx_port);
4 fwrite(rx_port, 1);
5 GetNavData(rx_port);
6 AR_NAV_CONFIG = sprintf('AT*CONFIG=2,\"general:navdata_demo\",\"TRUE\"\r');
7 fprintf(tx_port, AR_NAV_CONFIG);
8 GetNavData(rx_port);
9 AT_CTRL = sprintf('AT*CTRL=0\r');
10 fprintf(tx_port, AT_CTRL);
11 fprintf('Nav Data Started.\n');
```

```
12 end
```

C.3 GetNavData.m

```
1 function [success, header, state, response, data_size, battery, pitch, roll,
      yaw, z, x_dot, y_dot] = GetNavData(rx_port)
2 % --- Function to read and save a set of data from the AR Drone
3 success = false;
4 header = 0;
5 state = 0;
6 response = 0;
7 data_size = 0;
8 battery = 0;
9 pitch = 0;
10 roll = 0;
11 yaw = 0;
12 z = 0;
13 x_dot = 0;
14 y_dot = 0;
15 try
16     [nav_data, count] = fread(rx_port, 512, 'uint8');
17 catch error
18     fprintf('Nav Data Read Error: %s\n', error.message);
19     return;
20 end
21 if(count < 8)
22     fprintf('Insufficient Nav Data Read.\n');
23     return;
24 end
25 if(count >= 4)
26     header = typecast(uint8(nav_data(1:4)), 'uint32');
27 end
```

```
28 if(count >= 8)
29     success = true;
30     state = typecast(uint8(nav_data(5:8)), 'uint32');
31 end
32 if(count >= 12)
33     response = typecast(uint8(nav_data(9:12)), 'uint32');
34 end
35 if(count >= 16)
36     % vision = typecast(uint8(nav_data(13:16)), 'uint32');
37 end
38 if(count >= 20)
39     % tag = typecast(uint8(nav_data(17:20)), 'uint32');
40 end
41 if(count >= 24)
42     data_size = typecast(uint8(nav_data(21:24)), 'uint32');
43 %     fprintf('Bytes Read: %d\n', count);
44 end
45 if(count >= 28)
46     battery = typecast(uint8(nav_data(25:28)), 'uint32');
47 end
48 if(count >= 32)
49     pitch = double(typecast(uint8(nav_data(29:32)), 'single')/1000);
50 end
51 if(count >= 36)
52     roll = double(typecast(uint8(nav_data(33:36)), 'single')/1000);
53 end
54 if(count >= 40)
55     yaw = double(typecast(uint8(nav_data(37:40)), 'single')/1000);
56 end
57 if(count >= 44)
58     z = double(double(typecast(uint8(nav_data(41:44)), 'uint32'))/1000);
59 end
```

```
60 if(count >= 48)
61     x_dot = double(typecast(uint8(nav_data(45:48)), 'single'));
62 end
63 if(count >= 52)
64     y_dot = double(typecast(uint8(nav_data(49:52)), 'single'));
65 end
66 % z velocity measurement is not working (always 0)
67 % if(count >= 56)
68 %     setValues(handles, 'z_dot', double(double(typecast(uint8(nav_data(53:56)
       ), 'single'))));
69 % end
70 end
```

C.4 CalculateDroneValues.m

```
1 function success = calculateDroneValues(handles)
2 %--- Function to read current drone position, velocity,
3 % and acceleration values then save them in application memory
4 [tx_port, rx_port, drone_flight, old, yaw_offset, time] = getValues(handles,
      ...
5     'tx_port', 'rx_port', 'drone_flight', 'response_num', 'yaw_compensation',
      't');
6 read_timeout = 1.8;  % Read timeout
7 read = true;
8 read_handle = tic;
9 while(read);
10     [success, header, state, response, data_size, battery,...
11         pitch, roll, yaw, z, x_dot, y_dot] = GetNavData(rx_port);
12     if(success)
13         if(droneEmergencyCheck(drone_flight, state))
14             success = false;
15             LightDroneLed(tx_port, getCommandNumber(handles), 'solid_red');
```

```
16              return;
17          end
18          if(bitget(state, 31) == 1)
19              ResetWatchdog(tx_port, getCommandNumber(handles));
20              LightDroneLed(tx_port, getCommandNumber(handles), 'blank');
21          end
22          if(roll == 0 && pitch == 0)
23              % No meaningful data returned
24          else
25              if(response < old)
26                  fprintf('Old Nav Data Read.\n');
27              else
28                  read = false;   % Read until meaningful data is returned
29              end
30          end
31      end
32      if(toc(read_handle) > read_timeout)
33          fprintf('calculateDroneValues Timeout!\n');
34          success = false;
35          return;
36      end
37  end
38  % Save drone parameters
39  LightDroneLed(tx_port, getCommandNumber(handles), 'solid_green');
40  setValues(handles, 'response_num', response);
41  setValues(handles, 'drone_header', header, 'drone_state', state);
42  setValues(handles, 'data_size', data_size, 'batt_lvl', battery);
43  % Get last values
44  [x_last, y_last] = getValues(handles, 'x', 'y');
45  [x_dot_last, y_dot_last] = getValues(handles, 'x_dot', 'y_dot');
46  [roll_last, pitch_last, yaw_last, z_last] = getValues(handles,...
47      'roll', 'pitch', 'yaw', 'z');
```

```
48 [roll_dot_last, pitch_dot_last, yaw_dot_last, z_dot_last] = getValues(handles,
       ...
49     'roll_dot', 'pitch_dot', 'yaw_dot', 'z_dot');
50 % Calculate drone z velocity (measurement not working
51 if(z > 1000)
52     z = 0;
53 end
54 t = time(2);
55 z_dot = (z - z_last) / t;
56 % Calculate drone orientation rate of change values
57 yaw = yaw - yaw_offset;
58 roll_dot = (roll - roll_last) / t;
59 pitch_dot = (pitch - pitch_last) / t;
60 yaw_dot = (yaw - yaw_last) / t;
61 % Calculate drone position values
62 x_ddot = (x_dot - x_dot_last) / t;
63 y_ddot = (y_dot - y_dot_last) / t;
64 x = x_last + x_dot * t + 0.5 * x_ddot * t ^ 2;
65 y = y_last + y_dot * t + 0.5 * y_ddot * t ^ 2;
66 % Calculate drone acceleration values
67 z_ddot = (z_dot - z_dot_last) / t;
68 roll_ddot = (roll_dot - roll_dot_last) / t;
69 pitch_ddot = (pitch_dot - pitch_dot_last) / t;
70 yaw_ddot = (yaw_dot - yaw_dot_last) / t;
71 % Save measured values
72 setValues(handles, 'z', z, 'roll', roll, 'pitch', pitch, 'yaw', yaw, 'x_dot',
       x_dot, 'y_dot', y_dot);
73 % Save calculated values
74 setValues(handles, 'x', x, 'y', y);
75 setValues(handles, 'roll_dot', roll_dot, 'pitch_dot', pitch_dot, 'yaw_dot',
       yaw_dot);
76 setValues(handles, 'x_ddot', x_ddot, 'y_ddot', y_ddot, 'z_ddot', z_ddot,...
```

```
77      'roll_ddot', roll_ddot, 'pitch_ddot', pitch_ddot, 'yaw_ddot', yaw_ddot);
78  success = true;
79  end
```