IMPLEMENTATION OF A RULE-BASED OPEN-LOOP CONTROL STRATEGY FOR A HYBRID-ELECTRIC PROPULSION SYSTEM ON A SMALL RPA

THESIS

Collin M. Greiser, BSE

2d Lt, USAF

AFIT/GA/ENY/11-M05

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GA/ENY/11-M05

IMPLEMENTATION OF A RULE-BASED OPEN-LOOP CONTROL STRATEGY FOR A HYBRID-ELECTRIC PROPULSION SYSTEM ON A SMALL RPA

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Astronautical Engineering

Collin M. Greiser, BSE

2d Lt, USAF

March 2011

AFIT/GA/ENY/11-M05

IMPLEMENTATION OF A RULE-BASED OPEN-LOOP CONTROL STRATEGY FOR A HYBRID-ELECTRIC PROPULSION SYSTEM ON A SMALL RPA

Collin M. Greiser, BSE

2d Lt, USAF

Approved:

Frederick G. Harmon, Lt Col, USAF (Chairman) — 16 Mar 11 — Date

Christopher M. Shearer, Lt Col, USAF (Member) — 11 Mar 2011 — Date

Dr. Richard G. Cobb (Member) — 16 MAR 11 — Date

Abstract

Currently-fielded small electric-powered remotely-piloted aircraft (RPA) lack endurance desired by warfighters, and internal combustion engine (ICE) RPAs generate undesirable acoustic and thermal signatures. Hybrid-electric (HE) propulsion systems would use ICE power for cruise, electric power for endurance, and combine both electric power and ICE power for takeoff, climbing, and recharging onboard battery packs. Use of HE systems would eliminate undesirable signatures in addition to providing considerable fuel savings over time. Various combinations of six components were used in this HE system: the ICE, electric motor (EM), electromagnetic clutch, a one-way bearing, battery pack, and a propeller. Control of such a system in a small RPA has never been attempted before. A rule-based controller was developed in C code to manage this HE system. This system and its various sensors were analyzed on a custom-built dynamometer test stand that was developed in conjunction with other students. LabView screens were developed to aid this testing and interface with the sensor suite. The controller's performance over 9 distinct operating modes, including 4 operational flying states, were validated to provide the most optimal operation of a HE-RPA system of about 13.6 kg (30.0 lbf).

Acknowledgments

I cannot start acknowledging anyone until I first acknowledge God for granting me the strength, wisdom, and perseverance to complete this. Without him, I could not have accomplished anything that I have done today, including actually being here completing a master's degree. It is something I never imagined I would do. I must thank my advisor, Lt. Col. Harmon, for his (continued) patience with all of my questions and constant visits to his office. Much of Col. Harmon's ideas in his dissertation and code went into my programming. A huge, huge thank you goes to John Hagen, without whom I would never have gotten anywhere close to completing the C code on the microprocessor. My friends and roommates including Josh Fehd, Josh Hess, and Adam Hillier also deserve credit for keeping humor in my life through this process.

Another thank you goes out to Isseyas Mengistu, Todd Rotramel, and Matt Rippl, all of whom worked together with me to design and build the test rig for our dynamometer setup. Countless hours were spent together planning and executing the design and building the stand itself. The AFIT machine shop crew and ENY lab techs also committed a large number of man-hours to our project in designing the bearings and mounting plates, and they deserve a great deal of credit for that endeavor. Finally, I would like to thank the U.S. Air Force for being so generous as to send me here and actually pay me a salary to get a degree. There is no other opportunity like it in the world.

-Collin M. Greiser

Table of Contents

Page

Table of Contents

List of Figures

Page

List of Tables

Page

List of Abbreviations

Abbreviation	Description
AC	Alternating Current
A/C	Aircraft
AVGAS	Aviation Gasoline
CDM	Charge-Depleting Mode
COTS	Commercial Off the Shelf
CSM	Charge-Sustaining Mode
CVT	Continuously-Variable Transmission
DC	Direct Current
DC/DC	DC-to-DC
DoD	Department of Defense
ESC	Electronic Speed Control
EM	Electric Motor
FET	Field Effect Transistor
FLC	Fuzzy Logic Controller
HE	Hybrid-Electric
HE-RPA	Hybrid-Electric Remotely-Piloted Aircraft
IAW	In Accordance With
ICE	Internal Combustion Engine
ISR	Intelligence, Surveillance, and Reconnaissance
LIPO	Lithium Polymer

MATLAB	Matrix Laboratory
MAV	Micro Air Vehicle
MEP	Mean Effective Pressure
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
R/C	Remote Control
RPA	Remotely-Piloted Aircraft
RPM	Revolutions Per Minute
SI	International System of Units
SM	State Machine
SNR	Signal to Noise Ratio
SOP	Standard Operating Procedure
TDC	Top Dead Center
UAS	Unmanned Aircraft System
US	United States
USAF	United States Air Force
WOT	Wide Open Throttle

Nomenclature

Symbol	Description (Units)
b	Neural Network Bias (unit-less)
C	Capacitance (Farad)
dB	Change in Magnetic Field Strength (dB/dt)
dl	Current Element (A)
f_c	Low-pass filter break frequency (Hz)
GetRCDutyCycle()	Function to Retrieve RC Duty Cycle (unit-less)
GetTorqueRequest()	Function to Compute Torque Request (N-m)
GetTotalAvailableTorque	Function to Compete Available Torque (N-m)
I_{bat}	Battery Current (A)
I_{clutch}	Clutch Current (A)
I_{motor}	Motor Current (A)
I_o	Electric Motor No-Load Current (A)
K_v	Electric Motor Speed Constant (rad/V/s)
MaxICETorque	Maximum Available ICE Torque (N-m)
MEP	Mean Effective Pressure (Pa)
μ_0	Magnetic Moment of Clutch Dipole
N	Engine Rotational Speed (RPM)
n	Electric Motor Rotational Speed (RPM)
normalizedtorque	Normalized Throttle Command (unit-less)
n_r	Engine Revolutions Per Cycle

P	Internal Combustion Engine Power Output (W)
P_{in}	Electric Motor Power Input (W)
P_{out}	Electric Motor Power Output (W)
rad	Radians
rad/s	Radians per Second
R	Resistance (Ohm)
R_{int}	Battery Internal Resistance (Ohm)
R_m	Electric Motor Internal Resistance (Ohm)
r	Clutch Displacement Vector (m)
s	Seconds
T	Engine Torque (N-m)
torquerequest	Torque Request From Autopilot/Pilot (N-m)
U_{emf}	Electric Motor Electromotive Force (V)
U_m	Electric Motor Open Circuit Voltage (V)
u	Clutch Displacement Angle (rad)
V_{bat}	Battery Voltage (V)
V_d	Cylinder Displacement Volume (cm^3)
V_{input}	DC/DC IMON Reference Signal (V)
W_c	Engine Work Per Cycle (W)
w	Neural Network Scalar Weight (unit-less)
wp	Neural Network Input (unit-less)
ω	Motor Rotational Speed (rad/s)

IMPLEMENTATION OF A RULE-BASED OPEN-LOOP CONTROL STRATEGY FOR A HYBRID-ELECTRIC PROPULSION SYSTEM ON A SMALL RPA

I. Introduction

1.1 Background

When one thinks of the history of unmanned aerial aviation, thoughts immediately go to flying reconnaissance systems used in the 70's, or depending on one's definition, perhaps even the feared V-1 and V-2 rockets of World War II. However, hybrid-electric unmanned aviation is a far more modern concept. Prevalent use of unmanned aerial vehicles did not start until the latter quarter of the 20th century [1], and hybrid-electric propulsion system use in such vehicles is unprecedented. Hybrid-electric propulsion systems have been in use in road vehicles for several decades. Ferdinand Porsche arguably created the first hybrid car in 1903 (uncertain to its significance a century later) [2]. These hybrids were heavy and slow, but formed the foundation for what would come later, including use in aircraft.

However, hybrid technology in the early 1900's fell by the wayside as the internal combustion engine continued its prominent takeover from steam power and was refined again and again. Introductions such as the diesel engine in the 1920's and compressors such as the turbocharger and supercharger in the same decade allowed the combustion engine to continue its meteoric rise into man's history. The combustion engine continued to dominate land vehicle technological improvements well into the 1970's, when the first real fuel crisis hit North America.

This 70's fuel crisis stopped the trend of increasing horsepower and ignoring efficiency. Automakers to this point had been in a competition to try to create more and more powerful

engines. However, these vehicles were fuel inefficient; incredibly pitiful in terms of specific fuel consumption as compared to the cars of today. For example, a 1971 Mustang utilizing a 7.0 L V8 engine with 375 HP achieved 10 miles per gallon (MPG) [3]. However, a 2011 Mustang with a 4.6 L V8 engine with 412 horsepower can achieve up to 26 MPG. Once this fuel crisis hit, the inefficiencies of such cars were highlighted, and automakers scrambled to find ways to save fuel. Hybrid technology was explored briefly here, as work by Victor Wouk showed the usefulness of installing hybrid-electric power trains into a Buick Skylark [4]. Audi also seemed to take note, with the introduction of the Audi Duo in 1989. However, these hybrids remained largely unsuccessful due to the end of the gas crisis in the late 70's and the cheap availability of fuel in the 80's and 90's.

In the mid 2000's another gas crisis struck, and finally hybrids were thrust into the public spotlight. Cars such as the Toyota Prius and Ford Escape Hybrid flew out of showrooms as gas skyrocketed to nearly $5 a gallon in some states. With interest in hybrid technology at an all time high, applications for the concept have turned to other areas besides just road vehicles. Aircraft have been largely untouched by hybrid technology, with many aircraft still flying with engines that were designed decades ago. The interest in hybrid technology has led designers to try to apply this technology to aircraft.

Just as hybrid technology has advantages in road vehicles; it has several advantages in aircraft as well. These aircraft have to be specially designed and built around this propulsion system to take advantage of the benefits; hybrid systems need the advantages to outweigh the disadvantages in order to make them practical. The need for a motivation comes from hybrid systems inherent cost and weight penalties that come along with a hybrid system. Unmanned

aircraft, which do not have the added weight of a pilot, can easily take advantage of hybrid benefits. Additionally, unmanned aircraft themselves are at the same time being thrust into the defense spotlight. The world of unmanned aircraft is quickly becoming complex and heavily invested in as evidenced by the Department of Defense (DoD) recent investments in the field.

1.2 Motivation

The Department of Defense has ramped up the use of RPAs, with a goal of 54 combat air patrols by 2011 [5]. An RPA has an incredible allure as an intelligence, surveillance, and reconnaissance (ISR) vehicle because of the lack of a pilot to put in harm's way. Indeed, over 100,000 hours were flown in 2004 alone. [5] The use of surveillance unmanned aircraft was conceptual as far back as the 1940's but came into use in the 1950's as the U.S. started to focus their efforts on 'surveillance drones.' These drones were designed simply to be controlled by an operator on the ground via radar. The ability to now miniaturize systems and delete those that are unneeded (such as the cockpit), can make the aircraft far more efficient, especially in terms of weight. However, aircraft used today by the war fighter still come up short in a number of areas. Aircraft can still be noisy in ISR missions because of the internal combustion engine (ICE). Flight times can be limited by fuel use, and this is especially critical when target information can be in windows as short as minutes. Having an aircraft have to end its mission at an inopportune time due to fuel shortage could end up costing an effective data-collecting mission. Additionally, fuel type requirements are critical, with the use of glow fuel and aviation gasoline (AVGAS) dominating the RPA fuel type. These fuels, while cheap and plentiful in the U.S., are expensive and difficult to acquire overseas. Therefore, logistically, this makes things

more difficult because more of the expensive fuels are required. Fuels such as AVGAS and glow fuel also have lower flash points, and as such are more dangerous on naval ships.

Therefore, the hybrid-electric remotely-piloted aircraft (HE-RPA) design is highly practical. With the ICE providing power for longer range and the EM providing stealthy quietness and efficiency for ISR missions, the HE-RPA is becoming more and more a focus in today's world. Additionally, much work is being done on the adaption of small engines for the use of diesel fuel and JP-8. Diesel especially is far easier and cheaper to acquire overseas. A combination of a strategically viable fuel and a HE system make a potent ISR aircraft for the warfighter.

However, the one area that the HE-RPA needs more development and research is in the area of propulsion control. With the exception of a few, like German company Flight Design [6], the area of control of hybrid-electric systems has been mostly constrained to the automobile field. A main reason for this has been the level of complexity. The control strategy and code for a hybrid controller is far more complex as compared to a controller for a regular vehicle (or aircraft) [7]. Hybrid controllers must balance the requirements and parameters of several additional systems on top of the systems that a normal ICE controller would supervise.

1.3 Problem Statement

Today, fighters from all nations now employ advanced technologies or clever versions of common technologies to gather intelligence and attack the enemy. The history of unmanned aerial aviation book in particular lists 52 countries as having an association with RPAs, being manufacturing, operation, or both [1]. Surveillance, in particular, has been a huge focus. A primary requirement of ISR is stealth, and this is where many current RPAs need more

development and research. According to the *Unmanned Aircraft Systems Roadmap (2007-2030)* [5] , the DoD must invest in improved propulsive efficiency through alternative propulsion power sources for endurance and unwarned ISR. The internal combustion engine of today is mostly adequate for endurance, but is too noisy and can be detected easily both acoustically and thermally. An electrical propulsion system seems the logical alternative, as it is quiet and efficient. However, battery systems are woefully inadequate when endurance is considered and add a significant weight penalty. Therefore, by looking at the automotive world, where efficiency concerns have been high over the recent years, the hybrid-electric (HE) systems seem the logical choice, and in fact are a feasible alternative. However, the control of these systems is still under great study and debate. The primary concern of these HE systems on cars is maximizing efficiency, while the production of noise is only a secondary concern. An RPA, however, needs both; the efficiency for long endurance and the stealth for invisibility while on station. The *Unmanned Aerial Systems Roadmap (2007-2032)* even states the ISR missions with higher endurance requirements "Will require more sophisticated energy systems, such as fuel cells and hybrid systems." [5] Very little work, however, has been done in the field of control of the propulsion systems for these aerial vehicles. There are many areas of study among HE automobiles, and some of this can be paralleled in the aerial world. There are also some studies done on advanced controllers such as the neural network controller by Harmon [8]. In fact, Harmon states: "the control systems on a hybrid-electric remotely-piloted aircraft has three objectives: increasing range, providing time for the RPA to operate in electric-only (EO) mode, and provide battery power for the UAS's sensors. In this light, the problem that is being solved is implementation of a control strategy on a prototype propulsion system.

1.4 Research Objective

The research contained here focused intensely on implementing the rule-based open-loop control strategy on a propulsion system test bed for a small RPA. Therefore, there were two critical objectives for this thesis. The first goal was to design and implement an open-loop control strategy in C that would directly take commands given to it by a pilot or autopilot, and translate those commands into efficient operation of a HE propulsion system. This leads directly into the second objective, which is the validation of the control strategy at four specific design points for flight. These design points were cruise, climb, endurance, and cruise with regeneration. Validating the control strategy was broken down further into creating a test matrix and analysis system in LabView, and building a test stand in which to develop the hardware and sensors needed to validate the strategy.

1.5 Research Scope

The controller created here is a result of converting a simple flowchart into a much more complicated rule-based controller in LabView. The controller itself can be adapted for use in the actual airframe, but as is the set up is not for use in the aircraft. As such, wiring of components with diagrams of such wiring for aircraft use are not needed and ignored. Additionally, when the controller makes its computations, it makes basic assumptions about the amount of power needed to fly the aircraft. These equations are discussed in Hiserote's master's thesis [9] and are taken partially from Anderson [10]. In reality, the aircraft would have multiple sensors needed to determine things such as air density, airspeed, etc., and use these to make a more accurate judgment on the power needed to fly. This controller is designed for use on small RPA, but could easily be adapted for use on larger systems such as those suggested by Rippl [11] with the right sensor suite and controls.

1.6 Methodology

The author implemented traditional control theory and programming techniques in designing this controller. The controller is assumed to have 5 operational parameters to account for: rotational speed for both the ICE and the EM, torque output from the ICE and EM, and state of charge of the battery pack. The basic rule strategy for the operating modes came from the flowchart from the dissertation by Harmon, but multiple additional paths and starting points were used. The controller state machine has 9 different operating modes that it switches between to accomplish its mission, 4 of which are primary flight modes discussed briefly above. The optimal path of energy use is controlled by the pilot in the scope of this research. The path could optionally be determined using basic dynamic optimization strategies outside the controller and then preloaded. Aircraft design is the 'clutch-start parallel design' from Hiserote's 2010 thesis on 'UAS design' [9] and does not include any other designs he mentions in his research.

1.7 Thesis Overview

Chapter I of this thesis provides an introduction to the thesis and relevant background information. Chapter II is a review of literature that applies to this thesis. Chapter III discusses in detail the author's methodology, including the state machine and analysis tools. Chapter IV includes analysis of the controller in operations and results of the tests performed. Chapter V discusses these results and relays relevant conclusions that the author has determined.

II. Literature Review

2.1 Chapter Overview

Hybrid propulsion technology has long been a subject of intense study. The automotive industry in particular has led the charge in adapting hybrid-electric power trains for use in everyday life, and resulted in drastic increases in fuel efficiency. However, the aviation world has only just begun to intensely study the benefits of using hybrid propulsion in aircraft. The research of controls specifically has been almost overwhelmingly biased towards the auto industry, with very little work being done on various control methods for HE systems and their effects on aircraft. Most research has been done on more complex types of intelligent controllers such as fuzzy logic or neural networks. This chapter begins by briefly outlining the background of the development of hybrid propulsion control and the various strategies. This includes various sections on the components of a HE aircraft. The author then will analyze each type of control method based on current research and present reasoning on why a rule-based controller was chosen for the initial design.

2.2 Hybrid-Electric Propulsion and Configurations

Hybrid technology, by its very nature and definition, combines the use of two or more power sources for a variety of different uses and creates a more efficient vehicle. Hybrid technology has many variations; however most of the work done today has been in one of three areas: series hybrid, parallel hybrid, and the power-split hybrid. A great number of these designs incorporate the gasoline internal combustion engine as the prime power source; however other engines have been used such as diesel, gas turbine, or fuel cells.

The series hybrid is a hybrid that uses the EM as its prime mover. A typical configuration for the series hybrid is to have a gasoline engine drive a generator; this generator in turn is connected to one or more electric motors which propel the vehicle. Series hybrids are perhaps the oldest type of hybrid in use today; an early example was built by Ferdinand Porsche in the early 20th century [2]. Another great example of series hybrid is a diesel locomotive, which has been in use for many years. The reason that this hybrid has been in use for so long is its general simplicity. A figure of the series hybrid is shown in Figure 1. The main advantage of the series hybrid is that the internal combustion engine is not connected to the means of motive

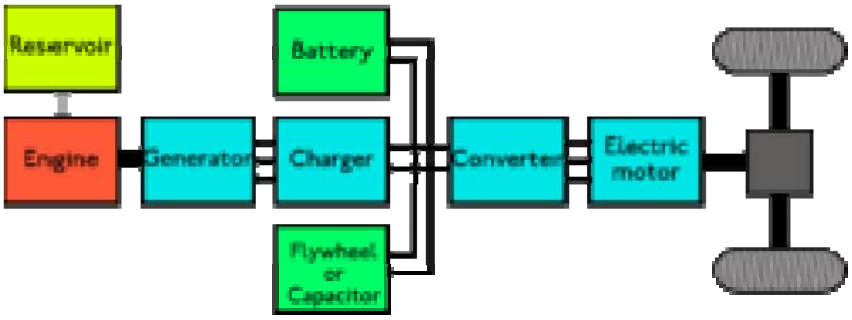


Figure 1: Series hybrid configuration [12]

force, and therefore can operate at its optimum efficiency all the time. An example of this would be the gasoline engine operating at the ideal operating line (IOL) continuously. However, the main disadvantages of this system are the various losses that occur. The electric motor must be sized exclusively for propulsion, and therefore will be heavy and provide a weight penalty to the vehicle. Additionally, the means of generation of electric power is not as efficient as a direct mechanical connection, incurring additional penalties. These losses are mitigated with larger systems, which is why this system is typically applicable to large transport systems such as buses and tow tractors [13].

A parallel hybrid is a hybrid where two or more systems are combined, and both have mutual or exclusive access to drive the vehicle. Consequently, this allows either the ICE or EM to power the vehicle, or both, depending on the vehicle and setup. Figure 2 shows this configuration. In automobiles, a parallel system is classified down even further into three

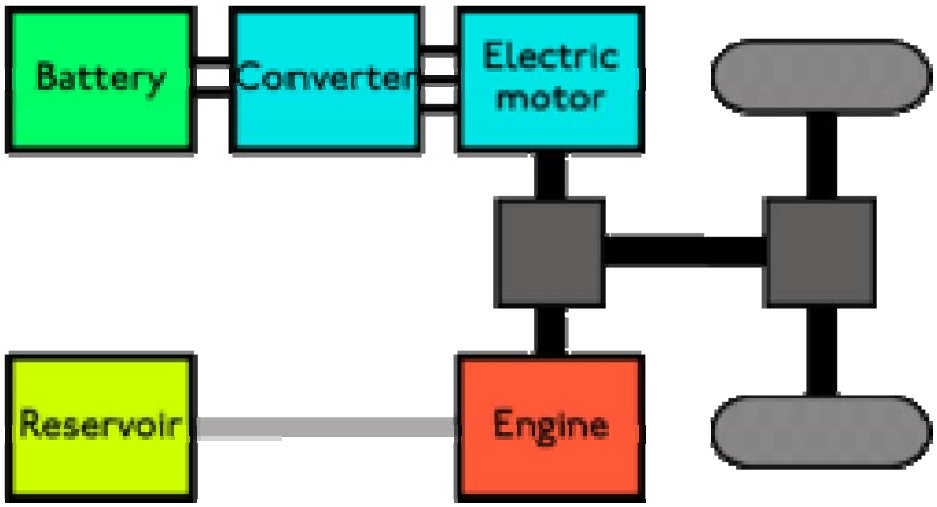


Figure 2: Parallel hybrid configuration [14]

subcategories: mild, power assist, and dual mode [15]. Mild parallel hybrids have a smaller electric subsystem that assists the ICE; generally to provide regenerative braking and perhaps an engine shutoff feature. A power assist system uses a larger electric subsystem to provide more capabilities, to include electric-only modes of operations and electric acceleration assist. Finally, the dual mode hybrid incorporates a still larger electric subsystem to account for 30% or more of the total system power of the vehicle [15]. Since the electric subsystem accounts for more power and is, in general, more efficient, the dual mode hybrid has the greatest efficiency of these three subcategories but in consequence costs more and is more complex. A variety of undergraduate and graduate research, including that which has been done at Virginia Tech [16], has focused on the parallel hybrid type.

The final type of hybrid, the power-split hybrid, is a combination of the series hybrid and the parallel hybrid. There is no direct connection, but rather there is a planetary gear set that allows transfer of power among the various systems to the road [13]. The power-split system essentially decouples the driver's actions from direct involvement from what is on the road. The comparative efficiency of the power split hybrid, due to its nature of combining the various sources more efficiently than other types of hybrids, is more effective at reducing fuel usage and emissions. However, it is also bound by this complexity in terms of cost and governing control strategies that are required for operation. Not only do the various elements of the system require controllers, but the strategy and ability to talk to one another and operate in harmony increases the complexity of the system as a whole. The power-split design is found on a number of vehicles today, including the Toyota Prius and the Ford Hybrid Escape.

2.3 Applications of Hybrid Power

Hybrid power trains, by their very definition, can be found in numerous applications. As stated in the chapter overview, automobiles are the field where the hybrid power train is most applied. Ferdinand Porsche actually built the one of the world's earliest gasoline-electric hybrids in 1903 [2]. However, the first well known hybrid vehicle in the world was the Toyota Prius in 1997 (in Japan), followed by the release of the Honda Insight in the U.S. in 1999. Both of these vehicles demonstrated that the hybrid vehicle was feasible and more importantly, more efficient than its gasoline powered brethren. In terms of RPA use, hybrid-electric power systems have not seen much use in the forms that are mentioned in this paper. When selecting the type of hybrid-electric system for use in an RPA, Hiserote has already completed a good deal of conceptual analysis and selected the dual mode parallel type for use. Specific components of this hybrid system will be discussed further in Chapters III and IV. However, control of these vehicles

remains an important factor, as the efficiency of the aforementioned vehicles would not be achieved without a solid controller strategy to realize it.

2.4 Hybrid-Electric Control Strategies

There are three main strategies currently employed when operating a hybrid-electric vehicle (HEV): electric-only mode, charge-sustaining mode (CSM), and charge-depleting mode (CDM). Electric-only mode refers to using the electric motor by itself to propel the vehicle or aircraft forward. In most cars this allows for low-speed operation, while in aircraft this can differ. Endurance mode in an aircraft, as discussed briefly in Chapter I, is most useful for ISR portions of missions. Charge sustaining mode refers to operating the ICE as the main method of propulsion and using some or all of the EM's available power for recharging the batteries. A typical strategy, depending on the durability of the batteries, would be to start charging at 20% capacity and stop charging at 30-40% (often called a "thermostat" method) [8] . As a result of the battery charge, the EM can then be used to either assist the ICE or provide low speed operation until the battery state-of-charge (SOC) has dropped to 20%. From here, the cycle would repeat. This will be explained in greater detail in Chapter III. The final strategy, the CDM, refers to using the EM and the ICE together to propel the vehicle, with no recharging being done. This means that the EM is supplying its power as a supplement to the ICE and drains the electric reserve power. Typically this is used in parallel hybrids to allow for "plug-in" use; in an aircraft this could be used for climbing. The CDM is used until the batteries reach the specified level and then the CSM is activated. In CDM, the batteries typically are used heavily at the beginning of the cycle while the pack has a high SOC. This makes the strategy perfect for "plug-in" use i.e. plugging the car in at the end of the day. For aircraft this is just as feasible, as the aircraft could be plugged in before the start of the mission and then plugged in following the

mission. The energy then used during the mission from a plug in source is far cheaper than regular gasoline, glow fuel, AVGAS, or diesel, and its efficiency is much higher. Hybrid vehicles on the road today use a combination of all three of these techniques to propel the vehicle and to try and achieve the greatest fuel efficiency.

An HE-RPA operating strategy, however, varies greatly based on the mission profile and aircraft design that the user wishes to assign. HE-RPA's are more restricted in many aspects, including noise and weight, which a regular automobile is not restricted in. Mission profile primarily dictates noise restrictions. The aforementioned endurance mode is greatly desirable in missions where stealth is of primary importance, however having a longer endurance mode requires larger battery packs, which increases weight. The weight of the aircraft greatly factors in to the available time on station. Therefore, the control strategy of the HE-RPA primarily needs to be designed around the given mission, or be able to switch between pre-loaded modes for differing legs of a mission.

The controller on the HE-RPA must be designed so that all the individual pieces operate in concert with one another to achieve the maximum efficiency for the HE system. Each piece has its own challenges when being operated which determine how the controller is designed. In the sections that follow each component will be discussed in detail and the challenges of control outlined.

2.5 Control Elements

The controller has various combinations of five main components that it must control: the electric motor, internal combustion engine, the battery pack, the electromagnetic clutch, and the one-way bearing. These five components are what make up the parallel hybrid that Hiserote

describes; this is the system that will be used to demonstrate the propulsion system concept. Each system has its own challenges in terms of control, but all center on the determination of the error between requested output and the actual output. The next few sections will detail how each component is typically controlled along with outlining the challenges of control.

2.5.1 Electric Motor

Electric motors are the first main component in the hybrid system. As described earlier, the electric motor is ideal because of its high torque at low revolutions per minute (RPM) (as compared to an ICE) and the fact that it gives the aircraft the ability to operate in 'stealth' mode (again, as compared to using the ICE). High torque at low RPM versus an ICE means that the EM delivers its torque at a very low RPM

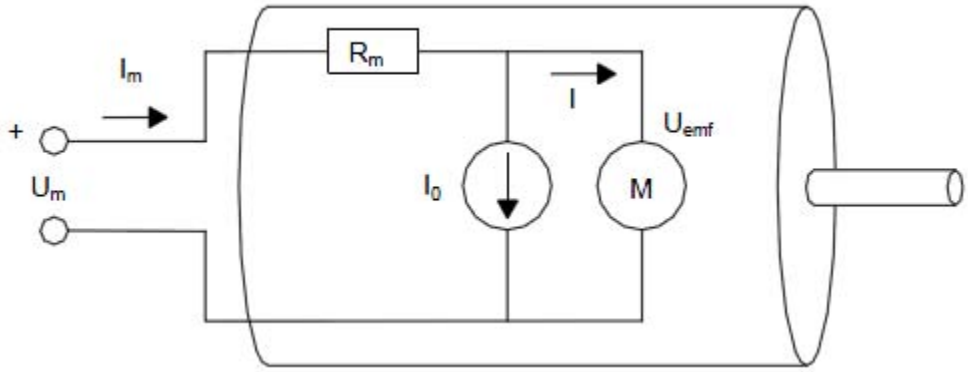


Figure 3: Electric motor model [17]

i.e. near zero, while even the strongest small diesels generally need to be above 1000 RPM. Each EM requires a controller to dictate its operation. Electric motors have several classical parameters that distinguish between them: K_v (motor proportionality constant), I_o (motor no-load current), and R_m (motor internal resistance). The data for these constants is usually given by the motor manufacturers. Losses are characterized by the no load current and the internal resistance,

which are very important things for the controller to have stored so it can accurately calculate the motor efficiency. A model of an electric motor is shown in Figure 3; in this model, U_m is the open circuit voltage and U_{emf} is the motor output voltage.

There are several governing equations that will be used throughout this thesis when discussing electric motors. These first order equations are presented by Lundstrom [17], and are shown below:

$$n = U_{emf} * K_v \tag{1}$$

$$P_{in} = I_m * U_m \tag{2}$$

$$P_{out} = I * U_{emf} \tag{3}$$

$$P_{out} = \tau * \omega \tag{4}$$

where n is the motor rotational speed in RPM, P_{in} is the power input to the motor in watts, P_{out} is the power output of the motor in watts, τ is the torque output of the motor in Newton-meters (N-m), and ω is the rotational speed of the motor in RPM. Here, n and ω are used as two different values for motor speed, where n is in RPM and ω is in radians per second (rad/s). These equations are commonly rearranged into the following equations, which relate P_{out} and n to input voltage (open circuit voltage) and current:

$$P_{out} = I * U_{emf} = (I_m - I_o) * (U_m - I_m * R_m) \tag{5}$$

$$n = K_v * (U_m - I_m * R_m) \tag{6}$$

The efficiency of the electric motor,η_m, is then:

$$\eta_m = \frac{P_{out}}{P_{in}} = \frac{(I_m - I_o) * (U_m - I_m * R_m)}{U_m * I_m} \quad (7)$$

Motor control, sometimes called electronic speed control (ESC), is how brushless electric motors are controlled. Brushless electric motors are alternating current (AC) machines. The ESC controls the input voltage to the motor by sending a pulse width modulation (PWM) signal. By varying the pulse length, amount of time the motor is receiving power is controlled, which thereby controls the speed of the motor. [18] Losses in the controller however, are harder to quantify. The motor manufacturer will occasionally give general numbers for losses due to electronics, but losses due to motor speed and duty cycle are generally not known and will be a challenge to model. Lundstrom describes losses at duty cycles less than 100% as being divided into two types: losses in the electronics due to additional switching in field effect transistors (FET), and other losses due to PWM signal losses. Lundstrom goes on to suggest that all motors in his paper performed with lower efficiencies than what manufacturers had reported. As the design of the RPA is primarily with off the shelf components, this is a huge challenge when attempted to model the motor in the controller flow chart. If the controller expects the motor to output a certain amount of power and it does not get the power due to unexpected losses, the system will perform sub-optimally. Losses in the motor need to be accounted for in the controller programming. Lundstrom also runs his tests with a variety of motor controllers, many of which are put into the selection process for the motor controller that will be used.

A direct current (DC) machine is another type of EM. These motors are typically brushed and use six main parts: a commutator, an armature, brushes, the axle, field magnets, and

the DC power supply. The motor works by flipping the magnetic field back and forth so the axle will spin (the axle being the torque transferring item). The brushes provide a mechanical means of transferring current to the electromagnet (field magnet). The commutator flips the electric field back and forth as it spins, which creates the motion. The armature then spins around the inside of the housing with its magnetic field flipping back and forth as the commutator changes the current direction. This simple type of motor has been in use since 1886 [19]; an example is shown in Figure 4. Simple DC motors are controlled with voltage; each motor is again specified a K_v value and this in turn controls the speed. The current controls the torque output of the motor as described in the electric motor equations above.

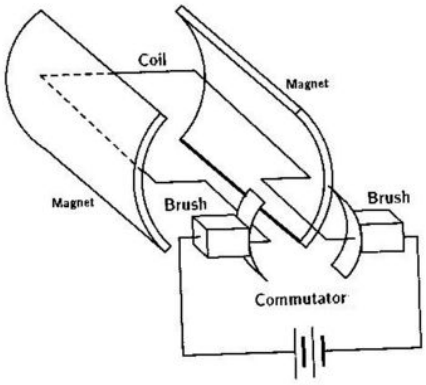


Figure 4: Diagram showing brushed DC motor [19]

2.5.2 Internal Combustion Engine

The second component of the hybrid-electric system is the internal combustion engine. This is the main power source for the RPA. While batteries combined with an electric motor have high efficiencies, they currently do not have the mission endurance that the U.S. military is looking for. Therefore, the ICE is considered the main power source. The ICE is a heat engine

that combusts fuel in an internal chamber to produce work. Model aircraft engines like the ones that are used on the propulsion system prototype are commercial off-the-shelf (COTS) two or four stroke engines. A sequence depicting a two-stroke engine cycle is shown in Figure 5. The

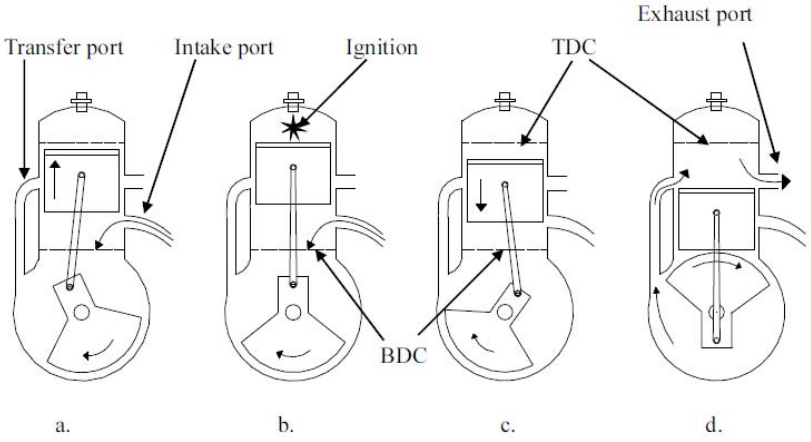


Figure 5: Two stroke engine cycle [20]

two stroke engine is fundamentally different, as its name suggests, because it produces a power stroke for every revolution of the crankshaft (*two* movements of the piston), while the four stroke engine produces a power stroke for every two revolutions of the crankshaft (*four* movements of the piston). There are many differences between the two, but this thesis focuses on governing equations and controlling the engine output, for which these equations apply to both engines. Figure 6 depicts the movements of a four stroke engine. From Heywood [21] several equations are shown that determine engine performance. It is then important to first define mean effective pressure (MEP), a crucial performance measure:

$$MEP = W_c/V_d \tag{8}$$

where W_c is the work per cycle, and V_d is the displacement volume of the cylinder of the engine. *MEP* is then defined as the work per cycle per unit displaced. *MEP* can then be related to power with the following equation:

$$P = MEP * V_d * \frac{N}{n_R} \tag{9}$$

Here *N* is the RPM of the engine and n_c is the number of revolutions per cycle (one for a two stroke engine or two for a four stroke engine). This is the resulting power output of the engine.

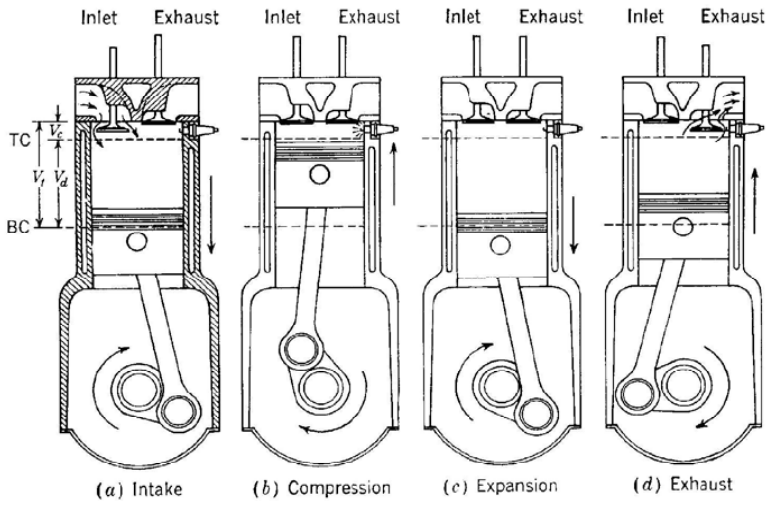


Figure 6: Four stroke operating cycle [21]

MEP can also be related to the torque output of the engine:

$$T = MEP * \frac{V_d}{2n_R\pi} \tag{10}$$

where *T* is engine torque, and is a critical parameter for engine control.

There are several challenges when controlling and accurately modeling ICE's. As with any off the shelf component, there is always the danger of the manufacturer over or under estimating the performance capabilities of their products. Without comprehensive testing to provide an accurate map of the engine capabilities, the controller must do its best to estimate the amount of torque and power the engine provides. There are a large number of papers and articles written on testing of engines and estimation of the various states. State estimation on the ICE centers primarily on one parameter: torque. Torque estimation is comparatively easy on larger hybrid-electric systems such as cars and buses. In these methods, the state estimator (in this case called an observer) uses one of many non-linear models, such as the sliding mode observer [22], to observe the engine torque. This method, while complex, has been proven to be accurate. However, the estimator uses a myriad of sensors that are simply not feasible on a small RPA due to weight and size constraints. Sensors such as rotational speed sensors (including their mounting hardware) that are mounted on the cars automatic transmission torque converter and its main crankshaft are far too heavy for use on an RPA, so other methods must be explored.

Another method used for torque estimation relies only on "signals that would be readily available in a mass-production car." [23] This refers to signals such as rotational speed and top dead center (TDC) positioning, which on a car are readily available. The above reference has correlations with modern RPAs in the sense that engine rotational speed is critical. This was a major parameter on the controller the author programmed. However, the measurement of TDC on a small RPA engine is simply not feasible. The TDC measurement is a very noisy signal and highly difficult to determine accurately [24]. Additionally, the estimator uses the following equation to determine torque:

$$I_e * \ddot{\theta} = T_{comb} - f(\theta, \dot{\theta}) * \dot{\theta}^2 - T_{load} \quad (11)$$

where θ is the engine rotation angle, T_{comb} is the combustion torque, T_{load} is the extended load torque, and I_e is the engine inertia. The difficulty in this equation is the engine inertia term. The term takes into account the mass of the oscillating parts, which is more difficult to determine in a small ICE accurately. Most small engine manufacturers do not provide this data.

A method that has been explored by numerous research students is the method of using a type of torque sensor to measure the engine output, most notably Menon [25]. While this method is great and reasonably accurate for bench testing of ICE engines, it is also not feasible on an aircraft due to the dimensions of the apparatus. Menon's setup used a moment arm that would deflect as the engine applied torque, which allowed him to use simple equations and a load cell to measure the reaction torque required to keep a freely rotating engine in place. However, Menon did use a method for measuring in-cylinder pressure which could be extremely useful. He used a sensor developed by Optrand [26], which is mounted inside the cylinder in a threaded hole. The sensor was used along with a basic equation for mean effective pressure to determine an estimate for engine output torque. While feasible, and research by the author suggests usefulness of an estimator, the torque estimator was not attempted due to time restraints with the HE system.

2.5.3 Batteries

The third component that needs to be controlled is the batteries. Batteries, in general, provide power for the electric motor during the section of the mission where endurance is most

crucial. Batteries generate electricity by converting chemical energy by redox reactions [27]. The electrical energy flows between two electrodes, designated the *anode* and the *cathode.* The anode is typically designated as the negative electrode, and the positive electrode is designated the cathode. A typical battery model is shown in Figure 7. Here the anode, cathode,

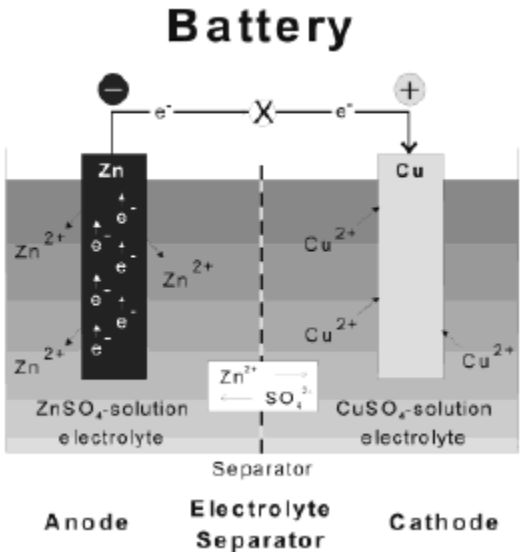


Figure 7: Battery model showing anode, cathode, and electrolyte separator [27]

and electrolyte separator are shown. The electrolyte separator is the boundary between the two chemicals and must be relatively impermeable to the two solutions to avoid short circuiting the battery. The electrolyte separator is usually made from a simple material that functions to block the two chemicals while being lightweight. Battery equations can be taken very simply from Ohm's law:

$$V_{bat} = I_{bat} * R_{int} \tag{12}$$

where V_{bat} is the battery voltage measured across its terminals, I_{bat} is the current flowing across the batteries terminals, and R_{int} is the internal resistance of the battery. The main parameters that

the controller will need to know from the batteries are voltage, current, and state-of-charge. Voltage and current are easily obtained from direct measurement, but determining SOC is another matter. There are many methods suggested for determining SOC, but two that are suggested and are feasible on an RPA are voltage-based table look-ups and current-based coulomb counting methods [28]. The method that the author implemented is a voltage based method, as this was more widely used in literature and the figures and curves for determining the SOC are readily available. When using this method, corrections must be developed for differences in actual voltage levels, temperature of the battery, discharge rate, and the age of the cell. More detail about this method is shown in Chapter III.

2.5.4 Electromagnetic Clutches

An electromagnetic clutch is a device that is operated electrically but transmits torque mechanically. There are many versions, but the primary version is a single-face design. The single-face clutch has four basic parts: the coil (as referred to as a field), a hub, an armature, and a rotor [29]. The coil is the primary magnetic piece that actuates clutch motion. It is typically made of carbon steel which combines magnetic properties with lightweight strength. By activating the electric circuit on the clutch, the coil is energized and produces an electric field. When the magnetic flux produced by this field overcomes the air gap that is between the armature and the coil, the armature is drawn to the rotor. This connection then permits the transmission of torque through the central clutch shaft. Magnetic and friction forces accelerate the armature and its hub to match the rotor speed. Rotor and armatures typically slip for about 0.02 to 1.0 seconds until the speeds match [29]. The single-face design is advantageous to the RPA because of its lightweight and simplistic design. A depiction of the electromagnetic clutch

is shown in Figure 8. By increasing the current to the clutch, the magnetic field strength will increase.

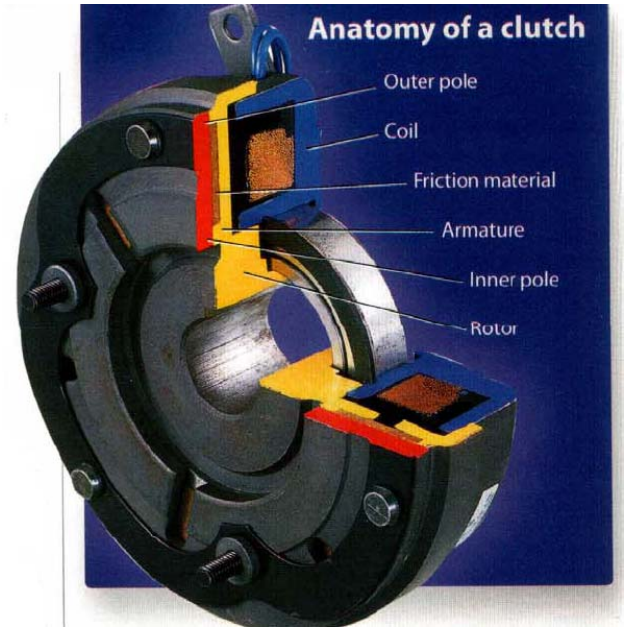


Figure 8: Electromagnetic clutch [29]

Magnetic flux and current are related by Equation 13:

$$dB = \frac{\mu_0 * I_{clutch}}{4\pi} \text{ X } \frac{dl * \sin(u)}{r^2} \tag{13}$$

where *dB* is the change in magnetic field strength, I_{clutch} is the clutch current in amps, μ_0 is the magnetic moment of the dipole, r is the displacement vector of the coil to the point of the magnetic field of interest, u is the angle between the vector and a current element dl. It becomes clear through this equation not only that increasing the current increases field strength, but applying the differing voltages to a clutch results in different currents, and therefore results in differing field strengths. Since the field strength is related to the grip strength of the clutch, this is of great importance when selecting a clutch for use.

2.5.5 One-Way Bearings

A one-way bearing is a simple device that allows power transmission from two sources. The bearing is typically a needle-type bearing with rollers. The rollers are mounted inside of a larger power transmission device, typically a gear. The torque from the shaft is transmitted by the rollers that wedge against the interior ramps when the input shaft spins. If the shaft does not spin, or in the case of the RPA the ICE is at idle, the larger gear will spin around the rollers and the bearing will not transmit torque. The specific setup of the HE-RPA will be discussed in Chapter IV, but a basic picture of the one-way bearing is shown in Figure 9.

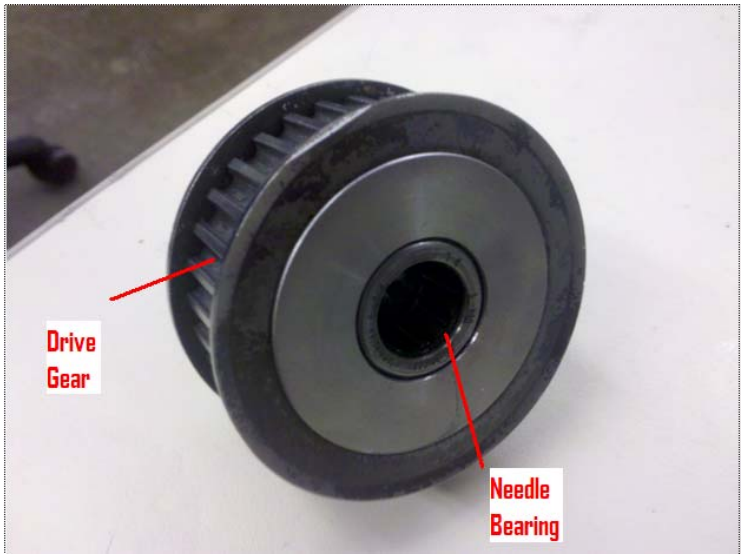


Figure 9: Example of a one-way bearing.

2.6 Testing Setups

The author would be remiss if testing setups were not discussed. One of the objectives of this thesis is validation of the controller model via testing versus an analytical model. Wilson [24] and Menon [25] are two previous researchers that have done a great deal of testing with small engines. Some pieces of their setups have been adapted or tweaked for use in the

controller test setup. Additionally, Rotramel [30] and Mengistu [31] have done a great deal of work on this particular test setup and have contributed greatly to its development. The basic challenges from testing setups come from signal noise and environmental disturbances. Ways to handle these noises usually involve implementing filters to correct and get rid of disturbances. Wilson in particular had to implement signal filters due to noisy operating environments with the small engines. The controller itself can also implement electronic filters to scale data and get rid of data that is noisy. The main goal is to increase the signal-to-noise ratio (SNR) to the best value. SNR is paramount, but there are other bigger challenges that mitigate this to a lesser role.

Another challenge is getting the controller to control the system as efficiently as possible, and this is indeed the main topic of this thesis. Different controllers handle differently in terms of how they optimize a system; and a discussion of each controller type and then the final choice of controller is discussed in section 2.9.

2.7 Rule-Based Controllers

Rule-based controllers are, conceptually, the simplest controllers to understand and implement. Rule-based controllers have been used throughout history and due to their ruggedness, provide very reliable designs. They can even be, in some cases, as efficient as their more ‘intelligent’ brethren.

A rule-based controller, at its very core, is simply a set of rules. The input to the particular system in question is a torque request. This torque request is fed into the controller, and the controller decides based on a given IOL which power mode to use. In this specific model, a torque request above the IOL then moves into the right hand side of Figure 10, where another decision based on available EM torque is made. Once the controller decides what

system to use and how much torque to be applied, the changes are fed into the system and the cycle repeats itself for as long as the system is operational. The controller is simplistic and easy to design, so the speed to make the calculations and decide on an action is comparatively fast. This can be easily demonstrated by a block diagram. This is a very basic model, but it

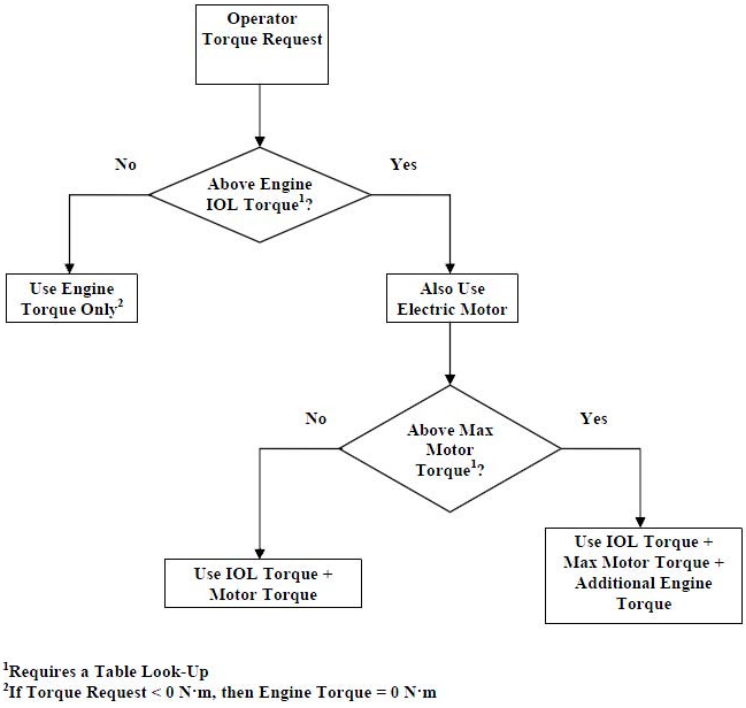


Figure 10: Rule-based controller block diagram [8]

demonstrates the advantages of a rule-based controller: simplicity and reliability. Additionally, since the number of rules determines its complexity (as in a fuzzy logic controller), the controller is only as complex as the designer makes it. Unlike a fuzzy logic controller, rule count does not exponentially increase.

There has been much research on rule-based controllers around the world. Harmon uses a rule-based controller to check his results on his neural network design. This controller uses a

concept that will be a topic of intense testing in this thesis: the ideal operating line concept discussed and tested in great detail by A.B. Fransisco from U.C. Davis [32]. The aforementioned IOL is essentially a line that describes points based on RPM and torque where the engine is operating most efficiently. The IOL is determined by varying engine speed and determining torque at its most efficient points. Theoretically, the engine will be restricted to operating at these points, and therefore efficiency will improve. A figure of Fransisco's IOL diagram for a car is shown in Figure 11. In the figure the IOL is shown in the yellow line. The strategy for

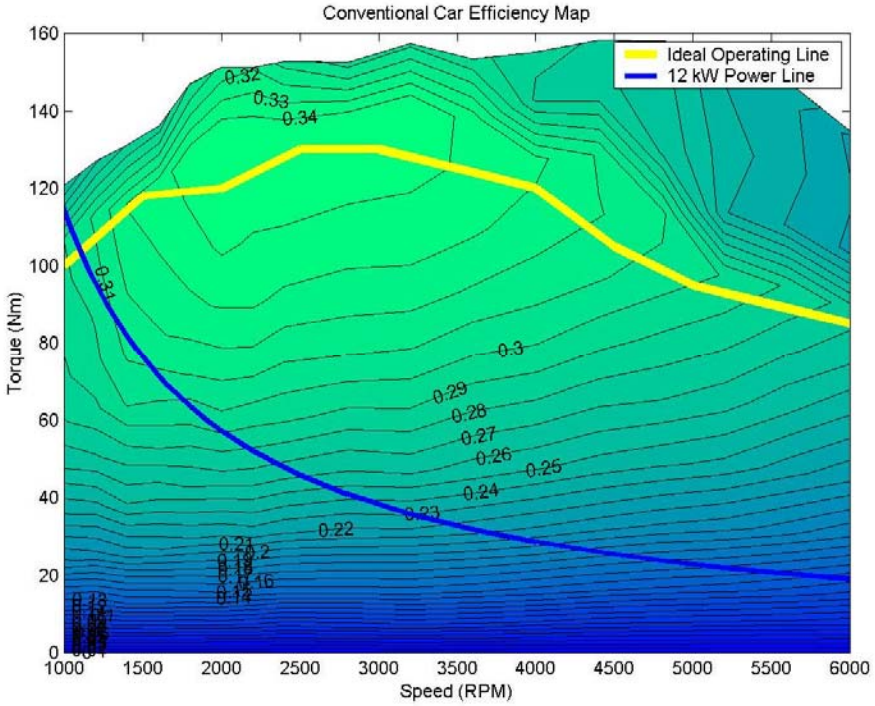


Figure 11: IOL for an automobile. Note conventional map with IOL overlay [32]

any type of controller would be to operate the engine at this line to maximize efficiency. A rule-based controller could possibly be ideal for this. There are several different strategies for rule based-control, and another thesis student, R.W. Schurhoff, has done quality work on explaining the different methods.

2.8 Rule-based Control Methods

There are many concepts for rule-based control that have been tested or used in the automotive industry. Most of these concepts, however, have not been used in an RPA application. Schurhoff, in particular, outlines in detail several concepts for rule-based control of a Continuously Variable Transmission (CVT) in his master's thesis [33]. These concepts are also repeated in a myriad of ways in other papers that discuss control strategies for HE systems. They will be discussed in detail below, as their relevance to the author's implementation of a control strategy is high.

2.8.1 Sloped Engine Engagement Strategy

In the sloped engine engagement, the engine is engaged and disengaged based on a simple measurement of vehicle speed (in this case the vehicle is a car). The vehicle starts in all-electric mode, and once the vehicle hits a certain speed the engine is engaged. If the speed

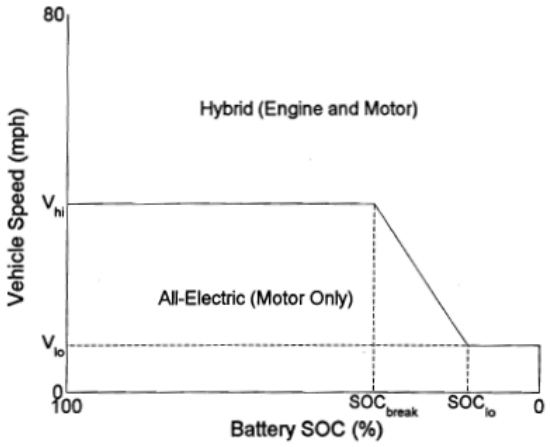


Figure 12: Diagram of sloped engine engagement strategy [33]

drops below a certain threshold, the engine is disengaged and the vehicle continues on in all electric propulsion. In this mode the SOC of the battery also affects engagement: if the SOC of

the battery drops to a set level, the engine will gradually be engaged to recharge the batteries. A diagram of this concept is shown in Figure 12. In this case the appropriate level of SOC recharge threshold would need to be selected based on vehicle design. Another thing to note is that the SOC_{break} and SOC_{lo} points have a slope between engine engagement and disengagement; this acts like a control gain between engagement and disengagement. If the slope gets steeper (the battery SOC is getting very low), the engagement speed will change more rapidly. In this particular control strategy, Schurhoff goes on to discuss limitations as they apply to an automobile. However, this design has different limitations when applied to an aircraft.

With an experimental RPA that is being designed for flight, several things that Schurhoff considered as limitations in his thesis are not applicable here. Primarily this includes emissions, because as the engine cools from being in extended EOM, tailpipe emissions are adversely affected. However, RPA's have no such limitation, and are not regulated in the same way cars are. Additionally, a simple 4-stroke engine that is used in small RPA's does not have an engine computer that will try to compensate for low engine temperatures by increasing fuel delivery. These small engines deliver roughly the same fuel/air mix provided the mechanical settings do not change.

Overall, this method is not viable for use exactly as described in Schurhoff's thesis. First, generation by the EM is nearly always going to be engaged in practice, as the HE-RPA has accessory loads that need to be compensated for by the EM in generation mode that a normal car would not have to compensate for. Most hybrid car designs use dual 12 V (for accessories) and a higher voltage system for drive, where as the HE-RPA will be using a single system for everything. Therefore, the EM is always going to be engaged and its engagement does not need

to be varied. However, for the purposes of testing, this method can be used and will be discussed in Chapter III.

2.8.2 The Stepped Engagement Strategy

Another strategy discussed is the use of a fixed line to choose when to engage the engine. The line would be preset based on the designer's choice for when to use battery recharging and how to operate the engine on its IOL. A figure showing this relatively simple engagement strategy is shown in Figure 13. This strategy's main strength is its reliable simplicity: it is

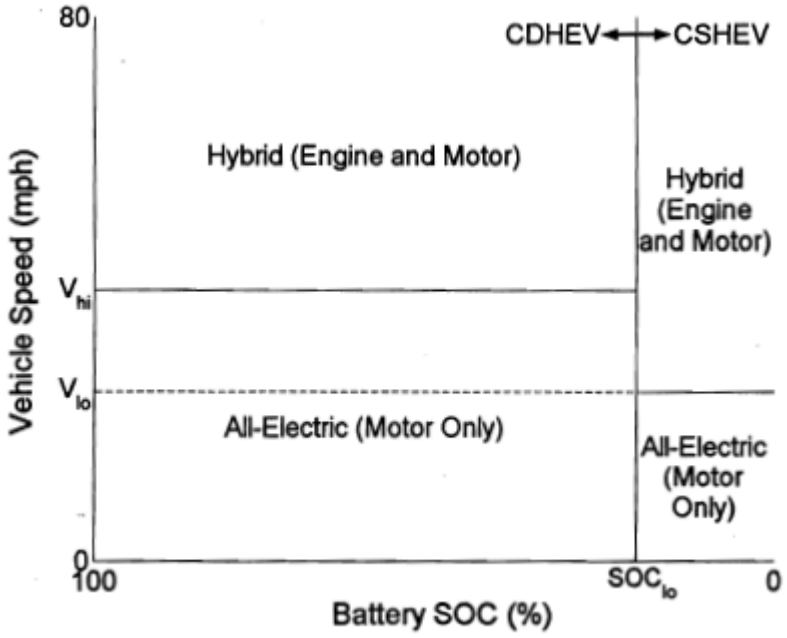


Figure 13: Stepped engine engagement strategy [33]

always known when the engine will engage and disengage, and buffers can be set in place to prevent undesirable rapid on/off cycles. Schurhoff again discusses many disadvantages to this

method in a land based vehicle, and again many of them are not applicable to an aircraft. In practice with a car, there is a negative torque applied to the power train as the engine is engaged from rest while the EM is propelling the vehicle. In this case, the clutch will slip as the engine is accelerated up to the EM speed. Passengers in this case will feel a slight jerk as the negative torque is applied, which is not desirable. However in an A/C, especially an unmanned one, this has no bearing as long as the negative torque does not adversely affect flight characteristics or sensors. The negative torque could, in theory, produce a large torque if the clutch moment arm was not along the centerline axis of the aircraft. This is explained in detail in Chapter III.

2.8.3 Two stage "pedal split" torque strategy

In the above cases, researchers primarily discussed how the engine is engaged and disengaged in respect to battery recharging and SOC. In the next two cases the primary discussion will focus on how to split the torque between the EM and ICE. First the simple method will be discussed. In the case of an A/C, the only input that the operator/autopilot directly has any command over is the throttle lever (with respect to the hybrid controller, directional controls i.e. ailerons and the rudder are not considered). The basic concept of this strategy is to split the throttle input into two regions: one for direct control of the EM torque and one for direct control of ICE torque. The throttle commands the ICE directly until its maximum torque is achieved, and then the EM continues to provide the remaining torque as the throttle lever is pushed to 100%. A diagram of this is shown in Figure 14. The limitations to this strategy are immediately apparent: they give no adaptability for differing conditions, and do not take into account the battery SOC. This particular strategy did give some insight into actual

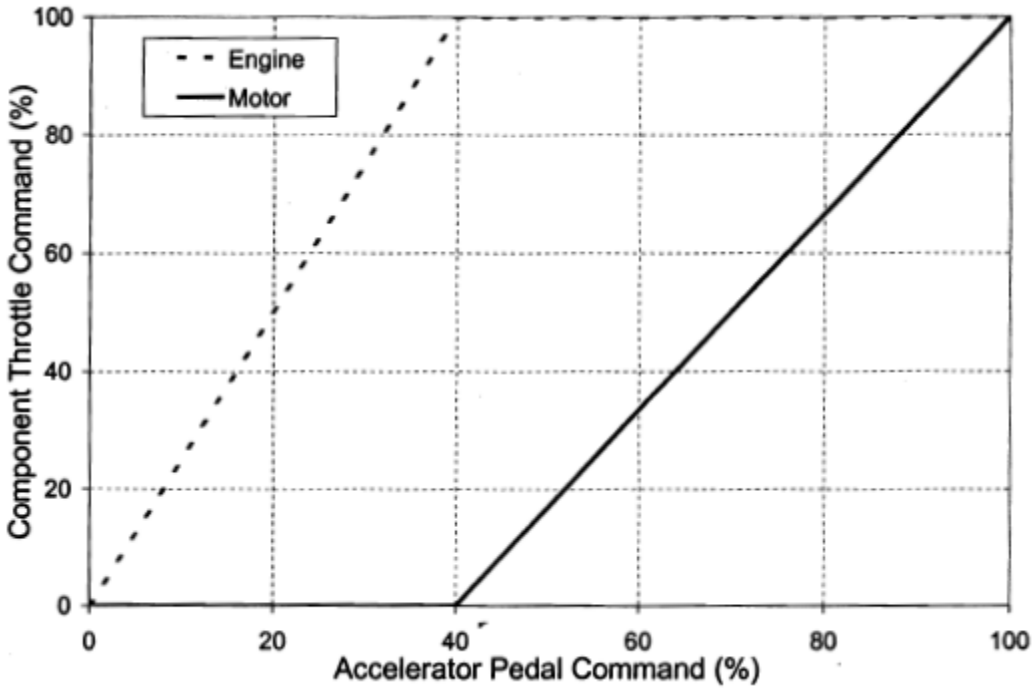


Figure 14: Two stage pedal split strategy [33]

implementation on the HE-RPA, and its modifications will be discussed later in Chapter III.

2.8.4 Three Stage Torque Proportioning

The three stage proportioning strategy is similar to the two stage strategy, with the exception that the top end has an additional setting for high power applications. In the case of a car, the first stage is engine use up to the IOL, the second stage uses EM torque, and the third stage uses engine fuel enrichment to give an additional boost torque for maximum power. The use of the IOL band for the engine, once again as in the two-stage method, allows for reasonably efficient use of the engine in its ideal region. The use of the IOL also allows the EM to be used conservatively to conserve battery power for endurance operation. The advantage here is the flexibility of the third stage. In an A/C with a mechanical carburetor, the third stage cannot be used for fuel enrichment since the fuel mixture is set before takeoff. However, the third stage

can be used for a myriad of other applications. For example, the third stage could be the "boost" phase of the EM, where it is operated outside of its continuous rated power regime for brief periods.

2.9 Intelligent Controllers

The field of intelligent controllers is one of intense study. There are several different types that are used currently on hybrid automobiles today. These types include fuzzy logic controllers and neural network controllers. Research that has been done on these types is described below.

2.9.1 Fuzzy Logic

Fuzzy logic controllers (FLC), at their very basic stage, are basically rule-based mathematical systems in which the logical variables can take on values between 0 and 1 rather than just 0 or 1 [34]. Fuzzy logic control, as described by Jantzen, is "control with sentences rather than equations." [35] Alptekin et al. describe fuzzy logic controllers as appealing for nonlinear modeling ability and robustness in the face of imprecise inputs [36]. At first glance, this may seem to be ideal in the airplane environment. In fact, fuzzy logic controllers have been implemented for autonomous flight control and such have been proven. Shown in Figure 15 are two diagrams for example fuzzy logic controllers.

However, there are drawbacks to a fuzzy logic controller. First, its' simple method for determining solutions to problems is also a hindrance. Since a fuzzy logic controller use rule sets to make decisions and the problems are often nonlinear, the control algorithm goes through "exponential rule expansion." Each input variable in this case has a separate rule formed

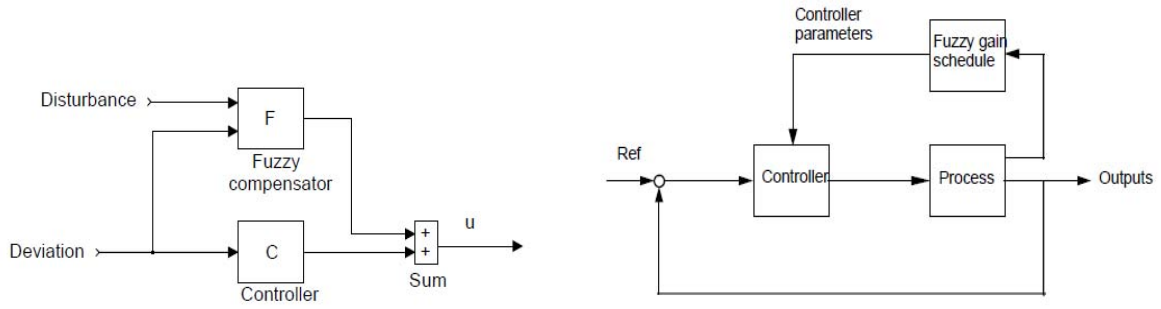


Figure 15: Block diagrams of two fuzzy logic controllers. Pictured left is a feed forward controller, right is an adaptive parameter fuzzy logic controller.

for each possible combination of input membership functions. The system works well for simple low-input systems, but the rule sets can quickly increase to an unmanageable number with even a small increase in the number of inputs. In the case of the RPA, there are 16 designed inputs to the controller. If the controller has **X** amount of membership functions for the input, this would require 16 (in some cases even more) raised to the **X** rules. It becomes easy to see that rule sets become unwieldy after X is greater than 4 or so. Usually an attempt is made to reduce the amount of rules by ignoring rules that would never appear in the operating environment, and this is probable in the situation of the RPA propulsion system. Many different fuzzy logic controllers have been used in HEV's, most notably authors like Salman, and Lee. Salman's controller in particular, focuses on the energy management for a CS type of HEV [37]. The controller has advantages when used for supervisory, task oriented control. It can also permit the designer to design the controller so that it will mimic his or her own preferences.

2.9.2 Neural Network Controllers

The neural network controller is another type of intelligent controller. These types of controllers are useful when the plant model is very difficult to model exactly. As the name may

suggest, a neural network controller attempts to mimic the human brain when modeling and controlling a system. Hagen and Demuth describe the neural network controller as a function approximator [38]. In this case of a neural network, the network is made up of many individual pieces called neurons. Each neuron has a scalar input which is multiplied by a weight, *w*. This

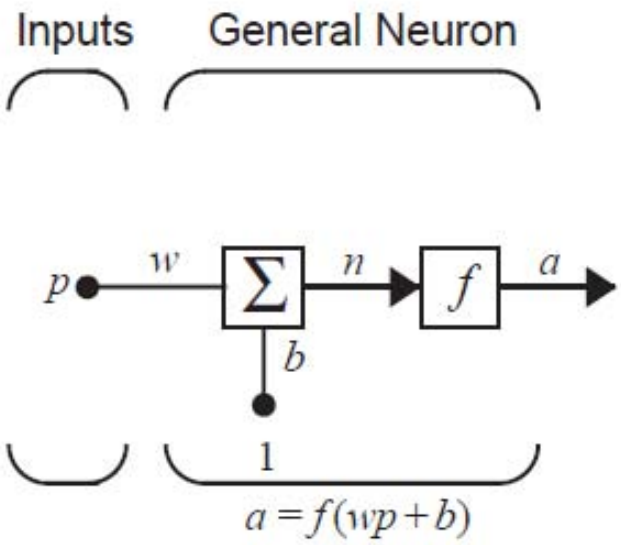


Figure 16: Basic neuron for a neural network [38]

input *wp*, is then added to a bias, *b*. The result is sent to a transfer function. The transfer function produces an output, which is the scalar output from this neuron. The transfer function is chosen by indentifying what particular piece of the problem that the neuron is attempting to solve. The above description is of a single input neuron, but neurons can have multiple inputs. To truly get the greatest level of control, multiple neurons are used and combined into several layers. These layers combine into the neural network, and are considered universal approximators [38]. Once these networks are formed, they are trained. Neural network training is basically another way of saying that the weights and biases need to be determined.

Neural networks require this training to learn and adapt their coding to new problems. Neural networks are trained using a variety of methods, some of which are based on gradient-descent approaches. The most popular way of training is by using the back-propagation method. As stated above, this is based on a gradient descent method. For multilayer networks, the output of one layer becomes the input to the next layer. The back-propagation method itself completes this process by using the gradient descent optimization procedure. To start the algorithm, a set of examples are provided that model proper network behavior. As each input is applied, the output is compared to the target output of the system. The algorithm then corrects and adjusts the network to minimize the error of the output and the target output. By using the steepest descent method, the negative of the gradients of the function will always guarantee a descending direction, but it is not necessarily the most efficient way to minimize the error. There are many other approaches, but as this is not the focus of this paper, they will not be discussed at this time.

2.9.3 Other Intelligent Controllers

Fuzzy-Logic and Neural Network controllers are two of the most prominent examples of intelligent control, but there are other methods that have been used. An *adaptive controller* is designed to react and adapt to unknown parameters in a plant [39]. Matthews has done research on designing an adaptive controller for micro air vehicles (MAV). A rigid, non-adaptive controller could be optimized for one path, but if unknowns force a deviation then that controller will either function less optimally or may even become unstable. An adaptive controller could adjust to this deviation and change parameters in the plant, thereby minimizing the error in the system. Most adaptive controllers achieve this by some form of state estimation, where the estimator is used in conjunction with changing controller parameters to adjust the control signal and minimizing the error [40].

Another type of intelligent controller is a controller that uses optimization based on genetic algorithms. This type of controller minimizes (or maximizes) the objective function using a multi-point search, as opposed to a single point search. In controllers without genetic algorithms, a single point search can be slow and computationally intensive. The search also can become stuck in local minima and not actually converge on an optimal solution. Additionally, the strict single point mathematical method requires the condition that the variables in the problem are continuous, which a genetic algorithm does not require. Genetic algorithms employ search procedures based on natural selection, and since it uses a multi-point search rather

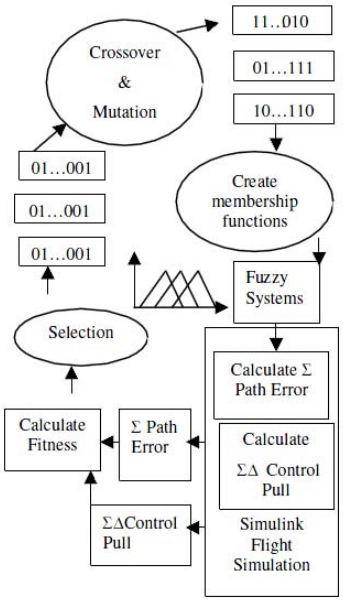


Figure 17: Genetic Algorithm block diagram [36]

than a single point search, it can adapt to irregular search spaces much better. Additionally, the algorithm is made better by the introduction of random changes to key conditions after each new 'generation' has been produced. A diagram of the genetic algorithm controller is shown in Figure 17.

2.10 Selection of Control Method

Based on the research done by the author and the advisor, a rule-based controller was selected for use in controlling the prototype RPA propulsion system. The rule-based controller was selected because of its inherent simplicity and because of the work that had already been previously done by Harmon. Since the aircraft will be flight tested and subject to rigorous government regulations, simplicity was deemed paramount. This enabled the author to base the thesis off this work and have a more efficient design that will make the propulsion system proof of concept much more effective.

III. Methodology

3.1 Chapter Overview

As outlined earlier, rule-based control can quickly become very complex when different scenarios and situations are considered. Chapter III outlines the methodology used by the author in developing the controller logic for control of the hybrid propulsion unit. The chapter begins by discussing various concepts of rule-based control that will be evaluated. This is followed by outlining the various states that are used inside the overall state machine of the controller. Worth noting is that the parameters listed in the tables in these sections are for specific test setups, but the controller is general enough that the parameters can be easily changed to adapt for different engines, motors, or gear ratios. The chapter concludes with an overview of the equipment used to test and verify proper operation of the controller, and the procedures used for validation.

3.2 Open-Loop State Machines

The controller used here is described as an open-loop state machine, but that requires some explanation. In an open-loop controller, the error of the true value and the actual commanded value are unknown to the controller. It does not use these values to make any fine tuning to the parameters. In a closed-loop controller the reverse is true; the controller has feedback to receive the values of error. It uses this to adjust the parameters accordingly. The rule-based controller presented here is not a closed-loop machine in a true sense, but it can have pseudo closed-loop behavior. The piece that actually closes the loop is the operator. The operator of the A/C has knowledge of the A/C speed and flight characteristics and adjusts the throttle accordingly. The controller interprets that signal and then decides how to split the power

sources on the A/C. This key fact is used throughout the code to establish an efficient open-loop state machine.

3.3 The State Machine

The controller for the HE-RPA uses a standard computer science "state machine" for governing the controller's decisions. The concept of programming the controller is to get a basic structure of the controller working, with many manual modes of operation. Then, as the controller is fine-tuned, more and more of the basic functions are automatically controlled rather than requiring the user to control it. The state machine (SM) of the controller contains many "states;" each of these states represents a different operating regime. These regimes are outlined by Harmon [8] and broken down further for the purposes of this thesis. The different regimes are (with state numbering system as used by the controller):

0. Reset
1. EM Rev
2. ICE Start
3. ICE Idle
4. Ground Roll Style Takeoff
5. Catapult Launch Style Takeoff
6. Climb
7. Cruise without regeneration
8. Endurance
9. Cruise with regeneration

Each of these modes will be described in detail in the following sections. The basic components over which the controller has direct authority are: the clutch [41], servos for the throttle and choke, propulsion DC-to-DC (DC/DC) converter, and the generation DC/DC converter. A flowchart of the controller's logic can be seen in Appendix B.

3.2.1 Reset

The reset state is the beginning state of the controller. In essence, it provides the operator confidence that every piece of the propulsion system is disabled prior to flight. It also serves to provide the operator a method of "emergency control" via a switch mounted on the outside of the system. If anything goes wrong, the operator hits the switch and the controller enters this state. The state disables every piece of the system as described in **Error! Reference source not found.**.

Table 1: **Reset state component control**

Component	Setting
Clutch	Disengaged
Engine Throttle Servo	0%*
Choke Servo	100%*
Propulsion DC/DC On/Off	Off
Propulsion DC/DC Output	0 RPM
Propulsion DC/DC Current Limit	0 Amps
Generation DC/DC On/Off	Off
Generation DC/DC Current Limit	0 Amps

*Servos controlled by PWM signal

As discussed earlier, the reset state can be controlled a number of ways. The primary way that the state is entered into is by means of a run/kill switch mounted on the side of the propulsion system. This run/kill switch allows the operator to manually control the system if there is a problem. The controller also defaults to this mode upon boot, which facilitates system safety. In flight, however, the controller is prohibited from entering this state by use of if/else loops within its programming. If the autopilot or operator is providing a PWM signal to the controller, it cannot enter this state to prevent accidents with the propulsion system components being disabled in flight.

3.2.2 EM Rev

The purpose of the EM Rev state is to prep the propulsion system for ICE start. There are various methods to enter this state, but the most basic way is when the operator hits the run/kill switch. Hitting this switch automatically enters this state as it signals the controller that the A/C is ready for flight. Any ICE engine can be started by various methods, but all methods require the ICE crankshaft to be spinning prior to initiation of the combustion event. Most Radio Control (R/C) A/C use an external starter motor designed explicitly for this purpose. While this is practical in a friendly airfield setting, it is not always viable for troops on the ground to carry around the additional starter, which adds weight. Therefore, the design of the controller includes this state, which revs the EM up and then rapidly engages the clutch to start the ICE spinning. While this may seem hard on the clutch itself to put a large load on it a short amount of time, the clutch is overdesigned specifically for this purpose and can endure the additional loads. Additionally, in preparation for engine start, the throttle to the engine is opened in accordance with (IAW) manufacturer instructions and the choke is half closed. Engine manufacturer

instructions necessitate manual operation of a cold engine for starting, and the code contains programming that can command the choke manually through clearly labeled switches on the transmitter. However, for a warm engine this state could be directly entered for easy starting. The engine throttle is opened to facilitate easier starting, and the closed choke is also a necessity to ensure ease of engine starting. Here note that 100% for engine servo indicates Wide Open Throttle (WOT) and 100% for the choke servo indicates a fully closed choke. The controller component control is shown in Table 2. Note also that the

Table 2: EM rev component control

Component	Setting
Clutch	Disengaged
Engine Throttle Servo	30%
Choke Servo	50%
Propulsion DC/DC On/Off	On
Propulsion DC/DC Output	4000 RPM
Propulsion DC/DC Current Limit	30 A
Generation DC/DC On/Off	Off
Generation DC/DC Current Limit	0 Amps

DC/DC converter selects an RPM here, not a voltage. This will be discussed later in section 3.2.8. Again, the parameters in these tables are specific to a certain configuration, and can easily be modified within the controller programming.

3.2.3 ICE Start

The ICE start state is where the ICE is started in preparation for flight. The state is entered once the controller detects that the EM has reached 4000 RPM. The controller component control is shown in Table 3. This is accomplished through use of a digital RPM

Table 3: ICE start component control

Component	Setting
Clutch	Engaged
Engine Throttle Servo	30%
Choke Servo	50%
Propulsion DC/DC On/Off	On
Propulsion DC/DC Output	4000 RPM
Propulsion DC/DC Current Limit	30 A
Generation DC/DC On/Off	Off
Generation DC/DC Current Limit	0 Amps

sensor which will be discussed in section 3.3. Once the RPM is approximately greater than 3800 RPM for three seconds the controller engages the clutch for engine starting. Three seconds is required for practicality, as the RPM in reality will never be exactly 4000 RPM for even one second. The EM then turns the ICE for approximately 6 seconds with the throttle at 30% and the choke half-closed. The controller then monitors the ICE RPM sensor for RPM that is over 5000 RPM. If this occurs it triggers the next state automatically. If this does not occur, the controller

automatically defaults back to the reset state, in which case the operator must toggle the run/kill switch from off back to on to repeat the process.

3.2.4 ICE Idle

The ICE Idle state is entered once the controller detects the engine RPM is greater than 5000 RPM. Component control for this state is shown in Table 4. Having the engine RPM at

Table 4: ICE idle component control

Component	Setting
Clutch	Disengaged
Engine Throttle Servo	25%
Choke Servo	50%-0%*
Propulsion DC/DC On/Off	Off
Propulsion DC/DC Output	0 RPM
Propulsion DC/DC Current Limit	0 A
Generation DC/DC On/Off	Off
Generation DC/DC Current Limit	0 Amps

*Note here that the choke is opened fully as the engine warms up, after initially being half closed.

5000 as the trigger for leaving the state is deemed acceptable as the maximum RPM is 9800 RPM, and even though the engine is not at WOT, it will easily reach 5000 RPM as it starts because the throttle is open partially. The 5000 RPM generality could also easily be changed in

the code as it is a simple constant. The purpose of this state is to allow the ICE to idle and warm up according to manufacturer instructions. Additionally, smoother and more efficient operation is achieved when the ICE is at operating temperature. The next state is triggered when the operator hits a switch on the remote to activate the change. The operator can see the engine temperature via a LabView screen which monitors the engine thermocouple. Alternatively, for the real A/C the operator could see this data through a telemetry data or through a memory storage method on the controller board.

3.2.5 Ground Roll or Catapult Style Takeoff

The next state for the aircraft is entered when the aircraft has indicated that it is ready for flight. This can be accomplished in a myriad of ways, but currently a switch on the remote is flipped to engage the aircraft for takeoff. As this is a prototype aircraft, the actual type of takeoff has not been specified for the final design. There are two methods for an RPA of this size: a ground roll style and a catapult launch style. Table 5 shows the component control for the ground roll style, as this was deemed the most likely method of being selected. Code for a catapult launch is included, but is currently out of the state machine loop.

For the ground roll takeoff, the clutch and electric motor are both engaged. In the case of this HE aircraft, additional power is required to get the aircraft up to cruise altitude due to the undersized nature of the gasoline engine. The controller code (shown in appendix A) is programmed to use the gasoline engine up to its maximum torque and then utilize the electric motor for the remainder of the torque request from the autopilot. How each component is specifically controlled and monitored will be discussed in later sections. Table 5 shows the component control for this state. For the remaining states in the machine, a linear progression is

not utilized. A linear progression would make little sense in an environment where the controller is switching states back and forth due to various demands on the system. At this time,

Table 5: Takeoff component control

Component	Setting
Clutch	Engaged
Engine Throttle Servo	Variable
Choke Servo	0%
Propulsion DC/DC On/Off	On
Propulsion DC/DC Output	Variable
Propulsion DC/DC Current Limit	10 Amps
Generation DC/DC On/Off	Off
Generation DC/DC Current Limit	0 Amps

the controller does not have the ability to shift states by itself. The operator has a bank of switches on the transmitter that are engaged in a fashion to enable each of the next four states as he or she desires. Ideally, an ICE only or a power regeneration mode would be engaged for cruise, the EM only mode would be engaged for endurance, and the climb mode for climbing By allowing the operator to control the modes of operation and states for flight, essential debugging of each operating mode is afforded without interference from the controller.

The four states discussed henceforth are the four most important states for the controller, and are the main focus of this thesis. Their implementation and execution make up the bulk of the advantages of a hybrid system when completed correctly and, as such, will be discussed at length.

3.2.6 Cruise without Regeneration Mode

The cruise without regeneration mode is engaged by flipping switch A on the transmitter as shown in Figure 18. There are several purposes behind a cruise without regeneration mode.

Figure 18: Transmitter depicting switch A.

Perhaps the most important: while a HE system utilizes both an ICE and an EM, the ICE is still the main focus of the propulsion system. This fact alone demands that the aircraft be able to be

flown with just the ICE at cruise speed. While it is undersized compared to a traditional aircraft of this size, and performance may be degraded with just the engine, the aircraft can still fly and make it home if something were to go wrong. Ideally, cruise without regeneration mode will either not be used when being flown automatically; or will be used in cruising when no regeneration is needed. However, it still has practical uses in training an autopilot or traditional pilot to fly the aircraft.

The basics of this mode carry over to the other three modes as well. Equations were developed by Harmon at U.C. Davis for use in a HE-UAV [8]; these equations are modified for use in the RPA. To begin the state, the controller disables the electric motor generation and propulsion, so the EM is essentially freewheeling on the ICE shaft. This will induce losses, but these are deemed unavoidable in order to avoid increased mechanical complexity. The controller then waits for an input from the autopilot or transmitter receiver to begin its command loop.

A traditional autopilot and its command style were utilized heavily in developing the equations for the controller commands. Traditionally, an autopilot is “tuned” to adjust the throttle on a RC aircraft correctly. This tuning involves getting the autopilot to “learn” how much available torque it has and how to adjust the throttle to get the torque neccessary to keep the aircraft in flight. The autopilot, when tuned, then sends a PWM duty cycle signal to a servo motor which is connected to the ICE throttle. This signal is between 0-1 (for 0-100% duty cycle). Therefore, if the autopilot theoretically had 3.0 N-m of total torque available to it, and it knew that it needed 1.5 to fly, it would adjust the throttle to 50%. This would occur continuously many times a second.

This process is exactly how the controller commands the ICE. The torque request is read into the controller by intercepting the signal sent by either the transmitter or the autopilot. The controller then does several things to convert this signal into a usable torque request that is passed as a throttle command. These are shown in Figure 19, which starts the process by reading in the requested torque from the source. In Figure 19, the function *GetTorqueRequest()*

```
double GetTorqueRequest()
{
	double throttleSetting = GetRCDutyCycle(AutopilotThrottle);
	// Will return 0-1.0

	double totalTorque = GetTotalAvailableTorque();

	return throttleSetting * totalTorque;
}
```

Figure 19: Function that gathers total torque request from controller inputs

calls the function *GetRCDutyCycle()*, which reads the total duty cycle request coming from the autopilot or human pilot via transmitter. It also calls *GetTotalAvailableTorque()* which gathers the total available torque from the hybrid system at its maximum. The controller then initiates several more functions that determine the actual throttle command.

First, it gathers information about how much available torque is currently available with *GetTotalAvailableTorque()*. This is done in two pieces. Previously in a traditional setup, the autopilot only has one torque providing piece. However, in an HE system there are now two sources of torque. Since the autopilot (or pilot) is tuned to know how much total torque it has, it now has to know the sum of the two at all times. For the EM this is relatively simple. An assumption is made that in this state the EM is never going to be spinning at its max speed. This assumption is important because the EM efficiency drops off as the maximum speed is attained. Therefore, the EM max torque available can be assumed to be a constant value, because for most

EM (this one included), maximum torque is provided from 0 RPM to a certain limit. Manufacturer information and test data confirms this assumption; manufacturer ratings show maximum torque is available for most of the motors speed range [42].

For the ICE torque available, the controller has to monitor the RPM via an optical sensor mounted near its output shaft. Programmed into the controller are a set of basic "maps" that identify the ICE maximum output torque at various RPM values. Unless the RPM map is infinitely stepped so that it has millions of possible RPM and torque combinations (impossible with limited storage space), it needs to interpolate when the RPM falls in between values and then pick a known RPM value. This is done via another function that handles the interpolation. The function then returns the max torque at this RPM to the controller. Once the value of max torque is known, it is converted into a throttle signal by normalizing it. This is done in Harmon's original code and is done the same way here by using Figure 20. Here each

$$normalizedICETorque = \frac{torqueRequest}{MaxICETorque}$$

Figure 20: Normalized ICE torque equation

variable is shown as it is coded; *normalizedICETorque* is the throttle command passed to the ICE throttle servo, *torqueRequest* is the torque request determined in Figure 19, and *MaxICETorque* is the maximum available ICE torque which is determined as discussed above.

In the case of the cruise without regeneration operation, the request will be based on the available torque of the EM and the ICE together, even though the EM is disabled. In the case of the prototype RPA, suppose the throttle stick on the transmitter is at max. The maximum torque that can be provided with both the EM and the ICE at 4000 RPM is 3.48 N-m; this is the torque

request. When divided by the maximum torque available (which for the ICE at 4000 RPM is roughly 2.1 N-m), this leaves a number that is over 1. A signal sent as such would prove problematic as the servos can only handle PWM commands between 0-1. Therefore, to avoid control issues the throttle commands are saturated so that the only possible values returned to the actual throttle command are between 0-1. In the above case, this would provide a max throttle signal of 1, and the ICE throttle would be at WOT. Actual performance of this state is detailed in Chapter IV.

3.2.7 Endurance Mode

Endurance Mode is engaged by flipping switch B on the transmitter as shown in Figure 21. The purpose of Endurance mode is to provide for quiet, efficient operation while the A/C

Figure 21: Transmitter depicting switch B.

is circling over the target. In this mode, the ICE will be idling. Having the ICE idle versus shutting it down completely was deemed acceptable for several reasons. First, in talking with operators of this type of A/C, restarting an ICE in flight can be difficult. Fuel can leak into the cylinder, creating a hydro lock situation where the EM cannot physically create enough torque to force the fluid out of the cylinder. The fluid in the cylinder also impedes the spark from actually igniting the fuel. Additionally, in personal experience with the Fuji engine, if the engine was on the compression stroke the torque required to turn the engine over is more than the EM can provide. This happened often due to the tendency of the engine to stop on this stroke. Finally, the ICE at idle is actually far quieter than the noise that an 18 inch or 20 inch propeller will make. Testing results proving this can be seen in Todd Rotramel's term paper on acoustic testing [43].

Therefore, once switch B is flipped, several things occur. The controller disengages the clutch, allowing the EM alone to power the A/C propeller shaft. The propulsion DC/DC converter is switched on and begins providing power to the EM. The throttle on the ICE is returned to its idle position. At this point the operator would check to make sure all these things occur before continuing. Actual control of the electric motor lies with the DC/DC converter. Traditional motor control is through the use of speed control. The voltage on a small controller is varied and this changes the speed of the EM. However, in order to use the equations developed and make a throttle input into a torque setting, the motor must be controlled by its torque output. This occurs by controlling its current.

Power out here then depends on the efficiency of the motor, and torque can be determined by measuring the rotational speed and the power output. Normal brushed EM's are

controlled by regulating the voltage to set a speed, and then allowing the current to be whatever it needs to be. By controlling the current however, more precise control over the motor is allowed based on its output torque. The torque request coming into the controller is similar to the cruise without regeneration mode, but instead of using maximum engine torque maximum motor torque is used, as shown in Figure 22. where *normalizedEMTorque* is the resulting

```
double normalizedEMTorque = GetTorqueRequest() / MaxEMTorque;
```

Figure 22: Equation for normalized torque request in endurance mode

torque request to the electric motor, *GetTorqueRequest()* is the same function that computes the requested torque from the pilot, and *MaxEMTorque* is the maximum motor torque as specified by the manufacturer. Once the torque request is read in, it is passed to another function that determines how to set the EM. The result of this is that the torque request is de-normalized into a value in Newton meters. By then dividing by the motors torque constant (specified by the manufacturer), the result is an amperage that will provide the required torque. Voltage from the battery will then drop to provide whatever amperage is required. In theory, by setting the EM voltage to be a maximum of 40.0V, the speed should be very high which would result in an inefficient propeller. However, since the EM is loaded down, the voltage will drop as the load is increased. Temperatures in the motor windings and shaft bearings will increase, but as long as they are monitored and kept below a threshold, optimum endurance can be achieved with careful torque control.

3.2.8 Climb Mode

In climb mode, both the EM and the ICE are used to provide torque to the A/C propeller shaft. By flipping both the A and the B switch on the controller (depicted in Figure 18 and

Figure 21), climb mode is engaged. Two strategies for implementing this mode were explored and tested, with results shown in Chapter IV. The first mode, as discussed in Chapter II, is the use of the ICE up to the IOL, and then continued use of the EM until the torque required is provided. The torque value is interpreted as in the previous two modes, with the incoming throttle signal being interpreted as a percentage of maximum torque that the two power devices can provide. The controller, knowing the engines RPM, decides if the engine can provide torque while staying at or under its IOL. If the engine can, the controller sends an equivalent throttle signal and the EM is not used. If it cannot, the controller feeds a throttle signal equivalent to the engines IOL line at that speed, and then uses the EM to make up the rest of the torque difference.

Since the controller bases its measurements off engine speed and does not adjust the throttle until after the calculation, it becomes easy to see that the engine could become "stuck" at a low RPM, which would force the EM to provide a large amount of torque. This is undesirable due to energy storage limitations and would drain the batteries very quickly. To avoid this, the controller runs an additional decision block embedded in the throttle setting procedure. This block triggers if the EM is providing more than 75% of the total torque for the system, in which case the controller bumps the throttle up by 10%. This is a simple method and is shown with

```
if(remainingTorque > 0.75*GetTorqueRequest())
{
      SetPWMDutyCycle(ICEThrottleServo, normalizedIOLTorque+0.1);
}
```

Figure 23: Throttle bump if statement

Figure 23. Here, *remainingtorque* is the torque command (between 0-1) that is passed to the EM command function, and *GetTorqueRequest()* is the same function as used in cruise without regeneration and endurance modes. Since the controller goes through the entire process many

times a second, this occurs very fast and does not adversely affect controllability. If the EM drops below this power level, the statement is ignored and the controller proceeds as normal.

As discussed earlier, the two or three stage torque splitting method could be implemented fairly easily on a hybrid A/C. In this method, the engine again is operated up to an ideal point, but in this case it uses several engine maps, not a simple line, to make the decision. There are three maps embedded in the controller, all of size 5 by 11. The maps are based on data collected by Isseyas Mengistu when bench testing the engines [31]. The first map includes torque values of the engine, where the X axis is engine speed and the Y axis is also torque. The second map contains the fuel usage numbers, again based on testing, that correlate to the torque the engine can provide. The third map contains the throttle setting required to achieve this torque and fuel use. By using these three maps, the controller reads in the torque request and can then decide on the lowest fuel usage point for the engine to provide the given torque. It then sets the throttle at this point, and uses the EM in the same way as in the basic mode, to make up the remaining torque. If the operator adjusts the throttle up or down, the controller reads in that request and resets the engine RPM with the throttle according to the transmitter stick positioning. The RPM measurement is used as a check to make sure the engine is operating at the correct point.

3.2.9 Cruise With Regeneration Mode

The purpose of this mode is to recharge the battery pack by using the electric motor as a generator. It is engaged by flipping all of the mode switches on the transmitter. Recharging in flight will typically be done in cruising conditions, as this is where the ICE has the most available torque to provide for regeneration. Recharging is deemed a necessary ability as the battery pack will be mostly drained for endurance operation, and the A/C power electronics need

power for the rest of the flight. Currently, the recharge mode cannot be used in any flight condition except cruise, although future iterations of the controller will have automatic recharging. Data passed back to the user will indicate the battery voltage and signal the user when it is time to recharge.

The mechanics of recharging are meant to be as simple as possible. Recharging is done through a separate SynQor DC/DC converter. The converter is the same as the one used for propulsion, but wired in reverse in order to provide power to the pack rather than drawing from it. By using current limiting, the pack can be safely recharged in flight.

Lithium Polymer (LiPo) batteries are extremely sensitive to heat and overcharging, so precise monitoring of the pack is necessary. The primary indicator of charge status, besides using complicated techniques to estimate power draw, is to watch the pack voltage over time. The voltage will drop as the pack becomes more and more discharged. In recharge mode, the controller monitors battery voltage and temperature. Overcharging, over temperature charging, charging too quickly, and charging a pack that has been depleted past a safe point are all dangerous and can cause a fire. Therefore, there are safety interlocks in the controller to force the user out of recharging mode if either a) the pack voltage drops below 21 volts (3.0 volts per cell) or b) the pack temperature is over 50 degrees Celsius. This interlock is designed so that even if the switches are still in the engaged position, the electric motor will not engage back into generation mode. The interlock is shown in Figure 24. The logic of the charging mode is taken from a commercial off-the-shelf (COTS) battery charger that is used with LiPo batteries [44]. The charger uses a common method to recharge these batteries, which involves charging in three stages. Due to limitations with the DC/DC converters, only two stages of the three are used.

The first stage is initial charging. This stage gives the battery a low current charge to bring the pack voltage up above the starting voltage, and is fairly brief. The second stage is the constant

```
double batteryVoltage = GetBatteryVoltage(ANPORT10);
//Do not charge if battery voltage drops below the 3.0 V min.
if (batteryVoltage < 21.0)
{
      propulsionState = ICEONLY_PROP;
      SetPWMDutyCycle(ICEThrottleServo, 1.0);
}
//Stop charging if battery temperature climbs too high
if(batteryTemp > 50.0)
{
      propulsionState = ICEONLY_PROP;
      SetPWMDutyCycle(ICEThrottleServo, 0.5);
}
```

Figure 24: Safety interlocks for battery charging

current charge which comprises the bulk of the charging. In this stage a constant current is applied to the battery, with voltage draw being whatever is necessary to accomplish this. Lower voltages will necessitate using a 3.0 Amp charge rate, which is very close to the maximum limit. Higher voltages will use a slower charge rate. The stage completes once the voltage hits 27.5 volts, which is when the COTS charger shifts its modes as well [44]. A section of C code depicting this is shown in Figure 25. Here, the function *setcurrentlimitNQ40* is a function that sets the current limit on the generation DC/DC converter. As depicted, the current limit will change depending on the state of charge of the battery. Once this stage is completed, the controller shifts into a constant voltage charge mode. The principle is the same, with the only difference being that the output voltage is not set instead of being allowed to float. A section of C code is depicted in Figure 26. Once the charge cycle is complete, the generation DC/DC converter is switched off. The ICE will continue to run without the motor generating any power,

so if this statement executes, the propulsion cycle is switched automatically back to ICE-only mode.

```
//Constant Current Charge


if(batteryVoltage < 22.0)
{
      SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 3.0);
}
else if(batteryVoltage < 24.0 && batteryVoltage > 22.0)
{
      SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 1.8);
}
else if(batteryVoltage < 27.5 && batteryVoltage > 24.0)
{
      SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 1.0);
}
```

Figure 25: C code for regeneration

```
//Constant Voltage Charge
if(batteryVoltage > 27.5 && batteryVoltage < 29.0)
{
      SetOutputVoltageNQ40(GenerationDCDCOutputVolt, 29.1);
      SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 1.0);
}


//Charge cycle complete, discontinue charging
if(batteryVoltage > 29.0)
{
      SetDigitalOutput(GenerationDCDCOnOffPort, GenerationDCDCOnOffPin,
FALSE);
}
```

Figure 26: Constant voltage C code, with “cycle complete” text

3.4 Test Setup

The controller, once programmed, was in need of a method of validation. A test setup based on an engine dynamometer was developed, and a LabView data collection program was written to assist in data collection. The outlying setup is shown below in Figure 27, and it will be broken down individually in the subsequent sections.

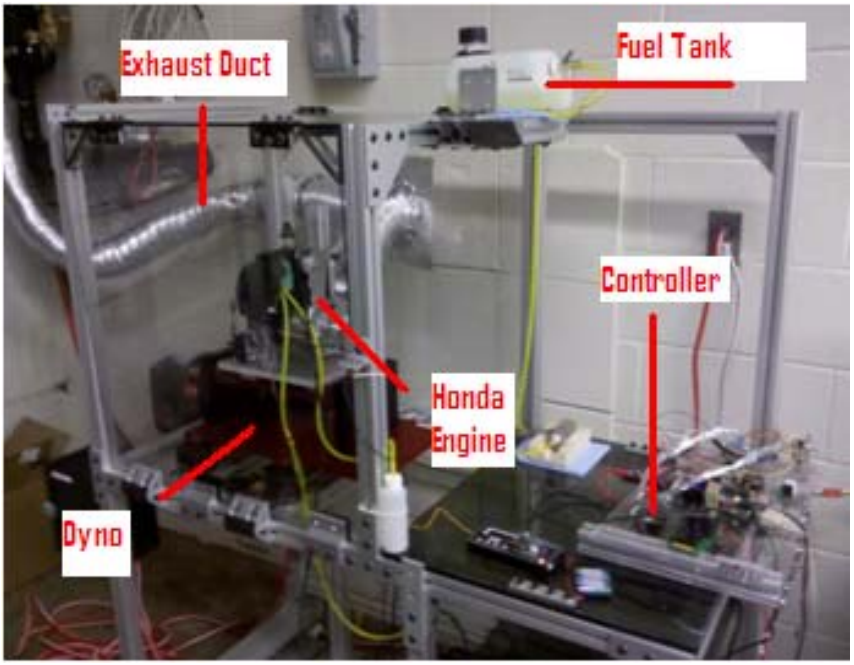


Figure 27: HE-RPA propulsion system test setup w/ Honda engine

3.4.1 Data Collection Setup

Two pieces of software were used to collect data. First, LabView was used to collect the majority of the data, including several controller parameters, engine and motor RPM, and current information. Second, the dynamometer proprietary software was used to collect torque information. The basis of the data recording and signal measurement was done by John Hagen, a computer engineer. The screen shown is a modification of the baseline block diagram that he developed while working for Harmon. All gauges related to the ICE are to the left, with the major gauge (RPM) colored in white. All gauges related the EM are on the right, and the EM RPM gauge is colored in gray. The purpose of this was to allow the test operator to easily glance up and read important component related data quickly. RPM is critical in determining torque,

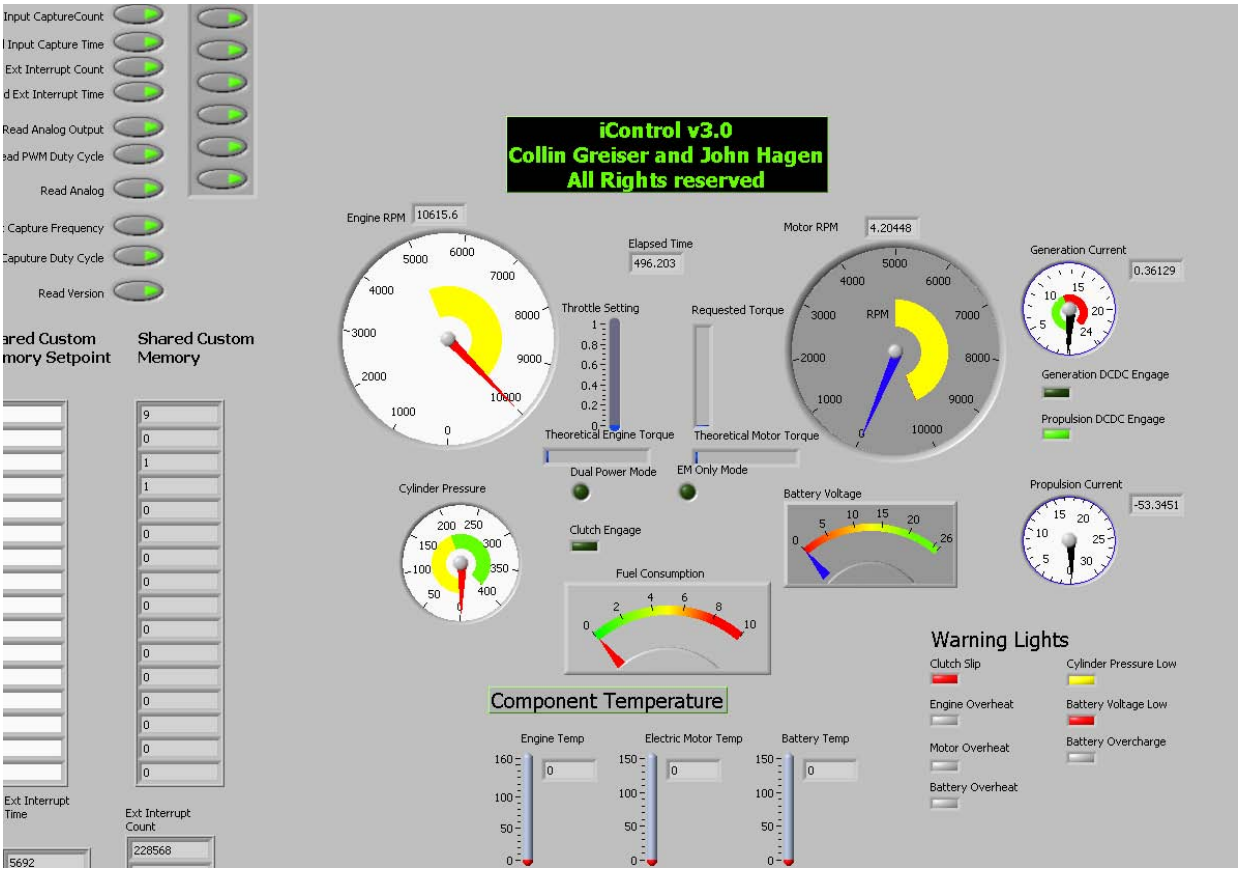


Figure 28: LabView data collection screen

power, and many other parameters so the gauges are the focal point of the setup. Other gauges monitor the engine cylinder pressure, DC/DC converter power, and battery voltage. Several minor lights are then included for convenience, such as controller state indicator lights and temperature readings. On the bottom right corner of the program a set of indicator lights are shown, much like the instrument panel on a car. As in a car, the lights are color coded either red or yellow. Red indicates a serious malfunction and imminent damage, such as the clutch slipping or one of the components overheating. Other lights are yellow, such as the engine cylinder pressure being low; to indicate that those particular parameters are in a warning stage and the operator should monitor the situation. In the center of the screen there was a timer

indicates the test run time, and a throttle setting gauge below to indicate the raw signal coming from the R/C transmitter. To the extreme left are the buttons that control which type of data is collected (usually all "on") and the shared custom memory blocks. These blocks display shared data between the microcontroller and LabView in real time, and were extremely useful in debugging code errors or fine tuning throttle commands.

All dynamometer sensors were hooked up to the controller, which then sends the data to LabView via a RS232 serial connection and was read many times a second. Data is automatically read into a text file, which was then easily read into MATLAB or Microsoft Excel and converted to graphs. The LabView program interfaced closely with "Lightning Stream.vi" which was a sub-vi written by Hagen. The sub-vi handles the actual parsing of the data into packets that LabView can read and exchange with the microcontroller. The vi has a variable time step that can be adjusted to take data ay varying intervals; for testing and controller optimality it was set to 0.2 seconds. To the left of the figure is the sub-vi. Split off to its right are all the individual arrays that represent each type of data being passed. The arrays pass data to the front panel, which displays it on a gauge, and also routes the information to do a number of other useful tasks. For example, as shown in the center of the figure, the external interrupt time is shown. This time is the reading from the RPM sensor; it counts the time between each pulse of the reflective tape passing in front of it. The time is then divided by 60000000 (converting microseconds into seconds and then into minutes) and then inverted to give a final RPM value. The data is then filtered to remove spikes and unsteady readings through another sub-vi. Once this data is split, the analog input data, from which most of the front panel information is derived, is moved to another section for processing. The figure shows the data collection part of the block diagram. will be discussed in Chapter IV. Each piece is broken down into a sub-array and

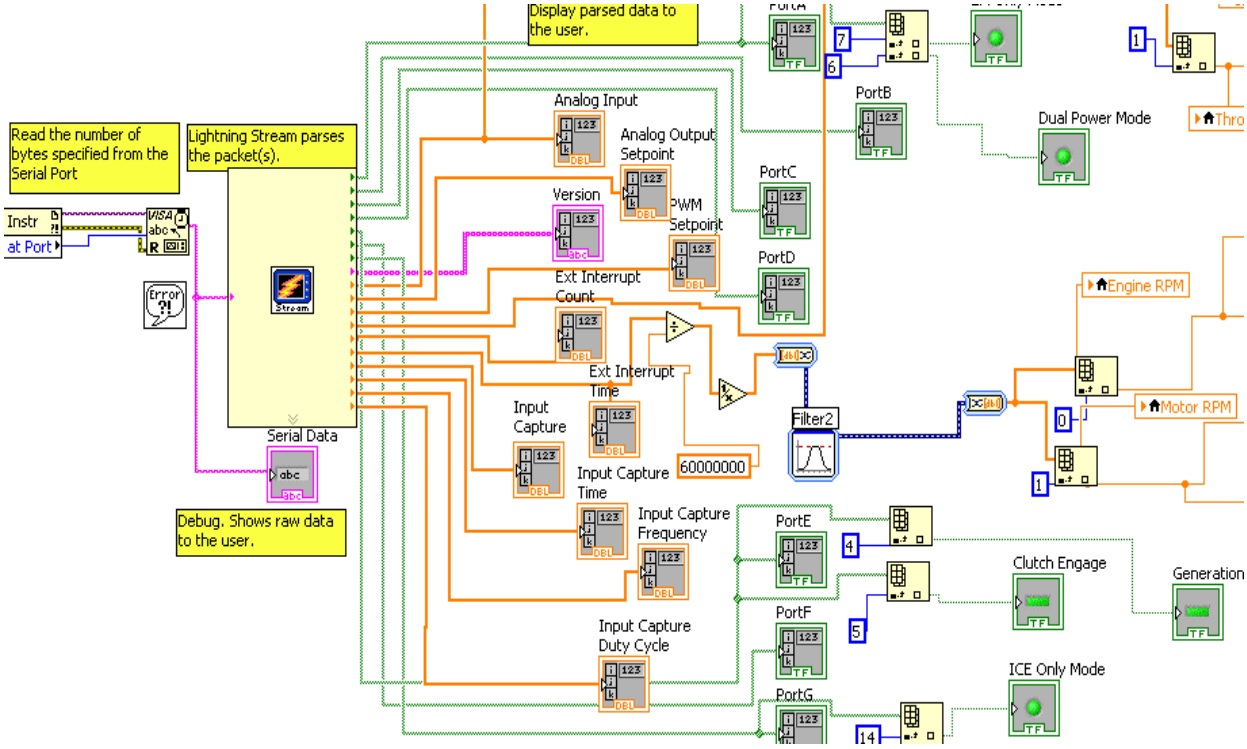


Figure 29: LabView block diagram showing streaming function to exchange data with the microcontroller.

then fed into both a local variable and a gauge for the front panel. The dynamometer's main function is to measure torque. To this end, the company provides a data collection program to collect the various data that the dynamometer measures. Shown is the main dynamometer data collection in Figure 31. The data collection utility provides many features that were useful such as the ability to change recording formulas, data collection rate, and control of engine inputs such as throttle. It is immediately obvious that most of the data taken is not needed, but this is easily filtered out from the data files. The large gauge in the upper left if the torque readout and the large gauge in the upper right is the RPM measurement. The torque was the main parameter measured, and it is added to the main data file for each run. The RPM is a nice reference to LabView's RPM and greatly aided in calibrating the RPM sensors, but otherwise is only recorded for reference. All other data is not used at this time.

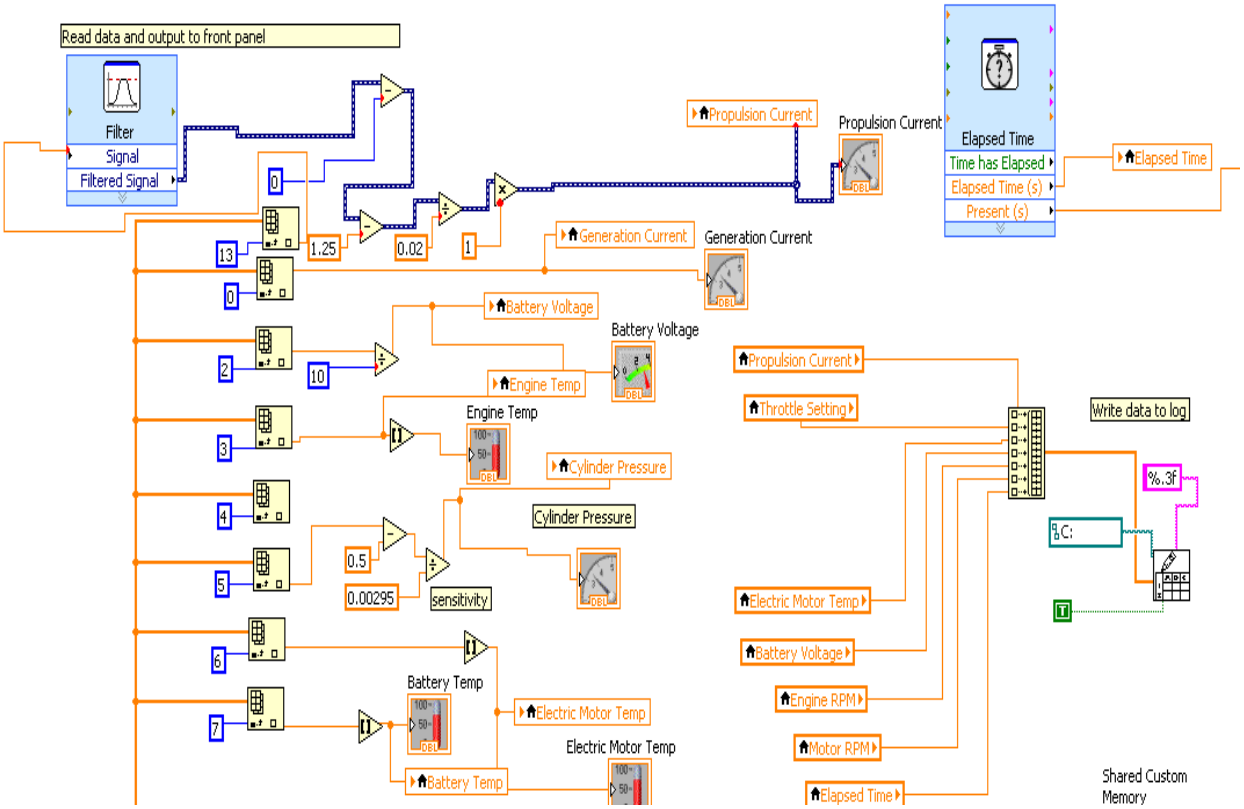


Figure 30: LabView block diagram showing analog data collection and some filtering.

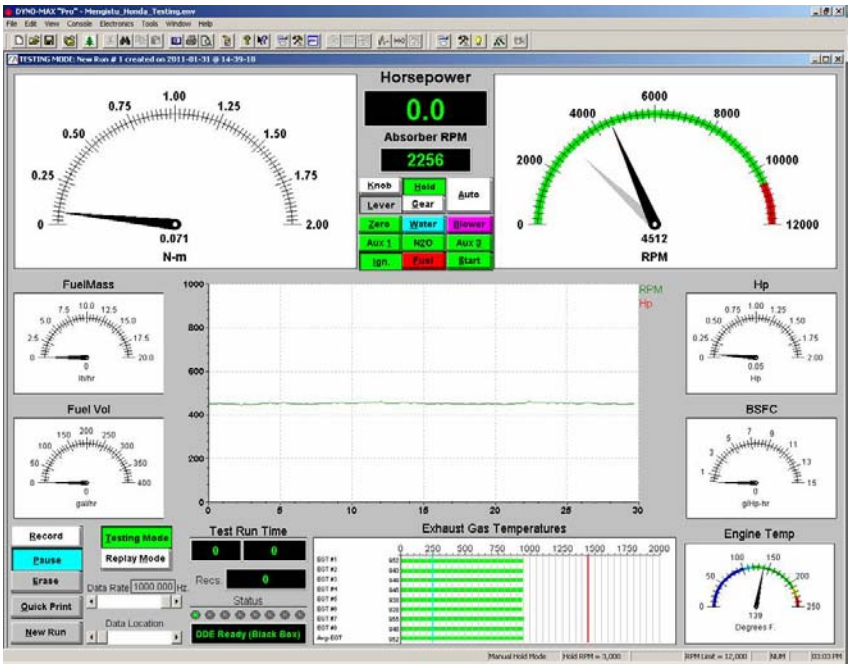


Figure 31: Dynamometer measurement screen

3.4.2 The Dynamometer

The small engine dynamometer built by Land and Sea Corporation is one of the few devices commercially available for small engine testing [45]. This necessitated its choice as the torque measuring device. Torque measuring was accomplished with Land and Sea's proprietary software and read into MATLAB for data analysis. The dynamometer is a cradle type, with a small strain gauge to measure a potential difference and convert this into torque. The load is applied with a 96 V eddy-current magnetic-brake (the large wheel on the right of the figure) on the end of a gear driven shaft. The shaft contains a 2:1 gear ratio, which is accounted for in the dynamometer software. A significant amount of time and energy was spent building a test rig that would house the dynamometer, the controller, and all related test equipment. The rig itself is built from 80-20 aluminum [46] with polycarbonate shielding [47] to protect the test operator. A 12 DC fan was used to exhaust the engine fumes from the test area. The controller was located away from the engine to mitigate noisy signal interference from its high voltage spark system.

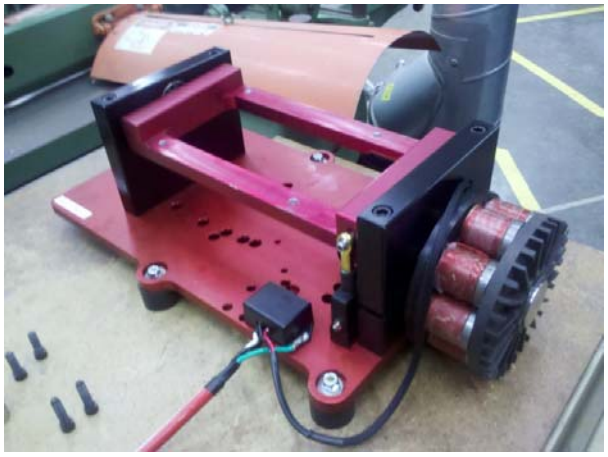


Figure 32: Dynamometer without any mounted equipment; note mounted strain gauge for torque measurement

3.4.3 Sensors

The LabView screen and code take their data directly from a number of sensors that were selected and then mounted on the dynamometer. Each sensor was chosen with an eye for portability and ease of use, which would make the transition into an actual aircraft simpler and easier. A total of 6 sensors were chosen: speed for torque sources, temperature for the EM, ICE, and battery pack, power measurement for the battery, and cylinder pressure for the ICE. Another benefit of the sensors chosen was that previous thesis students have had experience with them, and the challenge of debugging could be greatly eased. The students could advise if the sensors were malfunctioning or giving erroneous readings. For reasons explained in Chapter IV, only a handful of the sensors were implemented: the RPM sensors and battery voltage measurement.

The RPM sensors are one of the critical pieces of controller measurement and decision making. The sensors used were Monarch ROS-W, which are digital optical sensors [48]. They are lightweight, require very little power, and can be mounted very close to the measurement shaft which is useful in aircraft applications. Figure 33 shows the RPM sensor mounted to read a reflective strip on the ICE output shaft. Battery power and power draw rate to and from the battery are controlled by two SynQor DC/DC converters [49]. These are custom-built to allow current limiting, which is a critical parameter to torque application to and from the EM. The converters are controlled by a single on/off digital signal. Power is fed in differing directions depending on applications, with one converter used for propulsion and one used for generation. The converters have a current monitor built in, and this signal is used to measure current draw on

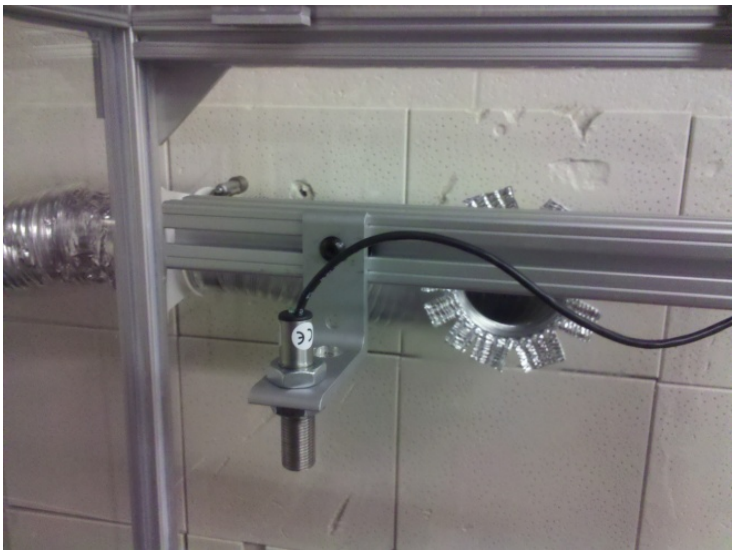


Figure 33: RPM sensor

each. Depending on the test being run, a steady, 26.9 V input power for propulsion is fed by either a Mastech DC power Supply [50] or a Thunder Power 7-cell LiPo battery. The DC power supply was incredibly useful for long-term testing periods where the batteries would normally be quickly depleted, and allowed precise current limitation for motor testing outside of the controller. Depictions of one of the converters are shown in Figure 34, and the power supply is shown in Figure 35.

Figure 34: DC/DC Converter being used for both propulsion and generation

Figure 35: Mastech DC power supply

3.4.4 Internal Combustion Engines

A total of three internal combustion engines were used in various configurations for the hybrid system. All of these engines were four stroke engines, chosen for their advantages in acoustics and fuel economy. Specifically, the four stroke designs produce less noise, emissions, and are more fuel efficient then their two-stroke counterparts. While four stroke engines are typically heavier, this penalty was deemed acceptable given the numerous advantages.

All three engines were modified to be started with a hobbyist starter; a simple high torque, 12 V DC motor depicted in Figure 37. Two engines were built by Fuji, model numbers BF34-EI and BF25-EI [51] (shown in Figure 36). Each of these engines is designed for model aircraft engine use, and are light and fuel efficient. Fuji recommends use of 87 grade automotive gasoline, which was also a big bonus for logistical reasons (87 is easier to come by then glow fuel or AVGAS). They also feature a spark timing mechanism to adjust spark for greater fuel

efficiency. Manufacturer specifications on the 34 model indicate a maximum rated horsepower of 2 HP, while the 25 is rated for 1.5 HP. Due to certain parameters explained later in Chapter IV, the engines are mounted from the bottom of the oil pan and additional mounting points are located under the crankshaft. Spark is provided by a 4.8 V battery which is mounted on the back plate, and each engine is started using a propeller cone mounted on the rear of the crankshaft. The third engine was built by Honda, model number GX-35 [53] (shown in Figure 38). This engine is primarily designed for portable equipment such as leaf blowers and string trimmers, and as such, is designed a little differently. Primary design differences include: a more robust crankcase, a recoil starter, more weight, and an overrunning clutch mounted on the shaft. Most of the extra plastic trim, the recoil starter, and the clutch were removed for testing (this is also typically how the engine would be configured in the aircraft). The Honda was also mounted

Figure 36: Fuji BF25-EI with mounting brackets (left) and BF34-EI (right)

from the bottom due to negative experiences with mounting engines from the backplate. The Honda is rated for a maximum of 1.3 HP and also runs on 87 grade automotive gasoline.

Figure 37: Sullivan DynaTron Hi-Torque starter [52] with 12V power battery

While the controller is designed for any engine, each engine was mounted in a different configuration on the dynamometer in order to facilitate easier swapping between engines. In a realistic case, the aircraft would have only one engine. The maps in the controller would also be programmed for that engine. Since this was not possible, each mode of the controller was programmed for the specific engine that was used. For cruise without regeneration mode,

Figure 38: Honda GX-35 engine with prop nut on shaft for starting

the Honda was the primary engine. For climb and cruise with regeneration, the Fuji 25 was used. The Fuji 34 was disabled due to a broken backplate early in the testing phase, so its results are not included in Chapter IV.

3.4.5 Electric Motor

One electric motor was used; this motor was the Maxon RE-50 Brushed DC motor [42]. There were several reasons behind using the DC motor versus a possibly more efficient AC motor. First, the DC motor used was not that much more inefficient then the AC motors on the market [9]. Secondly, the DC motor is much easier to control using an adaption of Harmon's equations. If the AC motor were used, a second 4-quadrant controller would have to be used to individually control this motor, which would add to the complexity significantly. Other independent sources have also confirmed that the additional controller is difficult to work with compared to a DC device [54].

The Maxon DC motor is rated for 200 W of continuous power, with up to 300 W of burst power for 30 seconds without overheating. 200 W power is rated at 24 V and 8.3 A continuously, while providing maximum torque and a claimed 94% efficiency for most of its operating range. Refer to Figure 39 for an image of the motor un-mounted from the test setup.

3.4.6 The Microcontroller

The microcontroller is the heart of the thesis and the most important piece to the setup. It is mounted externally away from the setup on an acrylic plate (acrylic because of dielectric properties). The controller itself is a PIC32MX development board manufactured by Microchip [55]. This particular controller was chosen because of its easy programming and debugging

abilities, fast processor speed, and its modified C code architecture. All of these qualities simplified the programming process.

Figure 39: Maxon DC motor with attached wire leads.

Running on a modified C compiler built by Microchip, the development tool also allows quick and easy changes to programming while allowing access to a multitude of features within C and C++ [56]. The controller is programmed by and debugged by a PiC-It 3 In-circuit debugger, which attaches to the top end of the controller. It is powered by a 5 V USB cable, which also conveniently grounds the controller commonly with the computer case for accurate measurements. The controller is shown in Figure 40. The controller is soldered onto a printed

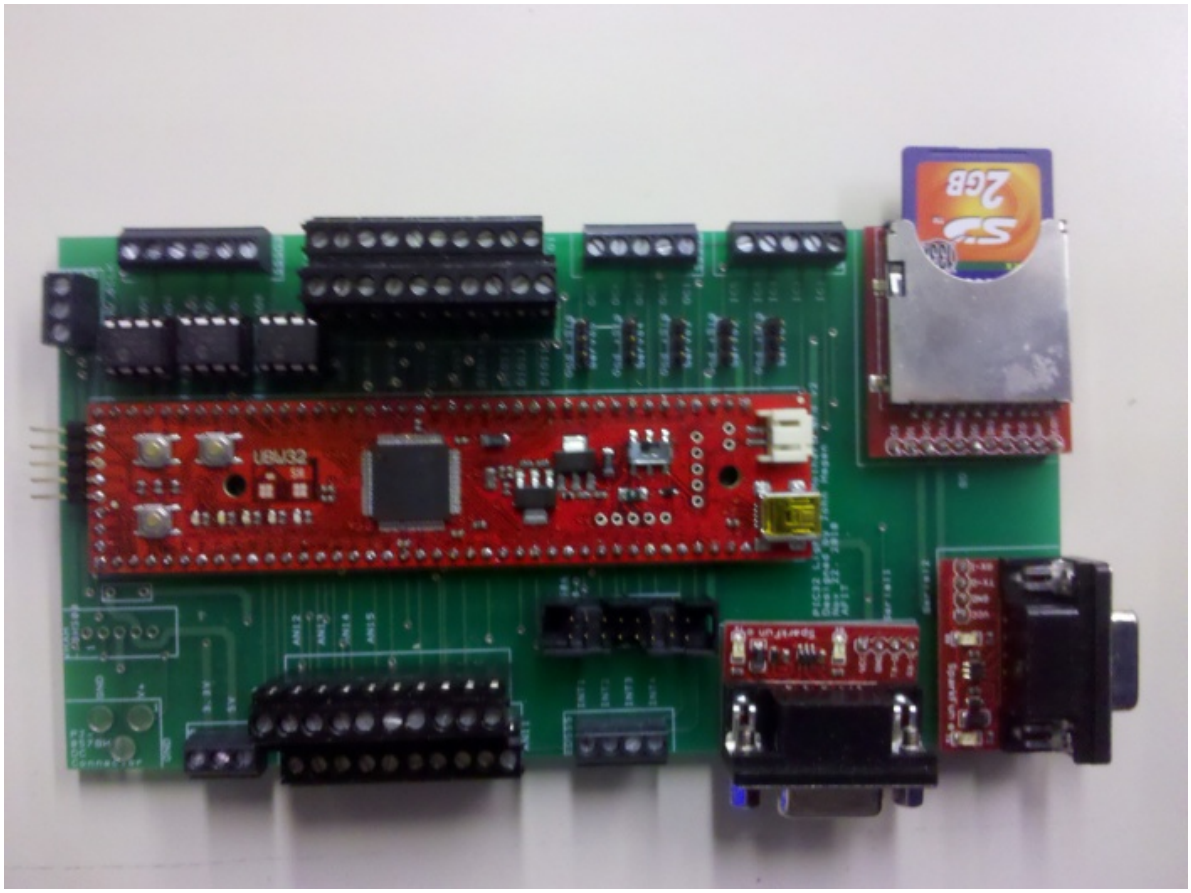


Figure 40: Microcontroller attached to PCB

circuit board (PCB) which conveniently routes important inputs and outputs from the main controller board to screw terminals on the PCB. Screw terminals made wires easier to connect and disconnect, and also ensured a more reliable connection in the circuit. The controller has 6 ports for analog outputs (i.e. controlling DC/DC converter current limits), 16 ports for analog inputs (temperature, battery monitoring, DC/DC converter current monitoring), 18 digital inputs (all control switches), 4 counter ports (RPM sensors), input capture ports (read in throttle signal), and servo outputs. The controller communicates with LabView through a RS232 serial data port. Each of these ports is shown in Figure 41. An example of the controller wiring diagram is shown in Appendix E.

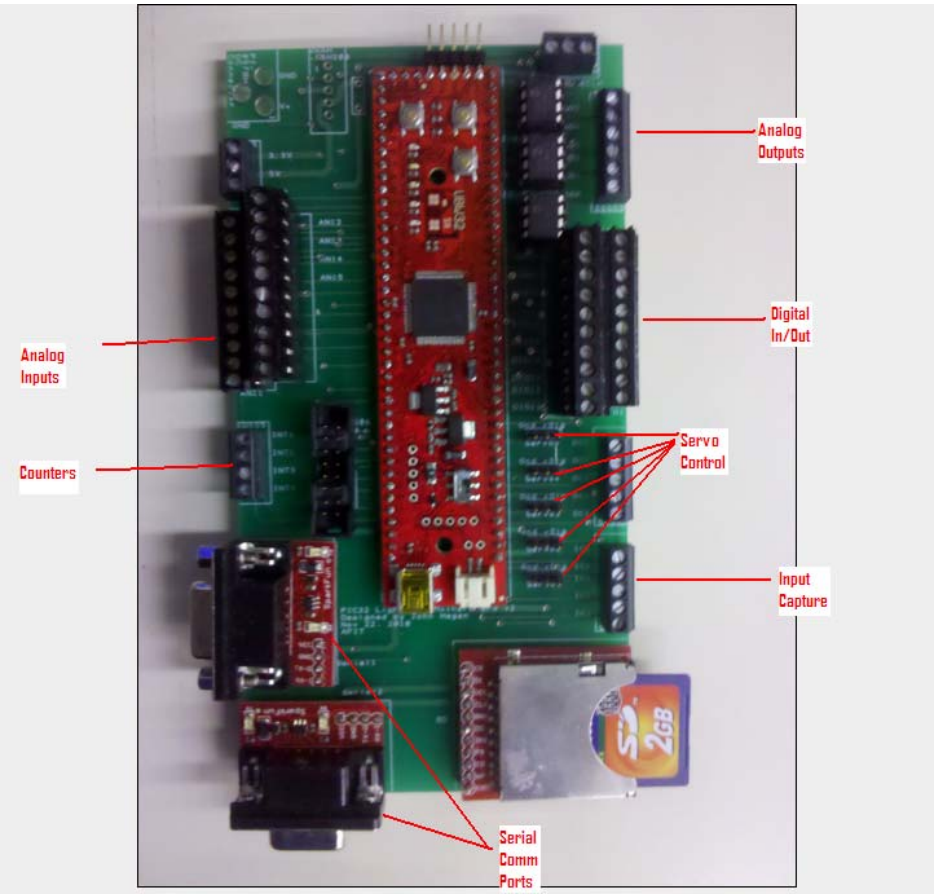


Figure 41: Microcontroller layout

3.4.7 Transmitter and Receiver

The transmitter and receiver were used to simulate the role of a human pilot in the controller's open loop. The transmitter used was a popular model with R/C hobbyists; a Futaba 8FGA transmitter and receiver kit [57]. The transmitter was chosen in particular for its ease of use, the ability to trim the servos for setup ease, and 8 output channels. The 8 channels were useful for controlling the individual controller states manually with the aid of BattleSwitch relays [58], as shown earlier in Chapter III.

Figure 42: R/C transmitter and receiver

3.5 Procedures for Validation of Controller and Setup

Once the controller was programmed and the LabView screens created and debugged, the validation process for the system was started. This process first involved creating test matrices that allowed the author to get a good understanding of what he was testing. A test matrix was created for each operational mode tested, for a total of 4. All of the test matrices can be seen in Appendix C; this gives examples of the types of data that were collected in the test phases.

Validation then contained a standard operating procedure (SOP) for each test. All tests initially began the same way, and then branched off when each went into their own specific operating regime. An example of a SOP is included in Appendix D. At the request of the

advisor, tests were repeated 3 times and then averaged to ensure some measure of data validity. The three trials were selected because of time constraints.

For each operating regime, specific data was collected. The one piece of data collected common to all operating modes was the throttle lever position, as the controller's reaction to this input was critical to operation. For the endurance mode, data collected included input voltage and current from the DC/DC converter and output torque and speed. For cruise without regeneration mode, primary data collected were throttle position, output torque, and output speed. For climb mode, the above two (endurance and cruise without regeneration) were combined. Cruise with regeneration replaced the propulsion DC/DC converter data with the generation DC/DC converter data. Data collected from experiments and analysis of this data is included below in Chapter IV.

3.6 Test Setups

The sheer magnitude of design iterations requires some explanation on which test setup was used for each section of results. To test the cruise mode without any regeneration, Figure 43 shows the Honda engine mounted on the dynamometer by itself. This particular setup is the simplest. The second setup in Figure 44 was used briefly to test endurance, climb and cruise with regeneration was the Fuji 25 engine with the electromagnetic clutch. This setup is mounted on an aluminum plate with the electromagnetic clutch. The motor is connected to the driveshaft after the clutch via a belt, which allows the clutch to disengage the engine from the power shaft. The plate sits on top of the dynamometer cradle which is rotated by application of the load to the magnetic brake. By mounting the HE power shaft directly above the dynamometer power shaft, no torquing moments were produced which could affect the final measurements.

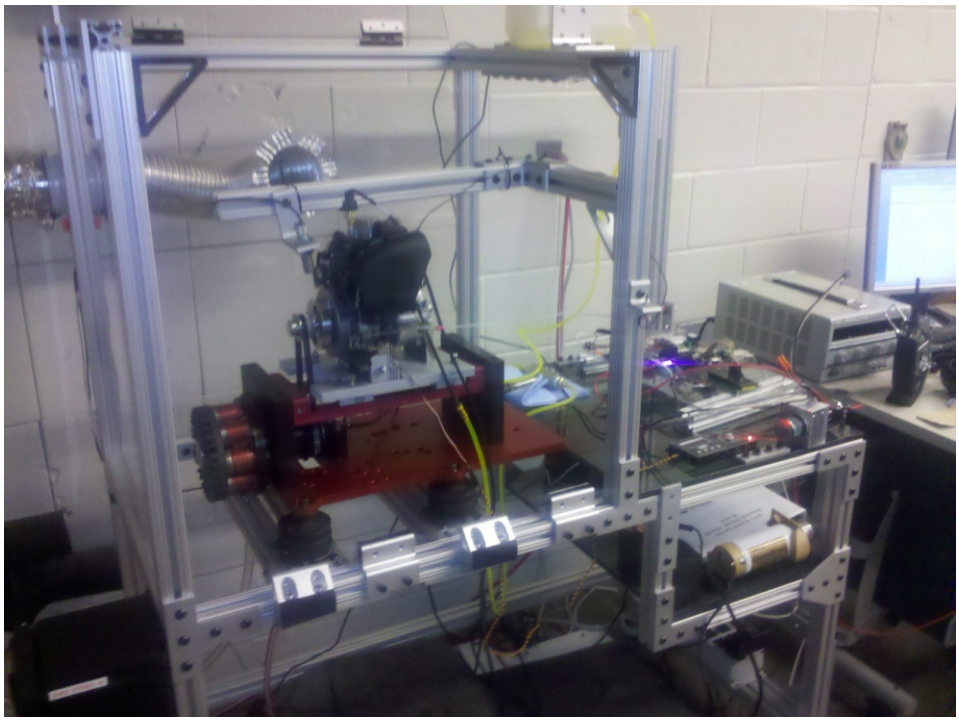


Figure 43: Honda engine mounted on the dynamometer

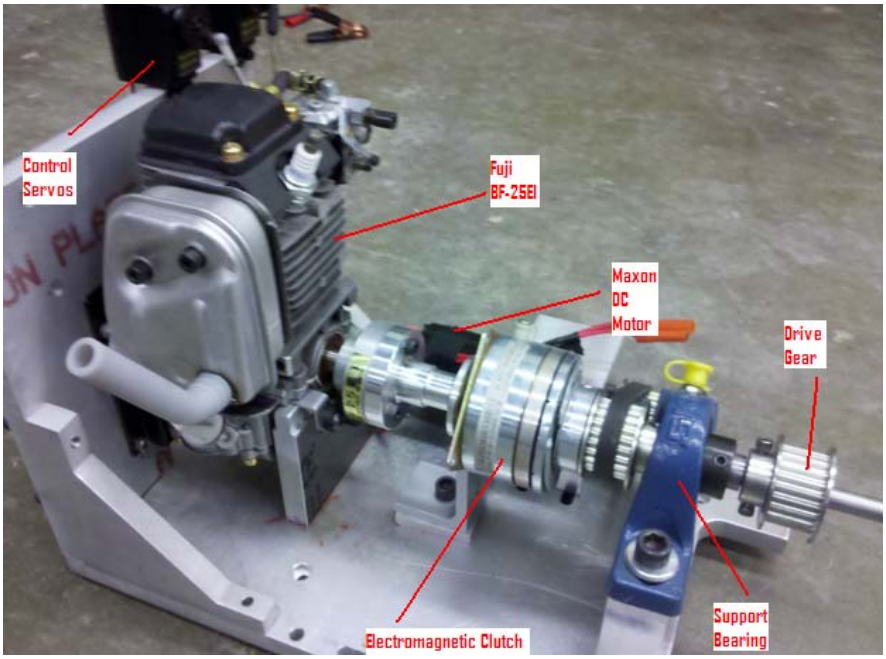


Figure 44: HE Configuration w/ Fuji 25 engine, Maxon Motor, and clutch

The third setup that was used for examination of endurance, climb and cruise with regeneration was the Fuji 25 engine with a one-way bearing instead of a clutch. Details of why this setup was used are discussed in Chapter IV, and a figure of this apparatus is shown in Figure 45.

Once the test setups were designed and built and the code was written, validation of the controller began in earnest. Testing took place over several months and used the SOP's and test matrices to aide in data collection and test repeatability. The test results yielded very positive results about the controller performance over the operating ranges, which will be discussed in further detail in the next chapter.

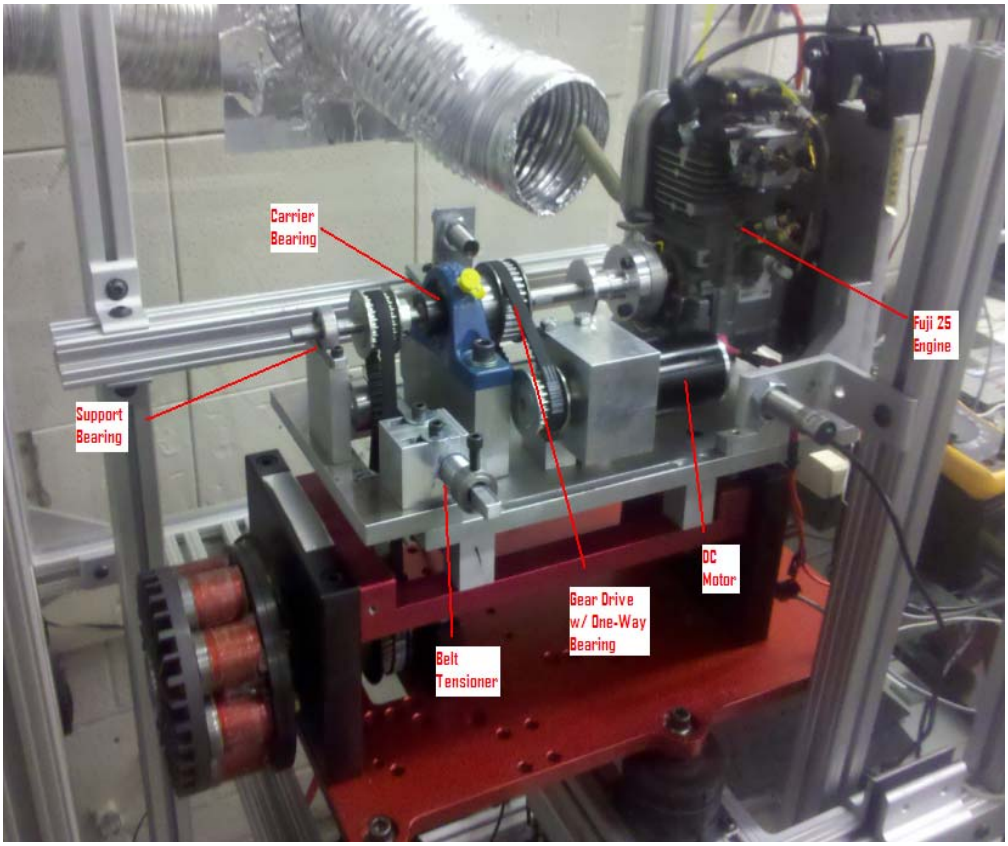


Figure 45: Test setup with one-way bearing

IV: Results and Analysis

4.1 Introduction

Chapter IV discusses the results of the validation for the controller and test setup. Many unique challenges were discovered while testing the system and collecting data. The chapter details each of the four flight operating modes during data collection and includes a section on restarting the engine with the electric motor. Each section then has its own analysis piece which thoroughly discusses the results of each test of each mode.

4.2 Cruise Without Regeneration Testing

4.2.1 Test Goals

The overarching goal of cruise without regeneration testing is to benchmark the controller's ability to command the ICE without any electric motor commands. As previously shown in Figure 43, this mode primarily used the Honda engine mounted alone on the dynamometer. This mode is essential in confirming the validity of the performance maps provided by Mengistu [31]. Another main objective was to observe the throttle commands in open loop in order to ensure that there were no undesirable effects in engine performance that could be caused by any number of outside sources. Specific data collected during these tests were engine throttle, RPM, and output torque. Power consumption by the DC/DC converter was considered insignificant for the scope of this test and was not recorded. Measurement of the engine throttle was completed using a special function to pass data from the microcontroller to LabView. The measurement is only the commanded engine throttle, not the true engine throttle

position. As discussed in Chapter III, the engine throttle command was calculated and then passed to the engine as a value between 0-1. This value was read into LabView using the aforementioned function within the controller that allows memory sharing between LabView and the C code.

The other two measurements were taken with sensors. The RPM measurement was primarily read into LabView with the Monarch RPM sensor. The torque was measured through use of the torque transducer on the dynamometer. The two primary measurements from each source were set to be taken at the same point (every 0.2 second) so there were no differences in measurement points, and they are then combined and plotted with MATLAB. For external reference, the RPM sensor on the dynamometer and the Monarch RPM sensor were compared, but variability was very low, and in fact the LabView data was smoother because of its data filtering utility. The LabView data was roughly 0.2 seconds slower than the dynamometer data because of this filter, and due to this MATLAB needed to be used to adjust for this inconsistency. The time lag on the dynamometer software necessitated that both programs were set to 'record' and then after 5 seconds, the test was started. To remove the differences, MATLAB was used to log the minimum points on the graphs. Using this point, time was readjusted so that both time vectors line up and was taken at the same point.

To start each test, the engine was allowed to attain its nominal operating temperature by idling. Once an optimal temperature of 150 degrees F (measured on the cylinder head) was attained, the test began. To accurately simulate the cruise conditions, the engine was first tested at the design point for cruise conditions. These conditions are at 5000 RPM as shown by Hiserote and Rotramel [9] [30]. At this RPM, the engine was examined to see if it could provide

the nominal torque for flight. After this torque was tested, other significant RPM points were investigated. The purpose of these tests was to make sure the controller could correctly set the throttle position to provide the torque that is specified as needed. An engine stall speed is determined from Mengistu's data [31]. Expectations from all these tests are that the controller could correctly set the throttle and provide the necessary torque without stalling the engine. See section 3.2.6 , for details about coding of this test.

4.2.2 Data Analysis

The Honda engine performed admirably under these loading tests. As noted by Mengistu, the Honda engine actually performs up to and sometimes exceeding its manufacturer ratings. Mengistu observed a peak horsepower of 1.4, which is higher than the engine rated power of 1.3 HP. Results from the cruise mode test are shown below for the Honda engine.

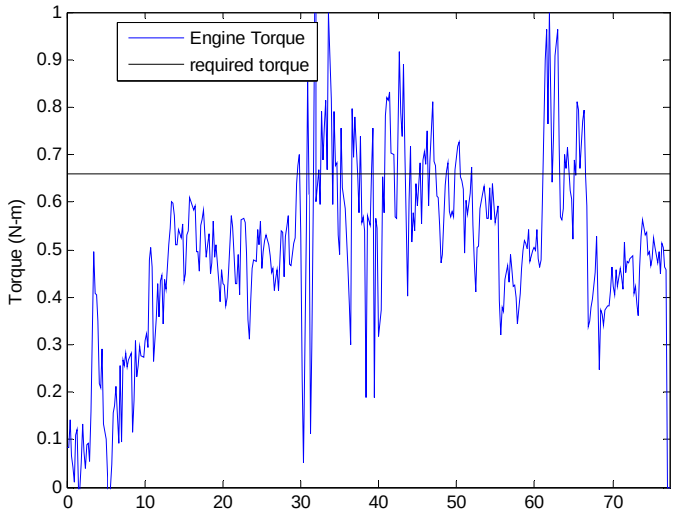


Figure 46: Honda GX-35 torque versus time

Figure 46 shows the Honda's torque versus time for the test. The graph shows one major flaw in the testing: torque was hard to get a precise reading on at any one point because of

several things. The engine itself rocked the cradle, and at these small torque measurements, this caused disturbances in the data. Cradle rocking can be attributed to torque spikes which these small single-cylinder engines provide. However bad the cradle rocking, general trends can still be looked at, of which the most important are the consistency of the engine to provide the required torque for flight at cruise. Rotramel, in his research, has shown that the required torque would be 0.66 N-m at 5000 RPM in order for the A/C to cruise at 40 knots [43]. When the Honda data is plotted versus time and the results are averaged over several tests, the trends are much better. In Figure 47, it can be easily seen that the required torque was provided.

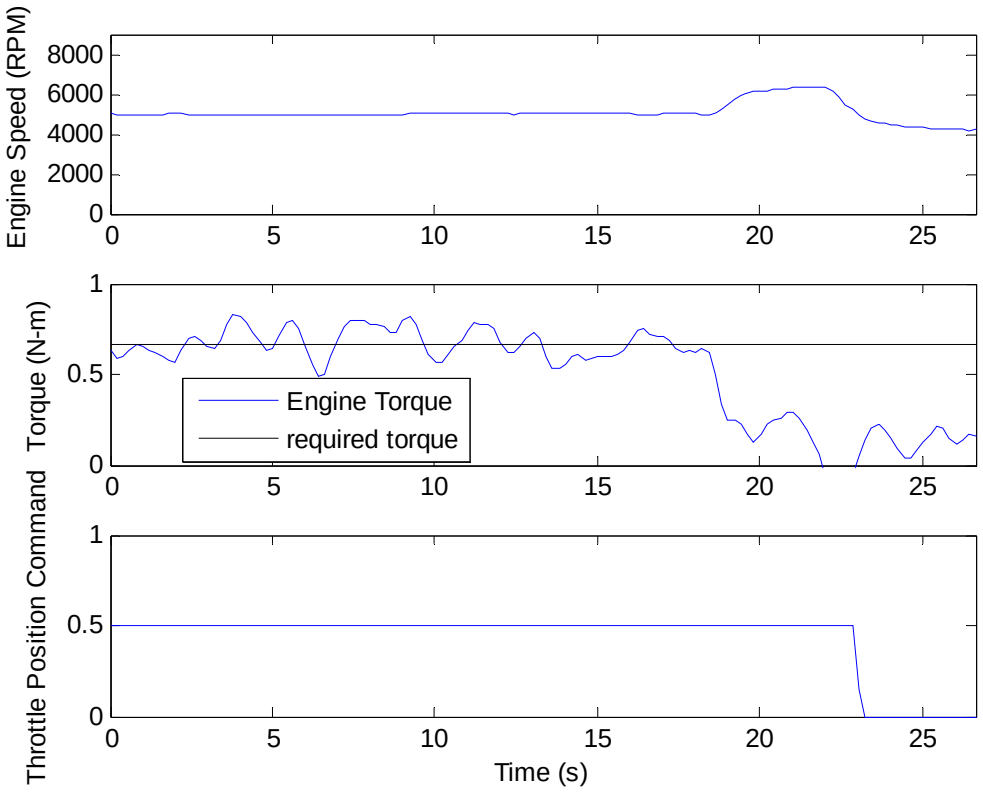


Figure 47: Honda GX-35 torque, engine speed, and throttle command versus time

As shown earlier, torque was not a steady measurement, even with data filtering. However, a general trend above the 0.66 point (black line) is seen until the load is disconnected

at 18 seconds into the test. The controller holds the throttle position without noise interference until the open-loop operator lowers the throttle to idle. Heavy saturation and filter commands needed to be applied to keep the signal to the servo steady and control the engine as best as possible. This was due to noisy servo signals that cause unstable engine control. Table 6 shows the average engine data. When under load, the engine provides the correct amount of average

Table 6: Average engine parameters for cruise mode testing

Engine Parameter	**Average Reading in SI (English)**
Output Torque	0.67 N-m (0.49 ft-lb)
Engine Speed	4997.5 RPM
Room Temperature	20.78°C (69.4°F)
Barometric Pressure	29.63 in Hg (14.56 psi)

torque as expected. As noted earlier, when testing the load was controlled manually. The reason for this was because the automatic load control feature controls the load by engine speed, not an actual set point. This type of load control has a tendency to stall the engine, making testing with it undesirable. Therefore, when testing the load was set to adjust the engine speed to the required point, in this case the 5000 RPM required for flight by Rotramel. At this speed, it can be noted with the trend data above that the Honda provides the required torque for flight. The controller, when given the command, attempts to set the throttle at half (0.5), which was right about where the author speculated the Honda would be able to hold 5000 RPM under load. Extraneous points on the outside of the graph designate the end of the test; the controller revs the engine up after the load was removed until the command is given to return the engine to idle. The engine data was produced with the weather data also shown on the table. The dynamometer software takes

this weather data and accounts for the pressure and temperature changes in the torque calculations. Weather data was measured by an Ambient Weather portable weather station [59].

Several interesting things were noted under testing. First, the RPM sensors are not as robust as the author would have thought. The sensors themselves are designed well on paper, but in practice several things attribute difficulty in using them. First and foremost, the signal coming from them was not consistent over time. Filtering needed to be applied through LabView to stabilize the signal. If the signal was not stabilized, the RPM reading could jump around, causing the throttle signal from the control to jump as well. This caused unstable engine operation that had to be stabilized by the user. Secondly, and most importantly, the sensors are prone to vibration issues. If the sensors experienced any vibration at all, the signal would become incredibly unsteady. This unsteady signal would cause the aforementioned problems with the reading and throttle settings. A solution was implemented that stabilized the sensors for operation, but a more thorough examination needs to done to acquire sensors that are vibration stabilized for the actual aircraft, as the A/C operating environment will be hardly vibration free. Figure 48 shows the solution to stabilize the RPM sensors. It was originally thought that the sensors would need to be mounted as close to the shaft as possible. This eventually was proven incorrect, as the sensors have a true operating range of over 6 inches. By mounting the RPM sensors on their own portable support towers, the sensors were both kept away from the harmful vibrations from the dynamometer and enabled them to be accurately aimed for better precision. The towers also allowed the sensors to be mounted on a slide made out of 80-20, which facilitated easier switching between the test setups.

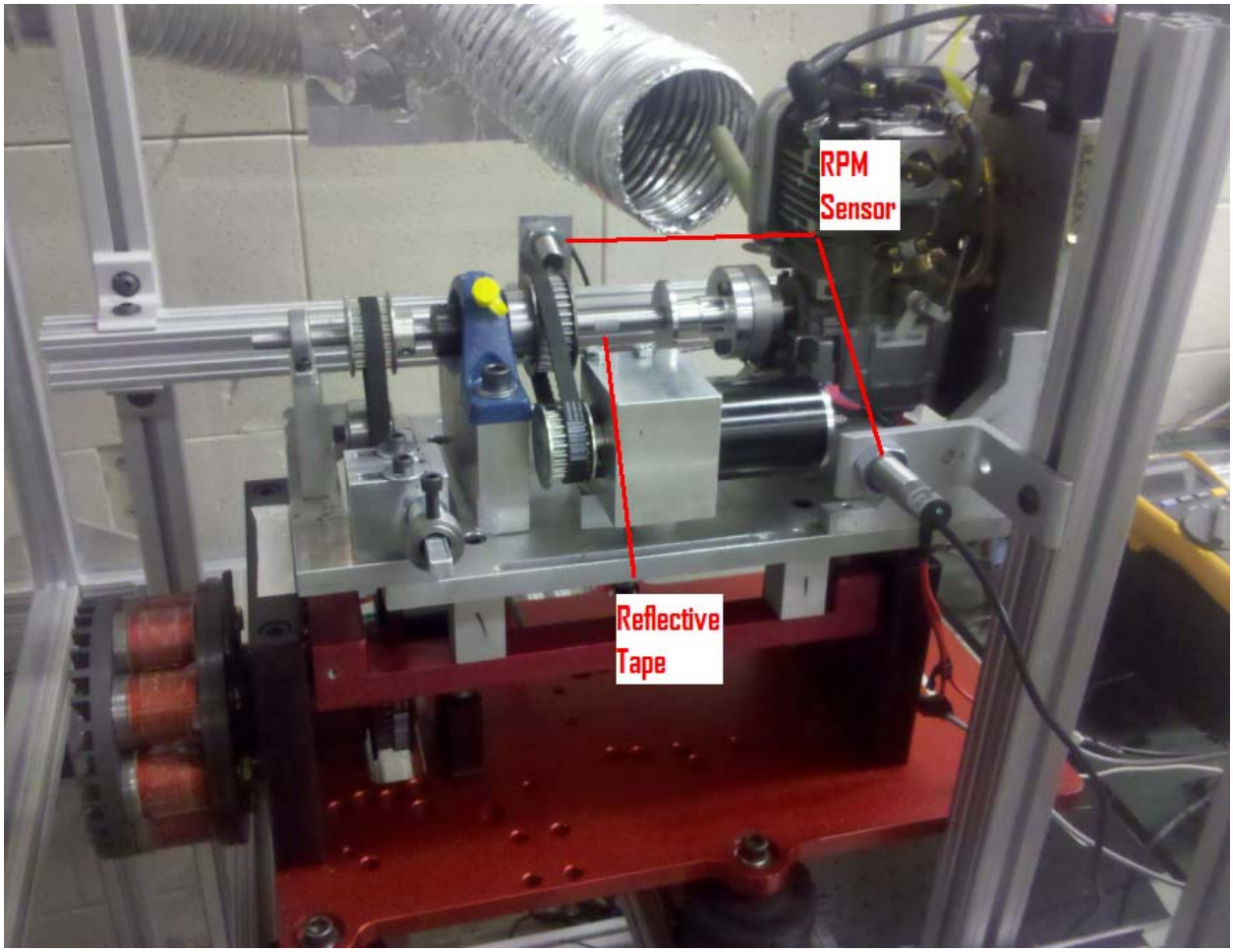


Figure 48: Stabilized RPM sensors. Sensors are stabilized with mounting towers (shown).

Another thing worthy of noting, which is true of practically any electronic setup, is the use of common grounds. The grounding point for all of the electronics must be common, or any signal received will not be with respect to each other, invalidating them. This was first observed early in testing. The controller was powered by 5 V from a USB style port. Initially, the controller was connected to a wall outlet adapter, with the thought being that the wall outlet will allow more current than a computer USB jack. However, this caused problems with servo operation. When connected to the wall outlet, the RS232 serial port and controller were on a different ground. The servos immediately became highly unstable as soon as a throttle command was

sent; rotating the full range of motion un-commanded. Once the controller main board was powered from the USB jack on the computer, the common ground was restored and the servo operation became very stable. As long as the ground in the aircraft is common to the airframe, this will not be an issue.

To conclude cruise mode testing, it can be seen that the Honda provided the torque needed for flight in cruise operation. The Fuji however, was much harder to operate and prone to stalling, so the author cannot recommend its use with the controller. Testing provided unstable torque spikes that could not provide useful data. Even with correction methods in place to avoid a stall, the engine was difficult to work with and would stall anyway. Additionally, choke manipulation was meticulous with starting the Fuji, as it needed to be operated whether the engine was warm or not. The Honda, however, is much more robust and is easier to control. Its choke only needed to be used rarely, even when cold. Often, the choke did not need to be adjusted. If the Honda was downsized just a little further, perhaps to the GX-25, additional fuel could be saved and the required torque for flight still provided. The controller can easily control either engine, and cruise mode parameters were achieved in the cruise mode testing with the GX-35.

4.2 Engine Restart

4.2.1 Test Goals

The engine restart test checks the engine start state on the controller. Its main purpose was to see if engine restarting in flight was feasible, and therefore whether or not the clutch start design was a feasible option for the HE-RPA. R/C operators of this class of aircraft have said

that restarting in flight was not feasible with small ICE engines, but the author wanted to test this method regardless. Positive test results would open up additional options in the cruise and endurance modes, such as shutting the engine down completely versus just letting it idle.

In this test there was only one type of analysis performed, which was the restart mode. The test utilized the two states of the controller, which were EM Rev, (section 3.2.2 EM Rev) and ICE Start (3.2.3 ICE Start). Data measured only was engine and motor RPM versus time. A consistent ICE RPM at idle was considered a successful test.

4.2.2 Test Analysis

The testing of this mode was unsuccessful for several reasons. The electromagnetic clutch was deemed the failure mode of the test. In trials, it could not handle the loads applied by the Fuji engine satisfactorily. The clutch spindle would over tighten inside the clutch housing, causing a resistance to be applied to the engine. In mild cases this would cause an undue amount of heat to be dissipated on the clutch shaft, and in extreme cases it would stall the Fuji completely. The heat being dissipated is one possible explanation of why the clutch was destroyed by warping the connecting electromagnets. This warp caused the shaft to wobble, which would cause the input spindle from the engine to over tighten and apply an even greater amount of resistance to the engine shaft. Not only was this dangerous to engine operation, but it caused increased wear on the support brackets due to uneven forces on the crankshaft bearing. The author theorized that the lack of a thrust bearing caused all of the above to become major problems. Another possible explanation is the voltage; the author theorized that if the voltage was higher, possibly 48 V, the clutch would have more grip and the problem of increased

resistance would be mitigated. The solution to this was to go apply a backup design, utilizing a one-way bearing.

The one-way bearing design, while not Hiserote's original recommendation for an HE-RPA, was actually a better solution in practice. The revised engine design used for climb and cruise with regeneration testing is shown in Figure 49. The one-way bearing is lighter and makes the system less complex. The drawback of this design is the loss of engine restart ability. This, however, was deemed acceptable for several reasons. First, the elimination of the clutch decreased the amount of things the controller has to control, which has the side-effect of making

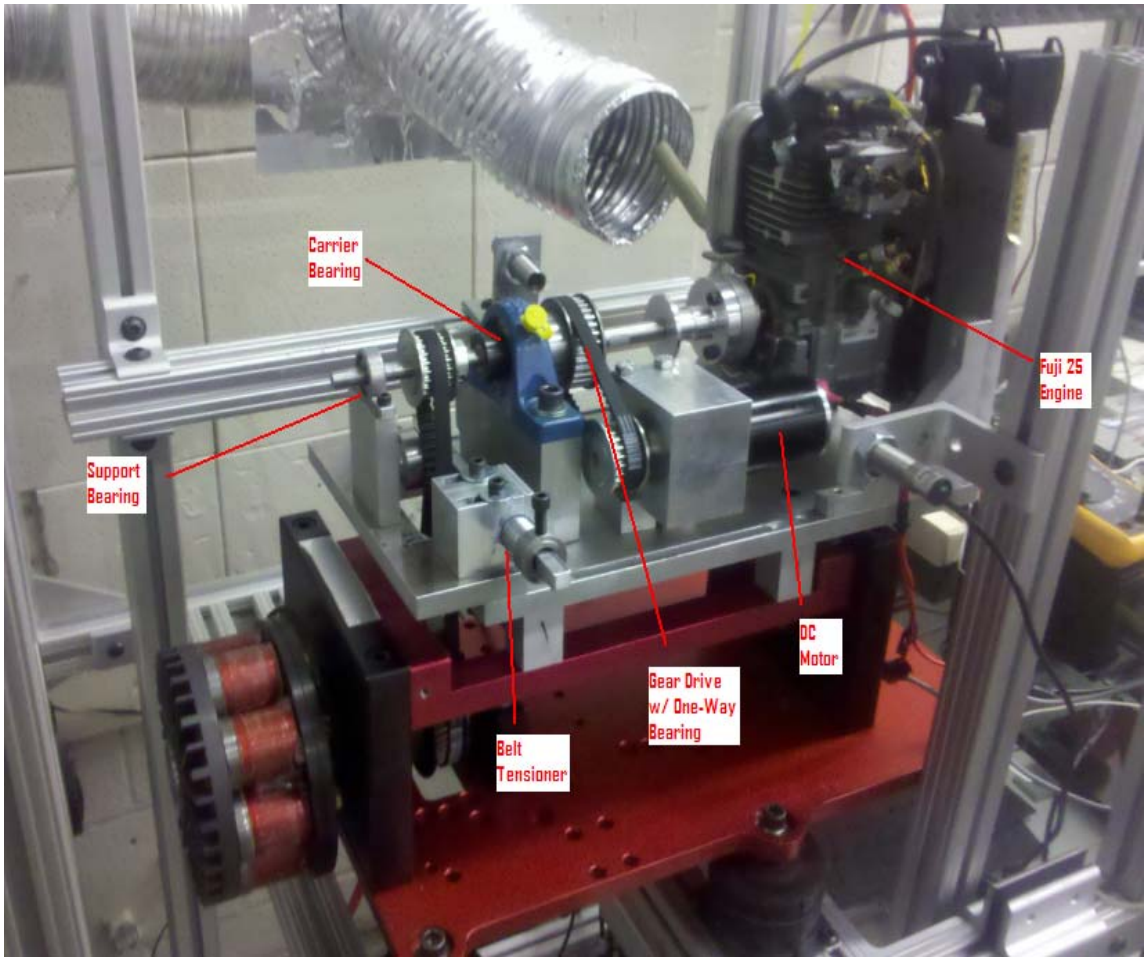


Figure 49: Final dynamometer test setup with one-way bearing.

the system safer through the lack of failure modes. Second, and more importantly, operators of this class of aircraft have stated that engine restart in flight is very difficult. Testing of this would be difficult on the ground, and in-flight testing is even more difficult because of the extreme cases of failure modes that could occur (such as engine stall or clutch lock-up). A solution to restore the engine restart functionality is through use of a very small starter motor. This motor would be secondary to the main electric motor and would provide minimal propulsion power. Through use of gearing, the starter could be very small and the use of gearing would provide the proper torque and speed to start the ICE. The one-way bearing is simpler, just as efficient, and easier to operate in practice. Using the hobbyist starter would allow the one-way bearing to have all the functionality of the clutch-start design without any of the reliability problems.

4.3 Endurance Testing

4.3.1 Test Goals

Endurance testing focused on the EM's ability to provide the rated power for flight over a specific time period. This mode has no torque provided from the ICE, so the engine was off for this evaluation. As shown in Figure 49, this setup utilized the one-way bearing. Primary measurements in this mode were the EM input current and voltage, EM RPM, EM torque output, and the input command to the EM. Since the one-way bearing setup was used, a gear ratio of 1.454545 was introduced into the system, which had several advantages discussed below. As before, the dynamometer and LabView had to be synced so the data points were taken at the same point in order to guarantee validity.

The first test conducted was a control test similar to the ICE test. As the method of controlling the EM is unique when compared to traditional methods, a general torque sweep was performed to check that the EM would provide the torque specified when the current limit was changed. The second test, and possibly more important, was an endurance speed test. With this test the EM was run at the assumed endurance torque required, 0.270 N-m. During this test the load would be steady and the EM and controller's interaction would be monitored. With this torque data, various analyses could be performed on expected range and flight characteristics. See section 3.2.7 for details about the coding of this test.

4.3.2 Test Analysis

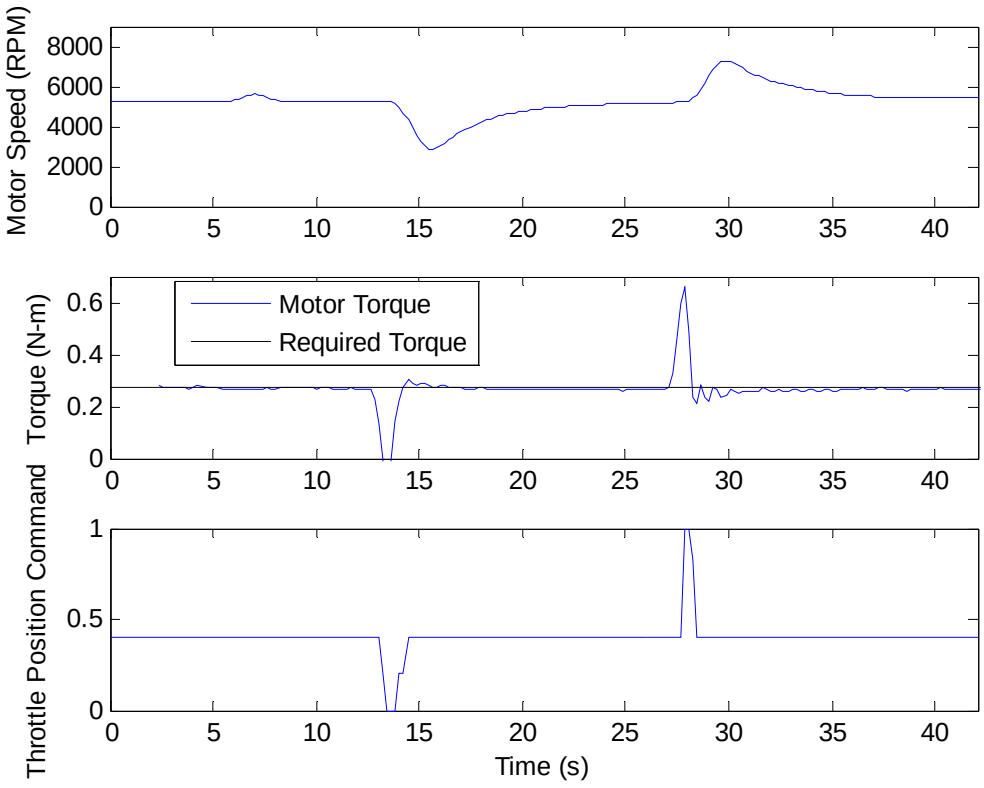


Figure 50: Motor torque, speed, and command versus time

Figure 50 shows the entire test performed, after the data has been averaged. The test was performed exactly the same each time, three times. The load was calibrated before the test, and then the test was performed over 45 seconds. The spike in the data at 6 seconds is caused by noise from the DC/DC converter. Values shown in Table 7 are a result of averaging between the

Table 7: Motor parameters at steady state endurance flight

Motor Parameter	Average Reading
Motor Gear Ratio	32/22 (1.454545)
Input Voltage	23.78 V
Input Current	7.29 A
Input Power	173.36 W (0.232 HP)
Output Torque	0.27 N-m (0.20 ft-lb)
Output Speed	5430 RPM
Output Power	153.53 W (0.205 HP)
Prop Output Power	153.53 W (0.205 HP)
Supplied Voltage	27.0 V
Supplied Current	6.6 A
Supplied Power	178.20 W (0.239 HP)
DC/DC Efficiency	97.28%
Motor Efficiency	88.56%

steady state portions of the test (not including the descent or climb adjustments). As shown on the table, the electric motor was able to properly provide torque for flight at the correct RPM.

Table values show and average value over the entire test. As before, the true prop speed in flight would be at 3800 RPM, so this was the primary test point. Without the gear ratio, the EM shaft speed is an average of 5430 RPM. At this point, the voltage was 23.78 Volts and the current was 7.29 Amps. This corresponded to an input power of 173.36 Watts, and the output torque was 0.27 N-m. Combining the torque and speed to get power, as shown in Equation 4, the resulting motor efficiency is 88.56%. The DC/DC converter efficiency, based on the ratio of input to output power, is 97.28%.

Rotramel's code calculates the torque and speed required including the 1.454545 gear ratio for an optimally matched propeller. The code takes into account all phases of endurance flight to make its calculation so the gear ratio is optimal for the same propeller. The dynamometer measures the torque with a strain gauge mounted on the cradle, so therefore the gear ratio does not affect this torque measurement. The dynamometer load is applied through the aforementioned 2:1 gear ratio, but this is accounted for in the software. Therefore, the motor torque represented in the figure is the actual motor torque being provided. The torque required line is also true motor torque required. When calculated back through the two gear ratios, the actual propeller power being provided by the motor is the same as the motor power being provided as shown in Table 7. This reveals the true purpose of the gear ratios, which is to keep the motor current low in order to increase the efficiency of the motor. Using the motor gear ratio also allowed effective measuring of current, which is explained later in this section.

Spikes on the figure are when the operator simulated a descent or climb condition, and tested the ability of the motor to return to is operating point correctly. These spikes were a necessary simulation because in less-than-ideal flight, the A/C may have to climb or descend in a

deviation of mission parameters. The controller correctly adjusts the current and lowers it or raises as the controller should once the climb or descent is finished.

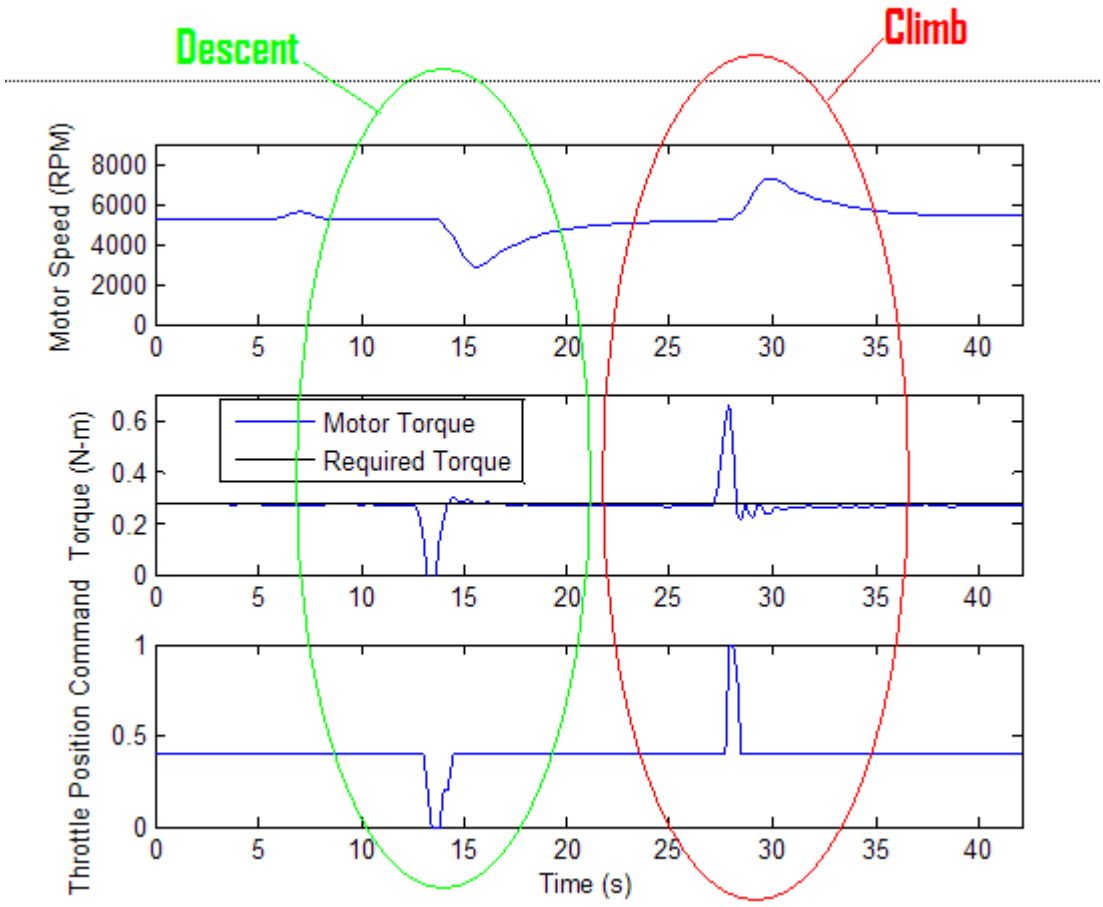


Figure 51: Descent and climb conditions under test

There were many things that contributed to difficulty in taking data for this test mode. The primary and most aggravating characteristic of the system that made things difficult was the DC/DC converter itself. While the voltage and current signals to the motor were not particularly noisy, the current monitoring pin (labeled as IMON in Appendix E) was very noisy. At the stock configuration, the pin reading would have a ±0.2 V difference on the signal. The reference signal is designed to be 1.25 when providing no current and will scale appropriately, never

greater than 3.3 V (this would damage the controller). The equation for calculating the current input to the motor, as provided by SynQor, is shown in Equation 14:

$$I_{motor} = \frac{V_{input} - 1.25}{0.02} \tag{14}$$

where I_{motor} is the input current to the EM and V_{input} is the reference signal provided by the DC/DC converter. It is clear by looking at this equation that small differences in the reference voltage will cause large differences in the current reading. Many correction measures were attempted to resolve this issue. First, a filter was applied in LabView. This filter smoothed the data out over several time periods, but this was of no help for several reasons. First, the input signal tended to vary on the positive side, meaning that the amperage reported tended to be high. Second, the input signal did not vary linearly. This difference meant that if the author tried to adjust with a constant gain, it would be correct at some points and not correct at other points. Varying the gain over several operating points was deemed time consuming and unnecessary.

A second method to alleviate the noise was also attempted. Upon oscilloscope application, the input 5 V to power the controller was found to have a noisy signal. This noise was being passed back along the common ground, affecting the DC/DC IMON signal. Therefore, a number of capacitors were wired into the circuit in order to provide a low-pass filter for the voltage signal. This filter is scaled based on the equation:

$$f_c = \frac{1}{2 * \pi * R * C} \tag{15}$$

where f_c is the break frequency of the filter, R is the resistance of the inline resistor, and C is the capacitance value in farads of the capacitor. Using this simple filter, the ground noise signal was

smoothed using ceramic 100 microfarad capacitors. However, the smoothing did not completely alleviate the noise problem. By increasing the capacitance and resistance, the signal could be smoothed almost indefinitely. Increasing the resistance past a very small amount interfered with the DC/DC converter signal, making the math supplied by the manufacturer no longer usable because it changed the reference voltage. Increasing the capacitance smoothed the signal, but larger capacitors required longer time to charge up, slowing the response time of the measurement to an undesirable rate.

Figure 52: Fluke Model 115 True-RMS Multimeter used for current and voltage measurements

It is also worth noting the reason for the noise. DC/DC converters use a switching method to convert the input DC voltage to the desired value of output DC voltage, and this rapid switching induces noise into the signal. This noise is unavoidable in this type of transformer. SynQor recommends using a relatively complicated external filter that was unable to be built in the time allotted to the author. Therefore, to record data a multimeter (shown in Figure 52) was wired in series with the motor to read amperage, and another was wired in parallel to read the voltage. This type of multimeter uses a series of internal filters to almost completely eliminate

the noise [60]. With the steady reading of the multimeter, the current on the motor could more accurately be measured.

Regardless of these problems, the motor proved reliable and robust when controlled by the microcontroller. Response time was reasonably quick, and the motor controls similarly to an AC motor that most hobbyists would employ on an aircraft. This similarity comes from the approximately linear motor responses to the input commands. The approximately linear response aids in teaching operators how to fly the aircraft, as it is not as hard to pick up and fly. An average motor efficiency of 88.56% was deemed acceptable. This efficiency is quite similar to what the author expected. The manufacturer rates the motor at up to 94% efficient, but this is not at the tested current. The manufacturer claimed efficiency is tested at a much lower current, which the manufacturer conveniently does not provide. Over 85% is still very good for a brushed DC motor, so the motor actually exceeded expectations in this regard. The DC/DC converter also exceeded expectations, as the manufacturer rated efficiency at 24-48V is 93% while the actually efficiency was over 97%.

4.4 Climb Testing

4.4.1 Test Goals

This mode was considerably more difficult to analyze because of the different torque sources. The first goal was to ascertain the torque that could be provided at maximum power (100% stick position). This torque could then be analyzed for a rough determination of climb rate for the aircraft. Climb performance was considered a steady state parameter and allowed the engine and motor together to provide torque. The second major goal was to adjust the stick to

each position and measure the output torque in order to check the controller set points for each component. These set points would then be compared to the output torque so as to confirm validity. The third test measured the controller's general stability when the clutch or one-way bearing was engaged or disengaged as the torque was varied. See section 3.2.8 for coding details about this test.

4.4.2 Test Analysis

Again because of the noisy signals provided by the IMON pin on the DC/DC converter, many types of dynamic measurements were difficult, if not impossible, to take. This would have been acceptable because for most scenarios, the A/C will be in some sort of rough steady state flight. In climb, this is the case. For climb, Hiserote lists power required as 367.9 Watts, and with Rotramel's 5421 RPM efficient propeller point, the required torque is then 0.92474 N-m.

However, due to numerous complications with the test setup, climb data was unavailable. Many reasons were the cause for this. The Fuji engine, by its very nature, runs very rough and applies harsh torque spikes to the test setup. These spikes were thought to be eliminated with the use of a belt that would adjust its tension to remove these spikes from dealing damage to components. This was indeed the case, but the belts that were supposedly recommended by the dynamometer manufacturer (who designed the dynamometer for engine with these types of spikes) could not take the loads and snapped repeatedly. Additionally, complications with the electromagnetic clutch not being able to take loads it was designed for also caused several redesigns of the system to simply took too much time. Therefore, the author simulated the code results without test data to back up these simulations. While the test results are absent, the

author has full confidence that the simulations reflect a close approximation to actual performance with some minor tweaking to controller coding. Figure 53 shows the controller's

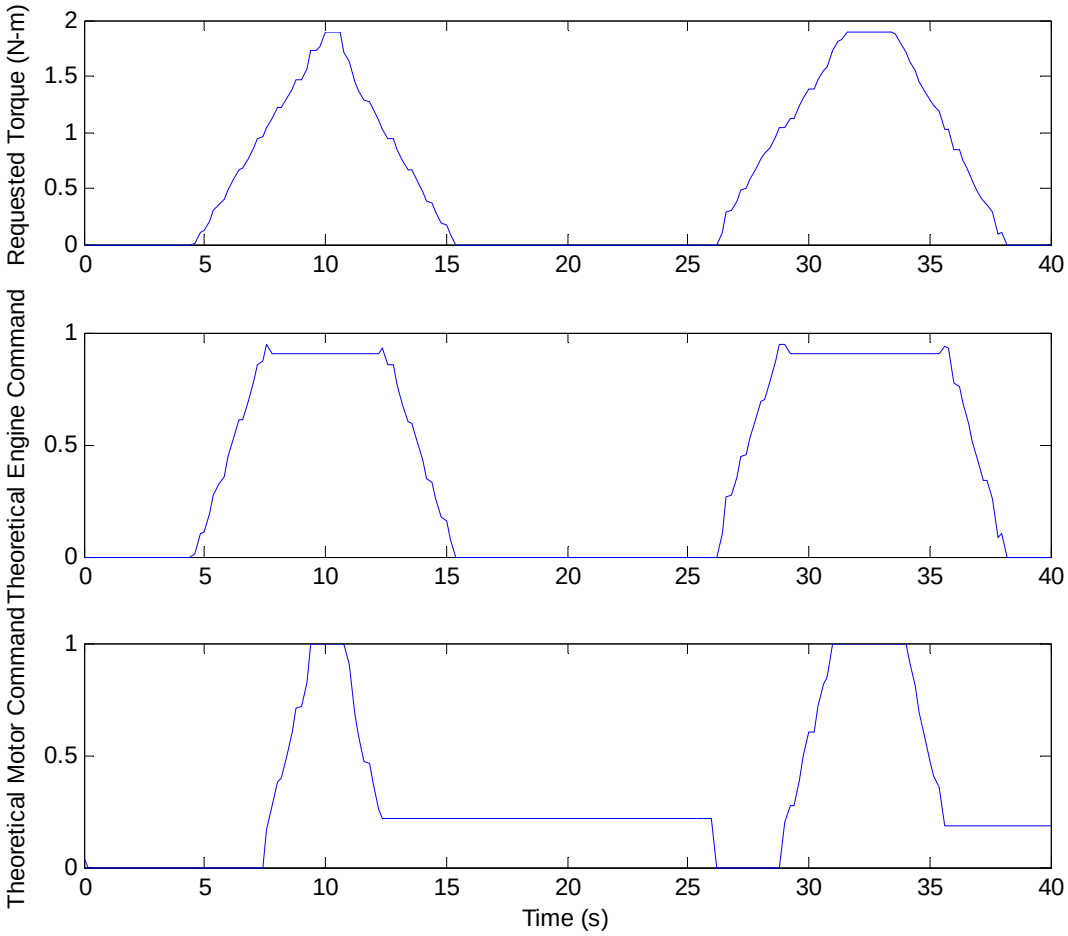


Figure 53: Controller commands for climb mode

output commands. Several things stand out upon initial review. First, the receiver input to the controller was not a perfectly linear signal, so small steps appear in the commanded torque plot. This was what causes the theoretical commands to both the motor and the engine to have small steps in them as well. What this will means in practice is the motor will hold a constant current

for a brief moment while the engine has to take up some extra torque. The time window for this was so small that in reality it should not matter.

Another thing of note was that the engine throttle command is relatively high. However, for climbing this is a necessity. The IOL programmed into the controller currently has the engine max torque at 1.1 N-m at 5000 RPM. The simulation shown above was simulated at an engine speed of 5000 RPM. Since the controller bases the engine throttle setting on the current RPM and the requested torque, the throttle setting on the engine will increase only after it gets past its set point. At 5000 RPM the engine will have a throttle setting of roughly 0.4, and the data makes sense in a realistic case.

The motor commands also look good at this point. Recalling the stepped engagement strategy proposed by Mr. Schurhoff, the engine and motor commands mimic this. The engine is used up to its IOL and the motor then kicks in and provides the rest of the torque. The flat line after the throttle command is released is a simulated result. In order to have the motor not drag the engine down with unnecessary back-torque, the motor command stays high enough to relieve the engine power shaft from having to power the motor shaft along with providing torque for flight.

4.5 Cruise with Regeneration

4.5.1 Test Goals

The primary goal of the cruise with regeneration test was to check the basic recharging model that the author developed to recharge the battery. Additional data was taken during this test to ensure safety of the operators and the equipment due to the unstable nature of LiPo

batteries during recharging. The temperature and charge rate of the batteries was under constant monitoring to guarantee full control of the process. Any error in charge rate or over temperature conditions nullified the test and forced a retest.

Testing methods were chosen based on the recharging model. In each test, the battery was inserted into the trial at a different state of charge: 25%, 50%, and 75%. Each of these regimes was determined to be a common battery SOC seen in the field. Recharge points lower than 25% were deemed unsafe due to the LiPo safety requirements [44], and recharge points over 75% were deemed unnecessary. Data collected in this test included ICE RPM, EM RPM, EM voltage, current draw, battery voltage and current received, and battery temperature. All of these measurements were taken versus time. See section 3.2.9 Cruise With Regeneration, for coding details about this state.

4.5.2 Test Analysis

Again due to numerous problems with the testing setup, true analysis of this mode was not possible. Simulation of the parameters would have provided results that were inconsistent, as the input to the controller is hard to simulate. In particular, the voltage input is difficult to simulate. The DC power supply is set up to provide power to DC/DC converter; having it also simulate an input voltage to the controller, while theoretically plausible, is impossible with the hardware in use. The power supply does not have the ability to split two voltage sources without a specialized electronic circuit. However, the controller model, being based off an actual recharger, will work well in theoretical flight. The only caveat to this is that the engine speed will need to be precisely controlled. Since voltage control is a major parameter to this recharge model, and the author's method of motor control relies on current limiting, the engine will

become a voltage regulator for the motor. In practice, this will mean that the operator will need to watch the engine RPM to make sure that it does not get too high or too low. In theory, the speed of the engine will not need to vary too much as the batteries will require at least 24 V. 24 V happens to correspond with the cruise speed of 5400 RPM, so the controller will be able to then regulate the current and recharge the batteries safely. More testing needs to be done to validate these statements.

V. Conclusions and Recommendations

5.1 Conclusions of Research and Testing

The DOD has specified a need for increasing missions from RPAs, and has also specified a need for increasingly quiet and more efficient RPAs. A hybrid-electric system, when implemented and controlled correctly as to take advantage of both electric power and gasoline power, could easily meet this need. The author's research sought to implement a controller in C for a proof-of-concept type aircraft that would control such a system in open-loop efficiently and effectively.

The research centered on the development of the C code. This C code controlled four distinct flight modes primarily, and had an additional 9 modes programmed for secondary functions. A test stand was developed to house the components of the HE system, and a LabView program was developed to interface with these components and the required sensors in order to record data. Validation of the code that was written was done on this test setup.

Initially, the C code was developed using the MPLAB IDE tool for programming microcontrollers. This code was built around several data collection and processing tools created by John Hagen. The code contained 9 initial states, with 4 states that would be active during A/C flight. The final base code file, its header file, and Hagen's data processing code can be seen in Appendix A.

With this code, work then started on developing a test setup to validate the code. The first step in this endeavor was creating a LabView interface that would work alongside the microcontroller and display various parameters throughout testing. This LabView screen was

based on modification of data collection LabView block diagrams again written by John Hagen. Data collection occurred through an RS232 serial port which passed data back and forth from the controller to the computer. Sensors such as the RPM sensors, temperature sensors, and throttle signal are examples of important data that LabView displays. Its screen also contains code in place for the future adaption of numerous other sensors such as cylinder pressure, fuel consumption, and battery voltage.

The final goal was to validate the test setup and code developed. A group of tests were then created, built on inputs from Rotramel's thesis work and Hiserote's original recommendations for a HE-RPA. Tests for the cruise without regeneration mode centered on a 5000 RPM operating point, outputting 0.66 N-m from the engine shaft. Endurance mode operating points were chosen at 5400 RPM and 0.27 N-m from the motor output shaft. Motor restart testing was also attempted, but proved unsuccessful and the setup shifted to using a one-way bearing instead of a clutch. Climb and cruise without regeneration testing originally were conceived but never realized fully, although climb mode was simulated with successful results.

Cruise without regeneration mode proved highly successful under evaluation. The Honda engine was easily the most stable engine of the trio that was purchased. The Fuji engines, while claiming to have better performance, had problems continuously throughout testing. The Fuji 34 engine was never tested due to a broken back plate suffered early. The Fuji 25 engine held together much better, but it was plagued by poor performance throughout the validation. Its torque spikes produced by the single cylinder design are extreme to the point of being unmanageable. Honda's engine was successful and correctly provided the torque the controller

commanded from it. Control of the Honda was stable throughout testing, with throttle spikes being eliminated by saturating the throttle commands.

Endurance mode, while initially troublesome, eventually had great results as well. Initially the challenge was getting a readable signal from the DC/DC converter IMON pin. Due to heavy switching noise, this signal was never stabilized, even with numerous different filters. The solution was to hook up multimeters in line with the motor current path to measure motor power usage. While not as precise as a constant signal with time, the method proved highly useful in gathering data about the motor. At the endurance mode primary flight point, the motor provided the torque required of 0.27 N-m at the correct RPM of 5450 RPM. While some noisy commands were unavoidable, the motor actually proved very stable once heavy saturation was implemented. Its efficiency surpassed the author's expectations, having 88.5% efficiency in endurance mode flight with a gear ratio of 1.454545. With a manufacturer rated efficiency of 93.5%, the motor is within a reasonable bound for actual efficiency versus measured efficiency.

Code was developed for climb and cruise with regeneration mode, but the code was never fully tested. While the author believes that the code is solid and will provide good open-loop performance, it needs to be tested more thoroughly. Improved test setups utilizing the Honda and the EM need to be built, and the DC/DC converter needs to have a more robust filter applied to its input and output terminals in order to get reliable data on these modes. The author has designed a revised version of the wiring for future students, shown in Figure 54. The main differences between the original design shown in Appendix E and the revised design are immediately clear. The inclusion of a DC/DC converter power filter will help to eliminate the noise coming from the switching converter, and can make the measurement pins on the converter

usable. The diagram also shows a dual ground design, where the controller has a ground for analog signals and a ground for digital signals. Conversations with electrical engineers have suggested that this will also help to eliminate noise, as the digital signals tend to produce high frequency noise that can feed back through the ground to the analog signals.

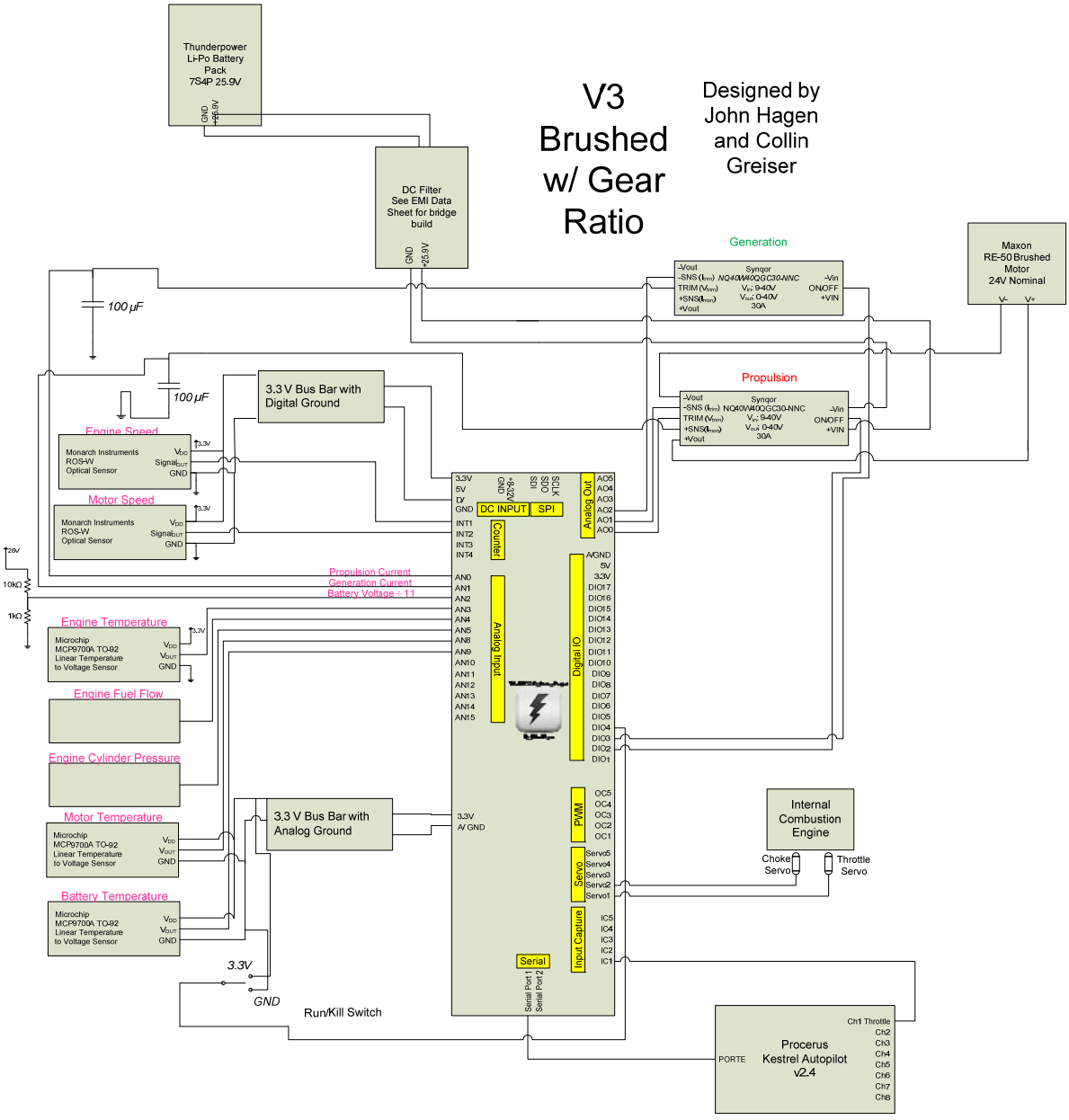


Figure 54: Revised controller wiring diagram to include filters and uncommon grounding points.

5.2 Recommendations for Future Work

As this was one of the first attempts in implementing a control method for an HE-RPA, there are infinite possibilities of paths and branches for exploratory research. Further refinement of control in particular could easily revolutionize the technology in today's warfighters hands around the world. However, there are a few key elements that could be improved upon and explored by future students.

Possibly the biggest challenge, and one that will be quite close at hand, is the actual flight testing of this control. While a great effort was undertaken in selecting sensors that would be of particular interest in the actual airframe, this task is much harder than simply picking small and lightweight sensors. The airframe, in particular, will go through many design iterations itself, which will require repositioning or possibly even replacement of components used by the author for validation. Future work is already being planned for this, but care must be taken to ensure communication between all design phases so that nothing is lost in translation.

On the topic of practical application of this control, another thing that needs to be accomplished is an exhaustive search to make certain that the controller is as robust as it can be before flight testing. Although the author made every attempt to ensure a robust, safe control method, there are always short sights and possible logic mistakes. The U.S. government already demands strict safety guidelines while flight testing, but every single failure mode must be tested further. Failure modes, such as overheating, were already tested or planned for; other unique failure modes such as weapons hits that disable parts of the propulsion system or sensor failures brought on by bird strikes are two examples of items that need more investigating. How the

controller will react to these types of situations was not thoroughly tested and needs to be investigated.

The author originally theorized that the cylinder pressure measurement, through equation analysis, could be used for an estimate of the engine's torque output, which would allow for a great deal of optimization based on the engine's current state. However, due to time constraints this was never implemented but only discussed in theory. A practical application of this theory could be examined by another student in follow-up work. Once the estimator has been developed, the test setup already implemented could be used to evaluate the estimator.

While it was tested and proved unsuccessful, restarting the engine with the electric motor needs a more thorough look. Due to mechanical limitations with a carburetor design on these model engines, fuel leaks into the cylinder while the aircraft is maneuvering in endurance flight. This fuel accumulates so that the electric motor cannot physically provide enough torque to turn the engine over since the fuel is approximately an incompressible liquid. The compression stroke of the engine can also be difficult to overcome; a combination of the two reasons would make restarting in flight very difficult. Two methods that could be tested to eliminate this phenomenon would include a fuel lockout valve with a simple servo or using an engine that is fuel-injected. Restarting the engine in flight versus allowing it idle could provide moderate increases in fuel economy with little to no penalty to weight or cost.

The revised wiring of the microcontroller, based on hands-on experience with the hardware, could be a great upgrade to the current setup. The DC/DC converter with DC motor setup, while easy to control, is a noisy system and measurements are hard to take in practice. With the filters that are built in to the new design, the noise could be limited or eliminated,

making the system far more robust. Inclusion of a dual grounding system would help to make the measurements more accurate by eliminating the digital noise feeding back through the ground.

Even though the DC/DC converter is simpler to control, an AC system could also be looked at. As stated in Chapter II, the AC motors are more difficult to control, requiring a separate controller just for the motor. However, if man-hours are spent on making the author's controller and the AC motor controller work in harmony, the AC system could possibly work much better than the DC system. Transmission losses which are readily inherent to DC systems could be avoided with an AC motor system. Challenges would be evident in using the author's current limiting method to regulate the motor torque, but these challenges could be overcome with enough time and effort.

The control methods mentioned in the literature review could also be applied. These methods such as fuzzy logic or neural networks, could possibly be installed on the same control circuit and have great potential to be more efficient than the rule-based method that the author employed. These control methods could make greater use of a closed-loop system which could track error in torque measurement and more accurately divide the torque being provided from the two sources. These methods could potentially eliminate reliance on engine maps as well, instead accurately predicting output torque by completing a dynamic optimization of the engine based on its current status (temperature, fuel usage, number of hours ran, etc.).

As with any type of design, there are always hurdles to overcome. However, designing the test system proved to be many times more difficult than originally expected. With the implementation of this system on a real airframe, problems encountered could be magnified many times. In order to avoid this, concentration should be placed on the design,

implementation, and trials of the system so that a safe and efficient transition is made from prototype to practical design.

There are still many avenues of research that need to be completed in order to further optimize the HE-RPA control problem. However, once this system is implemented, it will provide the best solution to meet the propulsion needs and challenges facing the warfighters of today. This effort will optimally work for the components described, but ultimately could be adapted to any HE system with the right software modifications. Modifications such as specific motor maps and engine maps, and power splitting strategies that make sense for the application are examples of things that could make the code adaptable to other systems.

Works Cited

[1] L. R. Newcome, *Unmanned Aviation: A Brief History of Unmanned Aerial Vehicles*. Reston, VA: AIAA, 2004.

[2] Hybrid Kingdom. (2008, January) History of Hybrid Cars. [Online]. http://www.historyofhybridcars.com

[3] Auto Editors of the Consumer Guide. (2010, January) www.howstuffworks.com. [Online]. http://auto.howstuffworks.com/1971-1972-1973-ford-mustang6.htm

[4] Judith Goodstein, "Godfather of the Hybrid," *Engineering and Technology*.

[5] Department of Defense, "Unmanned Systems Roadmap 2007-2032," Department of Defense, 2007.

[6] J. Paur. (2009, July) Hybrid Power Comes to Aviation. [Online]. www.wired.com

[7] Chan-Chiao Lin, Huei Pang, and J.W. Grizzle, "A Stochasitc Control Strategy for Hybrid Electric Vehicles," University of Michigan, DoD Research.

[8] Frederick G Harmon, "Neural Network Control of a Parallel Hybrid Electric Propulsion System for a Small Unmanned Aerial Vehicle ," University of California-Davis, Davis, CA. , PhD Dissertation 2005.

[9] Ryan M. Hiserote, "Analysis of Hybrid-Electric Propulsion System Designs for Small Unmanned Aircraft Systems," Air Force Institute of Techonology, Wright-Patterson Air Force Base, M.S. Thesis 2010.

[10] J.D. Anderson, *Aircraft Performance and Design*. Boston, MA: McGraw-Hill, 1999, ch. 2-3.

[11] Matthew D Rippl, "Sizing Analysis for Aircraft Utilizing Hybrid-Electric Propulsion Systems," Air Force Institute of Technology, Wright Patterson Air Force Base, M.S. Thesis 2011.

[12] P. Van den Bossche. (2010, May) Wikipedia. [Online]. http://en.wikipedia.org/wiki/File:Hybridpeak.png

[13] Fazal Syed et. al., "Derivation and Experimental Validation of a Power-Split Hybrid Electric Vehicle Model," *IEEE Transaction on Vehicular Technology*, vol. 55, no. 6, 2006.

[14] P. Van den Bossche. (2010, May) Wikipedia. [Online]. http://en.wikipedia.org/wiki/File:Hybridpar.png

[15] J. M. Miller, *Propulsion Systems for Hybrid Vehicles*. London, UK: IEEE Power & Energy Series, 2004.

[16] Jason Christensen, Lynn Gantt, Doug Nelson, Adam Robinson, and Michael Stover, "EcoCAR Design and Development Process for a Plug-In E85 Split-Parallel Architechture Hybrid Electric Vehicle," Virginia Polytechnic Institute and State University, Blacksburg, VA, ASME, 2009.

[17] D. Lundstrom, K. Amadori, and P. Krus, "Validation of Small Scale Electric Propulsion System Models," in *48th AIAA Aerospace Sciences Meeting*, Orlando, 2010.

[18] Castle Creations, *Phoenix ICE 50 and ICE 75 Brushless Controller User's Manual*. Kansas, U.S.A., 2009.

[19] Northwestern University. (2010, December) Brushed DC Motor Theory. [Online]. http://hades.mech.northwestern.edu/index.php/Brushed_DC_Motor_Theory

[20] J.B. Heywood and Eran Sher, *The Two-Stroke Cycle Engine: Its Development, Operation, and Design*.: SAE, 1999, p. 9.

[21] J.B. Heywood, *Internal Combustion Engine Fundamentals*. New York, NY: McGraw-Hill, 1988, ch. 1-2, 15.

[22] John J. Moskwa and Chung-hung Pan, "Engine Load Torque Estimation Using Non-Linear Observers," in *Conference on Decision and Control*, New Orleans, 1995.

[23] D. Khiar, J. Lauber, T. Floquet, and T.M. Guerra, "An Observer Design for the Instantaneous Torque Estimation of an IC Engine," in *Vehicle Power and Propulsion*, 2005.

[24] Cary Wilson, "Performance of a Small Internal Combustion Engine Using N-Heptane and Iso-Octane," Air Force Institute of Technology, Wright Patterson Air Force Base, M.S. Thesis 2010.

[25] Shyam K. Menon, "Performance Measurement and Scaling In Small Internal Combustion Engines," Univeristy of Maryland-College Park, College Park, MD, M.S. Thesis 2006.

[26] Optrand, Inc. (2010, June) Optrand Incorporated Website. [Online]. www.optrand.com

[27] Martin Winter and Ralph J. Brodd, "What are Batteries, Fuel Cells, and Supercapacitors?," *Chemical Reviews*, vol. 104, no. 10, pp. 4245-4269, September 2004.

[28] Woodbank Communications Ltd. (2010, May) Battery State of Charge Determination. [Online]. http://www.mpoweruk.com/soc.htm

[29] Frank Fleming, *Friction and Magnetism: The Basics Of Electromagnetic Clutches and Brakes*. Somerset, N.J. , U.S.A.: Ogura Industrial Corporation, 2009.

[30] Todd A. Rotramel, "Optimization of Hybrid-Electric Propulsion Systems for a Small Remotely-Piloted Aircraft," Air Force Institute of Technology, Wright Patterson Air Force Base, M.S. Thesis 2011.

[31] Isseyas H. Mengistu, "Small Internal Combustion Engine Testing for Hybrid-Electric Remotely Piloted Aircraft Propulsion," Air Force Institute of Technology, Wright Patterson Air Force Base, M.S. Thesis 2011.

[32] A.B. Fransisco, "Implementation of an Ideal Operating Line Control Strategy for Hybrid Electric Vehicles," University of California-Davis, Davis, CA, M.S. Thesis 2002.

[33] Robert W. Schurhoff, "The Development and Evaluation of an Optimal Powertrain Control Strategy for a Hybrid Electric Vehicle," University of California, Davis, CA, Masters Thesis 2002.

[34] P Singh, C. Fennie, and D.E. Reisner, "Logical Progression," in *Electric and Hybrid Vehicle Technology*., 2000, pp. 72-74.

[35] J. Jantzen. (2002) A Tutorial on Adaptive Fuzzy Control. [Online]. www.eunite.org

[36] Sema Alptekin, Diane Deturris, and Jon Ervin, "Optimization of the Fuzzy Logic Controller for an Autonomous UAV," 2005.

[37] M. Salman, N.J. Schouten, and N.A. Kheir, "Control Strategies for Parallel Hybrid Vehicles," in *American Control Conference*, Chicago, 2000.

[38] Martin T. Hagan and Howard B. Demuth, "Neural Networks for Control," in *American Control Conference*, San Diego, 1999.

[39] Joseph Matthews, "Adaptive Control of Micro Air Vehicles," Brigham Young University, Provo, Utah, M.S. Thesis 2006.

[40] Sidhartha Panda, N.P. Padhy, and R.N. Patel, "Application of Genetic Algorithm for FACTS-Based Controller Design," *International Journal of Computer, Information and Systems Science and Engineering*, 2007.

[41] Valcor Engineering Corporation. Electroid Company Catalog: Electromagentic Clutch Model EC-26C. [Online]. www.electroid.com/Catalog/CAT.pdf

[42] Maxon Precision Motors Incorporated. (2010, May) Maxon Motors RE50 Spec Sheet. [Online]. http://shop.maxonmotor.com/ishop/article/article/370354.xml

[43] Todd A. Rotramel, "Small UAS Propeller Acoustic Testing," Air Force Institute of Technology, WPAFB, Ohio, Term Paper 2010.

[44] Thunder Power USA, *Li-Polymer Charger/Discharger User's Manual TP-1010C*. Las Vegas, NV, USA.

[45] Land and Sea Incorporated, *DYNOMite Owner's Manual*. Concord, NH, USA, 2006.

[46] 80-20 Incorporated. (2008) T-Slotted Framing. http://www.8020.net/T-Slot-1.asp.

[47] McMaster-Carr Incorporated. (2010) Bullet Resistant PolyCarbonate Shielding. http://www.mcmaster.com/#bullet-resistant-polycarbonate/=b37pmf.

[48] Monarch Instruments. (2010, September) www.monarchinstruments.com. [Online]. http://www.monarchserver.com/Manuals/1071-4854-115%20ROS%20Eng.pdf

[49] SynQor Incorporated, *Technical Specifications NQ40x40QGC30*. Boxborough, MA, USA, 2009.

[50] Mastech Corporation, *DC power Supply Model HY6020E User's Manual*.

[51] Fuji IMVAC Incorporated, *Operators Manual for BF-25F and BF-34F EI 4-Stroke Engines*, 1st ed. Yokohama, Japan, 2004.

[52] Honda Motor Company, *GX35 Owner's Manual*, 9th ed. Japan, 2004.

[53] Sullivan Products. (2008) 12VDC Model Engine Starter Instructions. Electronic PDF.

[54] John T. Hagen, *The PIC32 Lightning Project: PIC32 Lightning Project User's Manual*, 1st ed. Fairborn, OH, U.S.A., 2010, (Personal Work and Interview).

[55] MicroChip Technology Incorporated, *PIC32MX3XX/4XX Family Data Sheet*. Chandler, AZ,

USA, 2008.

[56] Microchip Technology Incorporated, *MPLAB IDE User's Guide*. Chandler, AZ, U.S.A., 2005.

[57] Futaba Corporation, *8 Channel Radio Control System Instruction Manual*. Chiba, Japan, 2009.

[58] Dimension Engineering. (2010, February) Dimension Engineering Online Shop. [Online]. http://www.dimensionengineering.com/BattleSwitch.htm

[59] Ambient LLC, *Ambient Weather WS-1170 Advanced Weather Station User Manual*, 14th ed. Chandler, AZ, U.S.A., 2010.

[60] Fluke Corporation, *True-RMS Multimeters 114, 115, 117 User's Manual*, 2nd ed. Everett, WA, U.S.A., 2007.

[61] K. Ogata, *Modern Control Engineering*, 4th ed. Upper Saddle River, New Jersey: Pearson Education, Inc. , 2002.

[62] S. Zaloga, *Unmanned Aerial Vehicles: Robotic Air Warfare 1917-2007*. Westminster, MD: Osprey Publishing Ltd., 2008.

[63] Richard Botzum, *50 Years of Target Drone Aircraft*.: Northrop, 1985.

[64] Frederick G. Harmon, Andrew A. Frank, and Jean-Jaques Chattot, "Conceptual Deisgn and Simulation of a Small Hybrid-Electric Unmanned Aerial Vehicle," *Journal of Aircraft*, vol. 43, no. 5, October 2006.

[65] Bernard Michini, "Modeling and Adaptive Control of Indoor Unmanned Aerial Vehicles," Massachusetts Institute of Technology, Cambridge, MA, M.S. Thesis 2009.

[66] Optrand Incorporated, *AutoPSI Pressure Sensor Operation Instructions*, 5th ed. Plymouth, MI, USA, 2009.

[67] SynQor Corporation. (2002, June) SynQor DC/DC EMI Characteristics. [Online]. http://www.synqor.com/documents/appnotes/appnt_EMI_Characteristics.pdf

[68] Grand Wing Systems Incorporated, *MT-1, the Multi-Tester.* , 7th ed. Huatai Keji Yuanqu, Xiegang Town, China.

[69] Gates Corporation, *Belt Drive Preventative Maintinence and Safety Manual*. Denver, CO, U.S.A., 2008.

Vita

Lieutenant Collin M. Greiser graduated from Heritage High School in Newport News, VA in 2004. He completed his Bachelor of Science degree in mechanical engineering at Virginia Polytechnic Institute and State University in 2009. He then received his Air Force commission and had his first assignment as a graduate student at the Air Force Institute of Technology working on his masters in astronautical engineering. Lieutenant Greiser completed this degree in March 2011 and continued on to his next assignment at Los Angeles Air Force Base, California working for the Missile Defense Agency.

Appendix A: Controller Code

```
/*****************************************************************************
*****************************
FileName:         HybridPropulsionControl2.0.c
Author:           Collin Greiser & John Hagen
Project:          Hybrid Electric RPA
Description:      Implements a state machine that controls how user commands
are translated into different configurations of the Hybrid Electric UAV.
Handles controlling when the engine and motor are powered and when power is
taken to and from the battery pack.


*****************************************************************************
**************************/

/** PRIVATE PROTOTYPES *************************************************/
double GetTotalAvailableTorque();
double GetTorqueRequest();
double GetMaxEMTorque();
double GetMaxICETorque();
double GetICEIOLTorque();
void SetNormalizedEMTorque(double normalizedTorque);
double GetThrottleSetting();
/** PRIVATE PROTOTYPES *************************************************/

/***** NUMERICAL CONSTANTS*********************************************/
const double IdleThrottle = 0.0;
const double GearRatio = 1.0;

double MaxEMTorque;// Not constant, calculated in ConfigureHybridController()
const double EMTorqueConstant = 0.0396;   // Nm/A
const double DCDCMaxCurrent = 20.0;       // A

const int ICECountsPerRev = 1;
const int EMCountsPerRev = 1;

double ICEMapYValues[] = {1.1, 1.3, 1.3, 1.4, 1.4, 1.4, 1.4, 1.3, 1.2, 1.1,
1.0};
const int ICEMapLength = 11;  // ^Number of values MUST match ICEMapLength
const double ICEMapXStart = 4000.0;
const double ICEMapXStep = 500.0;

double IOLMapYValues[] = {0.2, 0.45, 0.632, 0.85, 0.95, 1.05, 1.15, 1.1, 1.0,
0.9, 0.7};
const int IOLMapLength = 11;  // ^Number of values MUST match IOLMapLength
const double IOLMapXStart = 1000.0;
const double IOLMapXStep = 500.0;

#define EngineMapLength 5
#define EngineMapWidth 11
#define MotorMapLength 5
#define MotorMapWidth 5
```

```
//Torque map for the Fuji 34-FI engine
double Fuji34TorqueMap[EngineMapLength][EngineMapWidth] =  {{0.8, 0.9, 1.0,
1.1, 1.3, 1.6, 1.9, 2.0, 1.9, 1.7, 1.6},

          {0.7, 0.8, 1.0, 1.0, 1.2, 1.5, 1.8, 2.0, 1.9, 1.7, 1.5},

          {0.6, 0.8, 0.9, 0.9, 1.1, 1.4, 1.7, 1.9, 1.8, 1.6, 1.4},

          {0.5, 0.7, 0.8, 0.8, 1.0, 1.3, 1.6, 1.7, 1.7, 1.6, 1.4},

          {0.5, 0.6, 0.7, 0.7, 0.8, 1.2, 1.5, 1.5, 1.5, 1.5, 1.3}
};

//Fuel usage for each torque point

double Fuji34FuelUseMap[EngineMapLength][EngineMapWidth] = {{1.0, 1.5, 1.5,
1.7, 1.8, 2.0, 2.4, 2.5, 2.7, 1.9, 1.9},

            {0.9, 1.4, 1.4, 1.5, 1.7, 1.9, 2.3, 2.4, 2.5, 1.9, 1.8},

            {0.9, 1.3, 1.3, 1.4, 1.6, 1.8, 2.2, 2.3, 2.4, 1.8, 1.7},

            {0.8, 1.2, 1.2, 1.3, 1.5, 1.7, 2.1, 2.2, 2.3, 1.7, 1.6},

            {0.7, 1.1, 1.1, 1.2, 1.4, 1.6, 2.0, 2.1, 2.2, 1.6, 1.5}
};

//Throttle position map for each torque point
double Fuji34ThrottleMap[EngineMapLength][EngineMapWidth] = {{1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0},

             {.90, .90, .88, .87, .86, .90, 1.0, .90, .85, .83, .82},

             {.80, .85, .83, .85, .88, .89, .90, .86, .83, .81, .78},

             {.60, .75, .77, .78, .80, .85, .85, .83, .80, .77, .75},

{.50, .65, .73, .74, .78, .78, .74, .77, .74, .72, .70}
};

//Torque Map for Maxon Electric Motor
//Scaled by speed (x-axis) and Power in (y-axis)

double MaxonTorqueMap[MotorMapLength][MotorMapWidth] =  {{0.5, 0.5, 0.5, 0.5,
0.5},

       {0.4, 0.4, 0.4, 0.4, 0.4},

       {0.3, 0.3, 0.3, 0.3, 0.3},

       {0.2, 0.2, 0.2, 0.2, 0.2},

       {0.1, 0.1, 0.1, 0.1, 0.1}
};
```

```
double MaxonEfficiencyMap[MotorMapLength][MotorMapWidth] = {{0.59, 0.62,
0.67, 0.66, 0.71},

            {0.69, 0.89, 0.87, 0.95, 0.85},

            {0.83, 0.95, 0.90, 0.94, 0.88},

            {0.75, 0.93, 0.95, 0.97, 1.0},

            {0.66, 0.87, 0.96, 0.0, 0.0}
};



/***** NUMERICAL CONSTANTS*********************************************/
//#include <GenericTypeDefs.h>
#include "..\Lightning\LightningScreen.h"
#include "..\Lightning\LightningIO.h"
#include "..\Lightning\LightningDrive.h"
#include "..\Lightning\LightningStream.h"
#include "HybridPropulsionControl.h"
#include "math.h"
void ConfigureHybridController()
{
      ConfigureDigitalIO(ClutchPort, ClutchPin, SETOUTPUT);
      ConfigureDigitalIO(PropulsionDCDCOnOffPort, PropulsionDCDCOnOffPin,
SETOUTPUT);
      ConfigureDigitalIO(GenerationDCDCOnOffPort, GenerationDCDCOnOffPin,
SETOUTPUT);
      ConfigureDigitalIO(RunKillPort, RunKillPin, SETINPUT);
      ConfigureDigitalIO(TakeoffIdlePort, TakeoffIdlePin, SETINPUT);

      MaxEMTorque = EMTorqueConstant * DCDCMaxCurrent;
}


#define RESET_PROP                      0
#define EMREV_PROP                      1
#define ICESTART_PROP                   2
#define ICEIDLE_PROP                    3
#define GROUNDROLL_PROP                 4
#define CATAPULT_PROP                   5
#define CLIMB_PROP                      6
#define ICEONLY_PROP                    7
#define EMONLY_PROP                     8
#define ICEANDEM_PROP                   9
#define EMONLYREGENBRAKE_PROP           10
#define ICEONLYGENERATION_PROP          11


int propulsionState = RESET_PROP;
void PropulsionControlStateMachine()

{
      //Get Component Temperature for safety checks
      double engineTemp = GetTemperatureTC1047A(ANPORT3);
```

```
	double motorTemp = GetTemperatureTC1047A(ANPORT5);
	double batteryTemp = GetTemperatureTC1047A(ANPORT9);


	SetSharedCustomMemory(0,propulsionState);

	switch(propulsionState)
	{

		//Reset_prop state is the default "dead" state for all components
		case RESET_PROP:
			SetDigitalOutput(ClutchPort, ClutchPin, FALSE);
			SetPWMDutyCycle(ICEThrottleServo, 0.0);
			SetPWMDutyCycle(ICEChokeServo, 0.0);

			SetDigitalOutput(PropulsionDCDCOnOffPort,
PropulsionDCDCOnOffPin, FALSE);
			SetOutputVoltageNQ40(PropulsionDCDCOutputVolt, 0.0);
			SetCurrentLimitNQ40(PropulsionDCDCCurrentLimit, 0.0);

			SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, FALSE);
			SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 0.0);
			break;


		//Spin electric motor to provide starting torque for the ICE
		case EMREV_PROP:  // Spin motor to 4000rpm
			SetDigitalOutput(ClutchPort, ClutchPin, FALSE);
			SetPWMDutyCycle(ICEThrottleServo, 30.0);
			SetPWMDutyCycle(ICEChokeServo, 50.0);

			SetDigitalOutput(PropulsionDCDCOnOffPort,
PropulsionDCDCOnOffPin, TRUE);
			SetOutputVoltageNQ40(PropulsionDCDCOutputVolt,
RPMToVoltageRE50(4000));
			SetCurrentLimitNQ40(PropulsionDCDCCurrentLimit, 30.0);

			SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, FALSE);
			SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 0.0);

			if(GetGenericRPM(EMSpeedPort, EMCountsPerRev) > 3800)
			{
				propulsionState = ICESTART_PROP;
			}

			break;


		//Dump clutch with EM spinning so ICE can be started
		//Added timer to shut off EM if ICE refuses to start
		case ICESTART_PROP:
			StartVirtualTimer(VTIMER1);

			SetPWMDutyCycle(ICEThrottleServo, 30.0);
```

```
                  SetPWMDutyCycle(ICEChokeServo, 50.0);

                  SetDigitalOutput(PropulsionDCDCOnOffPort,
PropulsionDCDCOnOffPin, TRUE);
                  SetOutputVoltageNQ40(PropulsionDCDCOutputVolt,
RPMToVoltageRE50(4000));
                  SetCurrentLimitNQ40(PropulsionDCDCCurrentLimit, 30.0);

                  SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, FALSE);
                  SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 0.0);


                  //Timer to regulate amount of time starter motor is
engaged.
                  int timer = GetVirtualTimer(VTIMER1);
                  if(timer>5000000)
                  {
                        propulsionState = RESET_PROP;
                        PauseVirtualTimer(VTIMER1);
                        ResetVirtualTimer(VTIMER1);
                  }
                  //Move to next state if ICE starts
                  if(GetGenericRPM(ICESpeedPort, ICECountsPerRev) > 3000)
                  {
                        propulsionState = ICEIDLE_PROP;
                  }


                  break;

            //Idle state to allow ICE to warm up per manufacturer
instructions
            case ICEIDLE_PROP:
                  SetDigitalOutput(ClutchPort, ClutchPin, FALSE);
                  SetPWMDutyCycle(ICEThrottleServo, IdleThrottle);
                  SetPWMDutyCycle(ICEChokeServo, 50.0);

                  SetDigitalOutput(PropulsionDCDCOnOffPort,
PropulsionDCDCOnOffPin, FALSE);
                  SetOutputVoltageNQ40(PropulsionDCDCOutputVolt, 0.0);
                  SetCurrentLimitNQ40(PropulsionDCDCCurrentLimit, 0.0);

                  SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, FALSE);
                  SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 0.0);

                  if(GetDigitalInput(TakeoffIdlePort, TakeoffIdlePin))
                  {
                        propulsionState = GROUNDROLL_PROP;  //TODO: be sure
this is correct
                  }

                  break;
```

```
            //Dual Power mode for a groundroll style takeoff
            //Disabled for bench testing
            case GROUNDROLL_PROP:

                  SetPWMDutyCycle(ICEThrottleServo, IdleThrottle);
                  SetPWMDutyCycle(ICEChokeServo, 50.0);

                  SetDigitalOutput(PropulsionDCDCOnOffPort,
PropulsionDCDCOnOffPin, TRUE);
                  SetOutputVoltageNQ40(PropulsionDCDCOutputVolt, 0.0);
                  SetCurrentLimitNQ40(PropulsionDCDCCurrentLimit, 0.0);

                  SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, FALSE);
                  SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 0.0);
                  //TODO: Check this
                  if (GetGenericRPM(ICESpeedPort, ICECountsPerRev) < 2000 &&
GetDigitalInput(TakeoffIdlePort, TakeoffIdlePin)); //Filler RPM, need to
determine Idle Speed
                  {
                        propulsionState =  CLIMB_PROP;
                  }


                  break;


            //State for a catapult style takeoff.  Unsused at the moment
            case CATAPULT_PROP:

                  break;

            //Dual power mode for climbing.  Roughly identical to
Groundroll_Prop
            case CLIMB_PROP:

                  SetPWMDutyCycle(ICEChokeServo, 0.0);

                  SetDigitalOutput(PropulsionDCDCOnOffPort,
PropulsionDCDCOnOffPin, TRUE);

                  SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, FALSE);
                  SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 0.0);

                  //Combine Electric Motor and ICE.  Use all availiable
torque from ICE, any remaining request for EM
                  double torqueRequest = GetTorqueRequest();

                  //Get torque request and command ICE
                  if(GetTorqueRequest() < (GetMaxICETorque()))
                  {
                        double normalizedICETorque = GetTorqueRequest() /
GetMaxICETorque();

                        // Saturate signal 0.0 - 1.0
```

```
                        if(normalizedICETorque > 1.0)
                        {
                              normalizedICETorque = 1.0;
                        }

                        //Note: Changed to 'IdleThrottle', makes more sense
than stalling the engine
                        else if(normalizedICETorque < 0.0)
                        {
                              normalizedICETorque = IdleThrottle;
                        }

                        SetPWMDutyCycle(ICEThrottleServo,
normalizedICETorque);

                  }
                  //If torque request is above torque availiable, use EM to
provide remaining.
                  else
                  {
                        double normalizedTorque = GetMaxICETorque() /
GetMaxICETorque(); //Filler, this just returns a 1

                        // Saturate signal 0.0 - 1.0
                        if(normalizedTorque > 1.0)
                        {
                              normalizedTorque = 1.0;
                        }
                        else if(normalizedTorque < 0.0)
                        {
                              normalizedTorque = IdleThrottle;
                        }
                        SetPWMDutyCycle(ICEThrottleServo, normalizedTorque);
//Continues to provide max engine power

                        double remainingTorque = GetTorqueRequest() -
GetMaxICETorque();
                        double normalizedEMTorque = remainingTorque /
GetMaxEMTorque();

                        // Saturate signal 0.0 - 1.0, eespecially important
as remainingTorque will often return a negative value
                        if(normalizedEMTorque > 1.0)
                        {
                              normalizedEMTorque = 1.0;
                        }
                        else if(normalizedEMTorque < 0.0)
                        {
                              normalizedEMTorque = 0.0;
                        }

                        SetNormalizedEMTorque(normalizedEMTorque);

                  }

                  break;
```

```
            //ICE Only operating mode
            //Switch C == True, D, G == False

            case ICEONLY_PROP:

                  SetPWMDutyCycle(ICEChokeServo, 0.0);

                  SetDigitalOutput(PropulsionDCDCOnOffPort,
PropulsionDCDCOnOffPin, FALSE);
                  SetOutputVoltageNQ40(PropulsionDCDCOutputVolt, 0.0);
                  SetCurrentLimitNQ40(PropulsionDCDCCurrentLimit, 0.0);

                  SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, FALSE);
                  SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 0.0);


                  //normalize torque request for throttle input
                  double normalizedICETorque = GetTorqueRequest() /
GetMaxICETorque();

                  // Saturate signal 0.0 - 1.0
                  if(normalizedICETorque >= 1.0)
                  {
                        normalizedICETorque = 1.0;
                  }
                  else if(normalizedICETorque <= IdleThrottle)
                  {
                        normalizedICETorque = IdleThrottle;
                  }
                  if(normalizedICETorque < 1.0 && normalizedICETorque > 0.95)
                  {
                        normalizedICETorque = 0.95;
                  }
                  else if(normalizedICETorque < 0.95 && normalizedICETorque >
0.90)
                  {
                        normalizedICETorque = 0.9;
                  }
                  else if (normalizedICETorque < 0.9 && normalizedICETorque >
0.85)
                  {
                        normalizedICETorque = 0.85;
                  }
                  else if(normalizedICETorque < 0.85 && normalizedICETorque >
0.8)
                  {
                        normalizedICETorque = 0.8;
                  }
                  else if(normalizedICETorque < 0.8 && normalizedICETorque >
0.75)
                  {
                        normalizedICETorque = 0.75;
                  }
```

```
                else if(normalizedICETorque < 0.75 && normalizedICETorque >
0.7)
                {
                      normalizedICETorque = 0.7;
                }
                else if(normalizedICETorque < 0.7 && normalizedICETorque >
0.68)
                {
                      normalizedICETorque = 0.7;
                }
                else if(normalizedICETorque < 0.68 && normalizedICETorque
>= 0.65)
                {
                      normalizedICETorque = 0.65;
                }
                else if(normalizedICETorque < 0.65 && normalizedICETorque >
0.6)
                {
                      normalizedICETorque = 0.6;
                }
                else if(normalizedICETorque < 0.6 && normalizedICETorque >
0.55)
                {
                      normalizedICETorque = 0.55;
                }
                else if(normalizedICETorque < 0.55 && normalizedICETorque >
0.53)
                {
                      normalizedICETorque = 0.55;
                }
                else if(normalizedICETorque < 0.53 && normalizedICETorque
>= 0.5)
                {
                      normalizedICETorque = 0.5;
                }
                else if(normalizedICETorque < 0.5 && normalizedICETorque >
0.45)
                {
                      normalizedICETorque = 0.5;
                }
                else if(normalizedICETorque < 0.45 && normalizedICETorque >
0.42)
                {
                      normalizedICETorque = 0.45;
                }
                else if(normalizedICETorque < 0.42 && normalizedICETorque
>= 0.4)
                {
                      normalizedICETorque = 0.4;
                }
                else if(normalizedICETorque < 0.4 && normalizedICETorque >
0.35)
                {
                      normalizedICETorque = 0.35;
                }
```

```
                  else if(normalizedICETorque < 0.35 && normalizedICETorque >
0.3)
                  {
                        normalizedICETorque = 0.3;
                  }
                  else if(normalizedICETorque < 0.3 && normalizedICETorque >=
0.2)
                  {
                        normalizedICETorque = 0.25;
                  }
                  else if(normalizedICETorque < 0.2 && normalizedICETorque >=
0.1)
                  {
                        normalizedICETorque = 0.15;
                  }
                  else if(normalizedICETorque < 0.1 && normalizedICETorque >=
0.0)
                  {
                        normalizedICETorque = 0.0;
                  }
                  else if(normalizedICETorque <= 0.0)
                  {
                        normalizedICETorque = 0.0;
                  }
                  //prevent a stall condition
                  if(GetGenericRPM(ICESpeedPort, ICECountsPerRev) < 3000)
                  {
                        SetPWMDutyCycle(ICEThrottleServo, 0.6);
                  }
                  int EngineSpeed = GetGenericRPM(ICESpeedPort,
ICECountsPerRev);
                  SetPWMDutyCycle(ICEThrottleServo, normalizedICETorque);


                  SetSharedCustomMemory(1, normalizedICETorque);
                  SetSharedCustomMemory(3, EngineSpeed);


                  break;


            //Electric Motor Only Operation
            //Switch D == True, C, G == False

            case EMONLY_PROP:
                  SetDigitalOutput(ClutchPort, ClutchPin, FALSE);
                  SetPWMDutyCycle(ICEThrottleServo, IdleThrottle);
                  SetPWMDutyCycle(ICEChokeServo, 0.0);

                  SetDigitalOutput(PropulsionDCDCOnOffPort,
PropulsionDCDCOnOffPin, FALSE);

                  SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, TRUE);
                  SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 0.0);
```

```
                  double normalizedEMTorque = GetTorqueRequest() /
MaxEMTorque;

                  // Saturate signal 0.0 - 1.0
                  if(normalizedEMTorque > 1.0)
                  {
                        normalizedEMTorque = 1.0;
                  }
                  else if(normalizedEMTorque <= 0.0)
                  {
                        normalizedEMTorque = 0.0;

                  }
                  else if(normalizedEMTorque < 0.8 &&  normalizedEMTorque >
0.4)
                  {
                        normalizedEMTorque = 0.40;
                  }
                  else if(normalizedEMTorque < 0.4 && normalizedEMTorque >
0.0)
                  {
                        normalizedEMTorque = 0.2;
                  }


                  SetNormalizedEMTorque(normalizedEMTorque);


                  SetSharedCustomMemory(1, normalizedEMTorque);

                  break;

            //Dual Power mode operation, both ICE and EM
            //Switches C, D == TRUE, G == FALSE


          case ICEANDEM_PROP:


                  //No need for choke with warm engine.

                  SetPWMDutyCycle(ICEChokeServo, 0.0);

                  SetDigitalOutput(PropulsionDCDCOnOffPort,
PropulsionDCDCOnOffPin, TRUE);

                  SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, FALSE);
                  SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 0.0);

                  //double torqueRequest = GetTorqueRequest();

                  /*** Basic Program, runs the engine up to IOL and then uses
the electric motor for the remaining
                   ***/
```

```
                if(GetTorqueRequest() < (GetICEIOLTorque() * 1.05))   //
Add 10% band
                {
                        double normalizedICETorque = GetTorqueRequest() /
GetMaxICETorque();

                        // Saturate signal 0.0 - 1.0
                        if(normalizedICETorque > 1.0)
                        {
                                normalizedICETorque = 1.0;
                        }
                        else if(normalizedICETorque < 0.0)
                        {
                                normalizedICETorque = IdleThrottle;
                        }

                        SetPWMDutyCycle(ICEThrottleServo,
normalizedICETorque);

                        SetSharedCustomMemory(1, GetTorqueRequest());
                        SetSharedCustomMemory(2, GetICEIOLTorque());
                        SetSharedCustomMemory(3, GetMaxICETorque());
                        SetSharedCustomMemory(4, normalizedICETorque);
                }
                else
                {
                        double normalizedIOLTorque = GetICEIOLTorque() /
GetMaxICETorque();

                        // Saturate signal 0.0 - 1.0
                        if(normalizedIOLTorque > 1.0)
                        {
                                normalizedIOLTorque = 1.0;
                        }
                        else if(normalizedIOLTorque < IdleThrottle)
                        {
                                normalizedIOLTorque = IdleThrottle;
                        }
                        SetPWMDutyCycle(ICEThrottleServo,
normalizedIOLTorque);   // Set engine to IOL
                        //double remainingTorque = GetTorqueRequest() -
GetICEIOLTorque();
                        double remainingTorque = GetTorqueRequest() -
normalizedIOLTorque;

                        double normalizedEMTorque = remainingTorque /
GetMaxEMTorque();

                        // Saturate signal 0.0 - 1.0
                        if(normalizedEMTorque > 1.0)
                        {
                                normalizedEMTorque = 1.0;
                        }
                        else if(normalizedEMTorque < 0.0)
                        {
```

```
                        normalizedEMTorque = 0.0;
                  }

                  SetNormalizedEMTorque(normalizedEMTorque);
//                SetSevenSegmentFloat(normalizedEMTorque);
                  if(remainingTorque > 0.75*GetTorqueRequest())
                  {
                        SetPWMDutyCycle(ICEThrottleServo,
normalizedIOLTorque+0.1);

                  }

                  SetSharedCustomMemory(1, GetTorqueRequest());
                  SetSharedCustomMemory(2, GetMaxICETorque());

                  SetSharedCustomMemory(4, normalizedIOLTorque);
                  SetSharedCustomMemory(5, normalizedEMTorque);
            }


            /*****Advanced Program, runs a check on the torque request
to see if the engine is providing the correct
            torque, and if there is another location where this torque
can be provided.
            *****/
                  //double normalizedICETorque = torqueRequest /
GetMaxICETorque();

            /***

                  //Run through function to find torque points and
return throttle setting
                  //TODO: Debug mode, change optimalthrottle
                  double OptimalThrottle =
GetThrottleSetting(GetTorqueRequest());
                  // Saturate signal 0.0 - 1.0
                  if(OptimalThrottle > 1.0)
                  {
                        OptimalThrottle = 1.0;
                  }
                  else if(OptimalThrottle < IdleThrottle)
                  {
                        OptimalThrottle = IdleThrottle;
                  }
                  SetPWMDutyCycle(ICEThrottleServo, OptimalThrottle);
                  //SetSevenSegmentFloat(0.0);



                  //double remainingTorque = GetTorqueRequest() -
GetICEIOLTorque();
                  double remainingTorque = GetTorqueRequest() -
normalizedICETorque;

                  double normalizedmotorTorque = remainingTorque /
GetMaxEMTorque();
```

```
                        // Saturate signal 0.0 - 1.0
                        if(normalizedmotorTorque > 1.0)
                        {
                              normalizedmotorTorque = 1.0;
                        }
                        else if(normalizedmotorTorque < 0.0)
                        {
                              normalizedmotorTorque = 0.0;
                        }

                        SetNormalizedEMTorque(normalizedmotorTorque);
//                      SetSevenSegmentFloat(normalizedmotorTorque);
                                    ***/


                  break;

            case EMONLYREGENBRAKE_PROP:
                  //Unused

                  break;

            case ICEONLYGENERATION_PROP:
                  //Switch C, D, G == True

                  SetPWMDutyCycle(ICEChokeServo, 0.0);

                  SetDigitalOutput(PropulsionDCDCOnOffPort,
PropulsionDCDCOnOffPin, FALSE);

                  SetDigitalOutput(ClutchPort, ClutchPin, TRUE);

                  SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, TRUE);


                  double batteryVoltage = GetBatteryVoltage(ANPORT10);

                  //Do not charge if battery voltage drops below the 3.0 V
min.
                  if (batteryVoltage < 21.0)
                  {
                        propulsionState = ICEONLY_PROP;
                        SetPWMDutyCycle(ICEThrottleServo, 0.5);
                  }
                  //Stop charging if battery temperature climbs too high
                  if(batteryTemp > 50.0)
                  {
                        propulsionState = ICEONLY_PROP;
                        SetPWMDutyCycle(ICEThrottleServo, 1.0);
                  }


                  //Initial Charging
```

```
                  //Constant Current Charge

                  if(batteryVoltage < 22.0)
                  {
                        SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 3.0);
                  }
                  else if(batteryVoltage < 24.0 && batteryVoltage > 22.0)
                  {
                        SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 1.8);
                  }
                  else if(batteryVoltage < 27.5 && batteryVoltage > 24.0)
                  {
                        SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 1.0);
                  }

                  //Constant Voltage Charge
                  if(batteryVoltage > 27.5 && batteryVoltage < 29.0)
                  {
                        SetOutputVoltageNQ40(GenerationDCDCOutputVolt, 29.1);
                        SetCurrentLimitNQ40(GenerationDCDCCurrentLimit, 1.0);
                  }


                  //Charge cycle complete, discontinue charging
                  if(batteryVoltage > 29.0)
                  {
                        SetDigitalOutput(GenerationDCDCOnOffPort,
GenerationDCDCOnOffPin, FALSE);
                  }

                  //Set ICE to react to throttle input, but add a torque band
to compensate for EM draw
                  //double normalizedICETorque = GetTorqueRequest() /
GetMaxICETorque();

                  // Saturate signal 0.0 - 1.0
                  if(normalizedICETorque > 1.0)
                  {
                        normalizedICETorque = 1.0;
                  }
                  else if(normalizedICETorque <= IdleThrottle)
                  {
                        normalizedICETorque = IdleThrottle;
                  }

                  SetPWMDutyCycle(ICEThrottleServo, normalizedICETorque+0.1);



                  break;
      }

      BOOL ICEOnlyMode = GetDigitalInput(ICEOnlySwitchPort,
ICEOnlySwitchPin);
      BOOL EMOnlyMode = GetDigitalInput(EMOnlySwitchPort, EMOnlySwitchPin);
      BOOL DualMode = GetDigitalInput(DualModeSwitchPort, DualModeSwitchPin);
```

```
	//Allows transmitter to manually control propulsion states
	//See header file for channel and switch listings
	if(!GetDigitalInput(RunKillPort, RunKillPin))
	{
		propulsionState = RESET_PROP;
	}
	else if(ICEOnlyMode == TRUE  && EMOnlyMode == FALSE && DualMode ==
FALSE)
	{
		propulsionState = ICEONLY_PROP;
	}
	else if(EMOnlyMode == TRUE && ICEOnlyMode == FALSE && DualMode ==
FALSE)
	{
		propulsionState = EMONLY_PROP;
	}
	else if(ICEOnlyMode == TRUE && EMOnlyMode == TRUE && DualMode == FALSE)
	{
		propulsionState == ICEANDEM_PROP;
	}
	else if(ICEOnlyMode == TRUE && EMOnlyMode == TRUE && DualMode == TRUE)
	{
		propulsionState == ICEONLYGENERATION_PROP;
	}
	//Defaults to Engine Only mode
	else
	{
		propulsionState = ICEONLY_PROP;
	}

}

double GetTotalAvailableTorque()
{
	return MaxEMTorque + GetMaxICETorque();
}

double GetMaxEMTorque()
{
	return MaxEMTorque;
}

double GetMaxICETorque()
{

	int currentICERPM = GetGenericRPM(ICESpeedPort, ICECountsPerRev);

	return InterpolateVector1D(ICEMapYValues, ICEMapLength, ICEMapXStart,
ICEMapXStep, currentICERPM);
}

double GetICEIOLTorque()
{
	int currentICERPM = GetGenericRPM(ICESpeedPort, ICECountsPerRev);
```

```
	return InterpolateVector1D(IOLMapYValues, IOLMapLength, IOLMapXStart,
IOLMapXStep, currentICERPM);
}

double GetTorqueRequest()
{


	double throttleSetting = GetRCDutyCycle(AutopilotThrottle); // Will
return 0-1.0

	double totalTorque = GetTotalAvailableTorque();

	return throttleSetting * totalTorque;
}

void SetNormalizedEMTorque(double normalizedTorque)
{
	// De-normalize torque value
	double requestedTorque = normalizedTorque * MaxEMTorque;
	double currentForTorque = requestedTorque / EMTorqueConstant;

// Torque/current control of brushed motor through current limited DC-DC
converter
// Set voltage to max, and limit current.
// The output voltage will then drop to whatever voltage will draw the
limited
// current from the motor
// 40V max

	SetOutputVoltageNQ40(PropulsionDCDCOutputVolt, 40.0);

	SetCurrentLimitNQ40(PropulsionDCDCCurrentLimit,currentForTorque);
	// Current control
	SetSharedCustomMemory(2, currentForTorque);


}


double InterpolateVector1D(double yValues[], int length, double xStart,
double xStep, double xInput)
{
	// Find Nearest X value to xInput
	int closestXIndex = 0;
	double closestDifference = 999999999999.9;
	int i;
	for(i = 0; i < length; i++)
	{
		double nextIndexValue = i * xStep + xStart;
		double nextDifference = xInput - nextIndexValue;
		if(nextDifference < 0)		// Absolute value for comparision
		{
			nextDifference *= -1;
		}
```

```
            if(nextDifference < closestDifference)
            {
                  closestXIndex = i;
                  closestDifference = nextDifference;
            }
      }

      // Select the indexes above and below xInput
      int xIndex1 = 0;
      int xIndex2 = 0;
      if((xInput - (closestXIndex * xStep + xStart)) > 0)
      {
            xIndex1 = closestXIndex;
      }
      else
      {
            xIndex1 = closestXIndex - 1;
      }
      xIndex2 = xIndex1 + 1;

      // Check if either index are outside the array
      // if so return the closest known yValue
      if(xIndex1 < 0) // xInput was less than xStart
      {
            return yValues[0];
      }
      else if (xIndex2 >= length)   // xInput was greater than the largest
known xValue
      {
            return yValues[length-1];
      }

      // Linearly interpolate the yValue that cooresponds with the given
xInput

      // Get slope between nearest two points
      // m = (y2 - y1) / (x2 - x1)
      double y1 = yValues[xIndex1];
      double y2 = yValues[xIndex2];
      double x1 = xIndex1 * xStep + xStart;
      double x2 = xIndex2 * xStep + xStart;
      double m = (y2 - y1) / (x2 - x1);

      // y = m(x - x1) + y1
      return (m*(xInput - x1)) + y1;
}

double GetThrottleSetting(double torqueCommand)
{
      int i;
      int j;
      BOOL Foundcommand = FALSE;
      int indice1;
      int indice2;
      double difference = 0.1;
      double previousmin = 99999.99999;
```

```
	for (j = 0; j<EngineMapWidth; j++)
	{
		for (i = 0; i<EngineMapLength; i++)
		{
			double x = fabs(Fuji34TorqueMap[i][j] - torqueCommand);
		if (x<difference)
			{
				double current = Fuji34FuelUseMap[i][j];

				if (current < previousmin)
				{
					indice1 = i;
					indice2 = j;
					Foundcommand = TRUE;
					current = previousmin;
				}
			}

		}

	}

	if(Foundcommand == FALSE)
	{
		indice1 = 4;
		indice2 = 4;
	}

	double BestThrottleSetting =  Fuji34ThrottleMap[indice1][indice2];

	return BestThrottleSetting;


}
```

Figure A-1: Main HE controller code

```
/*****************************************************************************
*****************************
FileName:          HybridPropulsionControl.h
Author:            Collin Greiser & John Hagen
Project:           Hybrid Electric UAV
*****************************************************************************
***************************/

#ifndef HYBRIDPROPULSIONCONTROL_H
#define HYBRIDPROPULSIONCONTROL_H

/** PUBLIC PROTOTYPES *************************************************/
void PropulsionControlStateMachine();
void ConfigureHybridController();
double InterpolateVector1D(double yValues[], int length, double xStart,
double xStep, double inputX);
/** PUBLIC PROTOTYPES *************************************************/

/***** NUMERICAL CONSTANTS*******************************************/
#define ClutchPort                        DIO1PORT    // E5
```

```
#define ClutchPin                           DIO1PIN
#define PropulsionDCDCOnOffPort             DIO2PORT    // G15
#define PropulsionDCDCOnOffPin              DIO2PIN
#define GenerationDCDCOnOffPort             DIO3PORT    // E4
#define GenerationDCDCOnOffPin              DIO3PIN
#define RunKillPort                         DIO4PORT    // G13; TRUE = Run,
FALSE = Kill
#define RunKillPin                          DIO4PIN
#define TakeoffIdlePort                     DIO5PORT    // G12;     TRUE =
Takeoff, FALSE = Idle
#define TakeoffIdlePin                      DIO5PIN
#define ICEOnlySwitchPort                    DIO6PORT   //Channel 4, SG
#define ICEOnlySwitchPin                    DIO6PIN
#define EMOnlySwitchPort                    DIO7PORT    //Channel 5, SD
#define EMOnlySwitchPin                     DIO7PIN
#define DualModeSwitchPort                  DIO8PORT    //Channel 6, SC
#define DualModeSwitchPin                   DIO8PIN

#define     ICEThrottleServo                OCPORT1     // D0, Channel 3, J3
#define     ICEChokeServo                   OCPORT2     // D1, Channel 1, SA

#define PropulsionDCDCOutputVolt            AOPORT4     // DAC1 VOUTA
#define PropulsionDCDCCurrentLimit          AOPORT1     // DAC1 VOUTB
#define     GenerationDCDCCurrentLimit      AOPORT2     // DAC2 VOUTA
#define GenerationDCDCOutputVolt            AOPORT3

#define ICESpeedPort                        INTPORT1    // E8
#define EMSpeedPort                         INTPORT2    // E9

#define AutopilotThrottle                   ICPORT1     // D8, Channel 3, J3


/***** NUMERICAL CONSTANTS************************************************/

#endif // HYBRIDPROPULSIONCONTROL_H
```

Figure A-2: HE control header code

```
%M-File for the subplotting of controller data runs
%Collin M. Greiser
%Master's Thesis, February 2011
%Wright-Patterson Air Force Base, Ohio


%Plot All Test Data

clear all; close all;
clc
%Read in raw data file from MS Excel
%User inputs data run number and test type
disp('Enter 1 for Engine Only, Honda');
disp('Enter 2 for Engine Only, Fuji 25');
disp('Enter 3 for Motor Only');
disp('Enter 4 for Dual Mode');
```

```
disp('Enter 5 for Regen Mode');
type = input('Enter the test type');

num = input('Enter the data run number');
%Read ICE Only Data
if type == 1;
    ICE = xlsread(['honda run',num2str(num),'.xlsx']);
elseif type == 2;
    ICE = xlsread(['fuji run',num2str(num),'.xlsx']);
%Read  Motor Data
elseif type == 3;
    MO = xlsread(['motor run',num2str(num),'.xlsx']);
%Read  Dual Mode Data
elseif type == 4;
    DUAL = xlsread(['dualmode',num2str(num),'.xlsx']);
%Read  Regen Data
else
    REGEN = xlsread(['regen mode',num2str(num),'.xlsx']);
end

%Subplot Engine Speed, Torque, and Throttle Commands
if type == 1 || type == 2;

    figure;
    subplot(3,1,1)
    plot(ICE(:,1),ICE(:,2));
    ylabel('Engine Speed (RPM)');
    axis([min(ICE(:,1)) max(ICE(:,1)) 0 9000]);

    subplot(3,1,2)
    plot(ICE(:,1), ICE(:,3));
    ylabel('Torque (N-m)');
    axis([min(ICE(:,1)) max(ICE(:,1)) 0 1]);
    hold on;
    plot([0 max(ICE(:,1))], [0.66 0.66],'-k')
    hold off;
    legend('Engine Torque','required torque','Location','Best');

    subplot(3,1,3)
    plot(ICE(:,1), ICE(:,4));
    ylabel('Throttle Position Command');
    axis([min(ICE(:,1)) max(ICE(:,1)) 0 1]);
    xlabel('Time (s)');
elseif type == 3;
%Subplot Motor Data
%Scale Data to adjust for time indifference
m = find(MO(:,4)==min(MO(:,4)));
m2 = find(m(:,1)==min(m));
m3 = m(m2,1);
n = find(MO(:,3)==min(MO(:,3)));
p = MO(m3,1);
q = MO(n,1);
s = abs(p-q);
x2 = MO(:,1)+s;
figure;
```

```
subplot(3,1,1)
    plot(MO(:,1),MO(:,2));
    ylabel('Motor Speed (RPM)');
    axis([min(MO(:,1)) max(MO(:,1)) 0 9000]);

    subplot(3,1,2)
    plot(x2, MO(:,3));
    ylabel('Torque (N-m)');
    axis([min(MO(:,1)) max(MO(:,1)) 0 0.7]);
    hold on;
    plot([0 max(x2)], [0.27 0.27],'-k')
    hold off;
    legend('Motor Torque','Required Torque','Location','Best');

    subplot(3,1,3)
    plot(MO(:,1), MO(:,4));
    ylabel('Throttle Position Command');
    axis([min(MO(:,1)) max(MO(:,1)) 0 1]);
    xlabel('Time (s)');
elseif type == 4;
figure;
    subplot(3,1,1)
    plot(DUAL(:,1),DUAL(:,2));
    ylabel('Requested Torque (N-m)')
    axis([min(DUAL(:,1)) max(DUAL(:,1)) 0 2]);

    subplot(3,1,2)
    plot(DUAL(:,1), DUAL(:,3));
    ylabel('Theoretical Engine Command');
    axis([min(DUAL(:,1)) max(DUAL(:,1)) 0 1]);

    subplot(3,1,3)
    plot(DUAL(:,1), DUAL(:,4));
    ylabel('Theoretical Motor Command');
    xlabel('Time (s)');
    axis([min(DUAL(:,1)) max(DUAL(:,1)) 0 1]);

end
```

Figure A-3: MATLAB plotting code

Appendix B: Controller Flowcharts

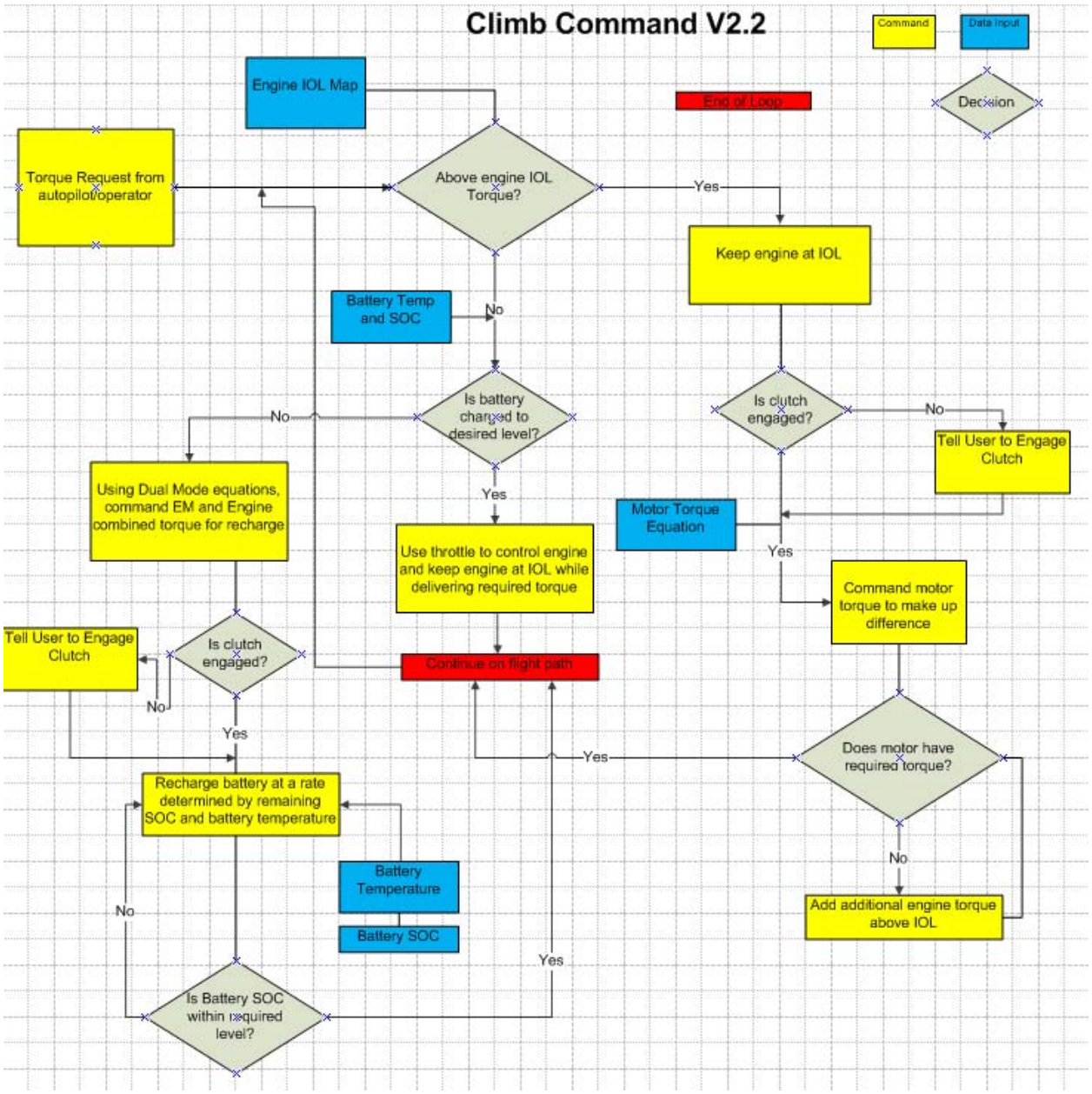


Figure B-1: Climb and cruise with regeneration flowchart

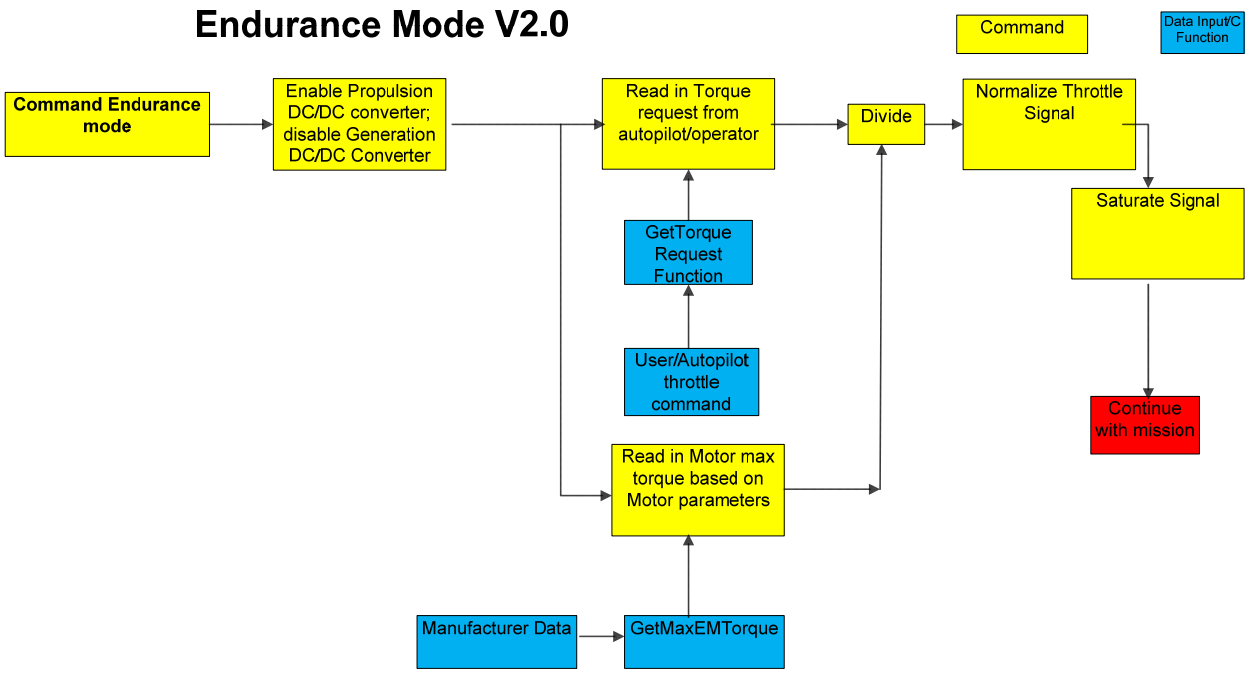


Figure B-2: Endurance mode flowchart

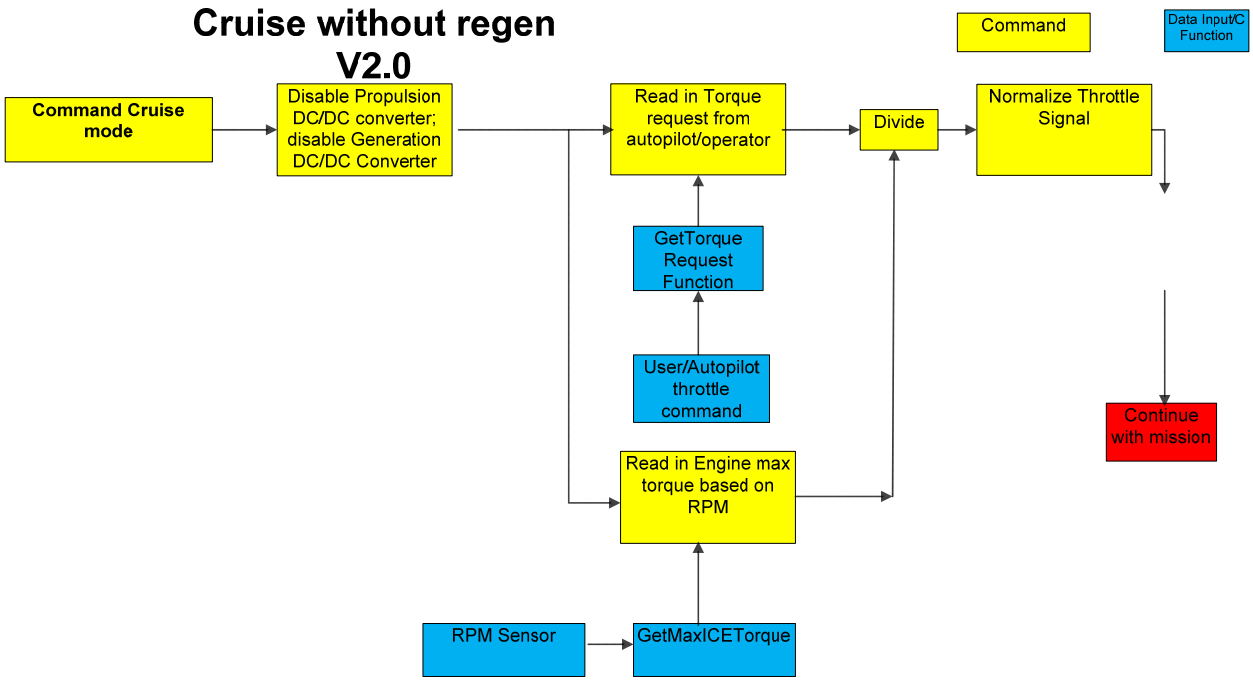


Figure B-3: Cruise without regeneration flowchart

Appendix C: Example Test Matrix

Table 8: ICE Only (Cruise) Test Matrix

Engine Only Mode Test (Cruise)

Futaba Remote Settings

Switches Engaged	Switches Disengaged
"A"	**"B", "C", "H"**
(Either Direction Engages)	(Center Position Disengages)

(Note: Choke and clutch are controlled automatically, "H" Will override choke command and "C" disables controller)

(Note #2: Switching all switches to the center position will put the controller in "Reset", disabling everything)

Controller Settings

Propulsion DCDC	Off
Generation DCDC	Off

Land and Sea Dynamometer Settings

Dynamometer Main Power	On
Load Cell Switch	Manual
Load Knob	No Load/Variable

LabView Panel Settings

All Data Collection Switches	On

Measured Parameters

Engine Speed (RPM)	Throttle Position (0-1)	Motor Speed (RPM)	Battery Voltage (V)	Battery Current (A)	Engine Temp (^{o}F)	Motor Temp (^{o}F)	Clutch Engagement (On-Off)	Choke Engagement (On-Half-Off)	Shaft Torque (N-m)
Idle									
2000									
2500									
3000									
3500									
4000									
4500									
5000									
5500									
6000									
6500									
7000									
7500									

Data File Name:

Table 9: Endurance Mode Test Matrix

Motor Only Mode Test (Endurance)

Futaba Remote Settings

Switches Engaged	Switches Disengaged
"B"	**"A", "C", "H"**
(Either Direction Engages)	(Center Position Disengages)

(Note: Choke and clutch are controlled automatically, "H" Will override choke command and "C" disables controller)

(Note #2: Switching all switches to the center position will put the controller in "Reset", disabling everything)

Controller Settings

Propulsion DCDC	Engaged
Generation DCDC	Off

Land and Sea Dyno Settings

Dyno Main Power	On
Load Cell Switch	Manual
Load Knob	No Load/Variable

LabVIEW Panel Settings

All Data Collection Switches	On

Measured Parameters

Engine Speed (RPM)	Throttle Position (0-1)	Motor Speed (RPM)	Battery Voltage (V)	Battery Current (A)	Engine Temp (^{o}F)	Motor Temp (^{o}F)	Clutch Engagement (On-Off)	Choke Engagement (On-Half-Off)	Shaft Torque (N-m)
Idle									
2000									
2500									
3000									
3500									
4000									
4500									
5000									
5500									
6000									
6500									
7000									
7500									

Data File Name:

Appendix D: Example SOP

Standard Operating Procedure for HE-RPA Controller Testing

Developed by Collin Greiser

4 JAN 2011

Cruise Without Regeneration Testing

Initial Setup (Dyno)

1. Ensure fuel tank is full with fresh gasoline (red container) and the bolts on the dynamometer are all tight and secure.
2. Ensure belt tensioner is in place and tight. Belt WILL break if the tensioner is not secure!
3. Prime the engine by pushing the fuel bulb until fuel fills the carburetor (Honda) or by turning the engine over until fuel is drawn into the carburetor (Fuji engines).
4. Set the choke to closed (gray lever on Honda, servo on Fuji).
5. Connect the 12V battery to the starter motor.

Initial Setup (Computer and Controller)

1. Power on the controller by flipping the switch on the main board to “USB.” All five lights on the board should light up and remain solid.
2. Ensure that optical sensors have power and their beams are pointed correctly.
3. On the computer, bring up the MPLAB main screen. Select “build all” to build the most recent version of the controller code. Once complete, select “Program.” The controller lights will dim, except for the blue light. Once complete, the controller is now programmed and ready for use.
4. On the computer, bring the main LabView and dynamometer screens up.
5. Ensure nothing is touching the dynamometer, and click “zero” to zero out and re-calibrate the torque sensor. The torque reading should now be very close to zero.
6. Turn on the transmitter and ensure battery voltage is above 7.0V. The LED on the receiver should now be green. Make sure the throttle stick is at zero (down).

Engine Warm Up

1. With the starter motor, place the motor as snugly against the propeller cone on the back of the engine shaft as possible.
2. Press the bottom of the starter pad to engage the starter. Spin the engine for no more than a maximum of 5 seconds. Refer to individual manuals for complete engine starting procedures.
3. Once the engine is running, allow idling for approximately one minute, then open the choke fully.
4. If the engine stalls, repeat steps 1-3.
5. Once the engine is running smoothly, disconnect the battery and move it out of the way.
6. Allow engine to warm up to operating temperature.

Testing

1. Give the throttle on the transmitter a quick bump to ensure engine operability.
2. Adjust the load and throttle setting to the desired point, then shut off the engine.
3. Zero the dynamometer.
4. Re-start the engine; data is now ready to be collected.
5. During testing, monitor the throttle output on the LabView screen. If the throttle appears "stuck" at 100%, disable the controller by clicking "build all" on the MPLAB screen. This will reset the throttle to zero.
6. To start collecting data, click "record" on the dynamometer screen and "run continuously" on the LabView screen. MATLAB will handle the time differences.
7. Power-down is reverse of start-up.

Standard Operating Procedure for HE-RPA Controller Testing

Developed by Collin Greiser

4 JAN 2011

Endurance Testing

Initial Setup (Dyno)

1. Ensure DC power supply is on. Turn the right-most knob to increase the supply voltage to exactly 26.9 volts.
2. Ensure belt tensioner is in place and tight. Belt WILL break if the tensioner is not secure!

3. Place the air hose so that air flows directly over the DC/DC converter. Turn on the air supply. If the converter is not cooled it WILL overheat.
4. Ensure the engine choke is closed and the engine ignition is disabled to prevent any accidental start-ups.

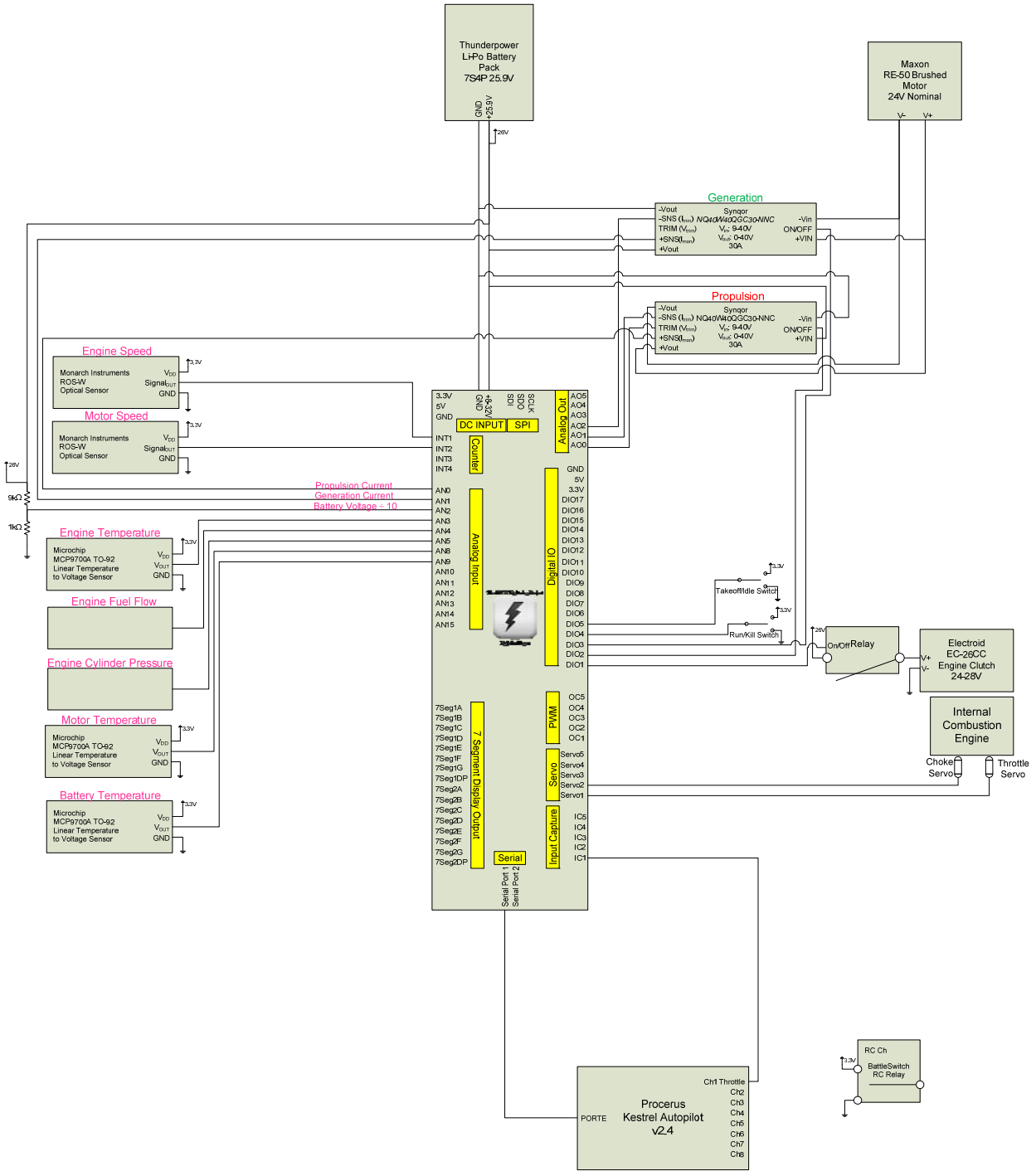
Initial Setup (Computer and Controller)

1. Power on the controller by flipping the switch on the main board to "USB." All five lights on the board should light up and remain solid. The motor will give a quick "blip." This is normal; it is due to small current being passed through the DC/DC converter on power up.
2. Ensure that optical sensors have power and their beams are pointed correctly.
3. On the computer, bring up the MPLAB main screen. Select "build all" to build the most recent version of the controller code. Once complete, select "Program." The controller lights will dim, except for the blue light. Once complete, the controller is now programmed and ready for use.
4. On the computer, bring the main LabView and dynamometer screens up.
5. Ensure nothing is touching the dynamometer, and click "zero" to zero out and re-calibrate the torque sensor. The torque reading should now be very close to zero.
6. Turn on the transmitter and ensure battery voltage is above 7.0V. The LED on the receiver should now be green. Make sure the throttle stick is at zero (down).
7. Turn on both multimeters. The parallel multimeter should read zero. The series multimeter will have zero current at this point.

Testing

1. Give the throttle on the transmitter a quick bump to ensure motor operability.
2. Adjust the load and throttle setting to the desired point, then stop the motor. The current multimeter will read approximately 0.3 A. This is NORMAL as the motor and DC/DC converter both have no-load currents associated with them.
3. Zero the dynamometer.
4. Data is now ready to be collected.
5. During testing, monitor the throttle output on the LabView screen. If the throttle appears "stuck" at 100%, disable the controller by clicking "build all" on the MPLAB screen. This will reset the throttle to zero.
6. To start collecting data, click "record" on the dynamometer screen and "run continuously" on the LabView screen. MATLAB will handle the time differences.
7. The one-way bearing will be noisy; this is normal.
8. Power-down is reverse of start-up.

Appendix E: Controller Wiring Diagram [54]



REPORT DOCUMENTATION PAGE		*Form Approved* *OMB No. 0704–0188*

1. REPORT DATE (DD–MM–YYYY)	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24 03 2011	Master's Thesis	Sept 2009-Mar 2011

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Implementation of an Open-Loop Rule-Based Control Strategy for a Hybrid-Electric Propulsion System On a Small RPA	5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Greiser, Collin M., 2d Lt, USAF	5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/ENY) 2950 Hobson Way, Building 640 WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GA/ENY/11-M05
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Ben Razidlo (Benjamin.Razidlo@wpafb.af.mil) Air Force Research Laboratory 1950 Fifth Street WPAFB, OH 45433-7251	10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RZPG 11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES
This material is declared work of the U.S. government and is not subject to copyright protection in the United States.

14. ABSTRACT

Currently fielded electric-powered small remotely-piloted aircraft (RPA) lack endurance desired by warfighters, and internal combustion engine (ICE) RPAs generate undesirable acoustic and thermal signatures. Hybrid-electric (HE) propulsion systems would combine both electric power for endurance and ICE power for cruise and climb modes. Use of HE systems would eliminate undesirable signatures in addition to providing considerable fuel savings over time. Five components were used in this HE system: the ICE, electric motor (EM), electromagnetic clutch, battery pack, and a propeller. Control of such a system in a small RPA has never been attempted before. A rule-based controller was developed to manage this HE system in C code. This system and its various sensors were analyzed on a custom-built dynamometer test stand that was developed in conjunction with other students. LabView screens were developed to aid this testing and interface with the sensor suite. The controller's performance over 9 distinct operating modes, including 4 operational flying states, were validated to provide the most optimal operation of a HE-RPA system of about 13.6 kg (30.0 lbf).

15. SUBJECT TERMS
Hybrid-Electric, Propulsion, Remotely-Piloted, Control, Rule-Based

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON .Frederick G. Harmon, Lt Col, USAF
a. REPORT	b. ABSTRACT	c. THIS PAGE	UU	168	19b. TELEPHONE NUMBER (Include Area Code) (937)786-3636 x7478 Email: (Frederick.harmon@afit.edu)
U	U	U			

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18