**MODULAR LASER COMBAT SYSTEM FOR REMOTELY OPERATED VEHICLES:
BRIDGING THE GAP BETWEEN COMPUTER SIMULATION AND LIVE FIRE**

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Thomas Edward Fulenwider

March 2010

© 2010

Thomas Edward Fulenwider

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Modular Laser Combat System for Remotely Operated
Vehicles: Bridging the Gap Between Computer Simulation
and Live Fire

AUTHOR: Thomas Edward Fulenwider

DATE SUBMITTED: June 2010

COMMITTEE CHAIR: John Saghri

COMMITTEE MEMBER: Jeff Gerfen

COMMITTEE MEMBER: Tina Smilkstein

ABSTRACT

Modular Laser Combat System for Remotely Operated Vehicles:

Bridging the Gap Between Computer Simulation and Live Fire

Thomas Edward Fulenwider

In the emerging industry of small unmanned vehicles, pioneered by small businesses and research institutions, a suitable combat system test platform is needed. Computer simulations are useful, but do not provide the definitive proof of effective operation necessary for deployment of a combat system. What is needed is an affordable simulated weapons system that enables live flight testing without the used of live weaponry.

A framework is developed here for the construction of a simulated weapon using Free Space Optical (FSO) infrared communication. It is developed in such a way to ensure compatibility with a variety of platforms including ground and aerial vehicles, so that identical but configurable modules can be used on any vehicle that is to take place in a live combat simulation. A proof-of-concept implementation of this modular laser combat system framework is also presented and tested. The implemented system shows the value of such a simulated weapons system and future areas of improvement are also explored.

Keywords: unmanned aerial vehicle, UAV, free space optical, FSO, infrared communication, IR, combat simulation.

TABLE OF CONTENTS

<u>LIST OF TABLES</u>	<u>ix</u>
<u>LIST OF FIGURES</u>	<u>x</u>
<u>1. INTRODUCTION</u>	<u>1</u>
<u>2. FREE SPACE OPTICAL COMMUNICATION</u>	<u>3</u>
2.1 Basic FSO Operation	3
2.2 Atmospheric Concerns	4
2.2.1 Absorption	5
2.2.2 Scattering	6
2.2.3 Scintillation	7
2.3 Modeling Atmospheric Transmission	8
2.3.1 Beer-Lambert Law	8
2.3.2 Kruse Relation	8
2.4 Merits and Shortcomings	9
<u>3. MODULAR LASER COMBAT SYSTEM FRAMEWORK</u>	<u>12</u>
3.1 Importance of Modularity	12
3.2 Combat Module	13
3.2.1 Simulated Weapon	13
3.2.2 Hit Detection Sensors	14
3.2.3 Statistical Recording	14
3.2.4 Registered Nonvolatile Configuration	15
3.2.5 Vehicle Communication and Control	15
3.2.6 Basic Add-ons	15
3.3 Command Module	16
3.3.1 Configuration User Interface	16

3.3.2 Review of Combat Scenario Results	16
4. TEST PLATFORMS	18
4.1 Aerial Platforms	18
4.1.1 Aerial Platform Overview	19
4.1.2 TomHe F-22 Raptor	19
4.1.3 Sport Trainer	21
4.1.4 Aerobird Challenger	22
4.1.5 Summary of Flight Characteristics	23
4.2 Ground-based Targeting	25
5. IMPLEMENTATION	27
5.1 Microcontroller	27
5.1.1 C Compiler Selection	28
5.2 Infrared Communication	30
5.2.1 IR Reception	33
5.2.2 IR Transmission	34
5.2.2.1 Optical Transmission	36
5.2.3 The Data “Bullet”	37
5.2.3.1 Bullet Length	38
5.2.3.2 IR Bullet Spot Size	40
5.3 Non-Volatile Configuration and Statistical Recording	42
5.3.1 Storage Structure	42
5.4 Control	44
5.4.1 Generic RC Servo PWM	45
5.4.2 X-Port	46
5.5 Hit Response	47
5.5.1 Motor Manipulation	47
5.5.2 Auxiliary Outputs	48

5.5.2.1 Activator Add-on Module	48
5.6 The Combat Module Device	49
5.7 Command Module	52
5.7.1 User Interface	52
5.7.2 Communication with Combat Modules	54
6. TESTING AND RESULTS	57
6.1 Communication	57
6.1.1 UART	57
6.1.2 Optics	59
6.1.3 Power Supply Interference	61
6.2 Ground Testing	62
6.3 Flight Testing	65
6.3.1 Preliminary Flight Testing	65
6.3.2 Static Vehicle Tests	65
7. CONCLUSIONS AND FUTURE WORK	67
7.1 Enhanced Optical Transmission/Reception	67
7.2 Combat Module Enhancements	69
7.2.1 Pan and Tilt Weapon with Video Targeting	69
7.2.2 Expanded Vehicle Communication Protocols	69
7.3 Command Module Enhancements	70
7.3.1 Non-Proprietary Platform	70
7.3.2 GUI enhancement	70
7.4 Conclusion	71
REFERENCES	72
8. APPENDIX A: COMPUTER SIMULATION BACKGROUND	75
8.1 Value of Computer Modeling	75
8.1.1 Cost	75

8.1.2 Safety	76
8.1.3 Logistic Simplicity	76
8.2 Insufficiency of Computer Modeling	76
8.2.1 The Problem of Incomplete Knowledge	77
8.2.2 Balance of Cost and Accuracy	79
8.2.3 Lack of Proof	80
8.3 Forms of Live Simulation	81
8.3.1 Live Fire	81
8.3.2 Routine Execution	83
8.3.3 Simulated Weapon, Real Operation	83
<u>APPENDIX B: SCHEMATICS</u>	<u>86</u>
<u>APPENDIX C: BOARD LAYOUT</u>	<u>92</u>
<u>APPENDIX D: C-CODE</u>	<u>96</u>

LIST OF TABLES

Table 2.1: IR Absorption due to water molecules [4].	6
Table 2.2: IR Absorption due to carbon dioxide molecules [4].	6
Table 2.3: Equation 2.1 variable definitions [3].	8
Table 2.4: Equation 2.2 variable definitions [3].	8
Table 2.5: Attenuation for different visual range conditions [1].	9
Table 2.6: The pros and cons of an FSOSW.	11
Table 4.1: Basic aerial platform flight characteristics.	24
Table 4.2: Performance related to aerial platform power rating.	24
Table 5.1: Comparison of basic microcontroller specifications.	28
Table 5.2: The data “Bullet” contents.	37
Table 5.3: Combat Module configuration registers.	44

LIST OF FIGURES

Figure 2.1: Example of full-duplex FSO network link [1].	4
Figure 2.2: Solar irradiance at sea level. Courtesy of NASA MODIS.	5
Figure 2.3: Concerns in FSO communication [1].	7
Figure 3.1: High level overview of Modular Laser Combat System.	17
Figure 4.1: TomHe F-22 Raptor aerial test platform.	21
Figure 4.2: KF airfoil used by the F-22 aerial test platform [7].	21
Figure 4.3: Sport Trainer aerial test platform.	22
Figure 4.4: Aerobird Challenger aerial test platform.	23
Figure 4.5: Stationary ground-based turret targeting test platform.	26
Figure 5.1: UART byte structure [8].	31
Figure 5.2: Spectrum spreading due to modulating UART signal.	32
Figure 5.3: A5 IR Sensor Array on Ground Station.	33
Figure 5.4: IR Sensor Array schematic.	34
Figure 5.5: IR LED drive schematic.	35
Figure 5.6: Cross section view of collimating lens tube assembly.	36
Figure 5.7: Relating bullet length to travel time and IR spot size.	40
Figure 5.8: Gaining similar successful hit areas with a single reception point and large spot size.	41
Figure 5.9: RC servo PWM communication [9].	45
Figure 5.10: Standard X-port communication pulse train [10].	47
Figure 5.11: The Activator high current Add-on Module (black stick in photo).	48

Figure 5.12: The Combat Module.	50
Figure 5.13: Most recent version of the circuit board for the Combat Module.	50
Figure 5.14: Combat Module schematic.	51
Figure 5.15: Command Module menu overview.	53
Figure 5.16: The Command Module.	55
Figure 5.17: The Command Module circuit board.	55
Figure 5.18: Command Module schematic.	56
Figure 6.1: PWM output showing approximately 40% duty cycle 40kHz square	58
Figure 6.2: UART transmission sending 0xAA and 0xCC showing PWM and inverted TX	59
Figure 6.3: Camera used for IR focusing with the removed IR blocking filter.	60
Figure 6.4: Relative responsivity compared to modulating frequency.	63
Figure 6.5: Spectral sensitivity of the TFMS5400 sensor.	64
Figure 6.6: Threshold irradiance in the presence of a bright ambient.	65
Figure 8.1: Live fire testing at Nellis AFB in Operation Red Flag [19].	82
Figure 8.2: Dogfight training with simulated weapon, Nellis Air Combat Training System. Top: flight video of dogfight. Middle: animated version of actual fight. Bottom: animated cockpit view showing a successful hit [19].	84

1. INTRODUCTION

Over the past few decades, there has been a strong technological movement away from physical testing of engineering projects and toward computer simulation. This movement has been spurred on by increasing computational capabilities and the attraction of low-cost implementation that computer simulation allows. Many companies and institutions rely solely on computer simulation to test a theory or product.

The sole reliance on computer simulation is simply not good engineering. Engineers must make products and create services for the real, physical world. While computer simulation is valuable and effective when used correctly, some form of physical testing should generally take place to ensure that a safe and reliable product is produced. Computer simulation can reduce the amount of physical testing processes, but it should not be expected to replace them entirely.

Safety and reliability are paramount in the design and production of unmanned and remotely operated vehicles. Unmanned vehicles are now used heavily for tactical military operations, both in reconnaissance and combat, as well as in the civil service sector by police and firefighters for bomb handling, tactical entry, and search and rescue. In any case, the performance of an unmanned system in these industries can determine the outcome of life and death situations. Reliability and repeatability are essential. These qualities in a product can only be maintained through effective and thorough testing, both in the virtual and physical worlds.

An area in desperate need of some method of physical testing is weapons and targeting systems for unmanned vehicles. These vehicles are being produced in greater number by private companies and research institutions without the funding or the logistic means to conduct regular live-fire testing. These systems must be tested physically and not just virtually to ensure reliability.

The proposed method of testing these unmanned weapons systems is a hybrid live and simulated approach. This Modular Laser Combat System used on an actual unmanned vehicle will provide a simulated weapon on a physical vehicle. By offering designers a tool to run physical tests while using a simulated weapon, design costs may be reduced, minimize testing collateral, and offer real life proof of accurate operation.

2. FREE SPACE OPTICAL COMMUNICATION

Free space optical (FSO) communication offers a very attractive means of wireless and fiberless communicating. This form of communication is under no circumstances, a new idea. Communication through the atmosphere via smoke signals, sea lanterns, and the heliograph date back centuries. Also, Alexander Graham Bell transmitted his voice through 183 meters of air using what he named the photophone, which used a reflected beam of sunlight [1]. Despite its long history, FSO communication is still an area of continuing research.

2.1 Basic FSO Operation

FSO communication uses a focused beam of invisible infrared light to transmit information. Data is translated by the transmitter into pulses of light that are beamed through the air to the receiver. The receiver collects the light and decodes the pulses into data again. See Figure 2.1 for a simplified FSO link example.

Figure 2.1: Example of full-duplex FSO network link [1].

“Optical wireless communications is becoming an attractive alternative medium to optical fiber, and radio frequency communications, because of its high bandwidth, low cost, ease of implementation, license free spectrum and freedom from interference” [2]. One of the most common uses of FSO is for “last mile” high speed network links. In these systems, a hard wired access point can deliver network access to nearby homes and office building without the need for expensive wire or fiber routing. It is especially useful for communication between large office building within line-of-sight since the laser transceivers can be set up in windows of the office buildings.

2.2 Atmospheric Concerns

FSO communication is actually a misnomer since it implies that the laser transmission is occurring in free space [1]. In actuality, transmission occurs through the atmosphere, which is far from a perfect optical transmission medium. Atmospheric transmission of light suffers from selective absorption, scattering, an scintillation [3].

2.2.1 Absorption

Atmospheric absorption occurs when incident photons interact with atmospheric atoms, molecules, and aerosols. This interaction destroys the photon and converts its energy to heat, causing a elevation in temperature [3]. In the infrared electromagnetic band, the molecules primarily responsible for atmospheric absorption are water and carbon dioxide. As can be seen in Figure 2.2 and Table 2.1, water molecules account for absorption bands at 940nm, 1.1um, and 1.375um in the near IR band. Table 2.2 shows that carbon dioxide is responsible for a weak absorption band at 1.4um.

Absorption can also occur due to aerosols suspended in the air. Aerosols are any minute solids or liquids that are present in the air due to very low gravitationally induced fall speeds. These can be anything ranging from smoke, dust, and ice and is between 10nm to 100um in size [3].

Figure 2.2: Solar irradiance at sea level. Courtesy of NASA MODIS.

Table 2.1: IR Absorption due to water molecules [4].

Water Absorption Bands in Infrared Bands	
Near IR	0.94 μm
	1.1 μm
	1.38 μm
Mid IR	1.87 μm
	2.7 μm
Thermal IR	3.2 μm
	6.3 μm

Table 2.2: IR Absorption due to carbon dioxide molecules [4].

Carbon Dioxide Absorption Bands in Infrared Bands	
Near IR	1.4 μm (weak)
Mid IR	1.6 μm (weak)
	2.0 μm (weak)
	2.7 μm
Thermal IR	4.3 μm
	4.8 μm
	5.2 μm
	15.0 μm

2.2.2 Scattering

Atmospheric scattering due to atoms and molecules occurs when part of the incidence light interacts with the molecules and the angular direction of the light is changed with the possibility of a change in wavelength as well. Scattering from aerosols occurs when the wavelength of the incident light is of the same order of magnitude as the aerosols and the light is refracted in all directions. This type of scattering happens primarily due to mist and fog in the air. It should be noted that, in general, a large number of small particles is more detrimental to an FSO link than a smaller number of large particles [1].

2.2.3 Scintillation

Scintillation is a refractive disruption interference that is caused by changes air temperature over the trajectory of a beam of light. These variations in temperature cause a variation in the refractive index of the local air. This change in refractive index can cause the light to bend unpredictably. Varying temperature gradients along a beam's trajectory can be caused by varying ground surface types along the path. Pavement will cause the air above it to be warm above it during the day. Water may have a cooling effect on the air above it.

For FSO links under a few hundred meters, the effect of scintillation is relatively small compared to the attenuation effects of atom, molecule, and aerosol absorption and scattering [1]. Note the overview of the transmission concerns of FSO communication in Figure 2.3.

Figure 2.3: Concerns in FSO communication [1].

2.3 Modeling Atmospheric Transmission

2.3.1 Beer-Lambert Law

For the purpose of modeling transmission of infrared light through the atmosphere for FSO communication, we use the Beer-Lambert Law:

$$\tau(\lambda, L) = \frac{P(\lambda, L)}{P(\lambda, 0)} = \exp[-\gamma(\lambda) L] \quad (2.1)$$

where:

Table 2.3: Equation 2.1 variable definitions [3].

$\tau(\lambda, L)$	total transmittance of the atmosphere at wavelength λ
$P(\lambda, L)$	signal power at distance L from the transmitter
$P(\lambda, 0)$	emitted power
$\gamma(\lambda)$	attenuation or the total extinction coefficient per unit of length

For communications links under a few hundred meters, we ignore all attenuation effects except for absorption and scattering. The simplified attenuation factor, in general, is then:

$$\gamma(\lambda) = \alpha_m(\lambda) + \alpha_a(\lambda) + \beta_m(\lambda) + \beta_a(\lambda) \quad (2.2)$$

where:

Table 2.4: Equation 2.2 variable definitions [3].

$\alpha_m(\lambda)$	molecular absorption coefficient
$\alpha_a(\lambda)$	aerosol absorption coefficient
$\beta_m(\lambda)$	molecular scattering coefficient
$\beta_a(\lambda)$	aerosol scattering coefficient

2.3.2 Kruse Relation

The attenuation coefficient is further simplified because the analysis of FSO links is most dependent on aerosol scattering attenuation. From this information, and semi-

empirical studies for distances of under 6km, we see that the attenuation factor becomes what is known as the Kruse relation [1]:

$$\gamma(\lambda) = \beta_a(\lambda) = \frac{17.0}{V} \left(\frac{550}{\lambda} \right)^{0.585V^{\frac{1}{3}}} \left[\frac{dB}{km} \right] \quad (2.3)$$

where V is the visibility in kilometers. See Table 2.5 for a qualitative comparison of visual range conditions to attenuation.

Table 2.5: Attenuation for different visual range conditions [1].

Description	Visual range	Loss (dB km ⁻¹)
Dense fog	40–70 m	250–143
Thick fog	70–250 m	143–40
Moderate fog	250–500 m	40–20
Light fog	500–1000 m	20–9.3
Thin fog	1–2 km	9.3–4.0
Haze	2–4 km	4.0–1.6
Light haze	4–10 km	1.6–0.5
Clear	10–25 km	0.5–0.1
Very clear	25–50 km	0.1–0.04
Extremely clear	50–150 km	0.04–0.005

2.4 Merits and Shortcomings

FSO communication offers incredible potential for use not only as a stationary network link, but also as a simulated weapon in an unmanned vehicle or targeting system. By using a laser or near-collimated beam of infrared light, a “bullet” can be fired from a pseudo-weapon and received by the target to record a hit. Thus, an FSO simulated weapon (FSOSW) uses light beams to virtually fire upon its targets.

FSOSW has a major advantage over both radio and ultrasonic simulated weapons in that it can be focused very easily and therefore serves as a better projectile substitute. The equipment for an FSOSW is also generally smaller and lighter than either its radio or ultrasonic siblings. FSOSW's also hold an advantage over radio because they neither cause interference in other systems and cannot be jammed.

An FSOSW is also a valuable projectile substitute because reduces the cost and improves the safety of any targeting system tests that are performed. No special testing grounds are needed to operate FSOSW's. There is no collateral damage on the target or the background because unlike a projectile, an FSOSW is non-destructive.

Unfortunately, an FSOSW is not a perfect option for testing targeting systems because it does not behave exactly as a projectile might. A projectile will travel, at best, in the supersonic range of flight, but a FSO beam will travel at the speed of light. Also, the way a projectile hits a target can greatly influence its effectiveness. Projectiles can deflect or ricochet off a target, they can hit a well-armored place on the target and cause little or no damage. Fundamentally, a light beam cannot strike the sensor on the target with the same effect as a projectile hitting the target.

As described previously, FSO suffers from strong atmospheric condition dependencies. Depending on the weather, the range and effectiveness of a FSOSW can vary significantly. High powered concentrated infrared light beams may need to be transmitted to achieve sufficient range to compensate for atmospheric attenuation. Because of this, FSOSW's can also present an eye safety hazard. This hazard is particularly important for infrared light between 780 and 920nm.

Eye safety is a concern because the cornea is transparent from the near violet to the near infrared range of wavelengths. As a result, the radiation at a wavelength of 875 nm $\pm$ 30 nm used in IrDA devices can cause thermal damage if focused onto the retina. This happens because the eye does not protect itself from damage by closing the iris or closing the eyelid as this wavelength is outside the visible range of light.^[10]

Table 2.6 below summarizes the advantages and disadvantages of using an FSOSW. Despite the problems associated with FSOSW, it offers significant potential as a projectile substitute for the purpose of unmanned vehicle and targeting system testing. Its low cost small characteristics as well as avoiding the need for special testing facilities make real life testing possible for small businesses and research institutions.

Table 2.6: The pros and cons of an FSOSW.

Advantages	Disadvantages
easily focused and directed cannot be jammed does not cause interference smaller and lighter equipment reasonable projectile substitute no special test sight needed low cost	range heavily dependent on weather does not behave exactly like a projectile presents eye safety concerns

3. MODULAR LASER COMBAT SYSTEM FRAMEWORK

It has been shown in Appendix A that computer simulation is insufficient for testing unmanned vehicles and targeting systems. It has also been shown that there is a need for a low-cost simulated weapon for use by small businesses and research institutions. A low-cost modular combat system should be developed. This kind of system either does not exist to date or it is at least not commercially available and is considered the author's intellectual property.

3.1 Importance of Modularity

The word modular is defined as an adjective meaning, “constructed with standardized units or dimensions allowing flexibility and variety in use” [5]. This is incredibly important for a simulated weapon combat system that is to be used in a wide variety of applications and platforms. Perhaps the greatest failure of any previous solutions is that they have been proprietary or exclusive in some way. Whether by

prohibitive cost, proprietary compatibility, or insufficient functionality other systems have not filled this void.

A modular system allows the system to be used on a wide variety of platforms, whether they be unmanned aerial vehicles of all sizes, ground-based targeting systems, or mobile ground unmanned vehicles. If one device could be used on all these platforms, it would reduce the complexity, cost, and time that must be invested in a simulated weapons test. It would enable all vehicles and targets involved in a live combat simulation to communicate with each other and inform each other when they have fire on or hit each other.

Within the system itself, modularity would offer the flexibility to offer different capabilities to each vehicle. By creating a system that is locally configurable, each vehicle's simulated weapons system could behave differently even though it has the same hardware and software platform. Additional hardware modules could be added to further enhance and differentiate vehicles and their weapons capabilities.

3.2 *Combat Module*

The heart of the proposed system is the combat module. The combat module is the hardware device that actually attaches to the vehicle under test. It consists of the simulated weapon, sensors to detect hits, a user-configurable microprocessor, and hardware to communicate with the vehicle to which it is attached.

3.2.1 *Simulated Weapon*

Due to the benefits presented in Chapter 2, the proposed system will use a FSO transmitter to serve as the simulated weapon. It will operate in the near infrared band of

the electromagnetic spectrum. Using this band is important simply because there is already a mature market for infrared communication devices. There are no complex hardware setups that must be researched because most of that work has already been done for FSO network links. Unlike a network link however, the FSOSW is not designed to remain in constant communication with its target. On the contrary, it is design to act more like a projectile and will only communicate with the target if it is aimed to deliver a “hit.” When the FSOSW fires what is called the simulated “bullet,” it will actually transmit an optical data packet containing information about the shooter and the weapon that is being fired. See Figure 3.1 for a high-level overview of the system.

3.2.2 Hit Detection Sensors

Each combat module will also have an array of sensors to enable proper reception of the firing signals, or simulated bullets, sent by opposing simulated weapons. This sensor array should ideally have a detection range of 360 degrees in all directions, but this may not be necessary in all cases. For example, when testing a ground-based targeting system against a UAV and assuming the UAV is not conducting inverted flight, the sensor array range on the UAV only needs to be in its underside hemisphere.

These sensors must be sensitive enough to detect data transmissions on the order of a hundred meters while still having the noise rejection capability to ignore the infrared noise due to solar activity.

3.2.3 Statistical Recording

The combat module must also be capable of recording important combat scenario data. Some critical information that must be stored are things such as number of shots

fired, number of hits received, origin of all hits, and information about the vehicle to which the combat module is attached.

3.2.4 Registered Nonvolatile Configuration

One very attractive feature of this combat system is the fact that one combat module can be configured in a variety of ways to provide many different types of combat scenarios, weapon differentiation, team membership, etc. This can be achieved by using internal registers on the combat module to store configuration data. It is important that these registers be non-volatile – data not compromised during power cycling – because power may not be reliable on an unmanned vehicle under test. Also, once a module is configured for a specific vehicle, it should not need to be changed very often, so the configuration should be stored indefinitely.

3.2.5 Vehicle Communication and Control

The combat module should be able to communicate with the vehicle under test. The vehicle should be able to command the combat module to fire when the vehicle or targeting system is ready to fire. Also, since the combat module detects hits and can communicate with other devices in the combat system, the combat module must be able to communicate and control the vehicle at some level. This way the vehicle can be alerted when a hit is received and it can act accordingly.

3.2.6 Basic Add-ons

It is accepted that the combat module will not have all the possible capabilities wanted by the end user. Because of this, the combat module should have some sort of expansion, or add-on capability so that hardware features could be added as needed.

These add-on modules could be provided as parts of the combat system or the end user could create their own add-on modules as necessary.

3.3 Command Module

While the combat module is completely configurable, it needs an external device to perform its configuration. The command module is a user interface device to set all of the configuration registers of the combat modules to be used in a scenario. One command module is sufficient to configure a group of combat modules.

3.3.1 Configuration User Interface

An easy way to configure the combat modules is through the FSO communication link that the combat modules use for the simulated weapon. The command module, therefore, should also be capable of FSO communication but does not need the higher optical power needed by the combat modules. The command module should have a user interface, with user inputs and visual feedback so that the user can setup a combat scenario with multiple combat modules. Once the scenario has been configured, the command module can be used to upload configuration data to each combat module that will take part in the combat scenario.

3.3.2 Review of Combat Scenario Results

After a combat scenario is complete, the command module can be used to review the results. The recorded statistics from each combat module can be downloaded to the command module. The statistics can be combined to create reports on the scenario results. This could include team statistics, scoring, firing accuracy data, etc. This data can

then be used to further modify and enhance the unmanned vehicle's or targeting system's operation.

Figure 3.1: High level overview of Modular Laser Combat System.

4. TEST PLATFORMS

The modular laser combat system is for use with autonomous and remotely operated vehicles, so it is only reasonable that the system be tested on some sort of remotely operated vehicle. Independent or autonomous vehicles are beyond the budget of this project and a simple remotely operated vehicle effectively shows the capabilities of this modular laser combat system. Two main vehicle types were used for testing of the combat system: remotely controlled (RC) airplanes, and a stationary remotely controlled ground-based targeting device.

4.1 Aerial Platforms

Three different RC airplanes were tested with the combat system, two were built from scratch using common parts, and one was a popular but proprietary beginner's RC airplane. The two scratch built airplanes show how the system integrates with industry

standard parts and communication techniques, while the proprietary plane shows that the system can be adapted to nearly any vehicle.

4.1.1 Aerial Platform Overview

Each of the airplanes used for aerial platform testing are quite small and lightweight. This is representative of the potential platforms on which the combat system is intended to be used. Small UAVs are becoming more prevalent and it is generally small companies and research institutions that are producing them. These small platforms present a non-trivial challenge of producing an extremely lightweight combat system that can be carried by these small airplanes.

4.1.2 TomHe F-22 Raptor

The TomHe F-22 Raptor is a scratch built foam airplane based on the plans made available online by Tomas Heller, whose online user name is TomHe [6]. Heller has a variety of free RC airplane plans available which are primarily targeted for construction with Depron foam. Depron is a type of extruded polystyrene foam that generally comes in 6mm thick sheets. This is the plane that was used for the majority of flight testing.

The F-22 that was used for testing in this project was built using Elmer's foam board as the primary structural medium. This foam board has a poster board paper layer laminated to both sides of a sheet of Depron foam. These poster board layers add significant strength and rigidity to the bare Depron, but also approximately doubles the weight of the board. Construction of an airplane using this structural medium is very simple and is also a very quick method. The only tool that is needed is a sharp hobby or X-acto knife. The airplane was assembled using a combination of Gorilla and Elmer's

glue along with packaging and fiberglass tape. An easily replaceable paper nosecone protects the front of the airplane during the highly unlikely event of a crash. See Figure 4.1 for the resulting aircraft.

The plans provided by Tomas Heller were also modified to include what is known as a Kline-Fogleman (KF). The original plans only had a flat foam wing shape. The Kfm2 airfoil, seen in Figure 4.2, was used on the F-22 Raptor. The airfoil has a step at 45% of the wing chord length. This step generates a small pocket of circulating air behind it that allows air coming over the top of the wing to slip over the pocket with no surface friction. This added airfoil increases the lift of the airplane, enabling a wider speed range and slower stall speed. The KF airfoil also dramatically increases high-speed stability, making the airplane much easier to fly.

This is the same sort of effect that makes driving on the freeway in a pickup truck with the tailgate *up* more efficient than one with the tailgate *down*. Having the tailgate down breaks the pocket of circulating air and increases drag. The airfoil was originally developed for long distance paper airplanes, but has found its way into a multitude of small aircraft designs

This type of airfoil is however speed limited to about 80mph. At that speed, the increased drag of the step overcome the benefits of the airfoil shape. This speed limit is why this airfoil is not typically used on any full sized aircraft. Fortunately, the 6 inch prop spinning at just under 20,000 rpm is only capable of driving this RC airplane at about 40mph, so the speed limitation of the airfoil is not of great concern.

Figure 4.1: TomHe F-22 Raptor aerial test platform.

Figure 4.2: KF airfoil used by the F-22 aerial test platform [7].

4.1.3 Sport Trainer

The Sport Trainer is a very basic scratch built airplane, consisting of only three main flat pieces of foam board; the wing, fuselage, and horizontal stabilizer. This airplane was built from plans found at MikeysRC.com. It has a flat wing without any special airfoil but has the look of a slow flying conventional aircraft like a Piper Cub. See Figure 4.3 for a photograph of the Sport Trainer.

Like the F-22, this plane was built out of Elmer's foam board. It is much less agile than the F-22, but is extremely stable thanks to its low center of gravity relative to its

wings. It is controlled via elevator and ailerons only, with no rudder. Its stability, ease of flying and ability to carry a small payload make it an ideal candidate for flight testing for this modular laser combat system.

Figure 4.3: Sport Trainer aerial test platform.

4.1.4 Aerobird Challenger

The Aerobird Challenger from HobbyZone is a basic RC airplane intended for novice pilots. Although it is capable of some basic aerobatics, its low power rating limits its performance significantly. Also, the small control surfaces on its V-tail make any quick or particularly agile maneuvers impossible. It has excellent glide characteristics however and can carry a small payload without noticeable degradation in performance.

The radio communication on this airplane is a non-standard protocol which makes communication with additional on-board devices a challenge. The plane does have an

activation button on the radio transmitter to enable additional features and this can be used to activate the Combat Module used in this project.

Figure 4.4: Aerobird Challenger aerial test platform.

4.1.5 Summary of Flight Characteristics

Understanding the flight characteristics of the aircraft used during testing of the combat system is essential to ensuring a safe flight test environment. The power rating shown along with other flight characteristics in Table 4.1 gives insight into the type of performance that a small unmanned vehicle will have. Table 4.2 relates power rating to a qualitative measure of performance. This performance level, along with the wing loading information, indicates how the aircraft will perform, particularly after a payload has been added.

Table 4.1: Basic aerial platform flight characteristics.

	TomHe F-22 Raptor	Sport Trainer	AeroBird Challenger
Wingspan [inches]	28.4	40	40
Length [inches]	39.3	30.5	30
Wing area (approx) [sqin]	288	288	160
Flying Weight [oz]	24	16.9	15.5
Wing Loading [oz/sqft]	12	8.45	13.95
Max Power Output [W]	201.1	201.5	47.4
Power Rating [W/lb]	134.07	190.77	48.93

Table 4.2: Performance related to aerial platform power rating.

Power Rating [W/lb]	Qualitative Performance
50-70	Minimum level of power for reasonable flight
70-90	Trainer and slow flying
90-110	Sport aerobatic and fast flying
110-130	Advanced aerobatic and high speed
130-150	Lightly loaded 3D aerobatics
150-200+	Unlimited performance aerobatics and 3D aerobatics

The power ratings calculated above for each aerial test platform is derived from a basic test procedure. A digital wattmeter capable of handling currents up to 100 amps was used measure the no load – no propeller installed – power usage of the aircraft drive motor. The propeller was then added to get a loaded power usage reading. In both cases, the motor was spun up to maximum throttle. The power output was then approximated as the difference between the fully loaded and no load power conditions.

This is not the most accurate test method for a variety of reasons, the first is that differencing full load and no load input powers does not account for copper or magnetization losses in the motor. Also, driving a propeller in a static test does not accurately represent the load that the motor will experience during actual flight. Despite these inaccuracies, it is important to note that only a first order approximation of the power rating is necessary. The power rating is only used for qualitative measurement of

performance and errors of even 15% in the power rating are unlikely to cause any significant disparities between expected and observed performance.

Note that the Sport Trainer actually has the highest power rating at 191W/lb. This can be slightly deceptive because the power rating is only part of the measure of performance. The power rating says nothing about the speed or pull-out torque of the aircraft. When compared to the F-22, which has a lower power rating, the Sport Trainer is actually quite sluggish. The F-22 is much heavier, which allows it to penetrate the air better, but leads to a lower power rating. Also, the power system on the Sport Trainer can only sustain the 191W/lb. rating for short bursts, as the battery heats up significantly at this power output level.

4.2 Ground-based Targeting

Due to a lack of qualified pilots during the testing process, a stationary ground-based targeting device was also created. This device also served to show that the system can be used in a variety of platforms and is not limited to unmanned aerial vehicles alone.

Due to its ease of use and availability of materials, Elmer's foam board was again used to construct the ground-based targeting device. It is a simple design, seen in Figure 4.5, consisting of a two-axis pan and tilt turret with a mount for placing on a tripod. The ground-based targeting device houses a wireless targeting camera and has mounting locations specifically for the modular laser combat system. It is controlled via a standard hobby radio system, much like the ones used in the aerial platforms. The difference, however is that the control commands sent through the hobby radio are decoded using a

custom microcontroller board to make the movement of the turret more realistic in operation and speed.

Figure 4.5: Stationary ground-based turret targeting test platform.

5. IMPLEMENTATION

For the purpose of proving the validity of a modular laser combat system, a proof-of-concept system has been created. This system is based upon a Microchip PIC microcontroller and hardware built around it to provide the necessary functionality. It should be noted that this is not presented as the one and only solution for the modular laser combat framework. This is merely a proof-of-concept design that could use further enhancements and functionality.

5.1 Microcontroller

This project was implemented using a PIC microcontroller as the heart of the system. The microcontroller provides all the input and output functions along with all the necessary processing. The microcontroller gives the system the flexibility and modularity needed for useful operation. By using a microcontroller rather than a more rigid hardware approach, each combat module can be configured independently.

Originally, the microcontroller that was used for the combat modules was the PIC18F1320. This was an 18 pin device with modest capabilities and characteristics. After an extensive search the PIC26K20 was chosen to be the controller for both the Combat and Command modules. This new device has many times the capability of the original microcontroller but at an even better price. The new device is by no means the most advanced controller available, but provides a balance between features and cost. See Table 5.1 for a comparison between the two microcontrollers. Both of these devices are considered part of Microchip's high-end 8-bit microcontroller class.

Table 5.1: Comparison of basic microcontroller specifications.

	PIC18F1320	PIC18F26K20
Program ROM (kB)	8	64
EEPROM (B)	256	1024
RAM (B)	256	3936
Pin Count	18	28
Price (\$)	2.24	1.93

5.1.1 C Compiler Selection

Due to the ease of coding and familiarity, C was the language that was chosen to program the firmware for the modular laser combat system. There are many available C compilers for use with PIC microcontrollers. The compiler that had been used extensively in the past was MikroElektronika's MikroC compiler for PIC. This is a very user friendly compiler and contains many compiler libraries to quickly and efficiently implement various hardware features on the microcontroller. The compiler handles all context saving for interrupt controls and does so seamlessly without fail. It is without a doubt the most comfortable environment to program firmware for PIC microcontrollers.

Unfortunately, due to budgetary constraints, only the demo version of MikroC compiler for PIC was used. This demo version has a firmware hex code limit of 2kB. Implementation of the entire system takes up more than 2kB on the microcontroller and the microcontroller that is used has 64kB of memory available for the program. Because this program size limitation was too restrictive and it was not reasonable financially to purchase the full version of the compiler, a new compiler had to be used.

Since Microchip's PIC microcontrollers were being used for the processing power in the modular laser combat system, the next compiler that was used was Microchip's C18 compiler. This is almost a strictly ANSI C compiler with very little user friendly features or abilities. There are some available libraries for implementation of special hardware functionality, but they are more difficult to configure and generally less flexible.

Also context saving for interrupt control is only handled by the compiler in the most basic sense and anything more advance is left entirely up to the programmer. This caused more trouble than it was worth since the firmware is almost completely interrupt driven. It was too troublesome to be concerned about context saving, especially after using MikroC, which handled context saving on its own. Because Microchip's C18 was not user-friendly by any stretch of the imagination and interrupt errors were common in the code it generated, yet another compiler was used.

The final C compiler that was used was HiTech's PICC18 compiler. This compiler had actually been purchased by Microchip and integrated easily into the programming environment that was already being used for Microchip's C18 compiler. This new

compiler by HiTech proved to be much more user-friendly and even generated smaller machine code given the same C code. PICC18 handles all context saving for interrupts on its own, unlike Microchip's C18. Unlike the MikroC compiler by MikroElektronika, the PICC18 compiler by HiTech did not have a code size limit on the demo version. Rather, the demo and full versions of PICC18 were differentiated by the level of optimization that the compiler conducted. For the purpose of this proof-of-concept design, however, all optimizations were disabled to help ensure that everything behaved as expected.

Changing between compilers would have been quite simple if they were all strictly ANSI C compilers. Unfortunately, each compiler had its own way of dealing with the hardware functionality of the PIC microcontrollers and they each deviated from ANSI C in subtle ways. This meant that every time a new compiler was used the firmware had to be re-written to operate correctly with the new compiler. This was a very tedious and time consuming process that hindered completion of this proof-of-concept design.

5.2 Infrared Communication

Infrared communication is accomplished through IR LEDs and IR sensors. The sensors used in this project have an optimum IR reception wavelength of 940nm. Whenever the sensor senses light at 940nm and modulated by 40kHz it drives its output low. This makes it an ideal sensor for UART-type communication.

UART was chosen as the communication protocol because of its ease of use with the chosen sensor, but more importantly because of its ease of use with the chosen microcontroller. The PIC18F26K20 has a hardware UART module built-in so that communicating via UART is as simple as setting the appropriate configuration registers.

Also, sending data over an IR connection is often times slow, so that fact that the microcontroller does not have to take the time to generate the entire transmission and reception processes in software saves a lot of processing time and power.

UART has separate lines for transmission and reception. It is an asynchronous form of communication so there is no clock necessary synchronizing communicating devices. Because timing errors between communicating devices can accumulate during a transmission, only 1 byte is sent at a time, and everything is timed off of the start bit. The UART transmission or reception lines, TX and RX respectively, idle in a high state. Therefore the start bit is a low value bit that is held for one baud period. After the start bit is sent, the data is sent with the least significant bit first. The data bits are active high, meaning that a digital “1” is sent as a high value and a digital “0” is sent as a low value. Each bit that is sent is held for one baud period. The stop bit is a high pulse for one baud period that signifies the end of the transmission. See Figure 5.1 for the structure of a UART data byte. This is the simplest and also most common form of UART.

Figure 5.1: UART byte structure [8].

The UART data is modulated at 40kHz to make it compatible with the IR sensor. This square wave modulation spreads the spectrum of the transmitted signal. In a transmitted square wave, the first and third harmonics contain roughly 90% of the transmission power, making the third harmonic an effective measure of bandwidth. Figure 5.2 shows the resulting spectrum of a modulated UART signal. The effective

bandwidth in this case is approximately 120kHz. But this signal is fed into an infrared LED with a wavelength of 940nm and an associated frequency of 319THz. The LED essentially shifts the entire UART spectrum to be centered around 319THz. Since the spectrum that is shifted contains positive and negative frequencies the resulting bandwidth of the LED modulated UART signal is double to approximately 240kHz.

Figure 5.2: Spectrum spreading due to modulating UART signal.

(W1) 2400Hz square wave representing UART signal, (W2) spectrum of UART signal, (W3) modulating 40kHz square wave, (W4) band limited spectrum of modulating wave, (W5) modulated UART signal, (W6) resulting spectrum of modulated waveform.

5.2.1 IR Reception

The sensor used in the implementation of this project is the TFMS5400. It has an effective viewing angle of $\pm 55^\circ$ on both the vertical and horizontal axes. This is by no means, enough reception range for the purpose of nearly 360 degree reception. The TFMS5400 does, however have open collector outputs, which enable multiple sensors to be chained together on the same UART RX. When one or more sensors in the chain senses an incoming signal, they will drive the output low, even if some of the sensors have not received any signal. The chained sensors can then be mounted in different orientations to construct a composite viewing angle that essentially can receive UART bytes via IR from any direction. See Figure 5.3 for the IR sensor array mounted on the Ground Station under test. It does not offer full 360 degree viewing, but enables reception from any reasonable direction.

Figure 5.3: A5 IR Sensor Array on Ground Station.

These sensors are highly susceptible to noise in the power supply, so it is recommended that a simple passive filter be used on the supply input. See Figure 5.4 for the IR sensor array schematic. Each capacitor is placed as close to a sensor's power pins as possible.

Figure 5.4: IR Sensor Array schematic.

5.2.2 IR Transmission

For IR transmission, the LVIR3333 IR LED was chosen because it was well matched with the TFMS5400 sensor and because it had a relatively narrow beam divergence of +/-10 degrees. It also has a high optical output and can handle short pulses of 3 amps. This is exactly what is needed for relatively long range IR communication.

The TFMS5400 sensor requires that incoming signals be modulated at 40kHz and the microcontroller must be able to drive this high current load. A surface mount dual MOSFET chip, the NDS9936, was used to provide the necessary switching power. See Figure 5.5 for the resulting IR LED drive circuit. Note that T1A and T2A handle the modulation of a UART TX signal.

Current can be sunk by these two transistors through the LED and an equivalent resistance of roughly 50ohms. This results in a low power transmission mode with a forward LED current of,

$$I_F = \frac{V_{DD} - V_{LED}}{R_1 \parallel R_2 + R_3 + R_{T2A} + R_{T1A}} = \frac{3.3 - 1.3}{10 + 39 + 0.1 + 0.1} = 40\text{mA}$$

T2B enables one of the load resistors to be bypassed so that the equivalent resistance is reduced to about 10ohms, and the forward LED current is boosted to,

$$I_F = \frac{V_{DD} - V_{LED}}{R_1 \parallel R_2 + R_{T2B} + R_{T2A} + R_{T1A}} = \frac{3.3 - 1.6}{10 + 0.1 + 0.1 + 0.1} = 165\text{mA}$$

This two-level transmission power is important because it enables long range firing while also limiting optical output during configuration by the Command Module. Also the maximum continuous current rating of the LED is 50mA. The high power output when I_F is 165mA is only possible because the LED is not on continuously.

Also note that the UART TX line idles in a high state. The IR LED needs to idle in an off state or else the LED would burn itself out due to large continuous currents. The TX signal is inverted via Q1 so that the LED is only active when TX goes low.

Figure 5.5: IR LED drive schematic.

5.2.2.1 Optical Transmission

Even though the LVIR3333 has a small beam divergence of ± 10 degrees, this is too much divergence to be a reasonable simulation of a weapon, unless the desired weapon characteristics were that of a shotgun. A large beam divergence corresponds to a wider firing angle and also a shorter range.

In order to narrow the beam divergence of the LED, a collimating lens must be added. See Figure 5.6 for a representation of a collimating lens assembly for an LED. A collimating lens is only effective at collimating incident diverging light if the light is coming from a source which is much much smaller in diameter than itself. This is to say that the light can only be collimated if the source of the light can be treated as a point source. Unfortunately, due to size and weight restrictions on the remotely operated vehicles under test, the lens that was used is only 20mm in diameter. The LED itself is 5mm in diameter so the assumption that the LED is a point source is very inaccurate.

Figure 5.6: Cross section view of collimating lens tube assembly.

This leads to what is known as parallax distortion. Since the diverging light is not coming from a point source, the focused beam can never be fully collimated. Thus, the

lens can only serve to reduce beam divergence and provide a near-collimated beam. The larger the lens is compared to the LED, the lower the parallax distortion.

It must be noted here that 940nm IR wavelength was used for all IR communication in this project because that is the hardware that was readily available. The 940nm band is actually intended for use in indoor IR communication such as TV remotes and the like. Looking back to the Free Space Optical Communication chapter in Figure 2.2 and also in Tables 2.1, it can be seen that 940nm light is strongly attenuated in the atmosphere. The effect of atmospheric absorption will reduce the range of the simulated weapon under test.

5.2.3 The Data “Bullet”

Every time that the simulated weapon is fired, a small IR UART data packet is sent that is referred to as the IR “bullet.” The bullet used in this implementation is only two bytes long. It contains four nibbles of information as shown in Table 5.2; the player ID, the team ID, the weapon type, and the weapon power.

Table 5.2: The data “Bullet” contents.

ID Byte								Weapon Byte							
Player ID				Team ID				Weapon Type				Weapon Power			
P3	P2	P1	P0	T3	T2	T1	T0	W3	W2	W1	W0	S3	S2	S1	S0

The term, player, is used from here on out to describe an active member or vehicle in a combat simulation. Team refers to a set of players that are aligned with one another in a combat simulation. All references to a weapon will be assumed to refer to a simulated weapon, as is the case with the IR weapon implemented in this project.

The player ID is a unique identification number for a player on any given team. Each team also has a unique identification number. These ID codes allow all the combat modules to know when they are hit and by whom they were hit. Note that in the current configuration, there is available codes for 16 teams with 16 players each. These numbers were chosen for simplicity and consistency, although it may be more useful to allow more players and fewer teams. As it stands however, it is doubtful that any team would require more than 16 players since this system is designed for small businesses and research institutions.

The next byte contains information about the weapon that was fired. This allows multiple simulated weapons to be utilized by any vehicle and deployed by the same Combat Module. Each vehicle can respond differently to a hit by a different type of weapon. Not only is there the possibility of different weapons, but each type of weapon may have a varying level of strength or effectiveness. This behavior is captured in the weapon power nibble.

5.2.3.1 Bullet Length

Since communication between moving objects using focused beams of infrared light transmitted through a very lossy atmosphere can be less than reliable, it is important to ensure that data that is sent is received accurately. It is possible that a bit error in the player ID byte during data reception could identify a hit from player X even though player Y fired the shot. Even worse, faulty readings from the IR sensors could mistakenly identify a hit from another player even though no other player was firing at the time.

To mitigate this issue for the purposes of this implementation process, each two-byte bullet is actually sent three times in rapid succession for each firing of a weapon. This enables the Combat Modules to compare the three bullets received for consistency. While the sample size of the received bullet samples is only three, the Combat Modules can make a determination of the bullet contents even if some of the data is corrupted.

This also means, however, that the composite bullet, consisting of three redundant bullets, will take three times as long to send and receive. The TFMS5400 IR sensor requires that data be modulated at 40kHz. Also, in order to register reliable data bits, each data bit must include 10 cycles of the modulating carrier frequency:

$$f_{bit(MAX)} = \frac{f_{carrier}}{10} = \frac{40\text{kHz}}{10} = 4\text{kHz}$$

The nearest common baud rate for UART communication that does not exceed 4kHz is 2400 baud. This is extremely slow, but is a limitation of the sensors that were available for this project. Since each byte that is sent via UART requires start and stop bits in addition to the eight data bytes, the length – in time – of a single byte is then,

$$\tau_{byte} = \frac{(1+8+1) \left[\frac{bits}{byte} \right]}{2400 \frac{bits}{s}} = 4.167 \frac{ms}{byte}$$

while a complete shot with 3 times bullet redundancy is,

$$\tau_{bullet} = 3 N_{bytes} \tau_{1\text{ bullet}} = 3 \left(2 \frac{bytes}{bullet} \right) \left(4.167 \frac{ms}{byte} \right) = 25 \frac{ms}{bullet}$$

This is actually a fairly long time to complete a transmission to a moving target. To ensure that this time is not too long for proper operation, consider a small unmanned

aerial vehicle flying at 60mph perpendicular to the the light of sight of a stationary Combat Module – see Figure 5.7. This is a reasonably fast speed for the small unmanned vehicles currently on the market and represents a worst case scenario for bullet reception.

Figure 5.7: Relating bullet length to travel time and IR spot size.

The speed of the vehicle is standard units is,

$$u_{UAV} = \left(60 \frac{mi}{hr} \right) \left(\frac{hr}{3600 s} \right) \left(1609 \frac{m}{mi} \right) = 26.8 m/s$$

During the time that it takes to send bullet, 25ms, the UAV will have traveled,

$$d_{UAV} = u_{UAV} \tau_{bullet} = \left(26.8 \frac{m}{s} \right) \left(25 \frac{ms}{bullet} \right) = 0.67 \frac{m}{bullet}$$

5.2.3.2 IR Bullet Spot Size

From this we see that the spot size of the IR beam must be greater than 0.67m, or 2.2 feet, in order to successfully receive the bullet. This is a very reasonable size and the

actual spot size should be larger. The spot size should be comparable to the size of the vehicle itself. This is do to the fact that for a real-life bullet, the point-like projectile will cause a “hit” if it strikes anywhere on the vehicle. With the Combat Module, however, a hit is only cause if the point-like sensor is struck with the IR bullet. Figure 5.8 shows that in order to simulate a similar successful hit area on the UAV, the IR beam spot size must be on the same order of magnitude as the UAV size itself.

Figure 5.8: Gaining similar successful hit areas with a single reception point and large spot size.

A notable benefit of bullet transmission latency is that it may actually offer a more realistic representation of a true weapon. No real-life bullets or rockets or other projectiles can travel at the speed of light, as the IR bullet is in this case. Real-life projectiles take time to reach their targets and when firing a projectile, it must be aimed just ahead of the target to compensate for the flight time of the projectile. In the same way, the simulated weapon in the Combat Module may need to be aimed slightly ahead of the target so that by the time the full IR bullet is transmitted, the vehicle will have traversed the entire IR spot. Failing to “lead” the target slightly with the Combat Module may result in an incomplete transmission as the vehicle may travel outside the IR beam.

5.3 Non-Volatile Configuration and Statistical Recording

The Modular Laser Combat system requires some form of non-volatile memory storage for the purpose of user device configuration and statistical recording during a combat simulation. The PIC18F1320 has built in EEPROM with space for 1024 bytes of information. Writing to EEPROM takes a considerable amount of processing time – on the order of milliseconds – so it is important to only save the required data when it is absolutely necessary.

5.3.1 Storage Structure

The current setup has 16 available bytes for configuration registers. This is not a hardware limitation and could easily be expanded by a firmware upgrade. The configuration register description can be found in Table 5.3.

There are three essential groups of configuration registers. The first is the Combat Module identification information. IDa and IDb are the player, team, weapon, and

weapon power information that was described in Table 5.2. These registers define who the Combat Module is and defines the information that is transmitted during a firing sequence.

The Player1 through PlayerN registers define the player and team identification numbers of all the other players in the current combat simulation. The NumPlayers register is the number of players that are to take part in the current combat simulation. The combination of these registers helps the Combat Module to know when it has received a valid hit from a player in the simulation or just an erroneous or corrupted IR signal.

The ControlType and FireThresh registers define the communication protocol between the Combat Module and the radio system on-board the unmanned vehicle. The Combat Module is capable of multiple communication styles to make it useful on a variety of unmanned platforms.

The other registers simply make the Combat Module as flexible as possible. Some of them have yet to be implemented in firmware.

An attractive feature of this combat system is the ability to store statistical information for later review and analysis. Only a basic form of statistical recording is implemented here for the purpose of proving that it is possible on this device. Each Combat Module stores the number of shots that it fires during a combat simulation. Each module also stored the number of hits that it receives from each other player in the simulation. This information can then be compiled to form firing accuracy statistics for each player.

Table 5.3: Combat Module configuration registers.

Combat Module EEPROM Registers

Register	Location (decimal)	Type	Description	Default
IDa	0	0	Personal player and team ID#s	0xAA
IDb	1	0	Personal weapon/power ID#s	0xBB
Handicap	2	0	Relative advantage setting for less skilled players	0x00
MotMin	3	0	minimum PWM pulse length of radio system motor input	0x9F
MotorRangeL	4	0	Low Byte - PWM pulse width range of radio system	0x64
MotorRangeH	5	0	High Byte - PWM pulse width range of radio system	0x02
HitType	6	0	reaction to hits (temp. disable, blood loss, friendly fire, etc.)	0x00
NumPlayers	7	0	Number of other players, N, in the scenario	0x02
GameType	8	0	Type of game (Free-for-all, VIP, Alamo, etc.)	0x00
ControlType	9	0	Input type to the module (RC PWM=0, HobbyZone X-Port=1, etc.)	0x00
FireThresh	10	0	Pulse width threshold for valid fire command	0x11,0x6E
Player1	10 + 1	1	Player1's player and team ID#s	0x55
Player2	10 + 2	1	Player2's player and team ID#s	0xCA
Player3	10 + 3	1	Player3's player and team ID#s	0x75
...	10 + k	1	Playerk's player and team ID#s	0xFF
PlayerN	10 + N	1	Last player's player and team ID#s	0xFF

5.4 Control

Each Combat Module is controlled via the existing radio hardware in the unmanned vehicles in which they are installed. The Combat Module can be configured to communicate using multiple protocols and two were tested for this project, generic hobby RC servo PWM and HobbyZone's proprietary X-Port.

The key inputs to the Combat Module are the FIRE and MOTOR inputs. FIRE tells the Combat Module when to transmit an IR bullet. MOTOR is the desired motor speed of the unmanned vehicle set by the pilot. The MOT_OUT output is the processed version of the MOTOR input, which may or may not have been altered by the Combat Module. It is important that each of these inputs and outputs communicate properly with the unmanned vehicle since glitches in communication may cause erratic vehicle behavior.

5.4.1 Generic RC Servo PWM

The generic method for controlling RC aircraft, such as the F-22 and Sport Trainer aerial test platforms used in this project, is through pulse width modulation (PWM) servo control. Hobby servos contain internal analog control systems that move a control arm from -90 degrees to +90 degrees off center. The reference, or desired, input to the servo is a positive pulse whose width determines the angle to which the servo is supposed to travel.

As a general rule-of-thumb, a 1ms pulse drives the servo in the maximum negative direction, a 2ms pulse drives the servo in the maximum positive direction, while a 1.5ms pulse would center the servo. See Figure 5.9 for a representation of this, but note that each servo is connected to a different control line contrary to what is apparent by the image. Each positive control pulse is repeated at a rate of approximately 50Hz.

Figure 5.9: RC servo PWM communication [9].

The FIRE input to the Combat Module in this communication scheme is simply a repeating 1.5ms pulse when no fire button is pressed, and a repeating 2ms pulse when the fire command is present. The MOTOR input is yet another servo PWM signal where 1ms corresponds to the desired motor speed being zero, and 2ms pulse width corresponds to maximum motor speed.

These type of signals are readily sensed by the microcontroller by connecting them to two external interrupt pins. Whenever the PWM signals change on either input change, the incoming pulses are timed by the microcontroller to determine the intended input. Controlling the MOT_OUT output is also straightforward as a timed pulse can be generated using the internal timers of the microcontroller.

5.4.2 X-Port

X-Port communication is a proprietary protocol for many RC airplanes from HobbyZone. These airplanes have an “X-Port,” or expansion port, where various add-ons can be plugged into the airplane. A button on the airplane's radio transmitter activates whatever is plugged into the X-Port.

The X-port control line consists of a repeating pulse train shown in Figure 5.10 which contains all control information for the aircraft. The figure shows the pulse train during normal operation. When the button on the airplane's radio transmitter is pressed, the normal control signal is replaced by a higher frequency square wave with approximate period or 0.32ms. Thus, if the width of the negative pulses is measured, the fire or activate command can be determined by looking for multiple short pulses in a row

The vehicle's motor cannot be controlled as completely while using the X-port as it was previously with the servo PWM control. With X-port, a line can be grounded to disable the vehicle's motor and no further motor speed control is possible.

Figure 5.10: Standard X-port communication pulse train [10].

5.5 Hit Response

Since the simulated weapon is just a beam of IR light that sends small packets of data, there must be a way to signify that the Combat Module has been hit when it receives an IR bullet. The Combat Module houses a piezo buzzer that can be triggered after a valid hit, but a sound alert alone is not enough necessarily enough to denote a hit.

5.5.1 Motor Manipulation

The most obvious way to signify a hit on an unmanned vehicle is to inhibit control of its motion. For aircraft or ground vehicles, this can mean disabling or reducing power to the drive motor. For the ground station tested in this project, disabling or limiting the speed of the tilt servo on the turret could also be the response to a valid hit.

Care must be taken while changing the motion control of a vehicle to ensure that safe operation is still possible. For example, the motor of an aircraft should not be

disabled indefinitely in case that vehicle is not in an area to make a safe landing. Instead, the motion of the vehicle should only be inhibited for a short time – on the order of seconds – so that a hit is obviously observable, but safety is maintained.

5.5.2 Auxiliary Outputs

The Combat Module is equipped with three additional outputs for the purpose of further hit response capabilities. One of the outputs is configured to be an additional servo PWM output to control another part of the unmanned vehicle. This could alter the orientation of a control surface on the vehicle or could release a flag or streamer to indicate a hit condition has been found. The other two outputs are configured as basic active-high outputs. These lines could be driven high after one or more hits.

5.5.2.1 Activator Add-on Module

The Activator is an add-on module which shows how the combat system can be expanded to include more functionality. The Activator is a dual-output high-current switch capable of sinking 5 amps per output. It is a small and lightweight device that can be simply plugged into the Combat Module's two active-high auxiliary outputs as seen in Figure 5.11.

Figure 5.11: The Activator high current Add-on Module (black stick in photo).

This device is useful for driving a higher powered siren, or for the ignition of a smoke trail to indicate the presence of a hit condition. Two LEDs are also on the Activator for debugging purposes. Each LED illuminates when its respective output channel is activated.

5.6 The Combat Module Device

The Combat Module, as implemented for this proof-of-concept project, can be seen in Figure 5.12. Since this was part of a prototyping process, this is the latest version of the Combat Module, although the previous versions which were also tested look similar. The casing was built using 3/32" thick balsa wood to render a strong, but lightweight structure.

The lens that was used is an acrylic plastic lens with a diameter of 20mm and a focal length of 34.5mm. It was glued in a paper tube, which was printed with black toner to try and minimize reflections of IR light off the tube's inner wall. The black toner also minimizes IR light penetration through the walls of the lens tube. The IR LED is mounted inside a wooden centering ring which ensures that the LED, lens tube, and lens optical axes are coaxial.

Although it is on its fourth physical iteration, only the latest version of the Combat Module circuit board is shown in Figure 5.13. It provides a programming port for easy firmware updates during the testing process. Input and output ports are available at the rear of the Combat Module for easy vehicle integration. The board can be powered from the vehicles drive system directly or can use a separate battery source. The board schematic can be found in Figure 5.14.

Figure 5.12: The Combat Module.

Figure 5.13: Most recent version of the circuit board for the Combat Module.

Figure 5.14: Combat Module schematic.

5.7 Command Module

In order to configure and review information from the Combat Modules in the field, a Command Module was produced. This was a straightforward device that demonstrates some of the potential for the modular and flexible nature of this combat system. Because there was a need for a cheap method of rapid and simple prototyping, the foam board that was used in the construction of the turret ground station was also used to make a useable case for the Command Module. Figure 5.16 shows the implemented Command Module and Figure 5.17 shows the circuit board fabricated for the Command Module. The circuit schematic can be found in Figure 5.18.

5.7.1 User Interface

User input is much like a handheld gaming device, with a directional joystick and two input buttons. Feedback is provided to the user through a 16x2 character LCD display. While the LCD used is not incredibly fancy or attractive, it uses a mature display technology that is easily viewable in full sunlight, as would be found in a testing field.

The layout of the Command Module menus are shown in Figure 5.15. There are four main menu options; CONFIG, UPLOAD, DOWNLOAD, and REVIEW. Selecting between these options is accomplished by moving the joystick up or down where the asterisks show the current selection the menu option can be entered by pressing the enter button, which advances the menu level by one.

The CONFIG category edits the configuration bits that are to be assigned to each player in a combat simulation. The player to be edited is selected in menu level 1 while the actual configuration bits can be edited in menu level 2. Selection of the configuration

bit and its respective value is accomplished through the use of the joystick. The back button is used to decrement the menu level by one. This enables navigation back to the main menu – menu level 0.

Figure 5.15: Command Module menu overview.

The UPLOAD category allows the user to upload the edited configuration bits for any given player. The player is selected with the joystick in menu level 1 and advancing to menu level 2 activates the transmission of the configuration bits to the Combat

Module. When transmission is complete, the menu level is automatically returned to level 1 so that other players can be configured.

The DNLOAD category enables the reception of data stored by individual Combat Modules during a combat simulation. There are asterisks under each player's number that has submitted its information to the Command Module since the last UPLOAD sequence. Advancing to menu level two sends a request to the Combat Module and receives the simulation information that it sends back. The menu level, as was the case in the UPLOAD category, is automatically returned to level 1 after a successful transmission and reception is complete.

The REVIEW category is a viewing tool for the information that was downloaded from the individual Combat Modules. Each player's information can be viewed in menu level 1 while the team statistics will be viewed in menu level 2. The compilation of team statistic data is still under development and this menu option is there as a placeholder for the moment.

5.7.2 Communication with Combat Modules

The communication hardware for the Command Module is nearly identical to that of the Combat Module. The differences are that the Combat module only has one IR sensor and that the IR LED is locked into the low power transmission mode. The Command Module does not need 360 degrees of reception because it is assumed that the Command Module will be pointed at the Combat Module with which it is communicating. Configuration of Combat Modules is likewise assumed to occur over a

transmission distance of a few feet, so high power transmission and a collimating lens are not necessary.

Figure 5.16: The Command Module.

Figure 5.17: The Command Module circuit board.

6. TESTING AND RESULTS

6.1 *Communication*

The first and foremost functionality that had to be tested was the Free Space Optical transmission using an IR LED and IR sensors. This was not necessarily the easiest thing to test and troubleshoot because IR light is invisible and cannot be analyzed electrically until after it has been sensed by a solid state device. This means that if there is an issue with the sensor, it is difficult to see by electrical means alone.

6.1.1 UART

The implemented system uses a modulated UART signal for communication using and IR LED. The microcontroller had to be configured to output this modulating 40kHz signal using an on-board PWM module. The 40kHz was dictated by the IR sensors used and a 40% duty cycle was chosen to ensure a strong output IR signal while minimizing the total power delivered to the IR LED. The microcontroller clock frequency was

10MHz, leading to an instruction clock frequency of 2.5MHz for the PIC18F1320 device. Because the instruction clock frequency is only 62.5 times that of the target PWM frequency, the resolution of possible PWM output frequencies, as well as the resolution of the PWM duty cycle is fairly small. Fortunately, Figure 6.1 shows that the modulating square wave generated by the microcontroller is very close to the target of 40% duty cycle and 40kHz.

Figure 6.1: PWM output showing approximately 40% duty cycle 40kHz square

The experimental frequency of the PWM signal is 39.97kHz while the duty cycle is as follows:

$$duty_{PWM} = \frac{\bar{t}_{high}}{\bar{t}_{high} + \bar{t}_{low}} = \frac{9.93 \mu s}{9.93 \mu s + 15.08 \mu s} = 39.7\%$$

The absolute error in modulating frequency is,

$$e_f = \left| \frac{f_{exp} - f_{calc}}{f_{calc}} \right| = \left| \frac{39.97 \text{ kHz} - 40 \text{ kHz}}{40 \text{ kHz}} \right| = 0.075\%$$

while the absolute error in duty cycle is,

$$e_{duty} = \left| \frac{duty_{exp} - duty_{calc}}{duty_{calc}} \right| = \left| \frac{39.7\% - 40\%}{40\%} \right| = 0.75\%$$

Because of the way that the IR LED is driven, the UART TX line had to be inverted to ensure that the LED idled in an off state. Figure 6.2 Shows an example

transmission by the Combat Module sending 0xAA and 0xCC. The stop bit of the second byte is not shown in this image. Note that channel A.0 in the image is the modulating PWM frequency where the display is slightly aliased because the plot is zoomed out so far in comparison to the 40kHz signal. Channel A.2 accurately shows that the UART TX line has been successfully inverted and the baud rate is approximately 2400.

Figure 6.2: UART transmission sending 0xAA and 0xCC showing PWM and inverted TX

The experimental baud rate is,

$$baud\ rate = \frac{\# \text{ of bits for a byte transmission}}{\text{time of one byte transmission}} = \frac{10}{4.14\ ms} = 2415.4\ baud$$

which leads to a baud rate error of,

$$e_{baud} = \frac{baud_{exp} - baud_{calc}}{baud_{calc}} = \frac{2415.4\ baud - 2400\ baud}{2400\ baud} = 0.644\ \%$$

This error can be explained by a combination of quantization error in the baud rate generator of the microcontroller as well as oscillator tolerance and/or temperature drift.

This error is extremely small and poses almost no threat to proper communication.

6.1.2 Optics

The focusing of this lens tube assembly is very important to the proper operation of the weapon. Without proper focusing, the weapon could experience a low firing range or non uniform spot pattern. From a practical standpoint, this presents a significant

problem because the light from the IR LED is invisible! In order to overcome this issue, there had to be some way to see the IR light.

The imaging sensors found in digital cameras are quite capable of imaging into the near-IR band. For everyday picture taking, however, IR light in images would drastically change the look of a photograph and result in a generally unsatisfactory image for the everyday photographer. Because of this, an IR optical blocking filter is added to most digital cameras to allow the visible spectrum through to the imaging sensor and reject the IR spectrum. An old digital camera was found and the infrared filter was carefully removed. The camera used is shown in Figure 6.3 along with the IR blocking filter that was removed. When viewed directly, the IR filter looks like clear glass, but is shown here at an angle, thus the slight hue.

Figure 6.3: Camera used for IR focusing with the removed IR blocking filter.

Due to the parallax distortion, the IR light coming out of the lens tube was impossible to collimate effectively. In the best case scenario, a 10 inch spot was projected onto a wall 6 feet away. The resulting half-angle divergence of this beam is,

$$\theta_{1/2} = \tan^{-1} \left(\frac{d_{spot}/2}{l_{travel}} \right) = \tan^{-1} \left(\frac{(10 \text{ in})/2}{(6 \text{ ft})(12 \text{ in/ft})} \right) = 3.97^{\circ}$$

Where d_{spot} is the diameter of the projected spot and l_{travel} is the distance from the source to the target. While this is a notable improvement over the 10 degree half-angle divergence of the IR LED alone, it falls short of the desired half-angle divergence. A spot size roughly the size of the target vehicle is desired, which will be approximated by a three foot diameter. So the beam divergence required to get a three foot spot on a target 50 feet away is,

$$\theta_{1/2 \text{ desired}} = \tan^{-1} \left(\frac{d_{spot}/2}{l_{travel}} \right) = \tan^{-1} \left(\frac{(3 \text{ ft})/2}{50 \text{ ft}} \right) = 1.72^{\circ}$$

So the actual spot size that was produced is 2.3 times the desired at any target distance. There is nothing fundamentally wrong with this resulting operation, but it does limit the performance of the simulated weapon. At 2.3 times the desired spot size, the weapon will have approximately 43% of the firing range that would be realized with the desired spot size. Also, at closer target distances, the target will be easier to hit because of the larger spot size. This is unfortunate, but unavoidable with the current hardware.

6.1.3 Power Supply Interference

During testing of the IR communication, it was discovered that the IR sensors used for this project are highly susceptible to power supply noise. This anomaly was

initially encountered when testing firmware without an infrared LED installed. When the Combat Module transmitted bytes of UART through the visible light LED that was being used for testing, the IR sensors picked up the signal even without the use of an IR LED. The problem was that the LED was sinking quite a bit of current in the on state and would cause ripples on the power supply line every time the LED turned on. These ripples in the power supply caused faulty readings in the IR sensors.

In order to inhibit noise from the power supply, a simple passive RC filter was added to the power input of the IR sensors. Also the voltage regulators for the main board and the sensors were separated so that it would be much less likely for power supply noise to occur in the first place.

6.2 Ground Testing

Extensive ground testing has been conducted both indoors and outdoors. All of the basic functionality works flawlessly indoors as expected. The Combat Module can communicate with the Command Module and visa versa. The Combat Modules can communicate and “hit” each other without any problems.

The results are less satisfying when the testing moved outside. All functionality still operated correctly, but reliability was the main concern. The Combat Modules could not always hit each other, even when properly aimed. Configuration of the Combat Modules by the Command Module through IR resulted in occasional corrupted configuration bytes. The problem was that there was so much solar interference in the IR band that the sensors are either saturating, or actually catching erroneous bytes from solar noise.

A closer look at the specifications for the TFMS5400 sensors used in this project reveal the limitations of the sensor in an outdoor environment. Figure 6.4 shows that the quality factor of the 40kHz modulation input filter is only 10, meaning that the modulation frequency can be off by 5% of the nominal value, and the sensor will still pass the signal. This makes the sensor susceptible to erroneous readings since optical noise incident on the sensor may very likely have an appreciable amplitude in the relatively wide band of 38 to 42kHz.

Figure 6.4: Relative responsivity compared to modulating frequency.

The spectral response of the sensor does not help the solar interference situation at all. Figure 6.5 shows the spectral response of the TFMS5400 sensor. While it is nominally a 940nm sensor, it will readily pass wavelengths from 875 to 1025nm. This bandwidth leads to a quality factor of only 6.3. Since the sensor is not very selective on the wavelength that it senses, it can easily saturate in a bright environment.

Figure 6.5: Spectral sensitivity of the TFMS5400 sensor.

Figure 6.6 shows the signal irradiance necessary to get a proper reading with the TFMS5400 in the presence of bright ambient light sources. From Figure 2.2, we can estimate an irradiance level for the 875 to 1025nm band. 500W/m² is a reasonable estimate of full sun irradiance. Even if only a quarter of the ambient light reaches and disturbs the sensor, that is 125W/m², the chart below shows that threshold irradiance increases very rapidly after 40W/m² ambient irradiance and 125W/m² is not even on the chart.

These sensors were used because they were what was available at the time, but it has become apparent that this sensor is not desirable for outdoor sensing purposes. Once again, the entire system did work in spite of the sensor issues, but the poor sensor performance limited the overall reliability of the system.

Figure 6.6: Threshold irradiance in the presence of a bright ambient.

6.3 Flight Testing

6.3.1 Preliminary Flight Testing

The preliminary flight testing was conducted with the Aerobird as the test vehicle and two Combat Modules, one on-board the aircraft and one aimed manually by a person on the ground. Both Combat Modules were able to fire on and hit each other in flight and when the Combat Module was hit on-board the aircraft, it successfully disabled the drive motor for preset amount of time. A firing range test was also conducted and successful hits were registered from 120 yards away. This is not the limit of the range, but the test site was only 120 yards long. Also this test was conducted in the late afternoon when solar interference was less of a problem.

6.3.2 Static Vehicle Tests

Unfortunately due to time constraints and uncooperative weather conditions, final flight testing was not completed. Instead, static vehicle tests were conducted to confirm

operation. The ground turret station as well as the F-22 and Sport Trainer were used for these tests. The ground station was mounted on a tripod as per usual but the aerial test vehicles were elevated on rails to get them off the ground but still enable static testing.

Whenever the ground station was hit by a shot, it was successfully disabled for the specified time and the tilt of the turret was moved to the vertical position to indicate the hit. When either the F-22 or the Sport Trainer were hit, the motor was disabled for the specified amount of time and was prevented from shooting. Each of the vehicles recovered from hits until they ran out of lives. The Activator add-on module was also used during this testing. Smoke canisters were ignited by the Combat Module during a hit to act as another indicator of a hit. All of the parameters like disable time, number of lives, and hit response were successfully set by adjusting the internal configuration bits of the Combat Modules.

7. CONCLUSIONS AND FUTURE WORK

What is presented here is a framework, not an end all solution. It can and should be expanded and improved upon. The implemented system was able to show the value of a modular laser combat system, but there are many aspects of the implemented system that were limited by both time and budget. In this chapter, the specific areas of improvement and intended future work is explored

7.1 Enhanced Optical Transmission/Reception

The blatantly obvious improvement for enhancement of the FSO simulated weapon would be to change the infrared wavelength on which it operates. As discussed previously, water molecules in the air significantly attenuate light transmission at 940nm, the exact wavelength used in the implemented version of this system. New hardware must be found that will operate at a more atmospheric-friendly wavelength. 850nm is

another common IR communication wavelength and does not have the same negative atmospheric attenuation.

As seen in Chapter 5, the collimating lens assembly fell short of the desired performance. Further investigation should be done to find the ideal lens setup for this application. The acrylic lens used in this project functioned, but a slightly larger lens, perhaps of a different material may give better transmission and collimation control. Care must be taken, however, to ensure that optimal lens performance is balanced with size and weight restrictions as well as the mechanical robustness requirements of a UAV.

The TFMS5400 sensors used in this project provided for simple implementation, but lacked advanced performance. These sensors were extremely susceptible to power supply noise and solar interference. Also, because each sensor in an array operated independently, a transmission was only received if the signal was strong enough to activate one sensor by itself. Even though they sensed independently, their open collector outputs were chained together so that there was no way to determine which sensor(s) received the transmission signal.

A new sensor array should be developed that is much more immune to power supply and solar interference. This new array should also consider the signals received by the array, as a whole, in addition to independent measurement. This summing of responses from all sensors in an array could improve reception fidelity during off-axis reception between multiple sensors. The signal may not be strong enough to activate any one sensor, but the sum of the signal between sensors may be enough to get an accurate

reading. While still considering each sensor independent of one another in the array, the new array could also give a reading as to the direction of the source of the transmission.

7.2 Combat Module Enhancements

Further improvement of the Combat Module itself could render the system much more useful. Some of these enhancements, as well as the IR communication improvements discussed above may require a more advanced microcontroller. There are many potential options for better microcontroller, but the main metrics of desired improvement would be to move away from an 8-bit controller to at least a 16-bit, and increase the clock speed to more than the current 10MHz.

7.2.1 Pan and Tilt Weapon with Video Targeting

The ground station tested in this project showed the value of a pan and tilt weapon in this combat system. This feature should be miniaturized and ruggedized for use with the Combat Module. Ideally, the Combat Module would be modified to accept weapon modules. One weapon module could be the one that is currently being used, just the collimating lens assembly, while the new weapon module would be the new pan and tilt version of the weapon. Another addition that could be made to the weapon module is the ability for video targeting. A camera could be integrated into either the pan and tilt or the fixed weapon modules to aid in aiming.

7.2.2 Expanded Vehicle Communication Protocols

The vehicle communication protocols implemented here are quite basic and limited. This is perfectly fine for the vehicles that were tested, but unacceptable for a system that is to be used on combat ready UAVs. Support should be added for common

and flexible protocols such as SPI and I²C. This type of communication could open up a plethora of future functionality as new features would not necessarily require more I/O pins since all the communication between the vehicle and Combat Module could be conducted on a single bus.

7.3 Command Module Enhancements

7.3.1 Non-Proprietary Platform

One goal of this system is to be as flexible as possible. The Command Module is a device that is not necessarily unique in its functionality, but a proprietary device was built for this project. All that is really needed for the Command Module is a user interface and a method of IR communication. Rather than building both of these features into a single device, the user interface of existing devices could be used and a hardware IR communication module could be added if necessary.

For example a laptop could be used as the Command Module with a small IR communication module attached to a USB port. Alternatively, a smart phone such as an iPhone or Droid, could be used to communicate via Bluetooth with an IR communication module. The advantage to these user interface options is that they are widely available and only require an application to operate as the Command Module. The only required hardware would be an IR communication device.

7.3.2 GUI enhancement

Along with using these other devices for the user interface, the GUI for the Command Module could be improved dramatically. The limitations of the microcontroller on the implemented Command Module make an advanced intuitive

interface very difficult to produce. Laptops or smart phones, on the other hand, are geared for attractive and intuitive user interfaces.

In the implemented Command Module, all configuration registers were accessed directly by their address, and altered by their hexadecimal values. This is perfectly fine from a developers point of view since all data is directly accessible, but from a field user's point of view, this can be confusing and cumbersome. With a different user interface device, these configuration registers could be abstracted to a high-level, human readable format. The review of combat simulation statistics could also be improved to include informative graphics.

7.4 Conclusion

The small, unmanned vehicle industry currently lacks an affordable and flexible method for testing weapons systems. The Modular Laser Combat System presented here offers potential for meeting this need. The implemented combat system has shown the validity of such a collection of devices and has also revealed areas that are in need of future improvement.

REFERENCES

- [1] KENNETH W. FISCHER, MICHAEL R. WITIW, JEFFREY A. BAARS, and T. R. OKE, "ATMOSPHERIC LASER COMMUNICATION: NEW CHALLENGES FOR APPLIED METEOROLOGY," *Bulletin of the American Meteorological Society*, vol. 85, May. 2004, pp. 725-732.
- [2] Salahuddin Qazi, "Challenges In Outdoor and Indoor Optical Wireless Communications."
- [3] Al Naboulsi, Sizun H., and de Fornel F., "Propagation of optical and infrared waves in the atmosphere," *International Union of Radio Science General Assembly Proceedings*, 2006.
- [4] "Centre for Remote Imaging, Sensing and Processing, CRISP."
- [5] Princeton University, "WordNet - About WordNet," *WordNet: A lexical database for English*.
- [6] Tomas Heller, "F-22 Raptor," *RC.TomHe.net*.

- [7] Dave Powers, “KF Airfoils :: explained by Dave Powers,” *RCPowers.com: One More Flight*.
- [8] “Universal asynchronous receiver/transmitter,” *Wikipedia, the free encyclopedia*.
- [9] “Servo Tutorial - How a Servo Works,” *Hooked-on-RC-Airplanes.com*.
- [10] Cy Brown, “HobbyZone X Port,” *Pasqualy.com*.
- [11] Paul Constance, “Army gets by with simulation; shift of money and technical priorities keep Army warfighting training alive,” *Government Computer News*, Jun. 1996.
- [12] N. Oreskes, K. Shrader-Frechette, and K. Belitz, “Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences,” *Science*, vol. 263, 1994, pp. 641–646.
- [13] Gürsel Alıcı, Romuald Jagielski, Y. Ahmet Şekercioğlu, and Bijan Shirinzadeh, “Prediction of geometric errors of robot manipulators with Particle Swarm Optimisation method,” *Robotics and Autonomous Systems*, vol. 54, 2006, pp. 956–966.
- [14] Jung W. Bang, Randall E. Schumacker, and Paul L. Schlieve, “Random-Number Generator Validity in Simulation Studies: An Investigation of Normality,” *Educational and Psychological Measurement*, vol. 58, Jun. 1998, pp. 430–448.
- [15] “NAVAIR Overview,” Nov. 2009.
- [16] K. Konno, O. Matsushima, D. Navarro, and M. Miura-Mattausch, “Limit of validity of the drift-diffusion approximation for simulation of photodiode characteristics,” *Applied Physics Letters*, vol. 84, Feb. 2004, pp.

1398-1400.

- [17] Maria J. BAYARRI, James O. BERGER, Rui PAULO, Jerry SACKS, John A. CAFEO, James CAVENDISH, Chin-Hsu LIN, and Jian TU, “A Framework for Validation of Computer Models,” *TECHNOMETRICS*, vol. 49, May. 2007, pp. 138-154.
- [18] Cristiano Cervellera and Marco Muselli, “Deterministic Design for Neural Network Learning: An Approach Based on Discrepancy,” *IEEE TRANSACTIONS ON NEURAL NETWORKS*, vol. 15, May. 2004, pp. 533-544.
- [19] Stephen Low, *Fighter Pilot - Operation Red Flag*.

8. APPENDIX A: COMPUTER SIMULATION BACKGROUND

8.1 Value of Computer Modeling

In order to dispel the notion that the intent of this paper is to discount computer simulation for engineering testing purpose, we must review the benefits and value of computer simulation.

8.1.1 Cost

Engineering is a matter of business and business is a matter of money. For every product or service that an engineer produces, cost will generally be an important concern. One of the most expensive aspects to product production is verification and testing.

With a reasonable capital investment in computers, software, and training a company or institution is capable of testing virtually any product in any environmental condition using computer simulation. No physical material is expended during computer simulation tests so there are significant savings on a per-test basis. Consider a rocket

engine for putting a satellite into orbit. The fuel is expendable and extremely expensive itself. A live, physical test would burn all the fuel, making each test expensive. Computer simulation is cost effective when many parameters are tested during many test runs.

The military spends millions of dollars on live combat simulations every year. Thanks to “constructive simulations,” where mobile field assets are simulated on a computer while the command center is staffed by real soldiers and officers who run the exercises, the Army has been able to save hundreds of millions of dollars [11].

8.1.2 Safety

Before any physical tests are done, computer simulation also helps to ensure that the results of a live test are predictable. In the rocket engine example, before it is test fired for the first time, computer simulation can verify that everything will operate as expected so that no one is injured or killed by a purely trial and error method.

8.1.3 Logistic Simplicity

Testing products via computer modeling also offers the benefit of logistic simplicity. For every test that is run, nothing needs to be refueled, nothing needs to be inspected, nothing needs to be transported to the testing site. With computer simulation, there just is not any logistic overhead. By enabling engineers to run multiple tests with no overhead, a company can also reduce its time-to-market.

8.2 Insufficiency of Computer Modeling

While computer simulation offers much needed insights to the performance of a system, it is not an end-all solution. In any system that requires complete and absolute

reliability, computer simulation is simply not enough to verify the operation of the system.

8.2.1 The Problem of Incomplete Knowledge

The only way that a computer model can offer a perfectly accurate and reliable result in any simulation is if the system being modeled is closed [12]. The problem of numerical modeling is that the input parameters are never completely known and therefore, the system can never be closed.

For example, if I say “If it rains tomorrow, I will stay home and revise this paper.” The next day it rains, but you find that I am not home. Your verification has failed. You conclude that my original statement was false. But in fact, it was my intention to stay home and work on my paper. The formulation was a true statement of my intent. Later, you find that I left the house because my mother died, and you realize that my formulation was not false, but incomplete. It did not allow for the possibility of extenuating circumstances.^[12]

While this example is basic, it reveals that even the most simple systems suffer from this problem of incomplete knowledge. Numerical models can only account for the influences on a system that are already known, and even when they are known, they are almost assuredly only approximations of the truth. Alici laments that in the case of deterministic robotic arm manipulation,

Even if it is possible to dismantle a robot manipulator and determine the

parameters in its kinematic model, the resulting model will still contain some inaccuracies arising from joint and link compliances changing with the manipulator configurations, steady state errors in joint positions, inaccurate knowledge of the kinematic parameters, and payload carried by the manipulator.^[13]

For any physical system, it is reasonable to assume that it is impossible to consider all the possible sources of introduced error into the system [13]. Even when we do account for errors in a modeled system, they are often represented as some form of noise. This can pose a problem if the source of error in a system is not noise but just an unknown or under-defined deterministic source.

An even greater problem with the validity of generated noise in a model is that noise is supposed to be random but any “noise” generated by a model is only pseudo-random; randomness is approximated by a deterministic equation in any modeled system. Before this issue is dispelled on the idea that the randomness of a pseudo-random number generator is “close enough” to random, it should be noted that the outcomes of simulations run with different random number generators can vary significantly [14]. Depending on the application, there could be a different random number generator that gives optimal results in different situations. Using the wrong random number generator could result in misleading outcomes and, while this may happen only rarely, it can be disastrous [14].

8.2.2 Balance of Cost and Accuracy

Although using computer simulation can provide significant cost savings for a company or institution, it may not be enough. When proof and validation are the goals of a simulation, that simulation must be extremely accurate or essentially perfect. This stringent requirement on the accuracy can potentially require significant engineering time to create a very accurate and complete model.

NAVAIR, the civilian engineering company which is part of the United States Navy has been working on simulation models for the F-18 fighter jet since it was first used by the Navy in the 1970's. Hundreds of thousands of engineering hours have been put into the computer model and it has cost the Navy a lot for that accuracy. The Navy now does not have to run as many physical tests and even when they do run a physical test, they have already run their simulation and have a good idea of the characteristics of the results [15].

For small private companies and research institutions, the decades of work and associated costs for accurate models may not be practical. Private companies live and die based on time-to-market, hence getting accurate results quickly is important. Research institutions are most often restricted by very limited funding and cannot afford to make the same kind of investment that the Navy has made on the F-18.

The F-18 is a very complicated system so one may argue that of course it costs a lot to simulate it accurately. But even for a comparatively simple system like a photodiode, the typical simulation models are not enough to generate accurate and reliable results. Konno et al. have shown that using the Drift Diffusion approximation and

the Boltzmann Transport equation should lead to accurate results, but in fact only result in only moderate accuracy. They admit that, “In order to simulate the photodiode more accurately within the framework of the [Drift Diffusion] approximation, we need to modify mobility models” [16] So even though they were working on a simpler system, and had reasonable models, more work still must be done to ensure proper accuracy.

The real cost of generating accurate models is validation. We must confirm that a model represents reality accurately before it can be used with confidence. “Validation should be viewed as an accumulation of evidence to support confidence in the model outputs and their use, and the methodology should allow updating of current conclusions as additional information arrives” [17]. This updating of values and evaluation of new information requires either some adaptive learning approach, as in [18], or physical testing and experimentation. Adaptive learning is still only a budding field of interest and so we are left with only physical testing and experimentation to confirm model validity.

8.2.3 Lack of Proof

How can it be known that a model is an accurate and reliable representation of the real world without physical testing? It cannot. Since it has already been established that there is incomplete knowledge of any system in the real world, it cannot be known that the outputs of an incomplete model is accurate. Since there can be no proof, computer models are only good enough to corroborate hypotheses by offering evidence to strengthen what may already be partially established through other means, such as physical testing [12].

Going back to the F-18 example, NAVAIR was tasked with confirming that a F-18 could survive a direct hit on its engines by small arms fire from the ground during flight. The engines have a Kevlar blanket surrounding them to protect them from just such an incident. Using their incredibly detailed computer model, they simulated the event and the simulation showed that the F-18 could in fact survive the hit. But knowing that this was not something that could be left to simulation alone since lives were at stake, NAVAIR ran a live static engine test and fired on the engine with a rifle. To everyone's surprise the engine lit on fire and tore itself apart. That physical test revealed a flaw in the computer model of the F-18 that has since been fixed [15]. The simulation, in this case, was inaccurate and hence not proof of reality.

8.3 Forms of Live Simulation

Since computer simulations are not enough to generate reliable products for use on unmanned vehicles, the different methods of live testing of unmanned weapons systems must be explored.

8.3.1 Live Fire

Live fire is both the most definitive and the most expensive method of weapons system testing. Live fire is where everything happens in the physical world; the unmanned vehicle is actually flying, driving, etc., the weapons are active, and there is a physical target. Live fire offers the closest possible representation to actual combat without actually having an enemy to fire upon.

Live fire is obviously expensive because actual weapons are very expensive to fire. There are numerous permits and authorizations required for live testing, live testing

can only be done on designated test fields, targets are actually destroyed, and live testing is inherently dangerous. Aside from a few final tests, there are no small companies or research institutions that can afford to do live testing.

Figure 8.1: Live fire testing at Nellis AFB in Operation Red Flag [19].

8.3.2 Routine Execution

One way that is often used to test unmanned aerial vehicles is routine execution. Routine execution is accomplished by having vehicles fly a predetermined path given a set of waypoints and tasks. If the vehicle can be in the right place at the right time, it has a considerable probability of being able to fulfill its mission when equipped with a live targeting system.

This type of testing is only the first step in live testing and can only prove that the vehicle can go where it needs to go. It cannot test the targeting system and is therefore incomplete.

8.3.3 Simulated Weapon, Real Operation

Since live testing has been shown to be necessary and live fire is too costly and dangerous, another method to test targeting systems should be found. A simple, yet elegant solution is to conduct live tests of vehicles and targeting systems with simulated weapons. These simulated weapons can be attached to the vehicles and targeting systems and communicate with each other to determine correct targeting operation.

The United States Air Force has developed an elite training program for fighter pilots known as Operation Red Flag [19]. It is an international event where pilots and ground crews conduct rigorous training missions at Nellis Air Force Base, Nevada. While the program includes numerous live fire exercises, each fighter jet is fitted with a simulated weapons system to “fire” on targets and detect when it has been “hit” by another simulated weapon. All aircraft are monitored at a command station that can alert the pilots when they have been hit or if they hit another aircraft.

Figure 8.2: Dogfight training with simulated weapon, Nellis Air Combat Training System. Top: flight video of dogfight. Middle: animated version of actual fight. Bottom: animated cockpit view showing a successful hit [19].

The simulated weapons system used in Operation Red Flag is part of the Nellis Air Combat Training System seen in Figure 8.2. This system is capable of tracking all live aircraft that are part of the combat training mission. Animated depictions of the actual training mission are generated and monitored by the command center. With this system, the command center can see everything that any of the pilots can see and more.

This system provides an enormous amount of data for both training and testing. The problem with this system is the cost. There is no way that anyone except the Air Force could afford anything like it. So there is a void in the market for a low-cost simulated weapons system with some similar features to those used in Operation Red Flag. A system should be developed for the small business or research institution.

APPENDIX B: SCHEMATICS

V0.02

V0.02

Ground-based Turret Controller

APPENDIX C: BOARD LAYOUT

Combat Module

V0.02

V0.10

V0.11

V0.12

Command Module

V0.02

Button Board

Ground-based Turret Controller

APPENDIX D: C-CODE

Only the latest versions of the firmware are shown here. Previous versions using other compilers are not shown.

Combat Module

V0.13

```
/*
 * Project name:
 *   CobraTag Combat Module V0.13
 * Copyright:
 *   Thomas Fulenwider 2010
 * Description:
 *   This is a test program for the Modular Laser Combat System for ROVs and
UAVs
 * Test configuration:
 *   MCU:          PIC18F26K20
 *   Board:        CobraTagV0.11
 *   Oscillator:   XT, 10.0000 MHz
 *   Ext. Modules: -
 *   SW:          Hi-Tech PICC18 Compiler
 * NOTES:
 *   None.
 */

#include <htc.h>
#include <usart.h>
#include <conio.h>
#include <stdio.h>
```

```

#include <delays.h>

// CONFIGURE MCU *****
__CONFIG(1,IESODIS & FCMDIS & XT);
__CONFIG(2,BORV18 & BORDIS & PWRTDIS & WDTPS32K & WDTDIS);
__CONFIG(3,MCLREN & LPT1DIS & PBDIGITAL & CCP2RB3);
__CONFIG(4,LVPDIS & STVRDIS);

// __CONFIG PWRT          = OFF
// __CONFIG BOREN         = SBORDIS
// __CONFIG BORV          = 18
// __CONFIG WDTE          = OFF
// __CONFIG WDTPS         = 32768
// __CONFIG MCLRE         = ON
// __CONFIG LPT1OSC       = OFF
// __CONFIG PBDEN         = OFF
// __CONFIG CCP2MX        = PORTBE
// __CONFIG STVREN        = OFF
// __CONFIG LVP           = OFF
//use default values for other config bits
//*****

// *** Default EEPROM Data ***
__EEPROM_DATA(0x55,0xBB,0x00,0x9F,0x64,0x02,0x00,0x02);
__EEPROM_DATA(0x00,0x00,0x11,0xAA,0xBA,0x65,0x00,0x00);

// *** Define I/O Aliases ***
#define MOT_IN      RB0
#define FIRE_IN     RB1
#define LED_PWR     RB4
#define BUZZ        RB5
#define MOT_OUT     RC2
#define PWM_OFF     TRISB3
#define TX          RC6

// *** Define Register Aliases ***
#define TMR1ON      T1CONbits.TMR1ON
#define TMR3ON      T3CONbits.TMR3ON
#define INTEDG0     INTCON2bits.INTEDG0
#define INTEDG1     INTCON2bits.INTEDG1

// *** Define Constants ***
#define SERVO_HOLD  5781
#define RXWait      2812      //maximum allowed separation between grouped UART
data (~9ms)
#define CNFG_BYTS   16
#define PLAYER1     11        //EEPROM location of first player ID
                                //each module considers itself PLAYER0
#define COMMAND     0xCC      //ID# of the command module
#define ENDCOM      0x00      //ID# of the command module
#define ACK         0x00      //acknowledge TX, sent as USART but read
as low pulse
#define HITWait     0          //starting value for hit timer (smaller =
longer)
#define HITCount    4          //number of times to run through hit timer - 1
#define BURSTLIM    3          //number of shots in a burst

/* *** Noteworthy Equations *** */
*      1 instruction cycle = 4 / 10Mhz = 0.4us

```

```

        therefore 100ms = 250,000 instruction cycles
*/

// *** Global Variables ***
unsigned char ctrl, pwr, IDa, IDb, IDaRX, IDbRX, RXData[6],RXNum;
unsigned int Motor, MotorOUT, MotMin, MotRange, FireIN, FireTHRESH;
unsigned char player[8] = {0,0,0,0,0,0,0,0}, playerHits[8] = {0,0,0,0,0,0,0,0},
players;
unsigned char hitTime = HITCount, burst, tempEEPROM[16];

typedef struct {
    unsigned group:1;
    unsigned vector:1;
    unsigned dataRec:1;
    unsigned FIRE:1;
    unsigned HIT:1;
    unsigned COM:3;
}RXSTATE;
volatile RXSTATE RXState;

void ReadCONFIG(){
    unsigned char i;

    IDa = eeprom_read(0x00);
    IDb = eeprom_read(0x01);
    MotMin = eeprom_read(0x03);
    FireTHRESH = eeprom_read(0x0A); //0x11 for normal,0x6E for X-port
    MotRange = eeprom_read(0x04) + (eeprom_read(0x05) << 8);
    ctrl = eeprom_read(0x09);

    players = eeprom_read(7);
    for (i=0; i<(players-1); i++){
        player[i] = eeprom_read(i+PLAYER1);
    }
}

void initialize(void){
    TRISA = 0xF6; //PORTA all inputs except RA0/3
    TRISB = 0x0B; //all outputs except MOT_IN and FIRE_IN and PWM(for now)
    TRISC = 0xA0; //all outputs except UART RX and RC5

    PORTA = 0x00; //PORTA unused
    PORTB = 0x00; //
    PORTC = 0x40; //idle TX high

    INTCONbits.INT0IE = 1; //enable MOT_IN interrupt
    INTCON3bits.INT1IE = 1; //enable FIRE_IN interrupt
    INTCON3bits.INT1IP = 1; //high priority FIRE_IN interrupt
    INTEDG0 = 1; //Set input interrupts to rising edge
    INTEDG1 = 1;

    PWM_OFF = 1; //disable PWM output
    PR2 = 62; //Set PWM freq to 39.682kHz
    = 1/((62 + 1) (4/10MHz))
    CCP2CON = 0x0F; //Set CCP to PWM mode
    CCP2L = 25; //Set PWM duty to ~40%
    PIR1bits.TMR2IF = 0;
    T2CON = 0x04; //Open Timer2 for PWM 1:1 pre- and
    postscalers

```

```

    /*TXSTA = 0b00100000;           //asynchronous transmit, low speed
    RCSTA = 0b10010000;           //continuous receive, enabled, 8-bit
non-parity
    BAUDCON = 0b00000000;         //normal UART mode
    SPBRG = 64;                   //set baud rate to ~2400
(2403.8)*/

//Initialize UART to 2400 baud
OpenUSART(    USART_TX_INT_OFF &
             USART_RX_INT_ON &
             USART_ADDEN_OFF &
             USART_BRGH_LOW &
             USART_CONT_RX &
             USART_EIGHT_BIT &
             USART_ASYNC_MODE,
             64);

//UART RX timeout Timer
T0CON = 0b00000010; //16bit, 1:8, off, source is instruction clk

//PWM input timer
T1CON = 0b10110000; //16bit, 1:8, basic timer, off
PIELbits.TMR1IE = 0; //disable TMR1 interrupt

//Motor PWM output timer
T3CON = 0b10110001; //16bit, 1:8, basic timer, on
PIE2bits.TMR3IE = 1; //enable TMR3 interrupt
IPR2bits.TMR3IP = 1; //high priority interrupt

ReadCONFIG();

MotorOUT = 0;
Motor = 0;
pwr = 0;
RXState.group = 0;
RXState.vector = 0;
RXState.dataRec = 0;
RXState.FIRE = 0;
RXState.HIT = 0;
RXState.COM = 0;

//Ready Alert
BUZZ = 1;
Delay_ms(20); //delay 20ms
BUZZ = 0;
Delay_ms(40); //delay 40ms
BUZZ = 1;
Delay_ms(20); //delay 20ms
BUZZ = 0;

RCONbits.IPEN = 1; //enable interrupt priority distiction
INTCONbits.GIEL = 0; //disable low priority interrupts
INTCONbits.GIEH = 1; //enable high priority interrupts
}

void cmd_com() {
    unsigned char i;
    LED_PWR = 0; //Ensure low power transmission
    //BUZZ = 1;

```

```

    if(IDaRX == COMMAND){
        if(IDbRX == COMMAND) RXState.COM = 0b111;
        if(IDbRX == ENDCOM){
            RXState.COM = 0b000;
            for (i=0;i<CNFG_BYTS;i++) {
                eeprom_write(i,tempEEPROM[i]);
            }
            initialize();
            //BUZZ = 0;
        }
    } else {
        tempEEPROM[IDaRX] = IDbRX;
    }
    //BUZZ = ~BUZZ;
    //RXState.COM = IDbRX;
}

char analyzeRX() {
    unsigned char valid = 1;
    unsigned char i;

    if (RXData[0] == RXData[2]) {
        IDaRX = RXData[0];
    } else if (RXData[0] == RXData[4] || RXData[2] == RXData[4]) {
        IDaRX = RXData[4];
    } else { valid = 0; }
    if (RXData[1] == RXData[3]) {
        IDbRX = RXData[1];
    } else if (RXData[1] == RXData[5] || RXData[3] == RXData[5]) {
        IDbRX = RXData[5];
    } else { valid = 0; }
    if (IDaRX == COMMAND || RXState.COM == 0b111) {
        valid = 0;
        cmd_com(); //if hit by the command module, communicate with it
    }
    if (valid) { //if valid data, check if it is from a known
player
        valid = 0;
        for (i=0; i<(players-1); i++){
            if (player[i] == IDaRX){
                valid = 1;
                playerHits[i]++;
            }
        }
    }

    RXData[0] = 0; //clear received data buffer
    RXData[1] = 0;
    RXData[2] = 0;
    RXData[3] = 0;
    RXData[4] = 0;
    RXData[5] = 0;
    return valid;
}

void hit() {
    unsigned int tmpTime = HITWait;
    TMR0H = tmpTime >> 8;
}

```

```

    TMROL = tmpTime;
    MotorOUT = MotMin;
    BUZZ = 1; //turn buzzer on
    if (ctrl == 1){ //if in X-Port mode
        MOT_OUT = 0; //pull Motor output low
        TRISC2 = 0; //enable Motor output
    }
    TMROON = 1; //enable hit delay timer
}

void interrupt isr(void){
    unsigned int tmpTime;
    //INTCONbits.GIEH = 0; //disable high priority interrupts

    //HANDLE UART RX TIMEOUT
    if (TMR0IF) {
        //PORTB.F1 = ~PORTB.F1;
        TMR0IF = 0;
        if (!RXState.HIT) { //handle GROUP DELAY TIMEOUT (not yet
hit)
            PIELbits.RCIE = 0; //disable RX INT while analyzing data
            RCSTAbits.SPEN = 0; //disable serial port
            TX = 1; //ensure TX remains high
            TMROON = 0; //turn group delay timer off
            BUZZ = 0; //turn buzzer off
            if (analyzeRX()) {
                RXState.HIT = 1; //set the hit flag
                hit();
            } else {
                PIELbits.RCIE = 1; //enable RX INT if not a valid hit
                RCSTAbits.SPEN = 1; //enable serial port
            }
            RXState.group = 0; //clear group bit after timeout
        } else {
            if (hitTime > 0) { //handle HIT TIMER
                hit();
                hitTime--;
            } else {
                TMROON = 0; //disable hit delay timer
                MotorOUT = Motor;
                BUZZ = 0; //turn buzzer off
                if (ctrl == 1){
                    TRISC2 = 1; //disable Motor output
                }
                hitTime = HITCount;
                RXState.HIT = 0; //clear hit flag
                PIELbits.RCIE = 1; //enable RX interrupt
                RCSTAbits.SPEN = 1; //enable serial port
            }
        }
    }

    //HANDLE UART RX INPUT
    if(PIR1bits.RCIF){
        if (!RXState.group) { //is received byte in a new
group?
            RXState.vector = ~RXState.vector; //toggle active input vector
            RXNum = 0; //clear number of valid bytes
            TMROON = 1; //enable group delay timer

```

```

        RXState.group = 1;
    }
    RXData[RXNum] = ReadUSART();
    if (RXNum < 5){ //overwrite 6th byte if more than 6 bytes
        are received (avoids unknown states)
        RXNum += 1;
    }
    tmpTime = 0xFFFF - RXWait; //reset group delay timer
    TMR0H = tmpTime >> 8;
    TMR0L = tmpTime;
}

//HANDLE MOT_IN INPUT
if(INTCONbits.INT0IF){
    INTCNbits.INT0IF = 0;
    if(MOT_IN){ //time high pulse
        TMR1ON = 1;
        INTEDG0 = 0; //check at falling edge
    } else { //measure pulse and reset
pulse timing
        tmpTime = TMR1L;
        tmpTime += TMR1H * 256; //read timer value
        Motor = tmpTime - MotMin;
        if(!RXState.HIT){
            MotorOUT = Motor;
        }
        TMR1ON = 0;
        TMR1H = 0x00;
        TMR1L = 0x00;
        INTEDG0 = 1; //wait for next rising pulse
    }
}

//HANDLE FIRE_IN INPUT
if(INTCON3bits.INT1IF){
    INTCN3bits.INT1IF = 0;
    if(ctrl ^ FIRE_IN){ //time high pulse for normal (RC ctrl=0), low
pulse for X-Port(RC ctrl=1)
        TMR1ON = 1;
        if(ctrl){
            INTEDG1 = 1; //check next rising edge for X-Port
        } else {
            INTEDG1 = 0; //check next falling edge for normal RC
        }
    } else { //measure pulse and reset
pulse timing
        TMR1ON = 0;
        tmpTime = TMR1L;
        tmpTime += TMR1H * 256;
        tmpTime /= 32;
        FireIN = tmpTime;
        if(FireIN > FireTHRESH){
            if(ctrl){
                RXState.FIRE = 0;
                burst = 0;
            } else {
                RXState.FIRE = 1;
            }
        } else {
            if(ctrl){

```

```

        } else {
            RXState.FIRE = 0;
            burst = 0;
        }
    }
    TMR1H = 0x00;
    TMR1L = 0x00;
    if(ctrl){
        INTEDG1 = 0; //check next falling edge for X-Port
    } else {
        INTEDG1 = 1; //check next rising edge for normal RC
    }
}

//HANDLE RC MOTOR OUTPUT
if(PIR2bits.TMR3IF){
    PIR2bits.TMR3IF = 0;
    MOT_OUT = ~MOT_OUT;
    if(MOT_OUT){ //set timer for high pulse output
        tmpTime = 0xFFFF - (MotorOUT + MotMin);
    } else { //set timer for PWM low
        tmpTime = 0xFFFF - (SERVO_HOLD);
    }
    TMR3H = tmpTime / 256;
    TMR3L = tmpTime;
}

//INTCONbits.GIEH = 1; //enable high priority interrupts
}

void main(void){
    unsigned char i,j;
    initialize();
    RXState.FIRE = 0;

    while(1){
        //BUZZ = 0;
        if(RXState.FIRE && !RXState.HIT && burst < BURSTLIM){
            BUZZ = 1;
            PWM_OFF = 0;
            if(pwr) {LED_PWR = 1;}
            WriteUSART(IDa);
            WriteUSART(IDb);
            while(BusyUSART()); //wait for byte to be sent
            Delay_ms(1); //1ms
            WriteUSART(IDa);
            WriteUSART(IDb);
            while(BusyUSART()); //wait for byte to be sent
            Delay_ms(1); //1ms
            WriteUSART(IDa);
            WriteUSART(IDb);
            while(BusyUSART()); //wait for byte to be sent
            //PORTCbits.RC6 = 1;
            PWM_OFF = 1;
            LED_PWR = 0;
            BUZZ = 0;
            burst++;
        }
    }
}

```



```

        j=0;
        while(j<2){
            Delay_ms(300);          //delay 300ms
            j++;
        }
    }
}

```

Command Module

V0.05

```

/*
 * Project name:
 *   CobraTag Command Module V0.05
 * Copyright:
 *   Thomas Fulenwider 2010
 * Description:
 *   This is a test program for the Modular Laser Combat System for ROVs and
UAVs
 * Test configuration:
 *   MCU:          PIC18F26K20
 *   Board:        CobraTagV0.11
 *   Oscillator:   XT, 10.0000 MHz
 *   Ext. Modules: -
 *   SW:          Hi-Tech PICC18 Compiler
 * NOTES:
 *   None.
 */

#include <htc.h>
#include <usart.h>
#include <conio.h>
#include <stdlib.h>
#include <lcd.h>
// #include <delays.h>
#include <adc.h>

// CONFIGURE MCU *****
__CONFIG(1,IESODIS & FCMDIS & XT);
__CONFIG(2,BORV18 & BORDIS & PWRTDIS & WDTPS32K & WDTDIS);
__CONFIG(3,MCLREN & LPT1DIS & PBDIGITAL & CCP2RB3);
__CONFIG(4,LVPDIS & STVRDIS);

// CONFIGURE DEFAULT EEPROM CONFIG BITS *****
__EEPROM_DATA(0xAA,0xBB,0x00,0x9F,0x64,0x02,0x00,0x02); //P1
__EEPROM_DATA(0x00,0x00,0x11,0x55,0xCA,0x75,0x00,0x00);
__EEPROM_DATA(0x55,0x66,0x00,0x9F,0x64,0x02,0x00,0x02); //P2
__EEPROM_DATA(0x00,0x00,0x11,0xAA,0xCA,0x75,0x00,0x00);
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00); //P3
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00);
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00); //P4
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00);
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00); //P5
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00);

```

```

__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00); //P6
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00);
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00); //P7
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00);
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00); //P8
__EEPROM_DATA(0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00);

//Define I/O pin aliases
#define X_ANA 1
#define Y_ANA 0
#define V_SENS 3
#define BUT0 RB0
#define BUT1 RB1
#define PWM RB3
#define PWM_OFF TRISB3
#define LED0 RC0
#define LED1 RC1
#define TX RC6
#define RX RC7

//Declare LCD Commands
#define _LCD_CLEAR 0x01

//Define UI Constants
#define JOY_TH 150
#define JOY_MID 530
#define LEVS 3 //number of menu levels
#define OPTS 4 //number of main menu options
#define SUBS 2 //maximum number of sub menu
options

//Define Communication Constants
#define RXWait 4096 //maximum allowed separation between grouped
UART data
#define COMMAND 0xCC //ID# of the command module
#define ENDCOM 0x00 //ID# of the command module
#define ACK 0xFF //acknowledge code for received commands/data
#define TRAIN 0x11 //weapon code for RC system training
#define CONFIG 0x22 //weapon code for combat module
configuration
#define PLAY 0x33 //weapon code for game
start initialization
#define DNLD 0x44 //command to download data
from combat module
#define CHG_PLAY 0x55 //command to change player ID number
#define SHRT_PULS 4 //number of short pulses before fire COMMAND is
registered (X-Port)
#define PLAYER1 11 //EEPROM location of first player ID
//each module considers itself PLAYER0

#define CNFG_BYTS 16
#define STAT_BYTS 8

//const unsigned char * cobraTag = "CobraTag";

//Declare global variables
unsigned char IDaRX, IDbRX, shortPulse, i,j, ctrl;
unsigned char RXData[6] = {0,0,0,0,0,0};

```

```

unsigned char player = 0, team = 0, teams = 8, players = 8, reg = 0, stat = 0;
unsigned char RXNum = 0;
unsigned int X, Y, V;
unsigned char Xchar[7], Ychar[7];

struct BITF{
    unsigned B0:1;
    unsigned B1:1;
    unsigned B2:1;
    unsigned B3:1;
    unsigned B4:1;
    unsigned B5:1;
    unsigned B6:1;
    unsigned B7:1;
};

typedef union{
    struct BITF bits;
    unsigned char data;
}BITFIELD;
volatile BITFIELD dnldState;

typedef struct {
    unsigned RXgroup:1;
    unsigned RXvector:1;
    unsigned RXdataRec:1;
    unsigned RXupld:1;
    unsigned RXdnld:1;
}RXSTATE;
volatile RXSTATE RXState;

typedef struct {
    unsigned lev:2;                                //current menu level (main level =
0)
    unsigned sub:3;                                //current sub menu option
    unsigned opt:3;                                //current main menu option
}MStat;
volatile MStat MenuStat;

unsigned char dataString[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
unsigned char REGISTERS[8][16] = {{0,6,0,10,0,2,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};

unsigned char STAT[8][8] = {{0,6,0,10,0,2,0,0},
    {0,0,0,0,0,0,0,0},
    {0,9,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0}};

unsigned int ADC_Read(unsigned char channel){
    unsigned int result;
    ADCON0 &= 0b00000011;
    ADCON0 |= (channel << 2);

```

```

        GODONE = 1;
        while (GODONE);
        result = ADRESH;
        return (result << 8) + ADRESL;
    }

void DelayFor18TCY(void) {

}

//void WriteLCD(unsigned char row,unsigned char col,const char * buffer){
//    SetDDRamAddr((row*0x40) + col);
//    putsXLCD( buffer );
//}
//void WriteVarLCD (unsigned char row,unsigned char col,unsigned * buffer){
//    SetDDRamAddr((row*0x40) + col);
//    putsXLCD( buffer );
//}

void Read_Config(void){
    for(j=0;j<8;j++){
        for(i=0;i<16;i++){
            REGISTERS[j][i] = eeprom_read(16*j + i);
        }
    }
}

void initialize(void) {
    ANSEL = 0x0B; //configure all pins as digital except
    for X_ANA, Y_ANA, and V_SENS //configure all other pins as digital
    ANSELH = 0x00; //configure X_ANA, Y_ANA, and V_SENS as
    inputs //Configure PORTB as output except for
    TRISA = 0x0F; //Configure PORTC as output except
    TRISB = 0x07;
    RB0/1 as BUT0/1
    TRISC = 0x80;
    for UART RX
    PORTA = 0x00;
    PORTB = 0x00;
    PORTC = 0x00;

    ADCON1 = 0x00; //use Vss and Vdd
    for reference //right
    ADCON2 = 0xAA;
    justified,12TAD,1/32 Fosc clk
    ADON = 1;

    PWM_OFF = 1; //disable PWM output
    PR2 = 62; //Set PWM freq to 39.682kHz
    = 1/((62 + 1) (4/10MHz))
    CCP2CON = 0x0F; //Set CCP to PWM mode
    CCPR2L = 25; //Set PWM duty to ~40%
    PIR1bits.TMR2IF = 0;
    T2CON = 0x04; //Open Timer2 for PWM 1:1 pre- and
    postscalars

```

```

//Initialize UART to 2400 baud
OpenUSART(    USART_TX_INT_OFF &
              USART_RX_INT_ON &
              USART_ADDEN_OFF &
              USART_BRGH_LOW &
              USART_CONT_RX &
              USART_EIGHT_BIT &
              USART_ASYNC_MODE,
              64);

OpenLCD(FOUR_BIT & LINES_5X7);
WriteLCDCmd(BLINK_OFF & CURSOR_OFF);
//WriteLCDCmd(_LCD_CLEAR);
WriteLCD(0,4,"CobraTag");
WriteLCD(1,1,"Command Module");

Read_Config();

//V = ADC_Read(V_SENS);

    T0CON = 0b00000010;          //TMR0 is 16-bit mode, 1:8, internal clk,
disabled
    T1CON = 0b10110000;          //TMR1 is 16-bit mode, 1:8, internal clk,
disabled
    T3CON = 0b10110000;          //TMR3 is 16-bit mode, 1:8, internal clk,
enabled

    PIE1 = 0; //0b00100001;          //Enable UART RX and TMR1 Interrupts
    PIE2 = 0b00000010;          //Enable TMR3 Interrupts
    IPR1 = 0b00100001;          //Set UART RX and TMR1 INTs to High Priority
    IPR2 = 0b00000010;          //Set TMR3 INT to High Priority

    INTCONbits.INT0IE = 1;          //Enable pin interrupt for BUT0
    INTCON3bits.INT1IE = 1;          //Enable pin interrupt for BUT1
    INTCON3bits.INT1IP = 1;          //Set BUT1 interrupt to high priority
    INTCON2bits.INTEDG0 = 1;          //Set input interrupts to rising edge
    INTCON2bits.INTEDG1 = 1;

    //ctrl = EEPROM_Read(0x0009);

    LED1 = 1;                      //activate LED1
    Delay_ms(200);                  //200ms
    LED1 = 0;                      //Turn off LED1
    Delay_ms(200);                  //200ms
    LED1 = 1;                      //activate LED1
    Delay_ms(200);                  //200ms
    LED1 = 0;                      //Turn off LED1

    RCONbits.IPEN = 1;
    INTCONbits.PEIE = 1;          //Enable peripheral interrupts
    INTCONbits.GIE = 1;          //Enable interrupts
}

void writeMenu() {
    unsigned char ind[5];
    unsigned char ind2[6];

    switch (MenuStat.lev) {
    case 0:

```

```

//MAIN MENU
WriteLCDCmd(_LCD_CLEAR);
if (MenuStat.opt >= 2) {
    WriteLCD(0,1,"DNLOAD");
    WriteLCD(1,1,"REVIEW");
    WriteLCD(MenuStat.opt-2,0,"");
    WriteLCD(MenuStat.opt-2,7,"");
    WriteLCD(0,15,"^");
} else {
    WriteLCD(0,1,"CONFIG");
    WriteLCD(1,1,"UPLOAD");
    WriteLCD(MenuStat.opt,0,"");
    WriteLCD(MenuStat.opt,7,"");
    WriteLCD(1,13,"...");
}
break;
case 1:
    //SubMenu #1
    WriteLCDCmd(_LCD_CLEAR);
    if (MenuStat.opt == 2){
        for (i=players;i>1;i--){
            utoa(&ind,i,10);
            WriteVarLCD(0,(i*2)-2,ind);
        }
        WriteLCD(0,0,"1");
        if (dnldState.bits.B0){
            WriteLCD(1,0,"");
        }
        if (dnldState.bits.B1){
            WriteLCD(1,2,"");
        }
        if (dnldState.bits.B2){
            WriteLCD(1,4,"");
        }
        if (dnldState.bits.B3){
            WriteLCD(1,6,"");
        }
        if (dnldState.bits.B4){
            WriteLCD(1,8,"");
        }
        if (dnldState.bits.B5){
            WriteLCD(1,10,"");
        }
        if (dnldState.bits.B6){
            WriteLCD(1,12,"");
        }
        if (dnldState.bits.B7){
            WriteLCD(1,14,"");
        }
    } else {
        utoa(&ind,player+1,10);
        WriteLCD(0,0,"Player");
        WriteVarLCD(0,8,ind);
        WriteLCD(MenuStat.sub,6,"<");
        WriteLCD(MenuStat.sub,10,">");
    }
}
if (MenuStat.opt == 3){
    utoa(&ind,reg+1,10);
    WriteLCD(1,0,"STAT");
    WriteVarLCD(1,8,ind);
}

```

```

        utoa(&ind,STAT[player][reg],10);
        WriteVarLCD(1,12,ind);
    }

    break;
case 2:
    //SubMenu #2
    if (MenuStat.opt == 1 || MenuStat.opt == 2){
        WriteLCD(1,9,"loading");
    }
    if (MenuStat.opt == 0){
        WriteLCDCmd(_LCD_CLEAR);
        utoa(&ind,reg+1,10);
        WriteLCD(0,0,"CONFIG");
        WriteVarLCD(0,8,ind);
        WriteLCD(1,0,"Value  x");
        utoa(&ind,REGISTERS[player][reg],16);
        WriteVarLCD(1,8,ind);
        WriteLCD(MenuStat.sub,6,"<");
        WriteLCD(MenuStat.sub,10,">");
    }
    if (MenuStat.opt == 3){
        WriteLCDCmd(_LCD_CLEAR);
        utoa(&ind,team+1,10);
        WriteLCD(0,0,"Team  ");
        WriteVarLCD(0,8,ind);
        WriteLCD(MenuStat.sub,6,"<");
        WriteLCD(MenuStat.sub,10,">");

        utoa(&ind,stat+1,10);
        WriteLCD(1,0,"STAT");
        WriteVarLCD(1,8,ind);
        utoa(&ind,STAT[team][stat],10);
        WriteVarLCD(1,12,ind);
    }
    break;
}
V = ADC_Read(V_SENS);
if(V < 589){//589
if(V < 574){//574
    WriteLCD(0,12,"BAT");
} else {
    WriteLCD(0,12,"bat");
}
}
}

void sendUART(unsigned short address,unsigned short outByte) {
    LED0 = 1;                //activate LED
    PWM_OFF = 0;              //activate PWM
    WriteUSART(address);
    WriteUSART(outByte);
    while(BusyUSART());       //wait for byte to be sent
    //_delay(2500);             //1ms
    WriteUSART(address);
    WriteUSART(outByte);
    while(BusyUSART());       //wait for byte to be sent
    //_delay(2500);             //1ms
    WriteUSART(address);
    WriteUSART(outByte);
}

```

```

    while(BusyUSART());          //wait for byte to be sent
    PWM_OFF = 1;                  //disable PWM
    LED0 = 0;                     //Turn off LED
}

unsigned char configCMBT(unsigned short player) {
    INTCONbits.GIE = 0;           //disable interrupts
    sendUART(COMMAND,COMMAND);
    RXState.RXupld = 1;
    Delay_ms(15);                  //5ms
    for (i=0;i<CNFG_BYTS;i++) {
        sendUART(i,REGISTERS[player][i]);
        //eeprom_write((16*player)+i,REGISTERS[player][i]);
        Delay_ms(15);             //4ms
    }
    sendUART(COMMAND,ENDCOM);
    //Delay_ms(20);                //20ms
    RXState.RXupld = 0;
    dnldState.data = 0;
    INTCONbits.GIE = 1;           //Enable interrupts
    return 1;
}

void interrupt_isr(void) {
    //unsigned int tmpTime;

    /*if (PIR1.TMR1IF) {
        //PORTB.F1 = ~PORTB.F1;
        PIR1.TMR1IF = 0;
        if (!RXdataRec) {         //handle GROUP DELAY TIMEOUT (data not
yet received)
            PIE1.RCIE = 0;        //disable RX INT while analyzing data
            RCSTA.SPEN = 0;       //disable serial port
            T1CON.TMR1ON = 0;     //turn group delay timer off
            PORTB.F5 = 0;         //turn buzzer off
            if (analyzeRX()) {
                RXdataRec = 1;    //set data received flag
                //hit();
            } else {
                PIE1.RCIE = 1;    //enable RX INT if not a valid data
                RCSTA.SPEN = 1;   //enable serial port
            }
            RXgroup = 0;         //clear group bit after timeout
        }
    }*/

    /*if (PIR2.TMR3IF) {          //handle PWM SERVO OUTPUT
        PIR2.TMR3IF = 0;
    }*/

    if (INTCONbits.INT0IF) {      //handle BUT0 input
        INTCONbits.INT0IF = 0;
        if (MenuStat.lev < LEVS-1){
            MenuStat.lev += 1;
            writeMenu();
            LED1 = 1;
        }
    }
}

```



```

if (INTCON3bits.INT1IF) {                                //handle BUT1 input
    INTCON3bits.INT1IF = 0;
    //Delay100TCYx(100);                                //4ms
    if (BUT1 == 1){
        if (MenuStat.lev > 0){
            MenuStat.lev -= 1;
            if (MenuStat.lev == 0){
                player = 0;
            }
            MenuStat.sub = 0;
            writeMenu();
        }
    }
}

/*if (PIR1.RCIF) {                                       //handle RX DATA
    //PORTB.F1 = ~PORTB.F1;
    if (!RXgroup) {                                     //is received byte in a new group?
        RXvector = ~RXvector;                         //toggle active input vector
        RXNum = 0;                                     //clear number of valid bytes
        T1CON.TMR1ON = 1;                             //enable group delay timer
        RXgroup = 1;
    }
    RXData[RXNum] = UART1_Read();
    RXNum += 1;
    tmpTime = 0xFFFF - RXWait;                         //reset group delay timer
    TMR1H = tmpTime >> 8;
    TMR1L = tmpTime;
}*/

}

void main(void){
    volatile MStat oldMenu;
    // _delay(100000);                                    //40ms
    initialize();
    //LED0 = 1;
    Delay_ms(250);
    Delay_ms(250);
    Delay_ms(250);
    Delay_ms(250);
    Delay_ms(250);
    Delay_ms(250);
    Delay_ms(250);
    Delay_ms(250);
    //LED0 = 0;
    WriteLDCmd(_LCD_CLEAR);
    MenuStat.lev = 0;
    MenuStat.sub = 0;
    MenuStat.opt = 0;
    writeMenu();

    while(1){
        //opt = MenuStat >> 5;
        //sub = (MenuStat >> 2) & 0x07;
        //lev = MenuStat & 0x03;
        X = ADC_Read(X_ANA);
        Y = ADC_Read(Y_ANA);
    }
}

```

```

        //itoa(Y,&Ychar);
        //itoa(X,&Xchar);
        //WriteVarLCD(0,2,&Ychar);
        //WriteVarLCD(0,10,&Xchar);

//      V = V * 13;
//      V = V / 101;

//IntToStr(X,Xchar);
//IntToStr(Y,Ychar);
if (X > JOY_MID + JOY_TH) {
    if (MenuStat.lev == 1 && MenuStat.opt != 2){
        if (MenuStat.sub == 0){
            if (player < players - 1){
                player += 1;
            }
        } else {
            if (reg < CNFG_BYTS - 1){
                reg += 1;
            }
        }
    } else if(MenuStat.lev == 2 && MenuStat.opt != 2){
        if (MenuStat.sub == 0){
            if (team < teams - 1){
                team += 1;
            }
        } else {
            if (stat < STAT_BYTS - 1){
                stat += 1;
            }
        }
    }
    if(MenuStat.lev == 2 && MenuStat.opt == 0){
        if (MenuStat.sub == 0){
            if (reg < CNFG_BYTS - 1){
                reg += 1;
            }
        }
        } else {
            if (REGISTERS[player][reg] < 0xFF){
                REGISTERS[player][reg] += 1;
            }
        }
    }
    writeMenu();
} else {
    if (X < JOY_MID - JOY_TH) {
        if (MenuStat.lev == 1 && MenuStat.opt != 2){
            if (MenuStat.sub == 0){
                if (player > 0){
                    player -= 1;
                }
            }
        } else {
            if (reg > 0){
                reg -= 1;
            }
        }
    }
    } else if(MenuStat.lev == 2 && MenuStat.opt != 2){
        if (MenuStat.sub == 0){
            if (team > 0){

```

```

        team -= 1;
    }
    } else {
    if (stat > 0){
        stat -= 1;
    }
    }
    if(MenuStat.lev == 2 && MenuStat.opt == 0){
        if (MenuStat.sub == 0){
            if (reg > 0){
                reg -= 1;
            }
        } else {
            if (REGISTERS[player][reg] > 0){
                REGISTERS[player][reg] -= 1;
            }
        }
    }
    }
    writeMenu();
}
}
if (Y > JOY_MID + JOY_TH) {
    if (MenuStat.lev != 0) {
        if (MenuStat.sub < SUBS - 1 && MenuStat.opt != 2){ //&&
MenuStat.opt > 2
        if(MenuStat.opt == 3) MenuStat.sub += 1;
        else if(MenuStat.lev == 2) MenuStat.sub += 1;
        }
        } else {
            if (MenuStat.opt < OPTS - 1) {
                MenuStat.opt += 1;
            }
        }
    } else {
        if (Y < JOY_MID - JOY_TH) {
            if (MenuStat.lev != 0) {
                if (MenuStat.sub > 0 && MenuStat.opt != 2) {
                    if(MenuStat.opt == 3) MenuStat.sub -= 1;
                    else if(MenuStat.lev == 2) MenuStat.sub -= 1;
                }
            } else {
                if (MenuStat.opt > 0) {
                    MenuStat.opt -= 1;
                }
            }
        }
    }
}

if (MenuStat.lev == 2) {
    if (MenuStat.opt == 1){
        LED1 = 0;
        MenuStat.lev = configCMBT(player);
        writeMenu();
    }
    if (MenuStat.opt == 2 && !RXState.RXdnlld){
//        writeMenu();
//        dnldCMBT();
    }
}

```

```

    }

    //MenuStat = 0;
    if ( MenuStat.opt != oldMenu.opt ||
        MenuStat.lev != oldMenu.lev ||
        MenuStat.sub != oldMenu.sub ) writeMenu();
    oldMenu = MenuStat;
    Delay_ms(150); //200ms
    }
}

```

Ground-based Targeting Device Controller

V0.01

```

/*
 * Project name:
 *   Ground Station Computer V0.01
 * Copyright:
 *   Thomas Fulenwider 2010
 * Description:
 *   This is a test program for an Onboard RC Computer that processes and mixes
RC inputs
 *   and controls the aircrafts motor/servos
 * Test configuration:
 *   MCU:          PIC18LF1320
 *   Board:        Onboard Computer V0.01
 *   Oscillator:   XT, 10.0000 MHz
 *   Ext. Modules: -
 *   SW:           HiTech PICC18
 * NOTES:
 *   None.
*/

#include <htc.h>
#include <stdlib.h>

// CONFIGURE MCU *****
__CONFIG(1,FCMEN & XT & IESOEN);
__CONFIG(2,BORV20 & BORDIS & PWRTDIS & WDTPS32K & WDTDIS);
__CONFIG(3,MCLREN);
__CONFIG(4,LVPDIS & STVREN & DEBUGDIS);

//*****

#define IN1 RB0
#define IN2 RB5
#define IN3 RB6
#define IN4 RB7
#define IN5 RB2
#define IN6 RB1

#define PAN IN1
#define TILT IN2
#define FIRE IN5

```

```

#define OUT1 RA3
#define OUT2 RA2
#define OUT3 RA5
#define OUT4 RA4
#define OUT5 RA1
#define OUT6 RA0
#define OUTPUTS PORTA

#define LED1 RB3
#define LED2 RB4

unsigned char i, oldIN = 0;
unsigned char outState, ChInc[6] = {0,0,0,0,0,0}, Inc[6] = {0,0,0,0,0,0};
unsigned int sigMin[6], sigMax[6], outMin = 300, outMax = 1200;
int in[6] = {0,0,0,0,0,0}, out[6] = {0,0,0,0,0,0}, centerOUT[6] =
{300,300,300,300,300,300};

void Delay_ms(unsigned char ms){
    unsigned char idx;
    for(idx=0;i<ms;i++){
        _delay(2500);
    }
}

void initialize() {
    unsigned short tmp;
    ADCON1 = 0xFF; //Configure pins as digital IO
    TRISB = 0xE7; //Configure PORTB as input except for RB3/4 as
LED out
    INTCON2bits.RBPU = 1; //disable PortB pull-up
    TRISA = 0x20; //Configure PORTA as output, RA5 always input
    PORTB = 0x00;

    T0CON = 0b00000010; //TMR0 is 16-bit mode, 1:8, internal clk, disabled
    T1CON = 0b10110000; //TMR1 is 16-bit mode, 1:8, internal clk, disabled
    T3CON = 0b10110000; //TMR3 is 16-bit mode, 1:8, internal clk, enabled

    //INTCON.TMR0IE = 0; //Enable TMR0 Interrupt
    //PIE1 = 0b00000001; //Enable TMR1 Interrupt
    PIE2 = 0b00000010; //Enable TMR3 Interrupt
    INTCON2bits.TMR0IP = 1; //Set TMR0 INT to High Priority
    IPR1 = 0b00000001; //Set TMR1 INT to High Priority
    IPR2 = 0b00000010; //Set TMR3 INT to High Priority

    INTCONbits.INT0IE = 1; //Enable pin interrupt for IN_1
    INTCONbits.RBIE = 1; //Enable pin interrupt for IN_2...IN_4
    INTCON3bits.INT1IE = 1; //Enable pin interrupt for IN_6
    INTCON3bits.INT2IE = 1; //Enable pin interrupt for IN_5
    INTCON3bits.INT1IP = 1; //Set IN_6 interrupt to high priority
    INTCON3bits.INT2IP = 1; //Set IN_5 interrupt to high priority
    INTCON2bits.RBIP = 1; //Set IN_2...IN_4 interrupts to high priority
    INTCON2bits.INTEDG0 = 1; //Set input interrupts to rising edge
    INTCON2bits.INTEDG1 = 1;
    INTCON2bits.INTEDG2 = 1;

    //ThreshHi = eeprom_read(0x0A);

    outState = 1;

    out[0] = centerOUT[0];

```

```

out[1] = centerOUT[1];

LED1 = 1;
Delay_ms(20);
LED1 = 0;
Delay_ms(20);
LED1 = 1;
Delay_ms(20);
LED1 = 0;

//INTCONbits.PEIE = 1;           //Enable peripheral interrupts
INTCONbits.GIE = 1;             //Enable interrupts
}

/*void checkCenters() {
    while (in[5] == 0){}
    INTCONbits.GIE = 0;           //Disable interrupts
    Delay_ms(100);
    centerIN[0] = in[0];
    centerIN[1] = in[1];
    centerIN[2] = in[2];
    centerIN[3] = in[3];
    centerIN[4] = in[0];
    centerIN[5] = in[0];
    centerOUT[0] = in[0];
    centerOUT[1] = in[1] - ELEV_OFF;
    centerOUT[2] = in[2];
    centerOUT[3] = in[3];
    centerOUT[4] = in[0]+40;
    centerOUT[5] = in[0]-40;
    pastFlaps = in[5] - centerOUT[5];
}*/

void computeOuts() {
    int pan;
    int tilt;
    int fire;

    pan = in[0] - centerOUT[0];
    tilt = in[1] - centerOUT[1];
    fire = in[4] - centerOUT[1];

    ChInc[0] = pan / 8;
    ChInc[1] = tilt / 8;

    if(Inc[0] == 0){
        Inc[0] = 32 - abs(ChInc[0]);
        if(ChInc[0] > 0){
            out[0] += 1;
        } else {
            if(ChInc[0] < 0){
                out[0] -= 1;
            }
        }
    } else {
        Inc[0] -= 1;
    }
}

```

```

}

void interrupt_isr() {
    unsigned int tmpTime;
    unsigned short test;

    if (PIR2bits.TMR3IF) { //handle PWM SERVO OUTPUT
        PIR2bits.TMR3IF = 0;
        switch (outState) {
            case 1:
                OUT1 = 1; //OUT1 = 1;
                break;
            case 2:
                OUT2 = 1;
                OUT1 = 0;
                break;
            case 3:
                OUT3 = 1;
                OUT2 = 0;
                break;
            case 4:
                OUT4 = 1;
                OUT3 = 0;
                break;
            case 5:
                OUT5 = 1;
                OUT4 = 0;
                break;
            case 6:
                OUT6 = 1;
                OUT5 = 0;
                break;
            default:
                T3CONbits.TMR3ON = 0;
                OUTPUTS = 0;
                outState = 0;
                break;
        }
        tmpTime = out[outState - 1];
        outState = outState + 1;
        tmpTime = 0xFFFF - tmpTime;
        TMR3H = (tmpTime >> 8);
        TMR3L = tmpTime;
    }

    if (INTCONbits.INT0IF) { //handle IN_1 INPUT
        INTCONbits.INT0IF = 0;
        if (IN1) { //if pulse detected, start timing it
            TMR0H = 0x00; //clear timer value
            TMR0L = 0x00;
            T0CONbits.TMR0ON = 1; //turn timer on
            INTCON2bits.INTEDG0 = 0; //check timer at next falling edge
        } else { //falling edge detected, clock pulse width
            T0CONbits.TMR0ON = 0; //turn timer off
            tmpTime = TMR0L;
            in[0] = tmpTime + (TMR0H << 8);
        }
    }
}

```

```

        INTCON2bits.INTEDG0 = 1;           //restart timer at next rising edge
    }
}

if (INTCON3bits.INT1IF) {                 //handle IN_6 INPUT
    INTCON3bits.INT1IF = 0;
    if (IN6) {                           //if pulse detected, start timing it
        TMR0H = 0x00;                   //clear timer value
        TMR0L = 0x00;
        T0CONbits.TMR0ON = 1;           //turn timer on
        INTCON2bits.INTEDG1 = 0;         //check timer at next falling edge
    } else {                             //falling edge detected, clock pulse width
        T0CONbits.TMR0ON = 0;           //turn timer off
        tmpTime = TMR0L;
        in[5] = tmpTime + (TMR0H << 8);
        INTCON2bits.INTEDG1 = 1;         //restart timer at next rising edge

        //computeOuts();
        tmpTime = 0xFF7F;               //Start output sequence
        TMR3H = (tmpTime >> 8);
        TMR3L = tmpTime;
        T3CONbits.TMR3ON = 1;
    }
}

if (INTCON3bits.INT2IF) {                 //handle IN_5 INPUT
    INTCON3bits.INT2IF = 0;
    if (IN5) {                           //if pulse detected, start timing it
        TMR0H = 0x00;                   //clear timer value
        TMR0L = 0x00;
        T0CONbits.TMR0ON = 1;           //turn timer on
        INTCON2bits.INTEDG2 = 0;         //check timer at next falling edge
    } else {                             //falling edge detected, clock pulse width
        T0CONbits.TMR0ON = 0;           //turn timer off
        tmpTime = TMR0L;
        in[4] = tmpTime + (TMR0H << 8);
        INTCON2bits.INTEDG2 = 1;         //restart timer at next rising edge
    }
}

if (INTCONbits.RBIF) {
    test = PORTB;
    INTCONbits.RBIF = 0;
    test = test >> 5;
    if (test == 0) {                     //falling edge detected, clock pulse width
        T1CONbits.TMR1ON = 0;           //turn timer off
        tmpTime = TMR1L;
        switch (oldIN) {
            case 1:
                in[1] = tmpTime + (TMR1H << 8);
                break;
            case 2:
                in[2] = tmpTime + (TMR1H << 8);
                break;
            case 4:
                in[3] = tmpTime + (TMR1H << 8);
                LED1 = 0;
                break;
        }
    } else {                             //if pulse detected, start timing it

```



```

        oldIN = test;
        LED1 = 1;
        TMR1H = 0x00;           //clear timer value
        TMR1L = 0x00;
        T1CONbits.TMR1ON = 1;   //turn timer on
    }
}

void main() {
    //Delay_ms(100);
    INTCONbits.GIE = 0;         //Disable interrupts
    initialize();
    //checkCenters();

    //
    // PORTB = 0xFF;             //LEDs on
    // Delay_ms(2);
    // PORTB = 0;                //LEDs off
    // Delay_ms(100);
    // PORTB = 0xFF;             //LEDs on
    // Delay_ms(2);
    // PORTB = 0;                //LEDs off

    //OUTPUTS = 0;

    while(1){
        NOP();
    }
}

```

Other Functions Written

Delay_10us()

```

#include <htc.h>
#include <delays.h>

void Delay_10us(unsigned char us){
    unsigned char ind;
    for(ind=0;ind<us;ind+=1){
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
    }
}

```

```

        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
    }
}

```

Delay_ms()

```

#include <htc.h>
#include <delays.h>

void Delay_ms(unsigned char ms){
    unsigned char ind;
    for(ind=0;ind<ms;ind+=1){
        Delay_10us(100);
    }
}

```

OpenLCD()

```

#include <htc.h>
#include <lcd.h>

/*****
*      Function Name:  OpenLCD                               *
*      Return Value:   void                                   *
*      Parameters:     lcdtype: sets the type of LCD (lines) *
*      Description:    This routine configures the LCD. Based on *
*                     the Hitachi HD44780 LCD controller. The *
*                     routine will configure the I/O pins of the *
*                     microcontroller, setup the LCD for 4- or *
*                     8-bit mode and clear the display. The user *
*                     must provide three delay routines:        *
*                     DelayFor18TCY() provides a 18 Tcy delay *
*                     DelayPORXLCD() provides at least 15ms delay *
*                     DelayXLCD() provides at least 5ms delay  *
*****/
void WriteLCDInit(unsigned char command){
    RS_PIN = 0;
    DATA_PORT = command & 0xF0;
    E_PIN = 1;
    E_PIN = 0;
}

void OpenLCD(unsigned char lcdtype){
    Delay_ms(15); //wait for power up
    WriteLCDInit(0x30);
}

```

```

    Delay_ms(5);
    WriteLCDInit(0x30);
    Delay_10us(16);
    WriteLCDInit(0x30);
    Delay_10us(16);

    WriteLCDInit(0x20); //4-bit mode
    Delay_10us(16);

    WriteLDCmd(0x28); //interface length
    Delay_10us(16);

    WriteLDCmd(0x08); //display off
    Delay_10us(16);
    WriteLDCmd(0x0F); //display on
    Delay_10us(16);

    WriteLDCmd(0x06); //entry mode
    Delay_10us(16);

    WriteLDCmd(0x01); //clear display
    Delay_ms(1);
}

```

SetLCDAdd()

```

#include <htc.h>
#include <lcd.h>

/*****
*      Function Name:  SetLCDAdd                                *
*      Return Value:   void                                    *
*      Parameters:     lcdtype: sets the type of LCD (lines)   *
*      Description:    This routine sets the cursor address of the LCD. Based
on      *
*                      the Hitachi HD44780 LCD controller.      *
*****/

void SetLCDAdd(unsigned char address){
    unsigned char temp;
    temp = address | 0x80;
    WriteLDCmd(temp);

    //
    //    return;
}

```

WriteLCD()

```

#include <htc.h>
#include <lcd.h>
#include <delays.h>

/*****
*      Function Name:  WriteLCD                                *
*      Return Value:   void                                    *

```

```

*      Parameters:      lcdtype: sets the type of LCD (lines)      *
*      Description:      This routine writes constant string to the LCD. Based
on      *
*                        the Hitachi HD44780 LCD controller.      *
*****/
void WriteLCD(unsigned char row, unsigned char col, const char * text){
    unsigned char i;
    SetLCDAdd((row*0x40) + col);
    while(*text)                // Write data to LCD up to null
    {
        Delay_10us(8);
        RS_PIN = 1;

        DATA_PORT = *text & 0xF0;
        E_PIN = 1;
        E_PIN = 0;
        //Delay_10us(16);
        DATA_PORT = (*text & 0x0F) << 4;
        E_PIN = 1;
        E_PIN = 0;

        text++;                // Increment buffer
        Delay_10us(8);
    }
    return;
}

```

WriteLDCmd()

```

#include <htc.h>
#include <lcd.h>

/*****
*      Function Name:   WriteLCD
*      Return Value:    void
*      Parameters:      lcdtype: sets the type of LCD (lines)
*      Description:      This routine writes commands to the LCD. Based
*                        on the Hitachi HD44780 LCD controller.
*****/

void WriteLDCmd(unsigned char command){
    unsigned char out;
    RS_PIN = 0;
    DATA_PORT = command & 0xF0;
    E_PIN = 1;
    E_PIN = 0;
    DATA_PORT = (command & 0x0F) << 4;
    E_PIN = 1;
    E_PIN = 0;
    Delay_10us(16);
    if(command == 0x01) Delay_ms(2);
}

```

WriteVarLCD()

```

#include <htc.h>
#include <lcd.h>
#include <delays.h>

```

```

/*****
*      Function Name:  WriteVarLCD
*      Return Value:   void
*      Parameters:     lcdtype: sets the type of LCD (lines)
*      Description:     This routine writes variable string to the LCD. Based
on      *
*                      the Hitachi HD44780 LCD controller.
*****/
void WriteVarLCD(unsigned char row, unsigned char col, char * text){
    unsigned char i;
    SetLCDAdd((row*0x40) + col);
    while(*text) // Write data to LCD up to null
    {
        Delay_10us(8);
        RS_PIN = 1;

        DATA_PORT = *text & 0xF0;
        E_PIN = 1;
        E_PIN = 0;
        //Delay_10us(16);
        DATA_PORT = (*text & 0x0F) << 4;
        E_PIN = 1;
        E_PIN = 0;

        text++; // Increment buffer
        Delay_10us(8);
    }
    return;
}

```