Autonomous Optical Guidance System

By:
John (Austin) Ciambriello
Ryan Shoaf
Brandon Staton
John Fridenmaker

1.0 Executive Summary

The following document is a full detail description for an autonomous optical guidance system that covers research, design, coding, and testing procedures. The document starts with the project description which includes the goals, motivation, and system specifications for both hardware and software. The goal of the project is to have a self guided airplane track and collide with a specified target.

The third section is all the research accumulated from reading similar projects, types of architectures, types of hardware designs and circuits, and possible software algorithms and approaches. This section explains the advantages and disadvantages of each hardware component for any design considered. The reason for using an airplane with optical tracking is also discussed in this section.

Section four logs all revision types for the system which change for each major adjustment to the design for either software or hardware reasons. The first two revisions started theoretically to narrow possibilities while researching the best design for success. The third design shows the initial structure and support for the system design with specific components. The third and four revisions briefly detail the overall system and what is included for the final design. Between the third and four revisions, the systems complete design is talked about, but the fifth section summarizes and explains the final design.

Section five summarizes the final design with both hardware and software which will contain the hardware schematic for the system and the class diagrams for all software coding. This section does not go into detail for software since section six is devoted completely to a thorough description of the software coding.

The software coding section outlines the complete structure of the systems code which includes flowcharts, diagrams, and charts to help explain the code's structure.

Section seven outlines all of the system's testing procedures which includes field testing, laboratory testing, hardware testing, and software testing. This section goes into details regarding how the tests will be executed as well as the interpretation of the results. Section eight combines all the administrative content such as milestones and budgeting for the project.

2.0 Project Description

During the first discussions of the project, the group needed to decide the scope of the project which was decided by the teams' motivations and goals. The project was developed around the future plans and interests of the group

members. After the group narrowed the current ideas to an aerial themed tracking system using the member's goals and motivations, the objectives of the project were created as guidelines for research and discussion. The next step included writing a set of system specifications which would dictate the boundaries of the autonomous system. The system specifications were split into two sections: software and hardware. In order to meet the system specifications, the team devoted days and weeks of research for specific hardware and software. The following sections outline, in detail, the motivation and goals, objectives, and specifications of the project.

2.1 Project Motivation and Goals

With advancements in technology, guided tracking systems have existed for many years now and have had a strong impact in the nation's military. Guided missiles and bombs are developed by many different companies which provide their products to every military branch the United States in existence. Companies nearby including Lockheed Martin and Northrop Grumman provide excellent opportunities to students at the university. Designing such a system at very little cost in comparison to existing ones may spark an interest in from these companies or any other company developing related work.

The group's desire to create a system which will autonomously guide a model airplane on a collision course with a specific target using image recognition stems from the group's interests as a whole. Members of the team have unique interests in specific sections of the project and this optical guidance system project offers each member interesting work. For one member interested in the optics and computer vision field, this project will provide hands on experience and develop research skills and techniques for a real world application in targeting and tracking as well as image storage and manipulation. For the electrical engineer in the group, the optical guidance system will consist of an embedded system which will require two microcontroller units as well as many other electronics devices not only for the tracking subsystem but also for the control subsystem. From synchronizing multiple microcontroller units to calculating altitude direction and angle, there will be many difficult tasks required by the hardware in order to achieve the final goal. Lastly, for the group members interested in embedded systems software, the project provides great experience in a broad range of areas popular with embedded programming.

Not only does this Autonomous Optical Guidance System spark an interest in each member of the group but it also provides an opportunity to promote

individual skills to nearby companies such as Lockheed Martin. The aspects of this project make it a well suited option for the entire group.

2.2 Objectives

The primary goal of the project is to create an autonomous system that can fly in a stable manner and also track and pursue a target at the same time. In other words, when the system is launched with a target destination, it will have the ability to track this target until impact much like a guided missile, but the technique which will be used is optical tracking. In order to achieve this primary objective, some important secondary objectives are required. Because the A.O.G. System covers such a wide variation of different components, it was broken it down into key objectives for each system of hardware and code.

Starting with the optical subsystem, the system must be able to recognize a target using an algorithm that can detect targets from a live image feed. Once a target has been selected, this optical subsystem must transform from an image detector to an image tracker. The system's new objective becomes the ability to locate and track its moving target.

The next key subsystem in the A.O.G. System is the system controller. The main objective of this subsystem is to control the object, which in this project is a model airplane. The system must maintain flight autonomously and be able to steer the object towards the target. This objective will require a significant amount of precision and balanced control to achieve.

The last and most essential component in the A.O.G. System is the communication components. Since the final design uses two microcontrollers, there are both control and data signals that must be reliable and accurate. This communication subsystem will be a high speed bus linking the two controllers together and its primary objective is to relay the correct information it obtains from the camera to the system controllers to correct the flight path.

2.3 Project Requirements and Project Specifications

The following specifications and requirements were created through numerous team meetings. These specifications were determined to be the adequate minimum for the autonomous tracking system. While developing these specifications many aspects were considered, such as time to accomplish all tasks, knowledge, and combined experience in order to set reasonable

specifications. If any specifications or requirements are not met by the A.O.G. System, it is considered to be insufficient and fails its duty. Below is a broken down summary of specifications and requirements for the overall project as well as for each component of the system starting with hardware and ending with software.

2.3.1 Software Specifications and Requirements

One of the most important requirements inherited by the idea of the A.O.G System is represented by the A, Autonomous, which in this system means that it can perform desired tasks in unstructured environments without continuous human intervention. Because the system has so many components just to control that airplane, a complete microcontroller handling this automation was designated. The software must be able to interpret the input signals from the second microcontroller as well as any other additional input signals and relay to correct output signals to the servos that steer the plane.

The next important software requirements come from the O and the G which stands for optical guidance. A second microcontroller is used to free the optical microcontroller to handle the image detection component of the system as well as the image tracking component. To handle the image detection the software must accept the input signals from the camera, build the image internally and detect objects that may be considered targets. When an object is considered a potential target it is marked by the software and waits for confirmation from the user. When handling the image tracking the software, which already knows the target, must locate the target in every frame and calculate the difference from the previous frame. This software acts as the vision for the autonomous system therefore it must be very reliable and accurate.

While the previous two paragraphs define the requirements of the system as two independent algorithms they must work together in order for the system to work as a whole. Since the A.O.G. System uses optics for its tracking and runs in real time a very large amount of data must be move around and processed. For the system to maintain real time processing the software must use very efficient techniques for handling the transfers of these images as well as the tracking of the object in these images. The team selected a relatively high camera resolution because distance of tracked targets should be maximized. This will put a burden the optical microcontroller, so optimizing the tracking component is essential.

2.3.2 Hardware Specifications and Requirements

The A.O.G. System will be implemented into a model airplane therefore strict board design specifications and requirements must be developed. Due to the small available space a single circuit board containing all electronic components will be used. The existing battery in the model airplane will be used for propulsion and a separate battery will power the guidance system. The board will contain an external power switch and a target select button for easy user interface during the initial targeting processes. In addition the board will contain a small LCD screen displaying self test results and recognized objects detected by the system. Other key components in the board design will include a 3-axis gyroscope, a barometric sensor, and two 32-bit microcontrollers.

To maximize the tracking distance of the A.O.G. System a higher resolution camera was needed. As a result, the minimum resolution allowed is expected to be 640x480. The set range for capturing the target is a minimum of 50 yards away. For ease and simplicity of storage and computation the selected camera encodes information as raw data. This will also help the team as issues are debugged during testing stages. The camera module must be small enough to have minimal effect on the plane’s flight characteristics and share similar voltages requirements as the rest of the system. The data format is a 16-bit RGB signal which can easily be interpreted and shared through the system.

Memory is used to handle the image storing and computations since there is not enough memory on the microcontroller itself to handle such large structures. The minimum specification to handle the images generated by the camera is 1 MB. Since the A.O.G. system runs in real time, data will be constantly flowing through the memory bus. To maintain efficiency and minimize bottlenecking the memory must have an access time of less than 20 nanoseconds. To simplify the system, the voltage requirements of the memory must be similar to the other sensor and logic components.

A barometric sensor and 3-axis gyroscope are used to measure the aircraft's altitude and attitude. The requirements for both sensors are similar to other component specifications. Similar operating voltage levels are mandatory as are the rest of the system components. These components should require very minimal encoding and be simple additions to the circuit. The barometric sensor and 3-axis gyroscope will both use I2C interface to communicate with the system controller. Their only I2C associated circuitry is the two pull up resistors on the bus.

The 4.3 inch LCD used to display the self test and target recognition will be mounted externally for easy access. The LCD has a backlight requires a 27 volt

input which adds complexity but the native input for pixel information requires no translation circuitry. This makes it very easy to display the raw data to the LCD. Apart from the backlight the display uses very little power.

The A.O.G. System uses two 32-bit arm cortex M-4 microcontrollers. One of the microcontrollers is used specifically for image rendering and computation, the other microcontroller is used to control the plane, in a stable manner, towards the target destination. The specifications for these microcontrollers are very similar for ease of parts acquisition, assembly, and future expansion. The microcontroller that handles the imaging computation is also connected to the memory, camera and LCD. As a result this microcontroller must have enough I/O pins for other interfacing requirements in addition to the data and address busses. It must be very fast in order to handle the large data image computations and a maximum frequency rate of 168 MHz it should suffice. The flight microcontroller is used to communicate with all of the control actuators and most of the sensors. This includes the servos, gyroscope, and barometric sensor. Because it's used to control the airplane, it is connected to several servo motors. This microcontroller must be capable of handling many I/O calculations from ADC conversions, five PWM outputs, I2C communications, and simple I/O toggles.

The last major section of the A.O.G. System is the power supplies. Their requirements include having passive cooling, capable of supporting 3.3, 5, and 27 volts and deliver a minimum of 1 amp at the rated output voltage. The size of these power supplies must be relatively small in comparison to the airplane and as light as possible to maintain a form of balance throughout it.

3.0 Research related to Project Definition

As the group formed and discussed ideas for this project it was discovered the group shared similar interests which involved two main components, one being optics and the other being robotics. These two areas of interest formed the backbone for the development of the project. As the team began to research and bounce ideas off each other, the group began to circulate a few ideas that fit the best. A consensus was reached when a member who also shared an interest in aviation came up with the idea of building a guidance system for some kind of aeronautical vehicle which could track a target using optical tracking. With the approval of this idea, the team needed to develop the function requirements and restrictions so the team began to research information related to it. The following sections will cover the research and approaches to how the specifications and requirements were selected. The first section explains the idea progression and

how research affected all project decisions. The second section covers the different architectures for the project design since there are multiple ways to design such a project. The third section explains the different types of communication hardware and protocols this project uses throughout. The fourth section explains the hardware approaches and the reasoning behind the choices made in the design. The last section is heavily influenced by each of the preceding sections since the hardware was selected as a function of this research and all discussions.

3.1 Existing Similar Projects and Ideas

The first major roadblock for the group was the exact project definition. Since the team has many ideas and interests, it was particularly hard to choose a project. The team first started by pooling all interests together and identifying the most common ideas, which were flying and tracking. The next step was to come up with ideas related to each topic, however, identified concepts covered a multitude of possibilities. Research of similar projects and past ideas for advice and approaches yielded helpful information. The topics the team researched were different types of tracking and types of flying projects.

3.1.1 Tracking

During tracking research, types of tracking were divided into two main categories which are either fixed or moving targets. Both types of tracking usually involved an initial object which is launched or directed towards a second object with the intent of collision. Since any motion tracking software can track stationary targets also, the group initially decided to implement motion tracking software for a stationary target since the tracker does not have to account for movement by target. The types of tracking the team considered initially were remote, command, laser, homing, and active. After reading and discussing each possibility, the software decision was to create a motion tracking system, or a type of active homing, which will use camera images. The functional requirements of this system would include the initial designation of the target by a user prior to takeoff, as well as the automatic dynamic tracking of the target after takeoff by the tracking system until collision with the designated target.

Remote guidance: A major consideration in the tracking of the target was with whether the target would be targeted at take-off or if the system would need to seek out a target based on statically designated characteristics. The team decided initially on a command guidance method in which the user of the system

could search and select a target before take-off through an LCD interface. This would allow the target's characteristics to be much more dynamic. This also eliminated the need to search for a target directly after take-off each time, which saves power and allows for a longer direct flight time. The research of the particular moving target tracking systems is discussed below.

Command guidance: There are two types of command guidance. They either go to the line of sight or off of the line of sight. The target destination would be calculated pre-launch by the user, and then the characteristics of the target selected would be passed to the system dynamically on each launch. In command to line-of-sight guidance research it was found that it would only use the angular coordinates between the plane and the target to ensure collision. This guidance system requires that the target be within line-of-sight of the initial tracking system on the plane before take-off. The group also looked into command off line-of-sight guidance which differs from command to line-of-sight in that it does not rely on the angular coordinates alone to calculate collision to the designated target. It also would need an additional distance variable.

Line of sight: With line-of-sight beam riding guidance, the target destination would be set statically by a physical laser beam to the target. This beam would be directly targeted at the destination, and detectors on the rear of the aircraft would detect the beam and keep the aircraft centered in this beam on its way to the target destination. There can be problems with weather as well as low distance that can be limiting in a line-of-sight beam riding guidance system; however, with this system, these limitations would not be a factor, since the requirements would not involve flying in bad weather or over very large distances. The team found in research that for the system that the additional hardware and software, as well as the increased cost of the system would not be worth the implementation of line-of-sight beam riding guidance.

Active homing: Homing guidance was another option looked into for tracking the target. While the team found that there would be some benefits in homing systems at first, upon further research it was decided that the complexity of these systems would not be acceptable for the particular application of this tracking system. The research into the particular homing system options is discussed below. An active homing system would require the use of a radar system on the aircraft to provide a guidance signal to the designated target. The radar is usually directed toward the target at all times, and the aircraft tracking system would then use these coordinates to realign its center toward the target. These systems can be limited by the particular radar resolution available, which would be determined by the size of the antenna on the aircraft. In this project, this limitation would not

be applicable, as the target distance would never exceed the distance limitation of the radar, however the additional cost and complexity did not make sense in the system.

3.1.2 Flying

The team's second decision involved the type of vehicle medium for the project which involved flight. The advantages and disadvantages of each particular type of vehicle were considered such as speed, stability, interest, space for components, cost, availability, construction, and general size**.** A major consideration regarding the type of aircraft used in this system is the need for the most compatible system to facilitate autonomous flight control. The two main types the team looked into were floating and flying types. Floating types all have the ability to hover and do not require propulsion to stay aloft such as helicopters, balloons, and hover vehicles.

Although balloons were considered as a possible approach for the project, the team decided to exclude them and focus on the hovercraft and helicopter. The balloon would be significantly affected by even the lightest winds, making testing conditions extremely difficult. The aerodynamic drag of a balloon would make it much harder to control and accurately propel than other types of vehicles. Flying aircraft appear much more suitable for concept development. Helicopters are an attractive option since they are much more common and versatile. However, since helicopters can be difficult to stabilize and control, the decision was made to research previous projects with hovercraft.

Unfortunately while evaluating previous hovercraft projects, the group decided it would not be a very practical choice. While there were quite a few different affordable options for RC hovercrafts, the issues with stability and control were prominent in the researched projects. The group also felt that the limited environments in which the hovercraft would be able to operate were not ideal. One significant issue with a hovercraft is the close proximity of the camera to the ground which would greatly reduce the field of vision for optical processing. The image at the lower edge of the camera field of vision would change rapidly as the hovercraft moved which would be difficult to process. The next area the team researched was about helicopters. Another issue involves the significant battery load created by the need for a hovering motor. Multiple batteries add weight that makes the hovering load even worse, further draining the batteries for hover thrust.

Helicopters are a very popular project and a multitude of helicopter projects were found. However, as feared, the most common problem discovered with helicopters was stability and control. For example, the rotors present significant rotating mass and contribute a force that will affect flight characteristics. During research, the team found there to be a great deal of similar projects doing autonomous helicopter flight, which provided a great resource for the project. The team found that the RC helicopters fit the requirements for weight as well as space for components; however, these other projects reported that stability and movement control issues were a major challenge to overcome. Concerns exist regarding stability of helicopters during turns. Many affordable and appealing RC helicopter options exist, however the stability issues make airplanes and rockets far more appealing.

Rockets also intrigued to the team because hitting a target with high speed would be an interesting challenge. Unfortunately, the velocity with which a rocket would strike the target would likely damage the vehicle. Aerodynamic constraints would require mounting the camera on the nose of the rocket which is where the highest damage occurs. Another problem caused by the location of the camera is the short data cabling requirement places the guidance system near the nose, which could adversely affect center of gravity. A rocket can very easily roll during flight, as a result the flight control software would have possess the ability to reference it's calculations with a changing axial reference or extra control surface mechanisms and system interfacing would be necessary. Additionally, the mounting structures necessary to prevent the guidance system and batteries from damaging each other would likely add unacceptable weight. The group also came to realize that timely guidance system response to the high vehicle speed might be much harder and could cause the chances of success to decrease. Since accuracy was one of the goals for the system, the airplane became the most appropriate research topic.

The airplane became the number one solution because it solved all potential problems with the previous designs such as moderate speed, stability in reasonable winds, space for component mounting, interest, reasonable cost, and availability. Since most airplanes fly at a moderate speed and are stable over a wide range of speeds, the speed can be adjusted to a level which suits the simplest solution. This speed is estimated to be around 20 feet per second. This will allow enough time for the image tracking software to process the data while maintaining enough reaction time. The airplane also provides inherent stability caused by airflow. This is a tremendous benefit because aircraft stability is far less of an issue compared with the helicopter and hovercraft. An airplane with

approximately a four foot wingspan and balsa construction should also provide enough space for all related components while remaining a manageable size.

A model airplane shares aerodynamic qualities and phenomena with full size airplanes. Movable control surfaces are mounted in various places that deflect the air stream, which changes the attitude of the aircraft. The control surfaces are mechanically connected to servos, which are electromechanical devices that interpret a pulse width modulated signal and rotate their output shaft to a fairly precise angular position. Most servos rotate their output shaft through a range of 180 degrees for a positive pulse of two milliseconds and zero degrees for a one millisecond positive pulse. Innovations in battery and DC motor technology have made electric propulsion of model airplanes a viable option. The airplane selected has a four foot wingspan, balsa monocoque construction, is partially constructed as supplied, is nimble, and has a significant wing dihedral for stability.

3.2 Possible Architectures and Related Diagrams

When the original project idea was decided many different architectural concepts were discussed. As these ideas were tossed around, many were discarded as they proved to be either overly complicated or not efficient enough for the task required by the A.O.G. System. The following sections discuss what architecture was decided upon and why it was chosen. Each idea that was a potential candidate was broken down into pros and cons and debated amongst the team. Many of these ideas came from similar existing projects where a full analysis of the systems difficulty and success was measured.

3.2.1 Architectures: Pros and Cons

The first architecture that was considered was an infrared sensor array and targeting emitter system. Eight sensors would be mounted in a cross pattern, axially centered on the front of the system. When the longitudinal axis of the system does not aim at the target, a difference in electromagnetic energy reflected from the target is detected by the sensor elements. Conversely, identical levels of infrared energy from the center two sensors on both the x and y axes would indicate the vehicle is aimed at the target. Identical levels of energy detected by the two sensors on one side of either the x or y axis would indicate the vehicle is extremely off course and a more significant turn in the opposite direction is warranted. If only one of the sensors which are located far from the

axial centerline detects any reflected energy, the maximum turn must be commanded by the guidance system controller. See the illustration below.

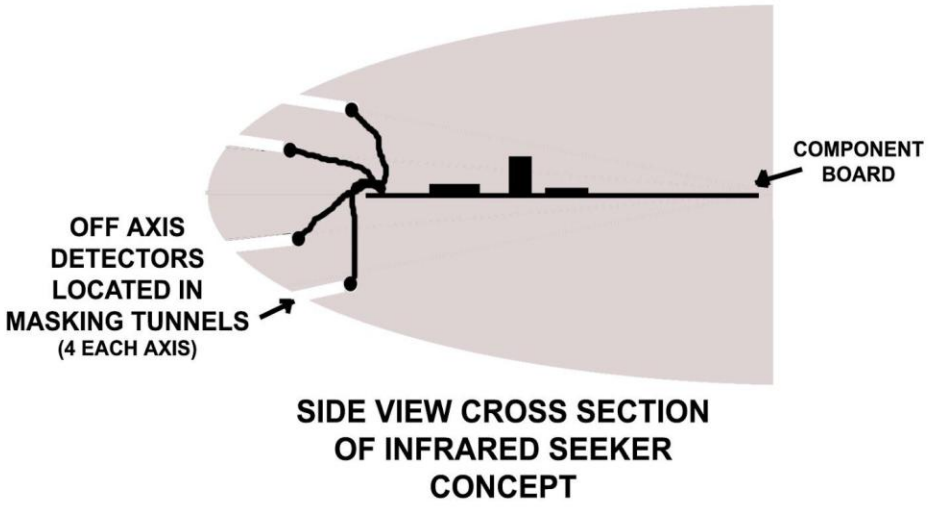

Figure 3.2.1: Infrared seeker concept

There are several obstacles involved with this architecture. It may be difficult to obtain a suitably powerful infrared emitter system that is portable both in terms of carrying and power connections. The sensors onboard the aircraft guidance system would have to be selectively sensitive to the wavelength of the emitter. Interference could occur from a wide variety of potential sources on any given day at any given test site and would have to be evaluated at the time of a test attempt. In addition, the quantization and comparison of received sensor levels will have to be performed at a very high bit resolution to maintain the highest accuracy. The primary benefit of this architecture compared to a camera system is the significantly lower number of data manipulation, movement, and calculation operations.

The next system considered receives, stores, and processes image frames. The storage of frames of raw digitized optical data allows a large degree of processing options. Such a flexible system can compensate for more adverse conditions such as variation of colors received due to shadows, for example.

The complexity of a camera based system involves the storage of a data word representing every pixel of multiple image frames. Over three megabytes of memory would be required if ten VGA resolution RGB565 video format were

stored at any moment. If the system must process these frames in one second including retrieving each frame from the camera, storing them, selecting each frame for comparison with the previous frame, and then sending steering data to the flight microcontroller, then over twelve million bytes must be transferred by the optical microcontroller in addition to the image processing operations.

The system was designed with speed in mind. Parts selection and system implementation was performed with an eye toward minimizing hardware and software “layers” and using relatively fast hardware. It is believed that tailoring design ideas with these factors in mind will minimize issues with performance in a relatively fast moving aircraft. Additionally, by implementing standard interfaces and protocols used by the most common hardware in radio control vehicles the system will also be easily transferred between vehicles.

3.2.2 How to arrange parts

Parts layout will segregate higher frequency communication lines from each other and from as many non-associated components as possible. For instance, the video data bus lines should be as far removed from the servo control lines, voltage regulators, etc. The regulators will need heat sinks and need to be away from the LCD panel, which itself will need to lay parallel to the circuit board to make the system compact and for aerodynamic and crash resilience purposes. The video camera obviously needs to face forward however the video bus will be as short as possible and the overall system will be as small as possible if the display lays flat, presenting its “vertical” dimension laterally with the aircraft. As such, the group envisions putting a 45 degree fold in the camera ribbon cable and mounting the display main cable, camera cable, and memory connectors and solder pads longitudinally. For crash resilience the camera will likely be atop a short spring returning pedestal. Surface mount packaging of selected devices will help minimize the size of the circuit board.

3.3 System communications

The A.O.G. System uses several different protocols to communicate amongst the devices. With the dual microcontroller design and several external components communication buses are used throughout the entire system. The key component communication protocols, which are described below, will be used on the camera to optical microcontroller bus, the external memory to optical microcontroller bus, the barometric pressure sensor and gyroscope to flight microcontroller and the MCU to MCU bus.

3.3.1 Types of communications

I2C data transfers are a simple and widely utilized standard for setting the camera registers and thus their configuration. The I2C hardware networks are simple to implement and easy to code for making it a strong choice for camera communication. Many I2C devices can be connected to the same bus with pull up resistors on each line near the master provided the slave devices were manufactured with unique and/or selectable addresses.

The camera will use the I2C protocol on its serial bus connections for the purpose of configuring its registers at the command of the optical microcontroller. The camera will also use an eight bit parallel bus to transmit raw video data formatted in the RGB565 format to the memory component and LCD display. The data to and from the memory and display will be raw binary 16 bit color video frames.

The communications between the two microcontrollers will be a simple 5-bit binary encoding of the group's own design. This 5-bit bus was based on the idea that every different type of signal necessary to communicate between the two microcontrollers could be handles with only 5 bits. This allows the systems to directly control these five registers and instantly send the signals to one another. The barometric sensor and gyroscope will send I2C serial data to the flight control microcontroller.

Some devices use the "serial peripheral interface" (known as SPI) for data transfers. The SPI architecture does offer more data throughput at the cost of more numerous connections. In addition, the devices that appeared more appealing to the team feature I2C protocols for data and configuration.

3.3.2 Data transferred

The video is 16 bit RGB565 data. This encoding is a "raw" color format, meaning no calculation must be performed to determine the levels of the primary colors that comprise the color of each pixel. It is transmitted from the camera in bytes which will be "assembled" in the optical microcontroller before being stored as required in the memory. This same data will be sent to the display when required. Some of the camera's registers will configured at startup to set the operating mode as VGA resolution RGB565 format.

3.4 Hardware

The follow sections summarize the researched components for the optical guidance system. Each section will discuss the primary points of interest for the specific component as well as the choices for each component. The two most important components of the project which consumed a majority of research time were the camera and the microcontroller. Each component of the system needed some extent of research in order to find the best hardware.

3.4.1 Cameras

There are several important attributes for the camera used in the system. The team practiced a design mindset that minimizing hardware and software interfacing and computational requirements increases the performance of the system and the chances of success. The size and complexity of the system is, in part, driven by the camera choice.

The attributes looked for in a camera sensor include resolution, documentation, price, packaging, whether or not it includes a lens, and color encoding. The highest possible resolution would be preferred except for the price of sufficient memory to store each frame.

Complete cameras like the GoPro HD and simple camera boards containing camera sensors were both considered. Most of the end product cameras like the GoPro perform various kinds of sampling and signal conversion for storage or display on televisions. Circuitry exists to convert component and composite video present an additional layer of processing and conversion of video data back to raw format. They frequently produce YUV color encoding which would increase the computational workload of the microcontroller. These kinds of cameras can output compressed MPEG video which would also require translation. Frequently, little or no documentation regarding the specific behavior of these camera sensors was found. Compared to “board cameras” that contain the minimum circuitry necessary to capture, encode, and send electrical representations of image data, the GoPro is also significantly heavier. The GoPro also draws several times as much current.

Many camera sensors are available mounted on small circuit boards. Their features vary and not all of them have documentation. Their main advantage is that many of them can output raw video data in many color encodings and already implement the appropriate power, filtering, and ground topologies recommended by the image sensor manufacturer. Unfortunately, many of the

camera boards evaluated have poor documentation as they appear to be meant for use with evaluation systems for particular microcontrollers.

The Aptina 9M033 camera sensor is used on several boards. The native color encoding of this sensor uses Bayer color encoding which would require pixel color computation by the microcontroller. This means the microcontroller would have to compute the color of every pixel by taking in the values of several neighboring pixels and performing mathematical calculations.

The Toshiba TCM8240MD sensor has a higher resolution of 1300X1040 pixels and a low cost. This sensor may be used in the final system if more resolution is necessary. This camera also offers many color encoding schemes and other options set through the internal registers. Where the sensor is sold it does not typically come with an external lens and holder. Additionally, the data sheet for this sensor is poorly written where the waveform diagrams and descriptions are concerned. This sensor may still be utilized if higher resolution is necessary; however its drawbacks are a source of concern.

Many camera boards equipped with sensors made by Omnivision were identified. These boards usually consist of the sensor and a handful of impedance matching resistors and filtering capacitors. A board based on the Omnivision OV7670 color camera sensor was selected. This sensor can be configured by writing values to internal registers to set many attributes. The OV7670 can produce RGB565 raw pixel color data which will reduce the computational workload of the microcontroller. The sensor transfers this pixel data in two sequential byte transfers. The video resolution is 640X480, which will require 307,000 bytes of storage per frame. A VGA resolution and an appropriate zoom factor will be used to capture enough optical data of the test target (experimental discovery is planned for this purpose).

If a VGA camera proves insufficient to provide enough resolution, a higher resolution Omnivision camera with similar specifications will be considered. The storage requirements of a higher resolution camera increase by a factor of at least three because of the target location and image processing requirements. As many as three images and/or frame fragments may need to be stored.

3.4.2 Microcontroller

The microcontroller requirements encompass the support of the sensors and the image processing. In addition to sensor communication and data transfer, the system will have to provide control and configuration signals.

The optical microcontroller will use a memory bus to shuttle pixel data back and forth between the camera, memory, and display. It must also provide I2C communications for configuring the camera and receiving data from the gyroscope and barometric sensor. Communications for steering advisory data will occur between the two microcontrollers using general purpose IO pins encoded in a binary word format.

The flight control microcontroller will use several timers to generate pulse width modulated control signals to the aircraft's servos. It will also monitor voltages of the propulsion battery and the 3.3 and 5 volt power supplies with analog to digital converters.

Ideally the microcontroller would be flexible in terms of its input voltage so that level conversion interfaces will not be necessary. Most of the system peripherals are compatible wholly or in part with a 3.3 volt DC source.

The STM32F4 microcontrollers feature as many as 14 timers, a memory bus compatible with several memory architectures, and several I2C channels. Members of the team also have experience programming the STM32 microcontroller.

The 32 bit ATMEL UC3 microcontrollers were initially considered for the project. They feature a large number of PWM outputs. ATMEL microcontrollers are also widely used, which means a large repository of user accounts should be easily found in a web search. Like the STM32 microcontroller, the ATMEL controller provides DMA options on many of their pins, which should provide high IO performance and minimized CPU utilization for data movement. It is believed that the STM32 product line had more options for peripherals, more program memory, and based on searches better application notes.

3.4.3 Memory

Several types of memory have been considered. The main factors are latency, cost, availability, and memory architecture support.

STM32 microcontrollers have the option of a memory controller which supports flash, static, and SDRAM memory modules. The existence of libraries for the use of these types of memory is an enormous advantage.

Flash memory is extremely inexpensive. Several gigabytes can be obtained for less than $25.00. Data would be stored in a nonvolatile condition with no data access interruptions for refresh operations. Latency is the single most significant obstacle to using flash memory in this system. Flash memory typically has access times that are many times longer than random access memory chips.

SDRAM memory chips have the advantages of high speed and low cost. Typical data density to cost ratios hover around $5.00 per gigabyte. The memory modules are commonly available. The design philosophy of requiring the optical microcontroller to do one thing at a time to maximize performance while dealing with latency issues with refresh cycles and the necessary buffer for video frame data while refresh operations occur, make the use of SDRAM a less attractive option.

Static random access memory needs no refresh operations. The access time of static RAM is typically around 20 nanoseconds. The primary disadvantage of static RAM is cost. A one megabyte static RAM module typically costs approximately $60.00. The low hardware overhead and minimal support operations make static RAM technologies the choice for the guidance system.

3.4.4 Gyroscope

A solid state three axis gyroscope will be used to provide the flight microcontroller with aircraft attitude readings necessary. These readings are critical for the flight microcontroller to maintain aircraft stability. Some companies offer Inertial Measurement Units that integrate a three axis accelerometer, a magnetometer, and a gyroscope. These units are plentiful and frequently come with good documentation and considerable premade libraries. If a three axis accelerometer is used code and libraries used make the code the guidance system runs more complex. For example, compensation for gravity must be made while integrating the combined readings of every axis. A gyroscope (or an accelerometer array with internal code that performs conversion) would reduce the code the flight microcontroller must run. The team selected the L3G4200DH because it has readily available documentation and I2C connectivity.

3.4.5 Barometric Sensor

Altitude sensing could be done by measuring barometric pressure or ultrasonic ranging. If ranging is used, the transducer must remain perpendicular to the ground. This will require more electromechanical hardware and code. Though the team is placing no altitude constraints on the operation of the guidance system, it

will likely keep the aircraft below an altitude of fifty feet. A high resolution barometric sensor with I2C connectivity fits the operational requirements of the system with minimal complication. The Bosch BMP085 has a .5 meter resolution and the appropriate interface.

3.4.6 Display

The viability of LCD displays with a diagonal length of around 4.3” was evaluated for the system. The Sharp LQ043 display has good documentation and some versions require very little associated hardware (beyond the microcontroller that will send pixel data and synchronization signals). This model was found for a cost less than $50.00. The uLCD43 display does have a controller built in and costs nearly twice as much. Other inexpensive LCD displays have various interfaces normally designed for audiovisual signal display which are unnecessary for this project. Minimizing hardware complexity is the better approach. The display will show power on self test indications and facilitate aiming the aircraft and guidance system at the target. The required timing, synchronization, and data signals will be generated by the optical microcontroller. Once the target has been selected, the LCD will be powered off to conserve power.

3.4.7 Battery

Because the flight time should be less than one minute and flight weight should be minimized, two smaller batteries will be installed in the aircraft. One will be connected to the propulsion motor and the other will be for the guidance system. This will ensure acceptable guidance system performance regardless of drain caused by the propulsion motor.

3.4.8 Servos

Servos offered for radio controlled vehicles universally respond to a standardized pulse width modulated signal. Their connectors lend themselves to ordinary pin headers. They are available in a wide variety of sizes and torque capacities for most load requirements.

3.4.9 Cabling

The guidance system design has very few off board high frequency connections. The camera will be connected via the shortest possible ribbon cable. If crosstalk or other electrical interference is observed, ribbon cables with interleaved

grounded conductors will be used. The remainder of the cables will conduct low frequency signals or direct current.

3.4.10 Switches

The system will have a power switch and a target selection push button. Inside the aircraft, a slider switch will be physically linked to a servo connector still connected to the radio receiver installed to test the aircraft's flight stability before the guidance system is installed.

3.4.11 Programming Hardware Interface

The group has STM32F4 discovery evaluation boards on hand, which can be used as an interface for programming other STM32 microcontrollers in abbreviated circuits. These evaluation boards provide the physical link between the personal computer used for software development and the guidance system. The guidance system will only need a simple pin header connection with little or no support circuitry with this design choice.

4.0 Project Hardware and Software Design Details

The following sections outline the thought process and the changes made through each design revision. The first two designs were completely theoretical to build a starting point. The initial design was formed from multiple ideas and possibilities which would have to be verified from research. Once the design was created, the components necessary to achieve the design would be researched to support the design theory. However, during the research phase, the hardware components selected could not support the initial design. The cycle of design and verification continued until a plausible design was finally formed.

4.1 Initial Design: Single Microcontroller with Advanced Tracking

The initial design was quickly revised with the number of problems encountered within the first few days. The system relying on one microcontroller to manage produced multiple faults, so the system was revised to contain two microcontrollers to solve the current issues. Other problems for the system included microcontroller resetting, batteries, specific cameras, and RAM.

4.1.1 Initial Hardware Design

As the initial hardware design with one microcontroller was discussed, many problems and possible issues were quickly realized. A few of the problems encountered include: synchronization, space, and overload.

The PWM (pulse width modulation) uses the time modules and they microcontroller need constant service from the core of the microcontroller could cause unacceptable delays in transferring data and synchronization problems.

Pin count and peripheral interfacing obligations create a density issue because the selected microcontrollers have many power and ground connections. The manufacturer suggests that a capacitor be placed near every power pin to prevent spurious resets. Many pins have multiple functions and the circuit board layout may interfere with interfacing all peripherals to a single microcontroller. The presence of the flight microcontroller segregates some wideband signals from the core of the optical processing circuitry and relieves the team of some of the burden of balancing all signal responsibilities across the microcontroller pins that are capable of suitable signals.

The effect of transferring data around while simultaneously controlling the servos is a significant concern. The hardware and software designs intentionally avoid the workload of multitasking to maximize the chances of success. The optical microcontroller will listen for the data from one image frame transmitted by the camera and store this information. Next it will identify the target and its two axis displacement from the centerline of the aircraft. Finally, the optical microcontroller will send steering advice information to the flight controller.

Another issue is that if the microcontroller suddenly resets, it may direct the aircraft servos to be at unsuitable directions for the aircraft attitude and velocity causing it to depart controlled flight. The devised word format for microcontroller to microcontroller communications must interpret a binary word of zero as advice to enter landing mode so that in the event of the optical microcontroller resetting, the flight microcontroller will safely land the aircraft.

Dual microcontroller architecture allows greater degree of flexibility. By utilizing a second microcontroller, one microcontroller can focus almost entirely on the image and target optical processing while the other microcontroller could focus on flight processing. Moving the pulse width modulated servo control signal responsibilities away from the optical microcontroller lightens the optical microcontroller processing load and segregates the harmonic rich square waves away from other signals.

Much deliberation occurred about using one or two batteries in the aircraft for propulsion and powering the guidance system. There is a physical “overhead” associated with the construction of every battery. As a result, two smaller batteries will occupy more space than a single battery with the same ampere-hour capacity. On the other hand, if a single battery is drained by the propulsion motor, the system may not be able to safely land the aircraft. In addition, the propulsion motor may introduce unacceptable electrical noise at the battery terminals. The decision to use two smaller batteries was made because safety of flight in paramount and the aircraft was in part specifically selected for its size and loading capabilities. The flight microcontroller will monitor the propulsion battery voltage and land the aircraft if necessary.

Data memory specifications are crucial for the timely accomplishment of optical microcontroller duties. The optical microcontroller contains very little data memory and the onboard flash memory will not present optimal system performance attributes. For optimal system performance, storage space for up to three image frames is desirable for memory architecture. The camera resolution is 640X480, resulting in 307,200 pixels. One megabyte of memory will store more than three image frames. Should the need to store larger amounts of data arise, additional memory will be multiplexed on the same interface.

Flash memory typically has read and write times as slow as 95 nanoseconds. A single image frame would take .03 seconds. Most flash memory uses serial data transfers, further hindering performance. A static random access memory module with 15 nanosecond time has 6.33 times higher data transfer rate. This higher performance triples the cost of system memory but is considered vital for optimum chances of success.

4.1.2 Initial Optical Software Design

The airplanes initial software layout was designed with extremely powerful image processing algorithms in mind for a complete project solution, from target locking to tracking. The target locking part would not be as complicated as the tracking part since there is much less to account for such as scaling, rotation, illumination, clutter, noise, and vibration.

Since detection and highlighting of every object in the frame during target locking was originally desired, a mildly complex algorithm would have had to be implemented in order to account for any type of object. The original idea for the algorithm was primarily an edge detection algorithm such as canny. Two other possible algorithms are Sobel and Prewitt; however, both approaches use a fixed

averaging kernel which disperses the weight of the surrounding area proportionally. Since the algorithm will average surrounding pixels using close weight values, this can cause a negative influence depending on the size of the kernel. Although this problem will not have a large impact the result, it was decided that use of an algorithm which would distribute the weight based on distance such as canny which uses a Gaussian kernel would be most desirable.

The tracking software would have been much more complicated compared to the target locking since tracking software was originally designed for any type of target. Tracking software designed to follow any type of object becomes very complicated since there are multiple areas that have to be taken into consideration such as the shape, structure, environment, and distance. Shape and structure impact the selection of the detection algorithm since different structures require different algorithms such as rigid objects with corners versus fluid objects without corners. If an object has corners or distinct rigid points, the Harris corner detector would be a good candidate since it finds and store all key points, or interest points, that reside in a frame. Using the Harris corner detector, the old frame with all interest points can be compared to the new frame's interest points to figure out the direction. This approach can account for noise and possibly vibration but scaling, illumination, rotation, and clutter are not solved using this approach. This aspect makes SIFT the logical choice for a tracking algorithm.

SIFT stands for scale invariant feature transform which will account for scale and rotation problems which the previous algorithm lacked. The algorithm is much more complicated than the Harris corner detector though which makes it much more complicated to implement. The most important problem using SIFT is the speed of the algorithm since it takes much more time to complete the algorithm. The effect of simultaneous real time calculations and need for a decent frame rate on hardware performance are a significant concern. Reconfiguration of the circuit architecture was driven primarily by the possibility of long processing time.

4.2 Second version: Dual Microcontrollers with Intermediate Tracking

In order to make up for the myriad of faults the initial design contained, the second design tried to average the difficulties of both hardware and software. The best solution would be to think about how to get the best data for image tracking while being able to keep up with processing. Since the actual values for speed of microcontroller for real time image processing as well as resolution needed to accurately detect and track objects are unknown, this stage of deliberation occurred for a pretty long time This is where research came into play.

The decision was made to go with a two microcontroller setup to make sure the image processing had almost exclusive access to processing data which will ensure that very few other operations could preoccupy the optical microcontroller while it should be processing image data.

4.2.1 Second Version Hardware Design

The team also considered three microcontrollers instead of two. One would simply process image data, one would move image frame data, and the third would control the aircraft and sense altitude and attitude. Significant problems include bus and video memory arbitration issues and may cause latency issue within one or more microcontroller because of the data handling workload. The advantages involved with the addition of a third microcontroller were also considered. Such a configuration would move the optical processing duties to the third microcontroller, leaving the optical data movement duties for the first microcontroller. Such a configuration likely fell into the category of diminishing returns for several reasons. Shuttling optical data between microcontrollers, one microcontroller would have to ensure image frame data integrity, somewhat negating the workload reduction. Board layout and signal segregation would be difficult at best. Finally, increased current consumption and maintaining regulator load well below specifications would likely require the addition of another regulator and its associated circuitry, which would increase the size of the system circuit board.

Since three microcontrollers would be very counterproductive, the decision to use two microcontrollers was solidified. This hardware configuration would utilize the second microcontroller solely as the flight controller which will interface with and take readings from sensors such as the barometer, servos, battery, and gyroscope. The optical microcontroller would be used to move data between camera, memory, and LCD as well as image tracking and transferring data to second microcontroller.

4.2.2 Second Version Software Design

Since processing time is the main concern for software processing failure, the decision was made to reduce the software level in order to make it faster. The original tracking software design was using advanced tracking algorithms which have much more versatility but take more time to process. The decision was made to reduce the software to use edge detection and optical flow since those algorithms are less computationally intensive compared to the burden of processing advanced algorithms,

Edge detection will still be used primarily for target locking which is very useful for seeing the contrasting objects which may not be seen easily, however the system will be using Sobel or Prewitt instead of canny. Using Sobel or Prewitt will facilitate hard coding the averaging matrix which is less computationally expensive versus calculating the Gaussian for a specific sized matrix. Although it is not distributed proportionally with respect to distance as the Canny algorithm, the averaging code also should not impact the resulting image which led to the decision to implement Sobel or Prewitt.

Image tracking software had a much more extreme change because of the change techniques from SIFT to optical flow. Optical flow calculates the direction of flow for each pixel which shows the objects movement instead of finding specific features to compare to each others. Optical flow could be very useful since it would show the direction of the target which could help reduce processing time by focusing on the area the target is moving to instead of scanning the entire image looking for the target. The difficult task would be finding the target to track since optical flow focuses on all pixel flow and not just the target. However, if another basic technique to find the target is used, optical flow and predicting the future movement could be simplified.

4.2.3 Second Version Optical Microcontroller Design

The optical microcontroller sub-circuit would interface to the camera, memory, display, and flight microcontroller. Primary duties would include transferring image frames, performing image processing, and providing steering advice data to the flight microcontroller.

4.2.4 Second Version Flight Microcontroller Design

The flight microcontroller sub-circuit would be added to the system and interfaced to the optical microcontroller, barometric sensor, gyroscope, servos, and batteries (for voltage sensing). Primary duties broadly entail controlling the flight and performing stability maintenance of the airplane.

4.3 Third Version: Combination approach

Once the system structure was decided with microcontrollers, camera, RAM, and other miscellaneous hardware, a full design was finally created. In order to put less strain on the optical microcontroller, the software level was downgraded

again since the amount of time required to process was still unknown. The final architecture is described in the sections below.

4.3.1 Third Version Hardware Design

After going through the two initial design architectures, a finalized hardware design with specific parts was developed for discussion. The hardware design will include: two microcontrollers, one for flight and one for optical, 1 MB of SRAM since it will be faster than regular RAM which will allow faster transfers, one camera with 640x480 resolution, one barometric sensor for altitude readings, one gyroscope for current plane angle, one LCD to display target, two battery supplies which is purely for motor control, four servos for aileron, elevator, rudder and motor, one crystal for each microcontroller, and a master clock oscillator for the LCD and camera.

4.3.2 Third Version Optical Processing Design

As more talks about processing capability of the STM microcontroller ensued, it was decided to try to reduce the tracking algorithm as much as possible to increase the chance of success. However, to reduce the algorithm would mean to have to find solutions to tracking problems which include occlusion, scale, rotation, illumination, noise, and vibration. In order to account for the problems of tracking, assumptions will be made which will help increase processing speed by eliminating problem areas. The assumptions allowed the team to reduce tracking software computation by scanning an image looking for a specific color value and determining the target size and location afterwards.

The final software design is broken into 2 main sections: target acquisition, and tracking. The acquisition mode will have more complex software in order to show and isolate the selected target which will use edge detection with the specifically selected contrast color. A histogram color sample of the target area will be used. The shape is currently unknown but rectangles have a nice advantage with computation and finding the area versus other shapes because of the integral image approach. The center image will also be theoretically “cropped”. This will isolate the center of the camera view for faster processing. Since the extraneous information of surrounding image is unnecessary, the system will only focus on the center where the target will be.

After the locking stage, the tracking stage will be entered. Taking into consideration the target distance, type of camera, resolution, zoom, etc, the group decided to go with a color histogram approach which would isolate the

area around the target in a box and compute the average color value which will provide a number and be within the tolerance allowed for a positive target.

The locking and target software is using a color histogram. Another idea for a faster processing speed was to use a basic color histogram which will take an average over an area and compare it to the original color histogram.

During tracking, the color histogram will eliminate the need for rotation and scaling since the color will be proportional to the size of the target at any size as well as the area that is visible. This should require less time than either optical flow or SIFT tracking approaches which will have to go through the image multiple times to determine where the target is and what direction the target is moving. However, this approach makes predetermining target direction nearly impossible so searching the entire frame until it finds a possible target to compute the color histogram on is necessary.

4.3.3 Third Version Optical Microcontroller Design

Memory and display Check Function:
Writes values to memory, and then sends these values to the display, verifying half the optical system in one cycle. This manner of testing will verify the operation of a large portion of the optical data path. Without timely synchronization signals and data transfer, the LCD will not cycle through the color patterns across it's dimension at regular intervals.

Camera to display Check Function:
When the user successfully sees and interacts with the display to select a target, this will indicate to the system that the timing signals from the camera and to the display, and the data transfers throughout the system are occurring in an appropriate manner. If camera images appear on the display as smooth video, the system video bandwidth capabilities are more than sufficient.

4.3.4 Third Version Flight Microcontroller Design

Once it was decided by the team to introduce the flight microcontroller into the system to fully handle the flight of the aircraft, a full software design of the flight microcontroller was needed. The following content details the outline of procedures, classes, and functions that the software will use to handle the requirements of the successful flight of the aircraft, as well as how the optical microcontroller will now interact with the flight microcontroller.

The flight microcontroller software will include several procedures to handle the different states the aircraft will be in throughout initialization, abort, take-off, altitude adjust, attitude adjust, flight, collision, and landing. These procedures will all call various class functions that will run while in the procedure to perform system hazard and safety checks, as well as control the flight of the aircraft as required by that procedure. These procedures will also be responsible for moving the aircraft into a different procedure if required. Procedures will be the only method to take the aircraft into a different procedure.

These procedures will all include procedure loops which will call upon various class functions to perform a specific role or check that the procedure requires. The classes available to these procedures are the altitude, attitude, battery, control, time, kill, and error classes. Each of these classes has various functions which perform a certain role or check that these procedures will use to control the aircraft, as well as monitor and ensure the safety of the aircraft.

Several functions will exist in these procedures that will also handle the physical hardware of the aircraft as well as the equipment and peripherals attached to it. This includes control of the gyroscope for stability monitoring, the barometer for altitude monitoring, the servos for controlling the surfaces of the aircraft, and the battery for battery level safety checks. The software will also include safety and hazard protocol functions that the procedures will use in passing of the aircraft into the appropriate hazard procedure if a safety or hazard condition is detected by one of these functions.

The flight microcontroller must communicate with hardware and peripherals in the system to perform routine checks and functions. Communication from the flight microcontrollers to the servos which will steer the plane left and right as well as control its ascending and descending patterns will be needed. In order to ensure and maintain the control of the airplane, safety checks must be calculated at every instance of the procedure loop. Aside from the battery level check the flight microcontroller must also perform an altitude check. This check will use an input value from the barometric sensor via an analog pin, complete an analog to digital conversion, and test its values against a specified threshold value. If the value is above or below the thresholds the aircraft will need to enter an altitude adjust procedure to immediately correct the altitude of the aircraft.

While the altitude check will determine if the aircraft is at an acceptable altitude or not, the flight microcontroller will also check the safety of the planes attitude. A 3-axis gyroscope will send signals to the flight microcontroller to verify that its attitude is within established thresholds. It is necessary to check these values to

ensure the aircraft does not lose its balance. Since this vehicle will be in the air, there are many directions in which it can roll over or flip to so all directions must be checked. While steering the vehicle the system will use a limit on how far it can turn left and right. When the plane begins to ascend at the beginning of the flight or when it descends during landing, limits will exist for how fast it can do either. If there was no limit to the angles the aircraft can begin to nose dive, stall out, or barrel roll into an uncontrollable savable state. Ensuring none of these cases happen during flight is paramount.

Full movement control of the aircraft through the servos will be handled by a servo pulse function which will control the servo pulse widths to each servo. The servo pulse function will be controlled by a control code received from the optical microcontroller control code bus. The function will also need to contain statically set coefficients for directional control, to compensate the input values with known aircraft movement patterns recorded during testing procedures. This function will also handle the throttling of the propulsion motor via the propulsion motor controller servo. When the aircraft is taken into different procedures, the propulsion motor will need to be adjusted to allow the aircraft to make the changes required in that procedure. This function will also be called when certain hazard conditions are detected, such as aircraft altitude and attitude conditions.

In the initialization procedure, the system will perform power on self tests (POSTs). These POSTs will be used to monitor the initial status of the aircraft to ensure it is clear for takeoff. These POSTs will also be used to set initial dynamically set variables that the classes will use when doing specific threshold checks. These POSTs will be an integral part of the flight microcontroller system, as they will help ensure the best possible chance for a successful flight by helping to alert the team to any possible failures that can be caught before the aircraft is in flight. This will also confirm that the flight microcontroller is communicating with the optical microcontroller correctly.

Safety and hazard condition functions will be written to monitor the state of the aircraft at all times, including the altitude check, the attitude check, the power supply check, as well as the battery level check.

The flight microcontroller system implements a barometer for use in altitude analysis of the aircraft by taking external pressure readings. This peripheral will be used to allow the altitude class to read in the altitude for use in monitoring the safe flight of the aircraft. Altitude values will be critical to ensure the flight path of the aircraft does not allow the aircraft to collide with the ground or take the aircraft to an extreme high altitude.

The flight microcontroller system will also implement a gyroscope for use in attitude analysis of the aircraft. The 3-axis gyroscope will be used to allow the attitude class to read in the attitude of the aircraft for use in monitoring the safe flight of the aircraft. Attitude readings will be critical to ensure the aircraft does not make drastic movements, as well as to monitor the current stability of the aircraft for use in flight safety checks. These readings will allow safety functions the ability to determine if the aircraft is past safe attitude thresholds.

The battery level check will be used in monitoring the current levels of the batteries, including the microcontroller's system battery and the aircraft motor battery. Monitoring the batteries of the systems will ensure a successful flight. While unlikely, it may be possible for one of the systems in the aircraft to lose power without these readings, which could cause the aircraft to crash. Therefore battery checks will be required to take place to prevent crashing.

The power supply check will be used to monitor that stable voltage levels are available for various elements in the system. This check will help determine whether any components in the flight microcontroller system may not be functioning correctly. While it will not be a guarantee by any means, this will give the system an additional check to help ensure that the flight microcontroller hardware and peripherals are ready for flight.

4.4 Fourth version: Combination approach

In order to increase success rate, another camera was added which would increase target recognition distance. The second hardware change was adding two servos for the camera and the deadman switch. The software changes would reflect the addition hardware as well as the optical processing software. The optical processing software was again downgraded to the least complicated and time consuming level which would use very basic target color searching.

4.4.1 Fourth Version Hardware Design

After discussing the possibilities of the previous hardware, the decision to add hardware for safety and success was made. In order to increase the target distance and produce more flight time, a second camera was added which would create more case scenarios inside the code. Also, two servos were added to control the camera and deadman switch. It seemed much safer to add a servo to keep the cameras level with the ground which would not impact the final design

or code heavily. The additional servo used for the deadman switch would make the plane much safer for flight as well as produce a way to land the plane on command in case of emergency situations.

4.4.2 Fourth Version Software Design

As more talk occurred about processing capability of the selected STM microcontroller, the group decided should try to reduce the tracking algorithm as much as possible to increase the chance of success. However reducing the algorithm would require solutions to issues associated with the tracking problem which include occlusion, scale, rotation, illumination, noise, and vibration. In order to account for the problems of tracking, assumptions were made which will help increase processing speed by eliminating problem areas. The assumptions allowed for the simplification of tracking software to scan an image looking for a specific color value and determining the target size and location afterwards.

For the finalized locking software, the group decided to use the previous method of edge detection to drawn the objects shape around the target which will highlight it. The flight mode would use the most basic approach by searching for the target color then finding the center. However, with the addition camera added, the code will have to check if the zoomed camera is selected while in the collision function since that could trigger a landing even though the plane is still far away. Also, if the plane enter seeks mode at any time, the camera will be switched to zoomed mode to give best opportunity to find target.

4.4.3 Fourth Version Flight Microcontroller Design

As further research and discussion into the functionality of the flight microcontroller system was done, it was determined that the flight microcontroller system would need to implement a kill switch, or what is also known as a deadman switch. This is a safety mechanism which gives the team control over the aircraft’s motor power in the event of an emergency that is not caught by the flight microcontroller software. The aircraft will be equipped with a radio frequency receiver that will be communicating with a remote containing an RF transmitter. This kill switch can be triggered in one of two ways. The first is when the aircraft exceeds the range limit of the RF communication to the point that the RF receiver is no longer receiving data from the RF transmitter. Once the RF receiver is not receiving data, it will trigger the deadman switch. This will keep the aircraft from getting too far away from the acceptable air space. The second way the deadman switch can be triggered is through a physical button on the RF transmitter remote. This button can be pressed to send data through the RF link

to enable the kill switch. Once this data is passed, the RF receiver will trigger the deadman switch, which will kill power to the motor.

Even though the motor power will be physically cut off, the flight microcontroller system will still be intelligent enough to take the aircraft into a landing procedure so that the flight microcontroller landing procedure software can handle the landing as smoothly as possible. This will allow attitude adjustments as the aircraft glides to the ground.

5.0 Design Summary of Hardware and Software

The following sections outline the final hardware and software design. Each design section will completely cover the entire design structure. The hardware section provides schematics while the software section will cover the design structure. Class diagrams and more detailed information will be covered in section 6.

5.1 Hardware: Structure and Design

The final hardware structure contains five main sections: video to optical microcontroller, optical microcontroller to memory, optical microcontroller to display, optical microcontroller to flight microcontroller, and support circuitry for both microcontrollers. The next few pages contain sections of the schematic, broken into readable portions. The pages beyond the schematics contain detailed descriptions of each section of this guidance system. Descriptions of components, signals, and data formats from a hardware standpoint are included on the next pages.

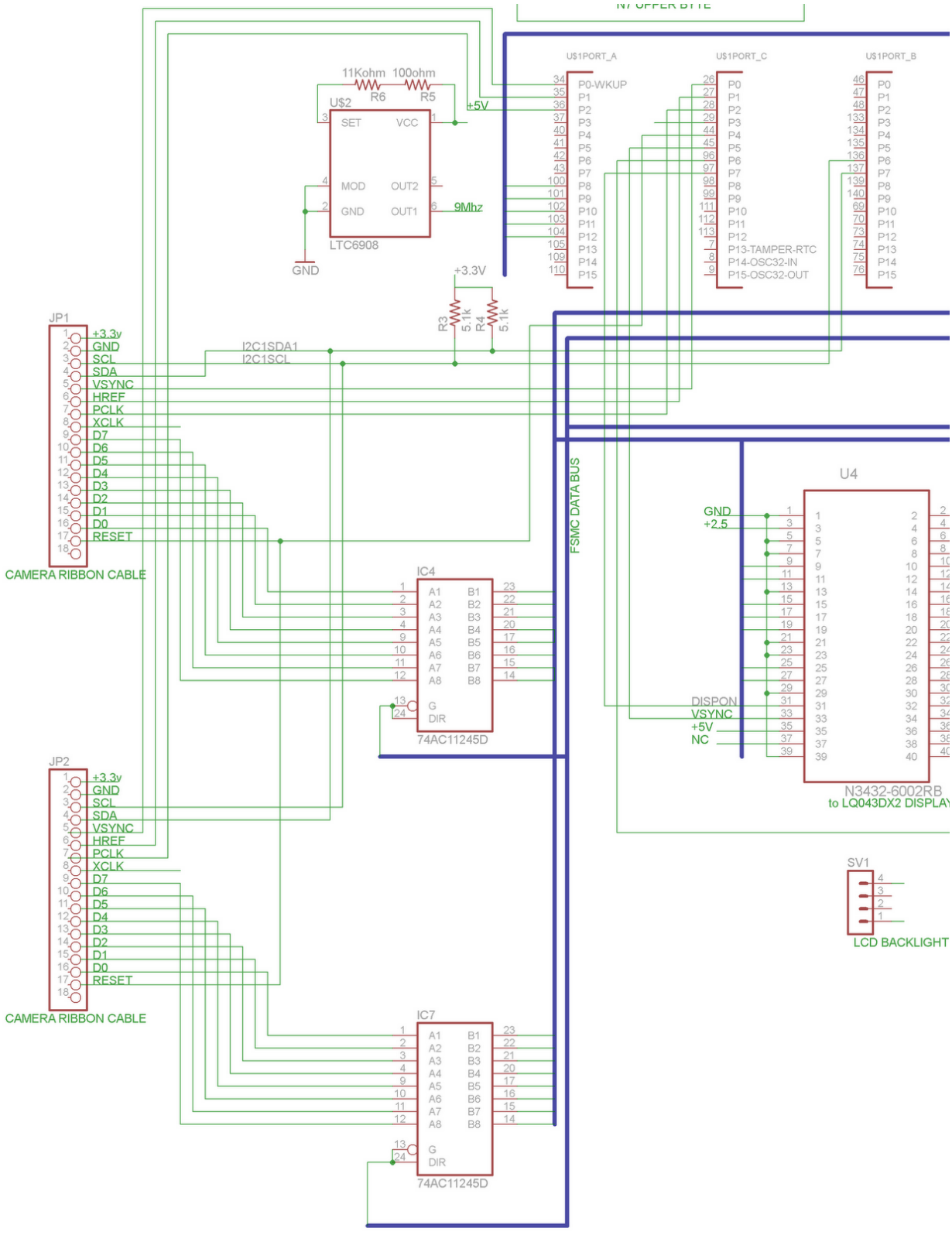

Figure 5.1A: Dual camera circuitry

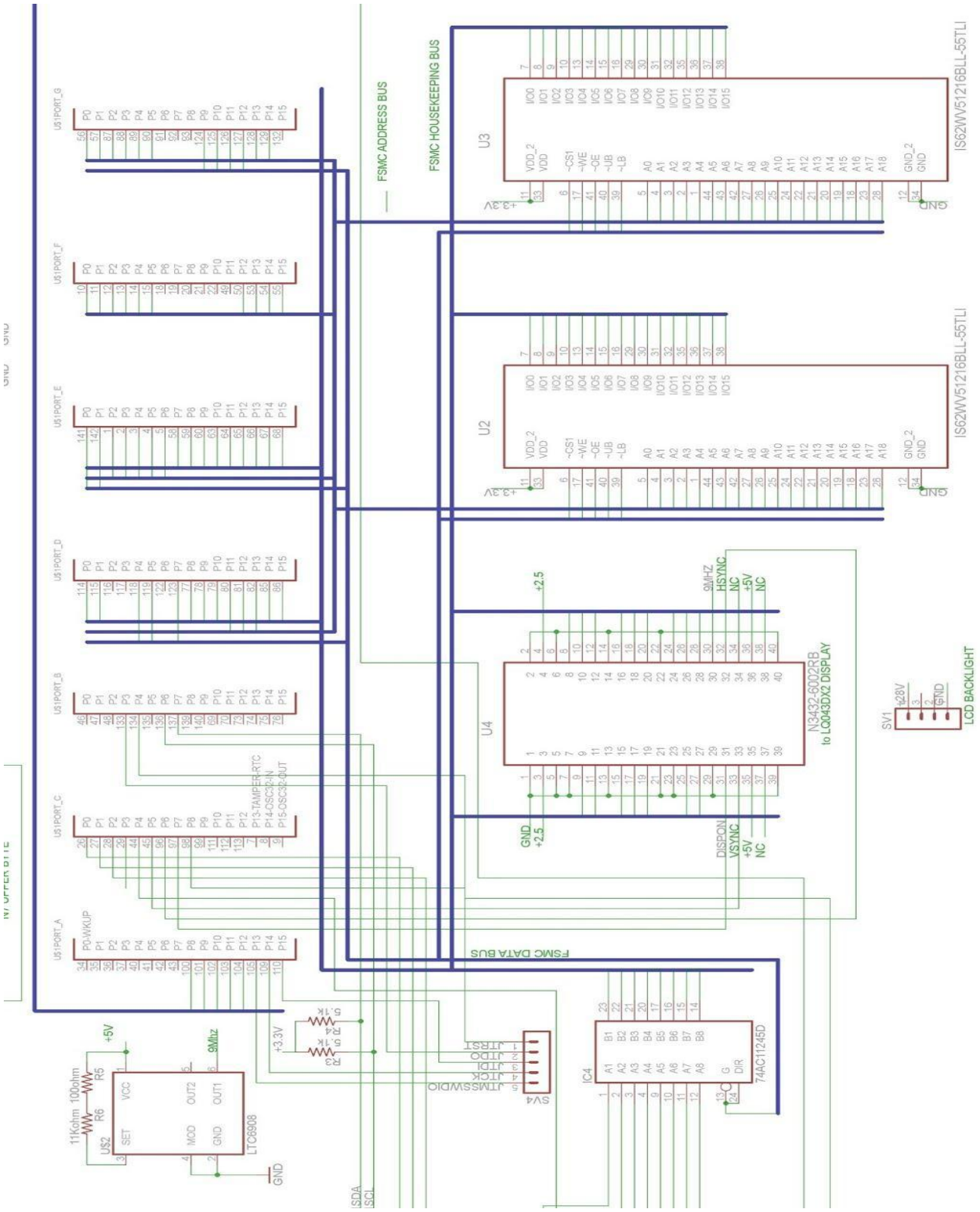

Figure 5.1B: Circuit showing optical portion which contains memory, microcontroller, LCD, and oscillator

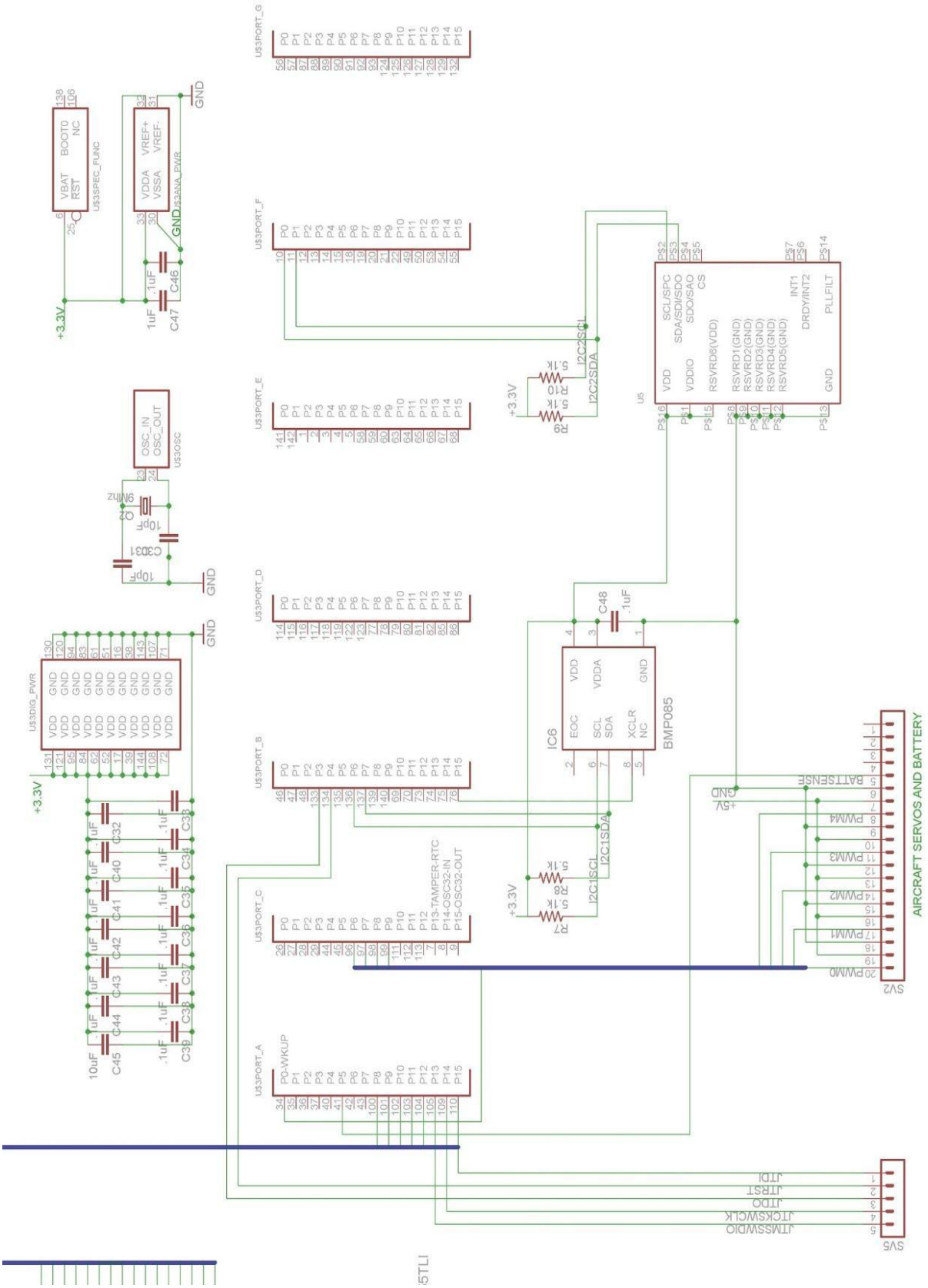

Figure 5.1C: Circuit showing the flight microcontroller, support circuitry, output headers, and sensors

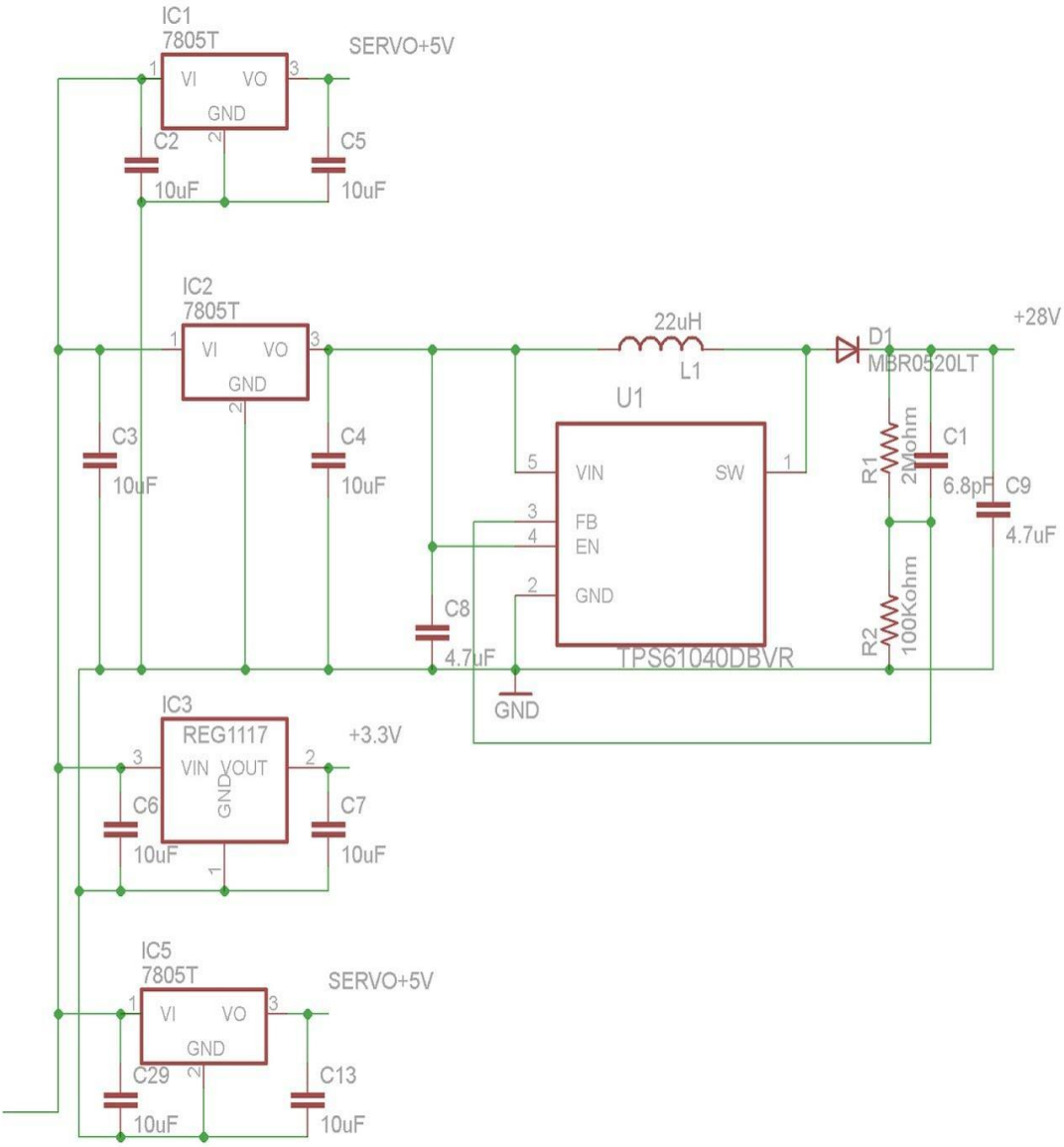

Figure 5.1D: Power supply circuitry

5.1.1 Video to Microcontroller

It was decided that the use of the most basic raw video color encoding scheme would minimize overhead of the system. In particular some color encoding scheme would require calculations of multiple pixels simultaneously to derive color value of each pixel. The raw video would be transferred from the camera as two bytes per pixel and assembled in the optical microcontroller. This has to happen in real time to maintain the integrity of the synchronization signals within each frame. It was also decided to use the dedicated memory bus facilities

integral the microcontroller which was selected in conjunction with direct memory access provisions.

For the first section, camera to microcontroller, focus is on the image from the camera getting to the microcontroller. The camera will take the image and use the software to convert it to information for the microcontroller. However, the image will first be stored in the memory in order to process. After the image has been stored successfully, the optical microcontroller will be able to compute the area the target is in and take appropriate action. This action will be sent to the second microcontroller, which is described in the next section.

5.1.2 Microcontroller to Memory

Video pixel data will be transferred between the optical microcontroller and system memory in 16 bit words. These transfers need only occur to provide real time image processing and properly synchronized data transfers heading to the display. The possibility of caching the video frame data within the optical microcontroller will be evaluated to examine if latency issues arise. Most of the memory available within the optical microcontroller is flash memory which has significantly higher read and write time and significantly increase latency of data movement. As a final option, separate hardware implementation of bus mastering architectures may be investigated. The decision to use asynchronous memory relieves the optical microcontroller of memory cell refresh duties.

5.1.3 Microcontroller to Display

The primary issue or workload associated with transferring data to the display is the microcontroller requirements that it maintains the timely transfer of video data while simultaneously generating the display synchronization signals necessary for the display to work correctly and for the image desired to be displayed properly. To offload workload of the optical microcontroller, a master peripheral hardware oscillator will be used as a pixel clock. At this point only vertical and horizontal synchronization signals for the display itself need to be sent from the optical microcontroller.

5.1.4 Microcontroller to Microcontroller

The microcontroller to microcontroller design process focuses on information transfer. The first microcontroller will transfer the relevant information such as the code for flight adjustment. A simple protocol for steering advice in binary format will sent from the optical microcontroller to the flight microcontroller. This data will

have to be held in a register in the optical microcontroller and read by the flight microcontroller when workload allows as opposed to some sort of interrupt or synchronization scheme which could adversely affect the tracking. If an interrupt scheme was implemented, there would have to be some type of priority setting inside code to signal the flight microcontroller. If there was no priority, the flight microcontroller could ignore the interrupt which would delay the adjustments and cause the plane to move off target or cause the tracking software to lose the target. A synchronization scheme would ensure the delay between microcontrollers to be low however the extra time needed could delay the process anyways. Since there would have to be communication between microcontrollers to keep synchronization, the delay needed could become excessive to cause the same effects as an interrupt scheme.

5.1.5 Microcontroller to Components

There are two main components which the flight microcontroller will control: sensors and servos. Sensors are the barometer, gyroscope which monitors the height and acceleration and the servos control flight control surfaces which are the rudder, motor, aileron and elevator and camera tilt.

Sensors: Data from the gyroscope and barometer sensors will be transmitting to the flight microcontroller via I2C communication protocols. The speed in which the flight microcontroller must process physical flight data then decide how to make control surface changes does not warrant a faster communication protocol. The sensors will be part of the criteria the flight microcontroller uses to maintain stability and proper steering of the aircraft to hit the target.

Servos: Typical mode vehicle servos simply require a pulse width modulated signal. Aircraft flight control surface position must be accurate and timely to maintain aircraft stability. The flight microcontroller will be required to simultaneously produce five pulse width modulated signals to control the aircraft and maintain camera level.

5.2 Software: Structure and Design

The software structure and design is broken into three sections which cover optical processing, optical microcontroller, and flight microcontroller. These sections are purely dedicated to outlining the software responsibilities.

5.2.1 Optical Processing Software

Since the optical processing software was restricted to bare minimums to avoid processing speed issues, there was a series of restrictions placed upon the target, area, and movement. These restrictions are summarized below in specific areas since some are mode specific.

The general assumptions and restrictions are as followed:
1) Target is within a certain distance because of camera resolution or a certain size, to make up for distance.
2) Target will need high contrast. Since optical processing algorithm is based on using color detection, target cannot have similar color to surround area since the plane could change target. High contrast also allows for defined edges for initial edge detection.
3) The target will not produce dramatic abrupt direction changes because target could be lost since slower processing time can allow for target to move out of camera. Depending on the distance moved, target could still be in range of camera which will not affect tracking. If a higher frame per second processing is possible after testing, the system's software could possibly pick up sharper turns and movements.
4) Target must be rigid or not be allowed to move independently. If the target is non-rigid, the calculation for target's center could be fail and cause erratic behavior. Also, if the target is non-rigid, the target might be affected by outside variables such as wind which will violate the abrupt movements.
5) During seek mode, it is assumed there is no object in the immediate surroundings which would cause a collision with the plane. This will allow the plane to enter seek mode and not present an issue with collision detection which would slow processing down.
6) The airplane cannot be launched at night since it could trigger an optical fail safe mode and cause the plane to land even though the target is available.
7) During target mode, there will be no objects obstructing the plane from hitting its target. Since the system does not implement object or collision detection, any object that is between the target and the airplane that is not big enough to lose the target will cause the plane to crash.
8) During edge detection computation, the system will assume that since the target is broken into quadrants which will reduce computational time by not searching opposing quadrants. For example, edge detection is used on quadrant I, which is based on the mathematically quadrants such that positive x and positive y values are upper right quadrant, negative x values will not be used, as well as move away from quadrant IV, negative y values. This approach bounds the target and reduces search time by eliminating known areas and will be applied to all four quadrants.

9) Optical fail safe mode will only trigger if the first pixel value for the frame is the bit equivalent of black. This will save time instead of having to break out of one routine into another routine to check if optical fail safe is triggered. This assumption should not impact the functionality of the project since the optical fail safe would trigger either through the optical microcontroller or will eventually set the first bit to black.

The optical processing software will be responsible for proper guidance and tracking of the target. This will require the software to have components which will process the data available from memory and produce the necessary result which will vary depending on the stage of flight. The two main stages of the software will be flight mode and seek mode which have various parameters. The goal of the flight mode is to find the target and continue producing adjustments for the airplane to converge and collide with target. The goal of the seek mode would be to find the lost target.

5.2.2 Optical Microcontroller Software

The optical microcontroller is responsible for handling the data flow of the images, manipulating these images and storing these images in memory. The microcontroller is will be directly connected to the non zoomed and zoomed camera, the external memory, the LCD display and the flight microcontroller. The best way to explain the optical microcontroller is to define it as the eyes and brain of the system. Since it is hooked up to both of the cameras it will be able to see where the plan is heading relative to where the target is. Since this microcontroller handles all of the frames and data flow to and from memory it must also interpret the frames. Therefore is responsible for sending the directions in which the plane should changes course to the flight controller.

The interpretation will be handled in the optical processing code which will be completely separate from the data flow and state machine code. Since the interpretations of the images are handled separately, the optical microcontroller system code will maintain the machines states as well.

Although the two microcontrollers have different state machines which will run independently, the optical microcontroller has the ability to change the state of the flight controller. For instance, when the user selected the target from the screen it’s the optical microcontroller’s responsibility to interpret the selected target and to change the state of both systems to flight mode. Since the flight controller is just the motor skills and flight sensors it will not know when to enter flight mode without the instruction from the optical controller.

Finally, one of the most important responsibilities of the optical microcontroller is the timing and synchronization control. Because certain are required to happen at certain times throughout the process, system timers will be used to make sure everything happens accordingly. Interrupts will be used to handle when fames should be captured, as well as for communications between microcontrollers.

5.2.3 Flight Microcontroller Software

The flight microcontroller software will be responsible for communicating with aircraft hardware and peripherals, such as the barometer, gyroscope, and servos. This software will be responsible for receiving and sending data to this hardware, and interpreting the data efficiently. Since the aircraft chosen for testing is an airplane with substantial velocity and that the software will be receiving data from hardware and peripherals during flight, it will be critical that interpreting and managing this data from the aircraft hardware is done correctly and efficiently. Aside from interpretation and management of the data received from hardware, the software will need to dynamically push changes through to hardware as well. Specifically, the software will need to push aircraft surface changes to servos throughout the aircraft. Since these updates are being pushed over during flight, it will be critical that these updates be accurate to ensure the safety and accurate flight of the aircraft.

This software will also be responsible for receiving instructions from the optical microcontroller. These instructions will be the eyes of the flight microcontroller. Any time an instruction comes in, the software will use this instruction to update the flight path of the aircraft. Without these instructions, the software will not know how to adjust the path of the aircraft. These instructions are also the only method of communication from the optical microcontroller to the flight microcontroller, so it is critical that the software appropriately read and handle these instructions so that it always understands the current state of the optical microcontroller.

Finally, the software will also be responsible for the safety of the aircraft during flight, as well as individuals and property within and around the aircraft's air space. Since this software is entirely in control of the hardware that controls the flight of the aircraft, it is absolutely critical that the software manage interrupts and hazard conditions appropriately. Without proper handling of these interrupts and hazards, it's possible that not only the aircraft, but individuals and property around the aircraft, could be put in physical danger. Therefore the code must be very thorough and consistent in safety and hazard checks during flight. If at any time during flight a hazard condition arises, the software will run an appropriate

procedure to handle it immediately, even if this procedure requires that tracking and interception of the target be aborted. To further help ensure the safety of the aircraft, as well as individuals and property around the aircraft, there will exist a deadman switch which can be physically triggered by the team if emergency situations arise.

6.0 Project Prototype Construction and Coding

The following sections will cover the construction plan with the first section covering hardware component acquisition. The second section discusses information regarding circuit layout and assembly while the final sections cover the software in detail. Each software section will cover the design theory and explain the coding process. These sections will provide flow charts, tables, and diagrams in order to facilitate understanding.

6.1 Parts Acquisition and BOM

Components will be sourced from larger vendors and resellers in an attempt to minimize the number of necessary packages and shipments. Resellers who will ideally provide discounts to college students will be strongly considered. Two key companies will provide a majority of the components needed throughout the testing and final phases of the project. After much research and pricing, Digikey and Mouser have proven to be the most cost efficient sources for parts. Each company will provide plenty of options for all necessities with their overwhelming presence in embedded systems component distribution.

6.2 PCB Vendor and Assembly

Routing files will be sent to 4pcb for circuit board production. Physical assembly, in particular the soldering, will likely be done by group or by the amateur radio club, although professional solutions will be identified and evaluated to cover all contingencies. This will minimize delays caused by shipments.

6.3 Optical Processing Coding Plan

This section will outline the code procedures and structure for the autonomous airplane. The code will start with initial functions to set the and then break into an infinite loop of case statements until either battery levels drop to insufficient values or collision procedures are activated.

The first procedure that occurs would be the power on self test which is explained below. Once the acknowledge has been confirmed by the optical microcontroller, the camera feed will be displayed onto the LCD with the crosshair and target detected adjusted which will be the target acquisition phase. If the button is pressed during this phase, the mode will then change to an overall tracking phase and the motor will start in preparation for launch. The overall tracking phase will be a case statement which looks at the first pixel in memory and then decide on the appropriate case.

If the first pixel in memory has a hexadecimal value of #0000, it will go into optical fail safe mode and either go into landing mode or go into tracking mode. If the first pixel in memory is the color of the target, it will go into collision subroutine which will then look at the other three corner pixels. If all four corners are the color of the target, it will access the collision mode. If all four corners are not the target's color, it will execute the normal tracking mode. If the first pixel does not trigger either optical fail safe mode, or the collision subroutine, it will default to the tracking mode.

Inside the tracking mode, there are two possible cases. The function will scan the image looking for the color value of the target, if found it will execute the function to find the center of the target which will then process the offset and send data to the flight microcontroller. If the color value of the target is not found in the image frame, it will default to seeking mode. Seeking mode would then continue until either the target is found again or until the airplane shuts down. Figure 6.3A below will show the overall flow for image processing.

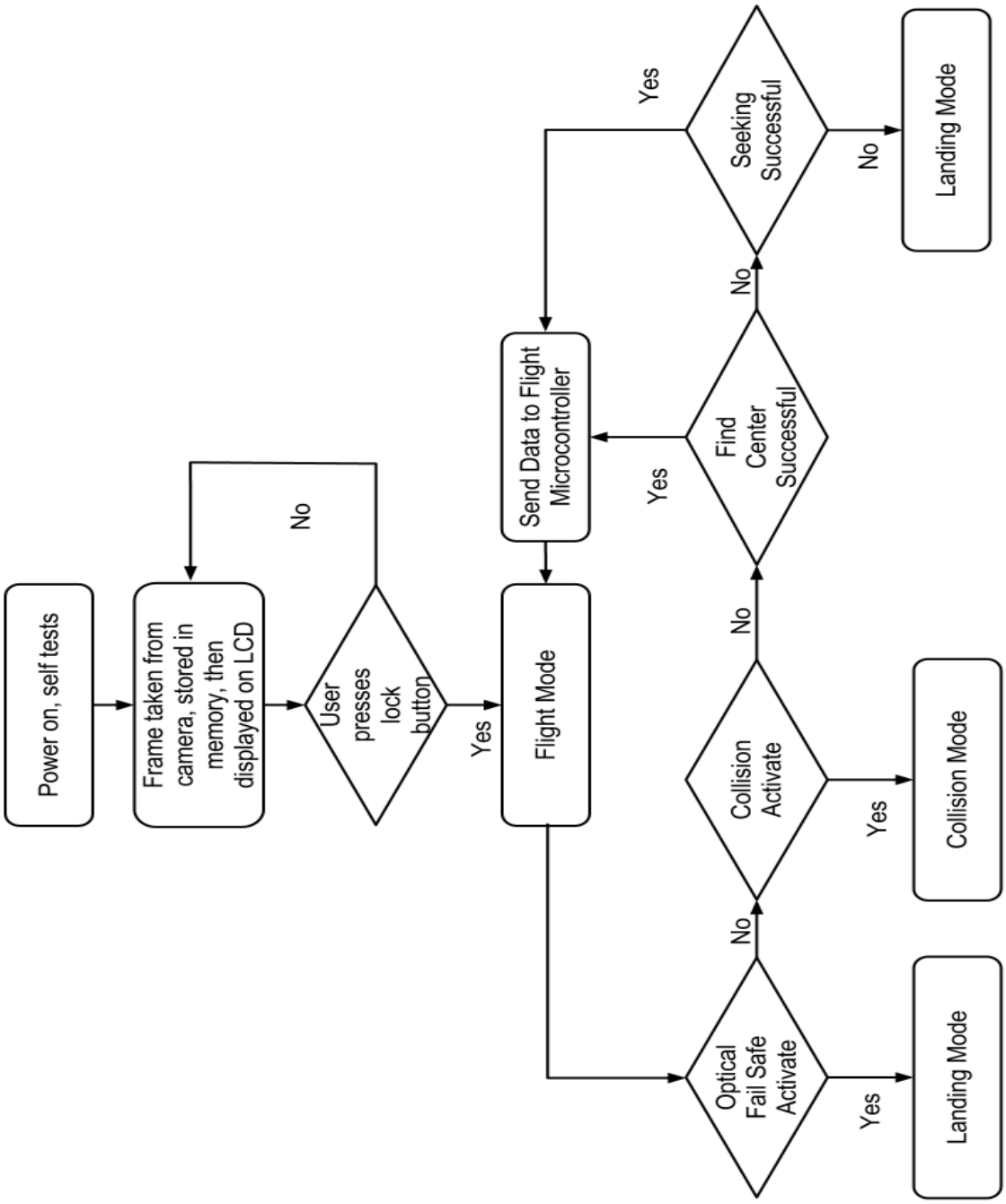

Figure 6.3A: Overall image processing flowchart

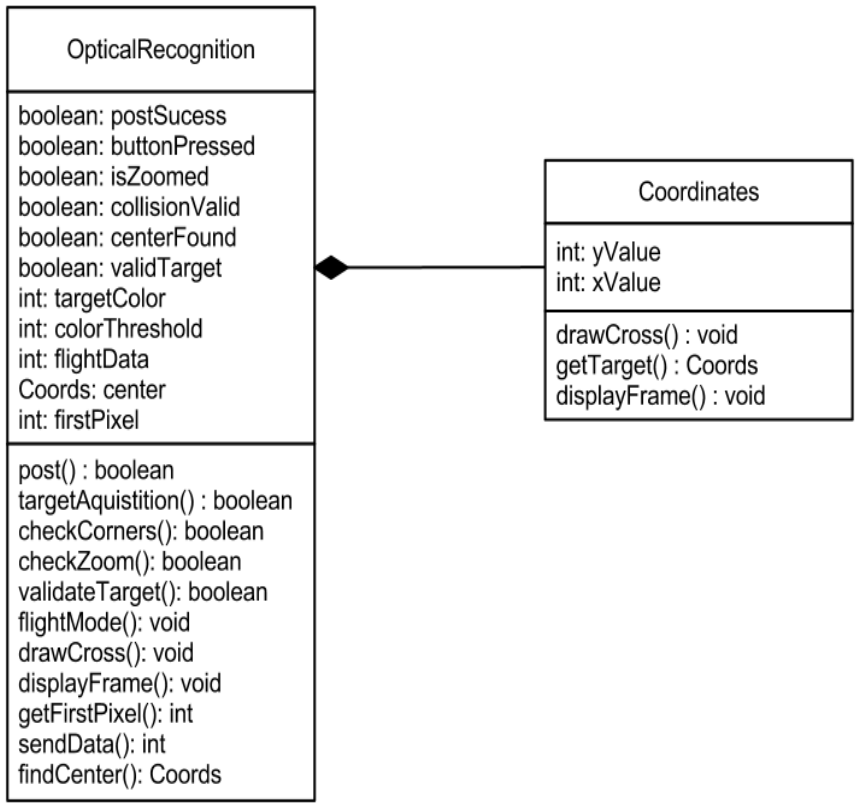

Figure 6.3B: Optical processing class diagram structure

6.3.1 Power on Self Test

The power on self test for the optical portion of the project is restricted to checking every pixel by going through the each primary color over a span of time by storing the rgb565 equivalents into memory then transferring to the LCD screen. The coded function will be broken into three sections, one for each primary color.

Since both red and blue have five bits to represent the color using rgb565, the functions will be identical except for the bit order which is changed. The 32 possibilities in both the red and the blue colors will be checked over a time span of two seconds which will ensure human vision can detect the changes of the tested primary color hue since it would be too fast for human eyes to accurately tell the changes otherwise.

Since the green will have 64 possible color changes because it uses six bits, the time used to transverse the spectrum will be increased to four seconds. This is done for the same reason as for the blue and red but increased because of more possible combinations.

While the basic code structure for all three functions will be very similar, there will be slight changes because of bit positions and quantity of bits. The function itself will start at the lowest bit for each color and store the corresponding value into memory for the size of the LCD. Afterwards, the content of the memory will be displayed onto the LCD which should turn every pixel of the LCD to the same color for a specified duration of time in order for human vision to detect. Next, the bit values will be increased by one and the same process of writing to memory, then displaying to LCD will occur for all possibilities of the color.

After the LCD tests are completed, the next step is for the optical microcontroller to send all possible bit patterns to the flight microcontroller. The bit pattern will be constantly sent for one second to verify that each component is moving correctly. Once all bit patterns are sent, the flight microcontroller will send an all 1’s acknowledgement back which the optical microcontroller will receive and turn the LCD to receive camera feed.

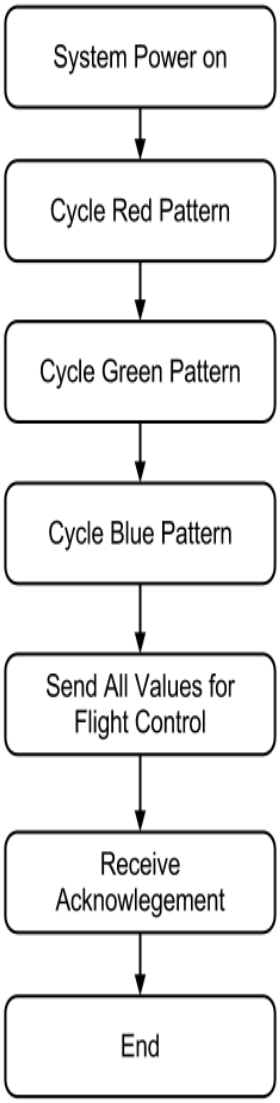

Figure 6.3.1: Flow diagram for power on self test procedure

6.3.2 Target Acquisition

The target acquisition mode will have three sections. Each section goes through sequential steps to make sure the current target is the correct target. The sections will shift through the initial lock on phase, and then a phase which will parameterize the target attributes, then to the final confirmation phase.

During the initial phase, an algorithm will be used it to do a primitive edge detection which will split the target into four quadrants and make a small assumption for each quadrant to decrease computational time. The algorithm will

start from the center and go up until the color threshold no longer applies. Since the assumption is being used, the algorithm can ignore all surrounding pixels except for the specific direction the target edge will be. The algorithm will use the assumption of starting in the first quadrant and continue to go right then down while tracing along the edge of the circle. The algorithm will expect to find the color value within the first two movements, if not, it will search the surrounding pixels while eliminating any possible pixels behind the original start position. There will be a special case near the end of the quadrant since the pixel will be almost in the same direction. This case will be absorbed into the surround pixel test by accepting a one distance movement.

During the next stage, the system will highlight the center 41 pixels in the shape of a “+” in the center of the LCD to make target locking easier. This will make a crosshair type structure to facilitate placing the target will be in the center of the screen. Once the target is in the center of the screen, the lock on button will be pressed. The lock on button will send a signal to the optical microcontroller which will start sending all data from current frame to store in memory.

The second phase will be the background processing code which takes the frame stored in memory and extract necessary information for target tracking. The first step is to extract the center pixel’s color value which will be stored as the target’s color value as well as an appropriate threshold to apply to the target color value to account for slight variation. From the center of the screen, the code will first branch in the positive y direction until the color value with threshold no longer applies. Afterwards, the system will reset back to the center and branch in the negative y direction until the color value threshold no longer holds true. It will then average the two values and adjust the y-center. The code will apply the same logic for the x-center which will get the target direction close to center. The result of the code will give the target’s center as well as the aspect ratio of the target. See figure 6.3.2 below for more details on target acquisition.

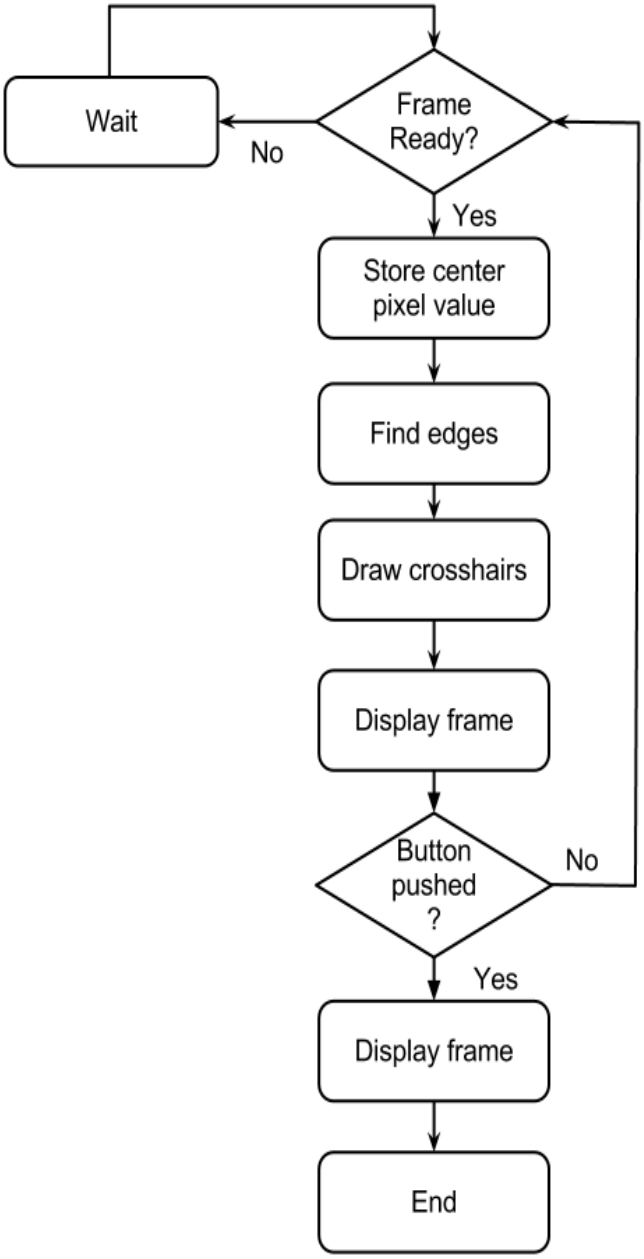

Figure 6.3.2: Flowchart for target acquisition

Since the project will use a spherical target to prevent an infinite loop, the centering logic will only be applied once. If the center of target happens to fall in between pixels, an infinite loop could occur because the algorithm would go back and forth trying to find the center. This could be prevented by either applying a tolerance threshold or limiting the number of times the algorithm executes. However, running the center algorithm once will provide an adequate target center location while minimizing the processing time.

The final phase of target acquisition is the confirmation stage which will display the image cropped to the size of the LCD with the target encapsulation using edge detection, the color value of target, and the aspect ratio

6.3.3 Target Center Calculation

Each incoming image will be taken and then scan each line going left to right, top to bottom. Comparison of the color value at each spot to the stored color value from the target acquisition phase will be made. Once the system finds a match to the stored color value, it will implement the code which is very similar to finding the center of the target. This will identify the center of the target with which the system can then calculate the offset to the center of the screen. There will be specifically designated sections of the frame which will be mapped to a code sequence. The sections will initially be proportional but depending on the adjustments necessary, the sections could be skewed for special reasons. The offset will fall into a designated section which is then sent to the flight microcontroller to adjust the planes current course. See figure 6.3.3A for breakdown designate section 6.3.3B for center calculation flowchart. During tracking mode, the code can trigger the optical fail safe mode, seeking mode, collision mode, and landing mode.

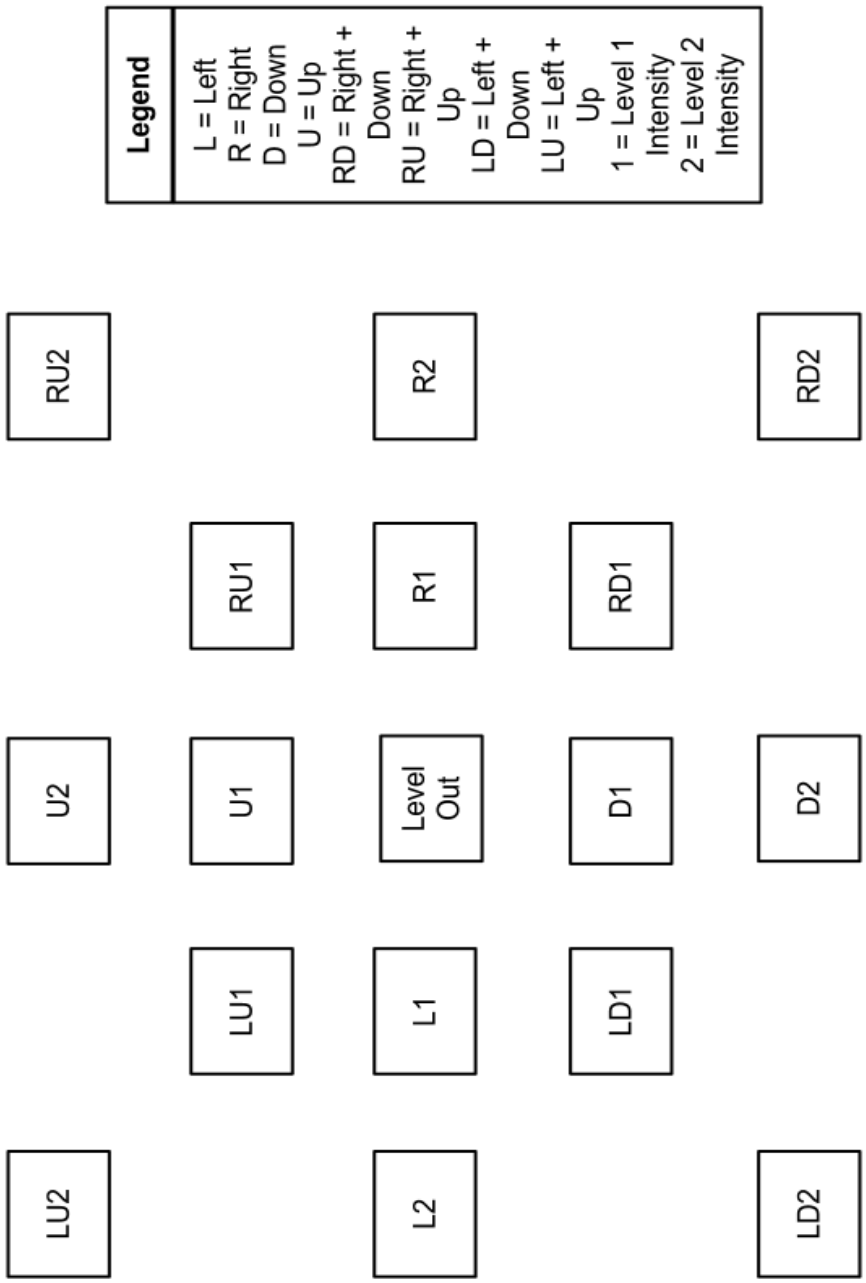

Figure 6.3.3A: Diagram showing points of interest for offset in respect to center of screen.

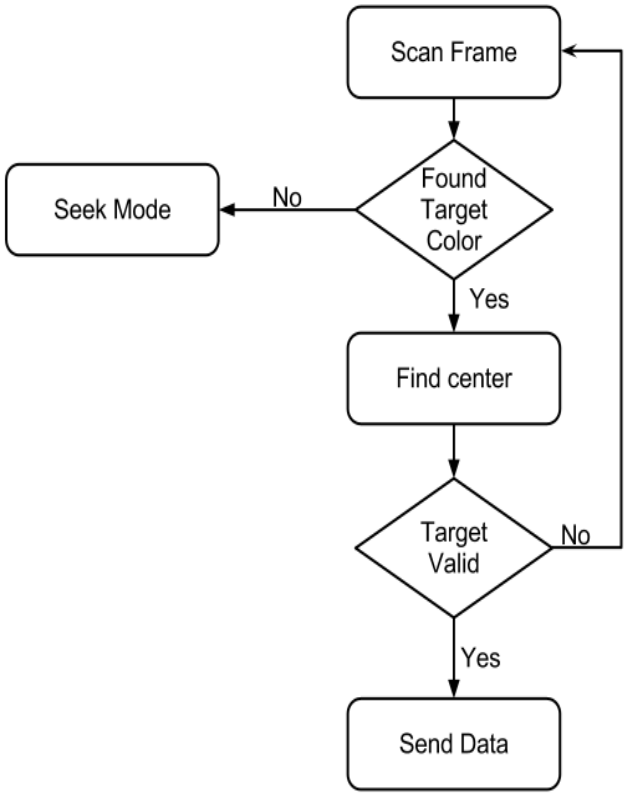

Figure 6.3.3B: Flowchart diagram for target center.

6.3.4 Optical Fail Safe

This section will describe the implementation for an optical fail safe. There are two situations which will cause the airplane to enter this mode. The first situation is if the microcontroller triggers the optical fail safe mode. The second situation is if the data transmitted is the hexadecimal value #0000. Upon either of these cases triggering, landing mode will be executed.

The microcontroller will be monitoring the incoming data from the camera. During normal operation, the camera supplies vertical and horizontal synchronization pulses for interpretation of the beginning and end of each image line and the whole image frame. These pulse trains occur at a fixed rate and if either signal is not received, the optical microcontroller will send the signal to land.

This is triggered by the data transmitted is the hexadecimal value #0000 which is essentially an all black screen. If the first value sent over to the microcontroller is #0000, the microcontroller will trigger a case break. This case break will cause the image to be briefly checked over specific areas for speed efficiency. In order to reduce the delay caused by the case break, coding to increase speed efficiency is preferred in case there is a false positive. If at any point the value check does not equal #0000, the case break will go back to code processing.

Inside the case break will check small boxes of 3 pixels by pixels spread over 9 locations in the image. Broadly and specific targeted areas of the image frame will be evaluated for a lack of color encoding possibly indicating possible camera system malfunction. Diagonal evaluation paths will facilitate efficient and quick evaluation of the entire frame. The locations will be hard coded into the function using a percentage of the resolution. Any locations using a percentage value, especially the locations which use 100% of the resolution, will have to be offset by the size of the box. This will keep the boxes in a diagonal line while not triggering out of bounds errors or possible wrap around. The first location checked will be the origin or (0, 0). The second location checked will be the end of the first line minus the area checked which would give the actual coordinates of (horizontal resolution - 3, 0). The third location checked will be 25% - 3 pixels of both resolution values which will give the actual coordinates of ((.25 * horizontal resolution) - 3, (.25 * vertical resolution) -3). The four location checked will be 75% - 3 pixels of the horizontal and 25% - 3 pixels of the vertical which will give the actual coordinates of ((.75 * horizontal resolution) - 3, (.25 * vertical resolution) -3). The fifth location checked will be the center of the screen and 50% of both horizontal and vertical resolution offset by 3 pixels which will give the actual coordinates of ((.50 * horizontal resolution) - 3, (.50 * vertical resolution) - 3). The sixth location checked will be 25% - 3 pixels of the horizontal and 75% - 3 pixels of the vertical which will give the actual coordinates of ((.25 * horizontal resolution) - 3, (.75 * vertical resolution) -3). The seventh location checked will be 75% - 3 pixels of the horizontal and 75% - 3 pixels of the vertical which will give the actual coordinates of ((.75 * horizontal resolution) - 3, (.75 * vertical resolution) -3). The eight location checked will be 0% of the horizontal and 100% - 3 pixels of the vertical which will give the actual coordinates of ((0, (vertical resolution -3). The final location checked will be 100% - 3 pixels of the horizontal and 100% - 3 pixels of the vertical which will give the actual coordinates of (horizontal resolution - 3, vertical resolution -3). See figure 6.3.4 below for location reference. Once optical fail safe mode has been triggered, there are two possible scenarios. The first is a shutdown sequence being sent to the flight microcontroller and the second is a return to tracking mode.

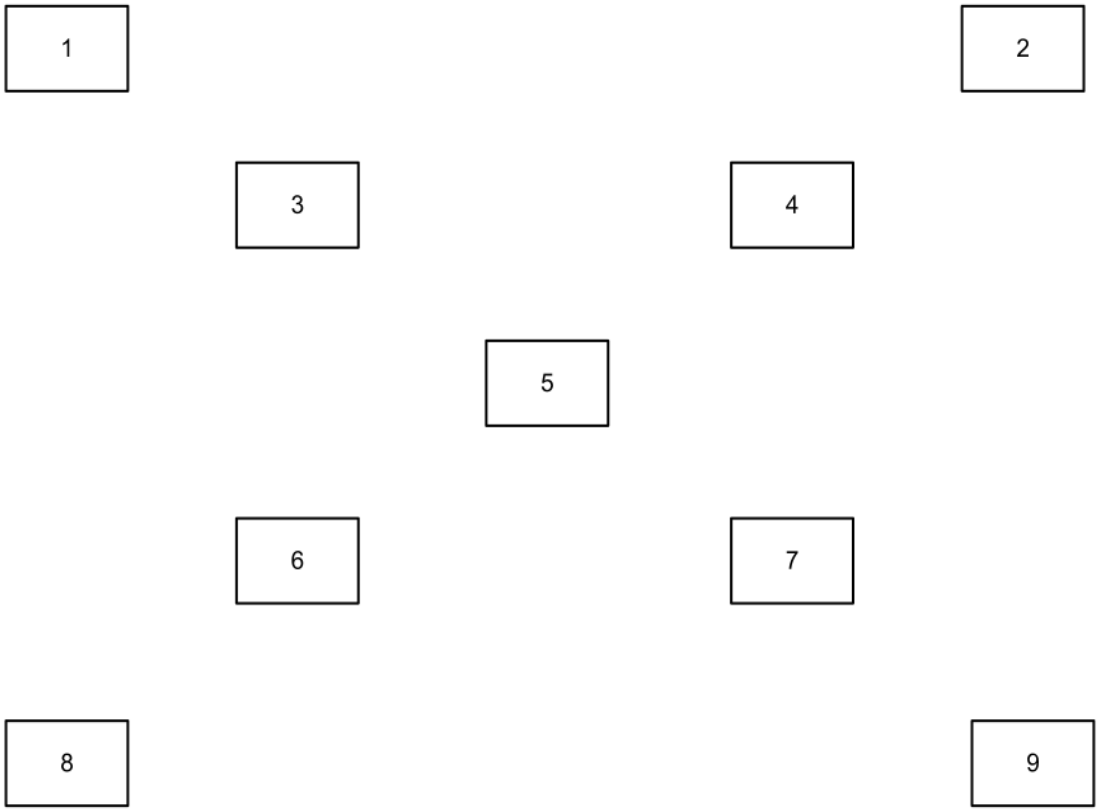

Figure 6.3.4: Diagram showing specific test spots for optical failure

6.3.5 Seeking

Seeking mode will be under the assumption there will be no objects in the area to avoid crashing into since collision detection will not be implemented. Seek mode will then go through multiple tasks which will have specific checks in order to break out of seek mode.

The first task that occurs once seeking mode is activated is to check the current camera. If the camera is currently on the short distance camera, it will switch to the long distance camera. If the camera is not currently on the short distance camera, it will stay on current camera.

The second task during seek mode would be to set a specific banking angle. The angle is specifically chosen in order to have the smallest radius while still keeping stability for the airplane. If the radius is too large, the risk of collision increases

greatly since the tested area would probably be the size of a football field which has approximately 160 feet wide.
The bank angle specifications relate to some limitations of airplanes. Most airplanes sacrifice symmetrical maneuvering capabilities to maximize flight stability. Specifically, while the aircraft turns fastest when baked to one side and using the elevator airstream for directional control, the aircraft can only turn quickly with the elevator in an up position. In addition, the code related to flight physics is far more complex if azimuthally course changes are allowed with the aircraft in an upside down attitude. For these reasons, the flight physics code will return the airplane to a right side up condition before attempting azimuthally changes. This consideration must be accounted for, regardless of the banking angle limitation, because of the possibility that the aircraft may be excessively rolled by the wind. Once the banking angle is set, the system will move onto the next task which is seeking for target.

The third task involves the coding process which is broken into two sections, the first seeking color and the second is confirming target. The first part will just be searching for the color value of the target including the threshold which is stored while the second part will be focusing on confirming target by implementing the target centering code. If the possible target has enough pixels to be declared the correct target, it will go back into target mode.

During the seek phase, the optical processing algorithm will start at the first line and compare each pixel to the stored value. If the pixel value match, it will move into the confirmation coding. If none of the pixels match on that line, it will then jump six lines. Since the target must meet a minimum size restriction of five pixels, it is a safe assumption that the system can jump six lines while not overlooking the target. This assumption is used mainly to save time for seeking since the team would like to get as many possible frames processed per second while in seek mode since it would allow the system to not to miss any portions of the area while seeking. The process of checking each line then jumping six lines will continue until a possible target is found, or until the frame is completely scanned. If the frame is completely scanned, it will wait until the next frame to process is ready and repeat this process until either the target is found or until the battery fail safe is trigger.

If the seek phase does match a color value, the confirmation phase occurs which will determine whether it is a false positive. This phase will use the target centering coding since it also incorporates the minimum size restriction which eliminates any possible false positives that might have been picked up in the surrounding area. If the target is valid and exceeds minimum size restriction

which is done in the confirmation phase, tracking mode will be executed. Figure 6.3.5 below outlines the seek mode procedure.

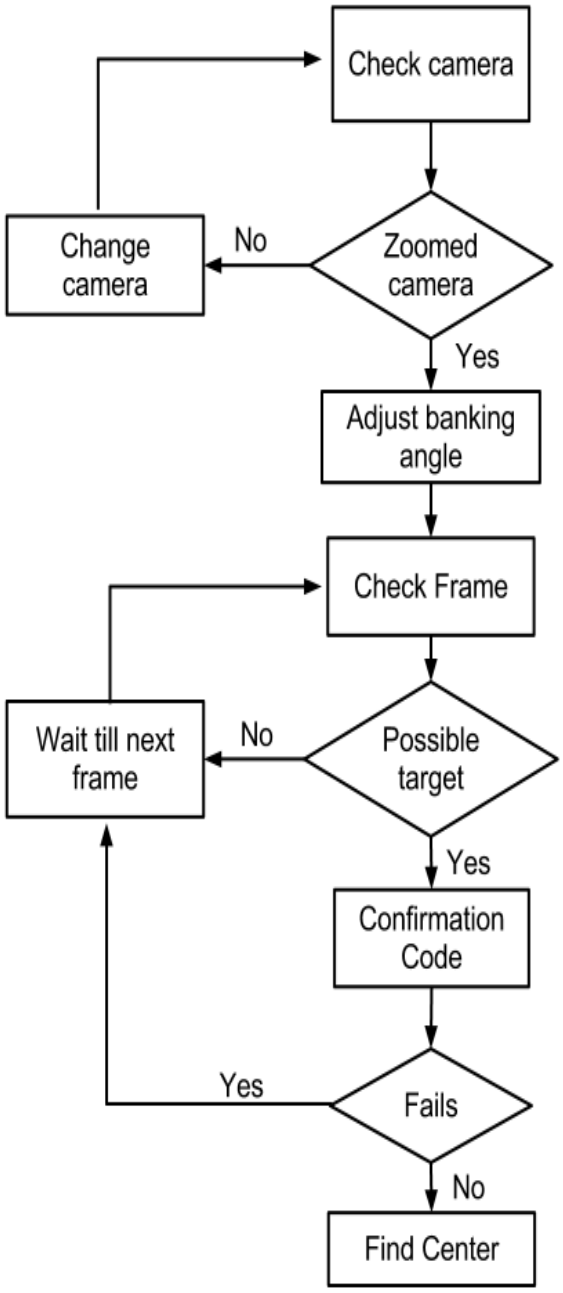

Figure 6.3.5: Flowchart showing the procedures of seek mode

6.3.6 Collision Mode

While in flight mode, collision mode can be triggered only by the first pixel matching the target pixel color. If the color matches, there will be two simple

checks before collision sequence is sent to the flight microcontroller. The first check will to see if all corners of the frame match the target pixel color which will imply the target has covered the screen of the camera. The next check is to see which camera is currently being used since the zoomed camera will be further away when the frame is filled. If both the non-zoomed camera is selected and all four corner pixel colors match the target, then the collision sequence will be sent to the flight microcontroller. However, if the zoomed camera is selected, the zoomed camera will be switched to the non-zoomed camera which will be done by setting a flag for the optical microcontroller. Figure 6.3.6 shows the procedure for collision mode to activate.

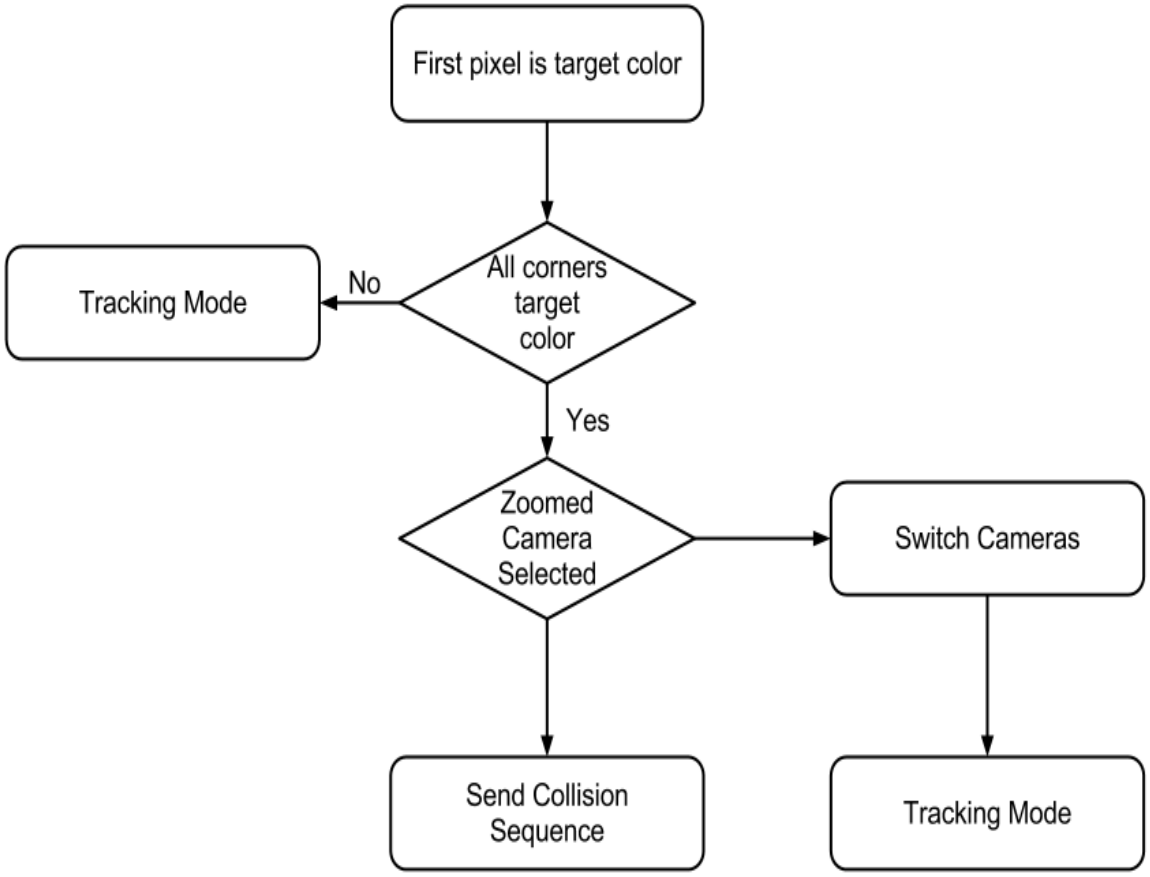

Figure 6.3.6: Flowchart showing the procedure of collision mode

6.4 Optical Microcontroller Coding Plan

The optical microcontroller is considered the “primary” controller of the system. Since it is in charge of the data flow between the core components of the system and making sure everything is synchronized it must be efficient and reliable. While the optical microcontroller does handles the image processing this section will only cover this section will only explain the procedures for how the optical microcontroller will control the flow of the data across the system as well has the steps for changing states, the procedures for each state and why the system is structured the way it is.

6.4.1 Optical Microcontroller Procedures

Initialization Procedure: On power up the optical microcontroller will start in an initialization mode. This procedure includes all necessary start up tests and verifications to ensure everything is connected and in synchronization with the system. Since this microcontroller will be connected to several external devices that are considered to be main components each one must be tested independently. The orders in which they are tested are as followed below but are subject to change if necessary.

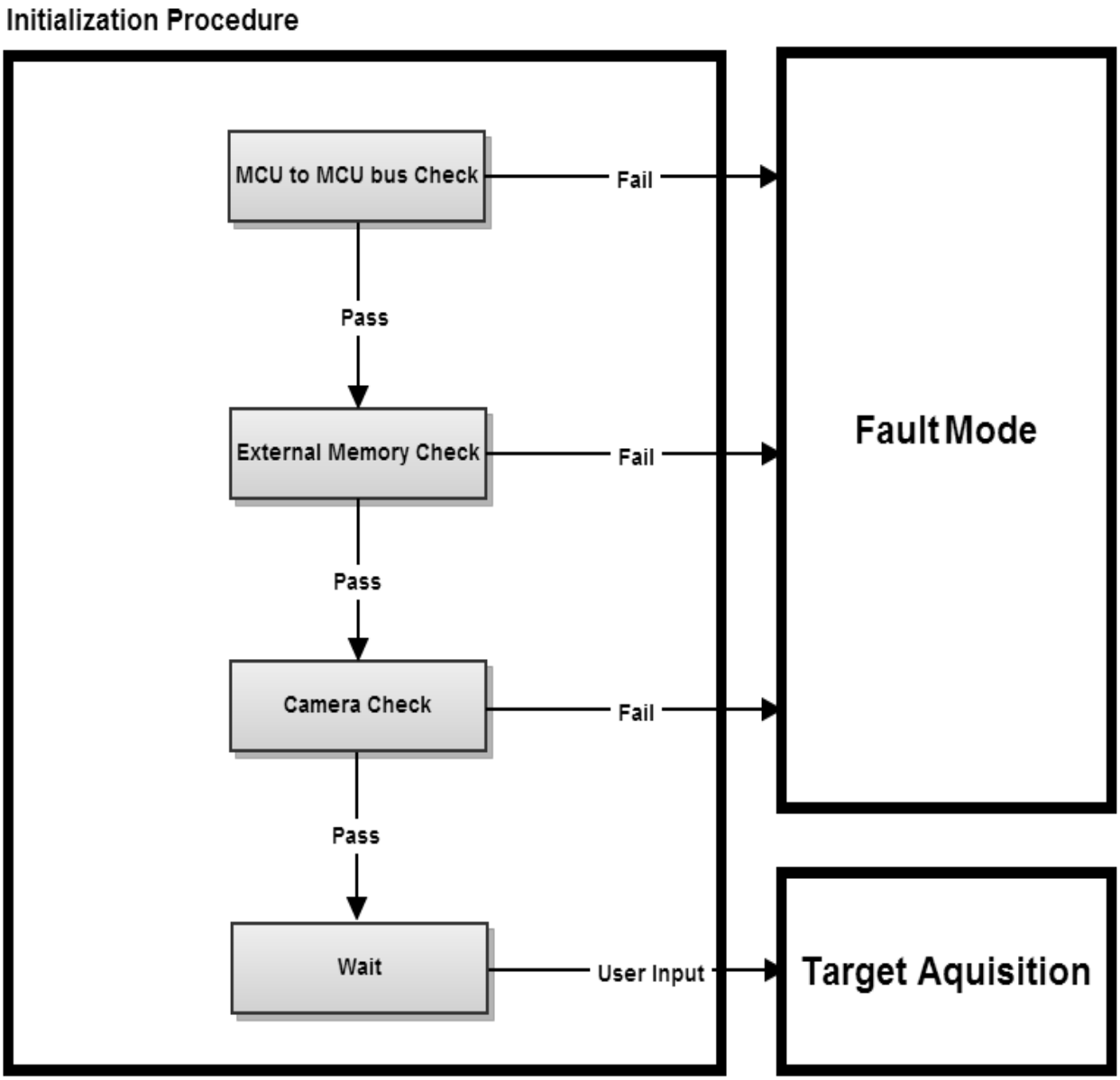

Figure 6.3.6: Flowchart showing the procedure for initialization mode for the optical microcontroller.

To verify that the microcontroller bus connecting the flight controller to the optical controller is working and in synchronization a simple handshake message will be sent across the bus once. If the message is received by the flight controller and is interpreted correctly it will respond with its own message. If this response back is also correct than the bus is working correcting. The messages will be simple 5-bit value since there is a 5-bit bus linking the two microcontrollers together.

The next verification the optical microcontroller must perform is on the external memory. Because the memory system is used to store the image information for the optical processing it is crucial to verify each of the 16 data lines in the bus are in synchronization and working correctly. In order to do so a sequence of 16 bit values will be sent to memory testing each case and will then be read back. If all values match the values sent than the memory verification step of the initialization process is complete.

The last component that will be verified in the power self test mode or the initialization procedures is the camera. This procedure is very simple in that all that needs to be done is to read in the 8-bit data input lines from the camera. Because the camera is constantly running, there should always be a flow of data across the bus which will be sampled on command starting from a fresh loop. This procedure will read the register values where the data is being stored over a few milliseconds and verify that the values are not constantly zero. If for any reason a specific bit does not change off of zero it can be assumed that there is some sort of malfunction between the camera and the optical microcontroller. This Test only ensures that there are no problems between the microcontroller and the camera. There is a separate function call from this controller that allows the optical processing code to run through its self test which is described in section 6.3.1.

Target Acquisition Procedure: After the optical microcontroller completes the power on self tests and everything checks out correctly, the controller changes states to Target Acquisition. In Target Acquisition mode the optical microcontroller is responsible for handling the data flow between three main components. The data will flow from the camera to the memory and from the memory to the LCD where it will be visible for the user to select the target they want to hit. With the two camera system in effect, the camera selection for acquiring the target will be determined based on the distance of the target. Four pins were used to allow switching between the full zoom camera and the standard camera lens. For the target acquisition procedures detection will be performed primarily with the zoom lens since this will be the furthest away from the object the system should ever be. In this state live streaming of the video feed from the camera to the LCD will occur but some camera data manipulation must occur. Because the data from the camera is much larger than the display on the LCD the image will be stored into memory and use a function to chop off the excess data. Despite the lower resolution of the LCD there should be plenty of information to be able to select the target, particularly considering the target must be in the center of the image. When the target is finally selected by the user a signal will be sent across the microcontroller bus to the flight controller

signaling the target has been set and the system should advance to the next state. Power to the LCD will be shut off and the process used to transfer the image to the screen will be halted. The last step in the procedure is to change the state of optical microcontroller to Tracking.

Tracking Procedure: In tracking mode, the optical microcontroller will handle data much like it does for the target acquisition but with some differences. The most obvious visual difference would be the disconnection of the LCD from the memory. Since the LCD will no longer be needed after the target is selected, in order to save energy the system will remove the power consumption of the LCD. By doing so battery life will be extended which will help ensure that the plane will reach the target destination without draining the battery dead. Behind the scenes there are many differences that separate the two modes. One major difference is the way images are captured and stored for image processing. While the camera can record the data and stream a live feed, the system will not be using this ability fully. Instead it will capture still images as quickly as the optical microcontroller can process the previous image and send steering advice. Because the airplane travels at a relatively low velocity, the system should be able to perform its duties sufficiently while processing at least five image frames per second.

Another feature unique to the tracking mode which may be implemented is the multi camera setup. In order to maximize the range of tracking, a second zoom camera will be used during the target acquisition and flight sequence. As the airplane gets closer to the target the optical processing code will calculate whether or not the system can begin to use the second, wider angle camera. The process of calculating when the correct timing is explained in section 6.3.6 above. Because the two cameras will be sharing the data bus as well as all of the time signals associated with the synchronization, camera switching will be performed between image frame transfers. By switching the pin high, it will automatically stop transferring the data from the zoomed camera and start transferring the data from the non zoomed one.

The key procedure for the tracking mode is how it handles relaying the information to the flight controller. A system tick will be used to maintain the capture rate. On a system tick the optical microcontroller will send a signal to the camera which will request a frame. After the frame has been captured and stored into memory function calls to handle the image processing will occur. After the image processing handles its computations the function will return values signifying where the target is in relations to the center point of the camera. These values will be used to send signals across the MCU to MCU bus to relay the

information which can be done by setting the register values associated with the bus which can simultaneously send all five bits across with one single function call. This is done to optimize the speed of the information to the flight control microcontroller. By constantly updating these registers and altering the data on the bus the flight controller can constantly check for any changes on the bus and run off interrupts.

Collision Procedure: When the optical microcontroller is in tracking mode it will constantly call the image processing functions. One of the features in these functions is collision detection. If the image processing function determines that a collision is going to occur it sets a flag in the optical processor. When this flag is set the optical processor sends a signal to the flight controller module across the MCU to MCU bus to set it into collision mode. In this mode the flight controller turns off the propulsion motor to avoid significant damage. After the optical processor sends the collision message to the flight controller, all image processing function calls and the synchronization signals to the cameras are stopped. It then changes state to standby mode.

Shutdown Procedure: This procedure consists of turning off all unnecessary components that consume power in the system. If this mode is to occur its either because the target has been hit in which case the flight controller will attempt to land the vehicle or a fault has occurred in which the plane must land because it’s in an unsafe state. In either case, neither the cameras nor the memory need to be working therefore they can be turned off. Along with shutting down these components, this state sits idle until a user response is sent to correct the error in which the system starts from the beginning with the target acquisition mode.

If the system is put into shutdown procedure while shutdown process is invalid, then an I/O pin connected to an LED will turn on signifying an error has occurred and the system used its safety landing mechanism to salvage the plane. Based on how frequent the led blinks, the fault causing error can be determined and debugged.

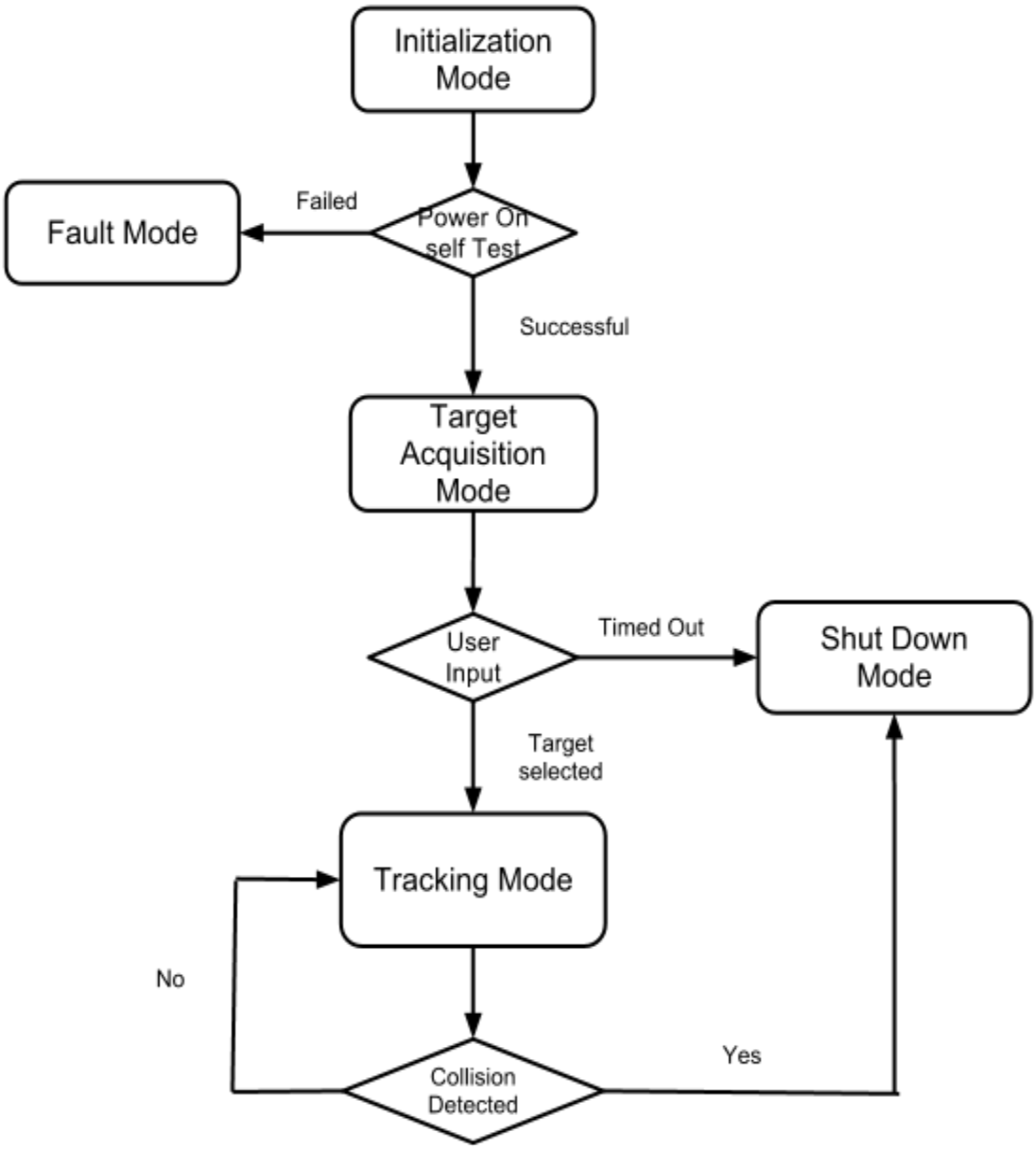

Figure 6.4.2: Flowchart showing the procedure of the optical microcontroller.

6.4.2 Optical Microcontroller Class Structure

The optical microcontroller is broken down into three major classes apart from the main class. The main class is structured around a state machine which is used to move through the A.O.G. Systems procedures. Two of the three major classes have to deal with handling peripherals. The first of these classes is the memory management class which is used to handle events that use the memory bus. The other is the interrupt class which handles communication with the camera as well has the second microcontroller that takes care of the flight routines. The third and most important class used by the optical microcontroller is the image processing class. This class is broken down into subclasses because it handles many different functions. This class's features are broken down in section 6.3.

6.4.3 Memory Management Class

One of the largest and most important classes for the optical microcontroller is the memory management class. This class is responsible for handling all of the image data transfers between the camera and memory, the transfers of the data from memory to the LCD and the manipulation of the image data during the processing. Because external memory is being used, functions will need to build to access memory addresses and to write to this memory bank. The memory management class diagram is shown in the figure below.

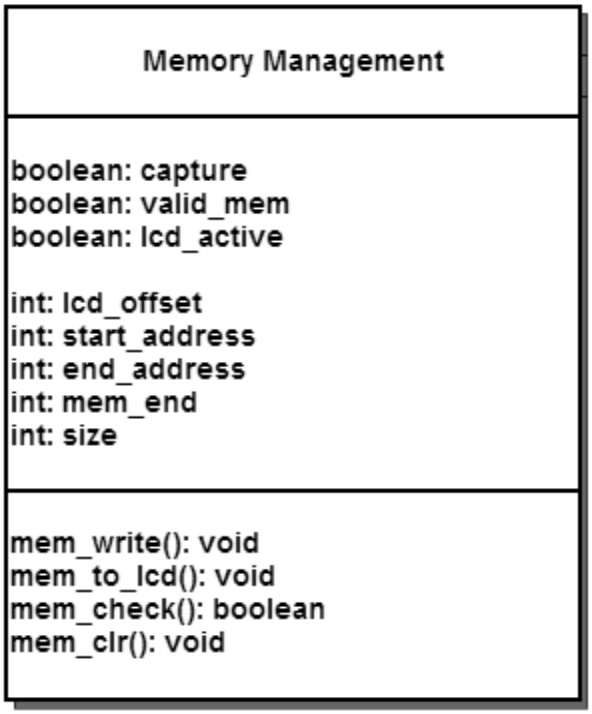

Figure 6.4.3: Memory Manage class diagram.

mem_write(): Since the camera will be taking the same size image for each capture, a static function can be used which will write the whole image to the memory bus. The mem_write function will send a synchronization signal to the camera which will allow the flow of data across the memory bus. It will also send a signal to the memory address bus defining where the image should be stored. Using the synchronization pulse from the camera the function will know when the end of the frame has occurred and shut off data flow to the memory bus. If the function works correctly each time it is called an entire frame will be stored in the memory which can be accessed by its starting memory address.

mem_to_lcd(): If a direct read on an image to the LCD screen was attempted, it would be distorted and not display properly because the camera has a larger resolution than the LCD. In order to fix this resolution problem, the edges of the image will be trimmed off during the target acquisition phase of the system. This function will not destroy or manipulate the actual image in anyway but it will ignore specified regions when outputting the LCD display. In order to keep the resolution of the camera, only specified regions in memory will be ignored. Since the target should be centered in the image anyways the outer edges are unnecessary for detection anyways.

mem_check(): The memory check function will be used to verify if a large enough memory block is available for the next image. Since image frames will be constantly added to the memory bank and because the image frame will be quite large, there will be an image check to ensure the image will fit into one memory block without truncating. This function will also allow the option of fitting as many images into the memory bank as possible before the need to remove an older frame. Instead of writing over an existing frame every time and potentially receiving altered data, the entire memory field will be written across before erasing.

mem_clr () - When the memory becomes full there needs to be a way to clear the bank for a new set of frames. The mem_clr function will erase the entire segment of memory when a flag is set stating that the memory bank is full. If this flag is set while an image is trying to be written, the image will be stored temporarily, the memory bank will be cleared and then the image will be inserted into memory for the image processing to evaluate.

6.4.4 Interrupt Class

The A.O.G. System is very reliant on timing and ensuring that all components stay in synchronization throughout the whole test procedure. Therefore, it is essential that an interrupt class be created. This interrupt class will contain the key functions to ensuring the timing of the system stays in sync. The interrupt class will also be responsible for determining when event should occur as well as supporting the communication between components. When dealing with interrupts it is very important to make sure priorities are set for each interrupt. Key interrupts that must be handled first should have the highest priority over others. When structuring this class it will design in such a manner as to preserve the process of the interrupts. These interrupts should contain relatively short procedures that should simple signal for a request or handle minor routines. The Interrupt class diagram is shown below.

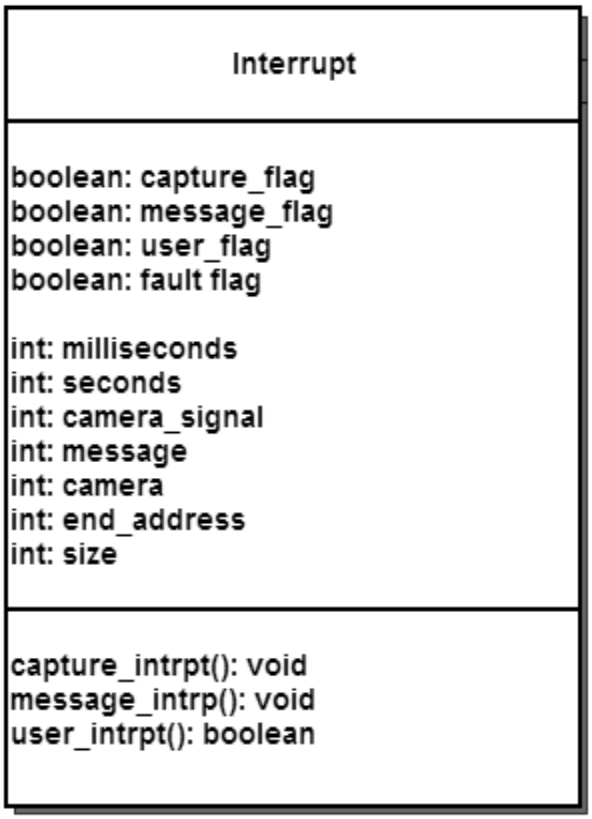

Figure 6.4.4: Interrupt class diagram.

capture_intrpt () - This interrupt will be used to determine when the optical processor need to capture a new image. Since the A.O.G. System will be running on a system tick, it is guaranteed that events will happen at specific times which enable the use internal timers that can be easily modified. For instance, since the specifications require the system to take five frames a second, every two

hundred milliseconds this interrupt will occur. Inside the interrupt will be a routine to signal to the camera which will allow the flow of data to the memory bus.

message_intrpt () - Because the system contains two microcontrollers each handling their own routines independently, there must be a communication link between them to make sure both controllers are in the correct states in relations to each other. For instance if one system is in the flight mode so should the other. While both state machines don't have the same states they do share some form or parallelism meaning when one microcontroller is one a specific state, the other should also be in its specific state. The way this is done is across the 5-bit bus. In order to ensure that messages between the devices are handled immediately an interrupt will be used. This interrupt will monitor when a change has occurred across the bus and stop what it’s doing to receive the message. If a message is received, then state changes can happen immediately. The interrupt routine will set a message flag signifying a message has come across the bus which will trigger a separate event in the main code to handle it.

user_intrpt () - The A.O.G. System is an autonomous system but does require some user input, specifically when selecting the target from a live image feed. In the target acquisition stage of the system, the user must push a button when the optical processing is selecting the right target. Since there maybe be more than one option on the screen the user must have some control to guarantee the correct target. Since this mode will be running in an infinite loop because it will be streaming a video an interrupt must be used to capture the target from the screen and to allow the system to change states. This interrupt will just wait until the user signal goes high then set a flag that will trigger some optical processing events. These events would capture the target from the image. This flag will also allow the optical microcontroller to change states and send a signal to the flight controller for it to changes its states too.

6.5 Flight Microcontroller Coding Plan

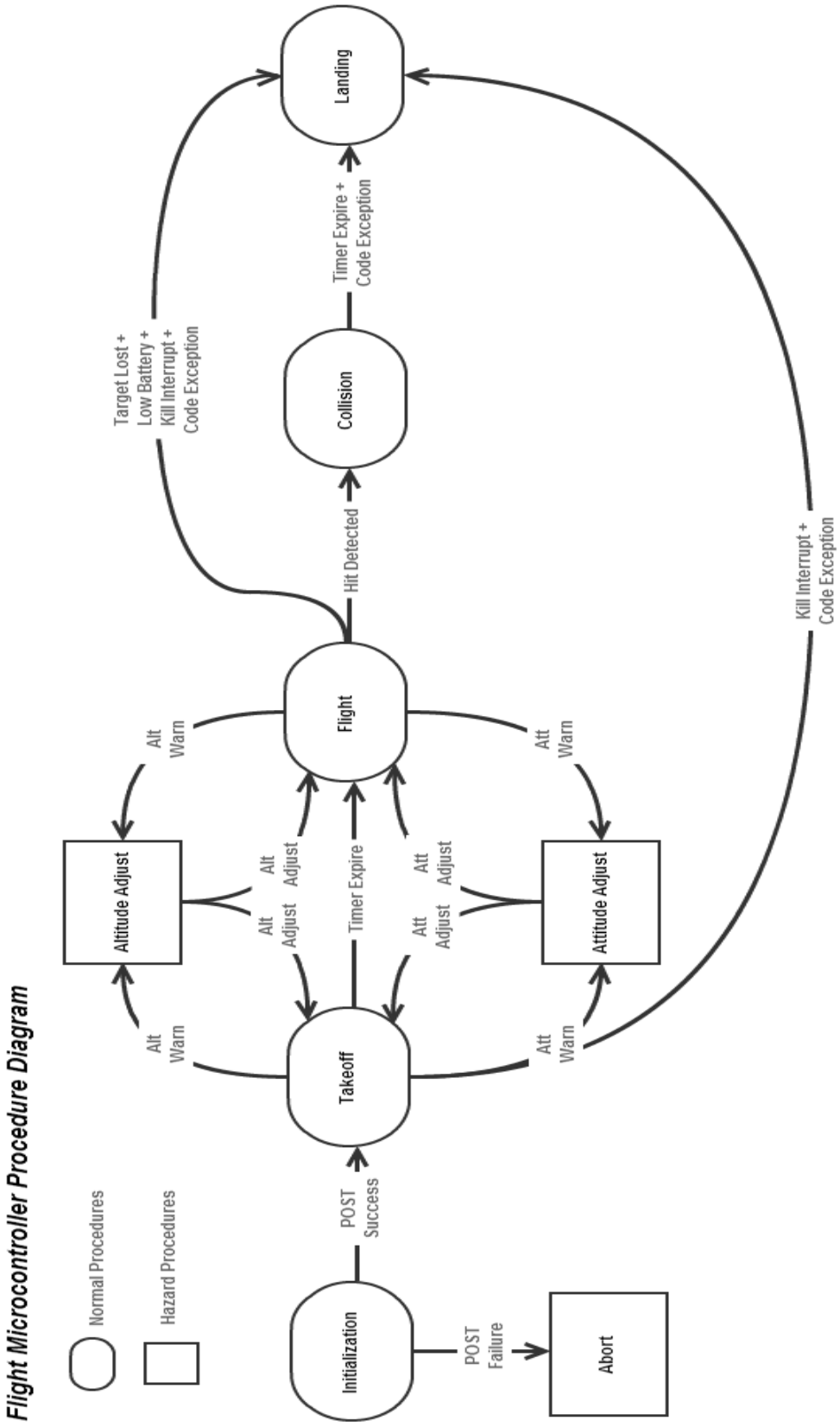

Fig. 6.5A: Flight microcontroller procedure diagram.

6.5.1 Flight Microcontroller Procedures

The flight microcontroller procedures will be used to control what procedure function loops will be occurring at any particular time throughout the flight of the aircraft which can be thought of as states. The aircraft will always be in single procedure at any point in time. The particular procedure that the aircraft is in will control what functions and checks are performed while in that procedure. This will allow the aircraft to adapt to different requirements more easily as they arise.

The current procedures to be implemented are the initialization, abort, takeoff, altitude adjust, attitude adjust, flight, collision, and landing procedures. Each of these procedures is described in detail below.

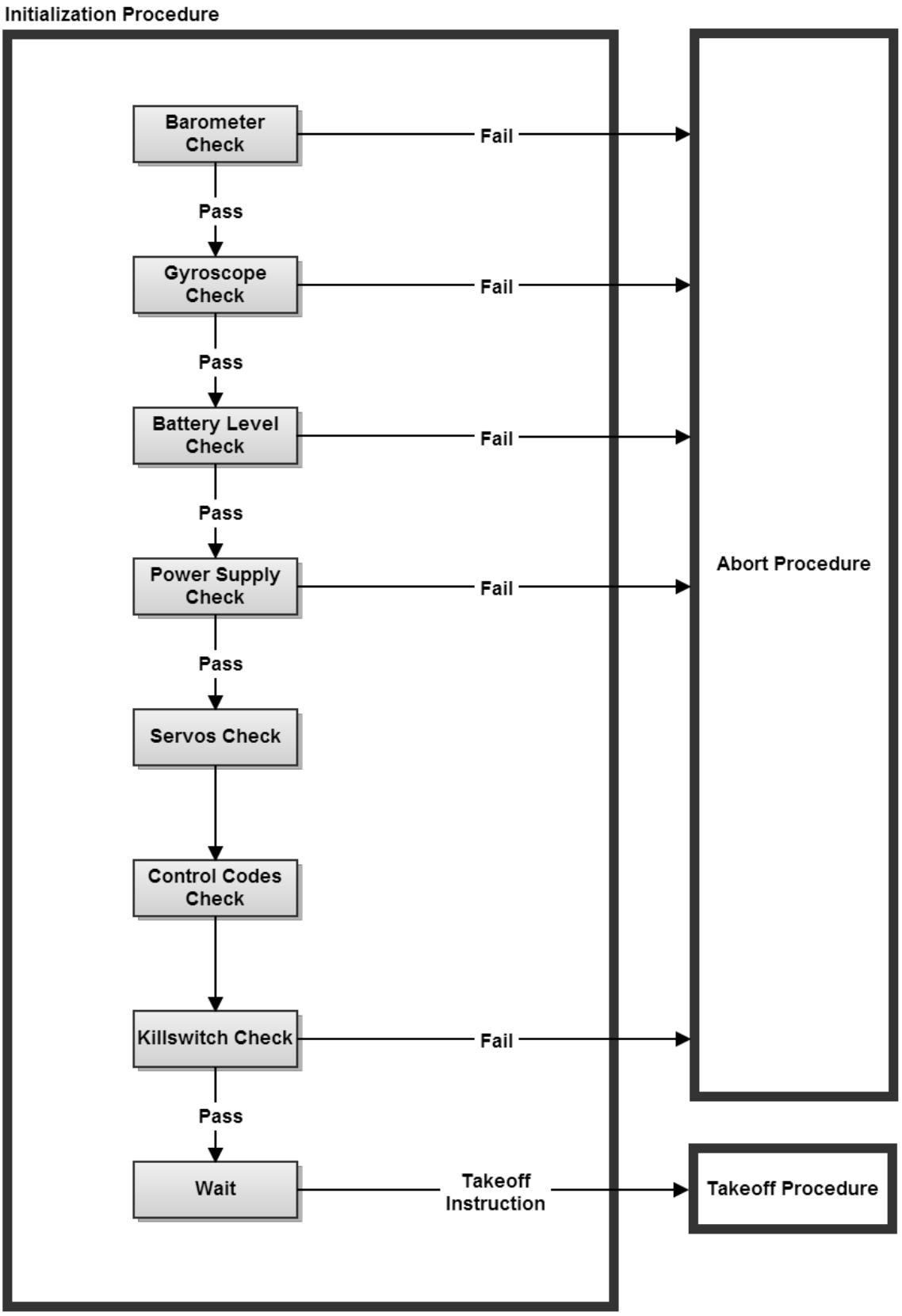

Fig. 6.5.1A: Flight Microcontroller Initialization Procedure.

The initialization procedure will be called upon the startup of the aircraft power systems. This initialization procedure will initialize the components under the control of the flight microcontroller, including the gyroscope, barometer, and servos. This procedure will also include the execution of all flight microcontroller power on self tests before the aircraft is cleared to be taken into take-off procedure.

In the initialization procedure the optical microcontroller will be waiting for an acknowledgement message from the flight microcontroller across the MCU to MCU bus. Once this message is received and acknowledgement is sent back to verify communication, the optical microcontroller moves to the next check of the initialization process. Therefore it will be required that the flight microcontroller send over an acknowledgement message in the initialization procedure before attempting to perform control code checks.

The first step in the initialization procedure will be a barometer check. This check will actually take multiple data readings from the barometer and compare them to statically set threshold values in the code to help ensure that the barometric readings received are accurate. If the barometric readings are found to be accurate, an initial altitude zero reading will be stored in a register for use later in the altitude check function. If the barometer check fails, the aircraft will be taken into the abort procedure, however if the barometer check passes, the initialization procedure will continue to the gyroscope check.

The second step in the initialization procedure will be a gyroscope check. This check will read values from the gyroscope and simple compare them to statically set threshold values in the code to help ensure that the gyroscope readings received are accurate. If the gyroscope check fails, the aircraft will be taken into the abort procedure, however if the gyroscope check passes, the initialization procedure will continue to the battery level check.

The third step in the initialization procedure will be a battery level check. This check will read the current battery levels of the system and compare them to statically set values to check whether the aircraft will have sufficient battery power to complete the required flight. If the battery level check fails, the aircraft will be taken into the abort procedure, however if the battery levels are found to be adequate, the initialization procedure will continue to the power supply check.

The fourth step in the initialization procedure will be a power supply check. This check will perform checks on all power supply lines available to the system. This

check will be used to help determine whether there could possibly be errors in hardware and peripherals of the aircraft caused by inaccurate power levels. If the power supply check fails, the aircraft will be taken into the abort procedure, however if the power supply check passes, the initialization procedure will continue to the servos check.

The fifth step in the initialization procedure will be a servos check. This check will simply move each of the servos through the appropriate pulse width ranges to ensure that the aircraft surfaces are moving correctly. Since there will be no data available directly to the flight microcontroller that the aircraft surfaces are indeed moving as required, a visual human inspection of these movements will be required during this step. If a visual inspection finds that the aircraft surface movements are not accurate, or not being made, the person can trigger the deadman switch to ensure that the aircraft will be taken into an abort procedure. If the aircraft surfaces are however moving correctly, the user will simply do nothing and the initialization procedure will continue to the control codes check.

The sixth step in the initialization procedure will be a control codes check. At this point the flight microcontroller will send a synchronization message over to the optical microcontroller to ensure their communication is now functioning correctly. The optical microcontroller will then begin transmitting multiple control codes for the flight microcontroller to handle. These incoming control codes will be handled as they come in. Since there will not be any data available to the flight microcontroller indicating that the aircraft surfaces are moving accurately for the control code sent over, a human visual inspection will again be required. If a visual inspection finds that the aircraft surface movements are not accurate to what the control code surface movements should be, the deadman switch will be activated to terminate procedures. If the aircraft surfaces are found to be adjusted correctly for the control code passed over, the person will simply do nothing and the initialization procedure will continue to the kill switch check.

The seventh step in the initialization procedure will be a kill switch check. This check will simply check whether or not the kill switch has been enabled. Since this check will happen after human visual inspections are required, any issues found by the person during those inspections that would have caused them to trip the kill switch will be caught here and the aircraft will be taken to the abort procedure as requested. If the kill switch signal is found to be 1, the aircraft will be taken into an abort procedure, however if the kill switch signal is found to be 0, the aircraft will continue to the wait step.

The last step in the initialization procedure will be a simple wait step. Since the flight microcontroller will have no guarantees that the optical system will be ready for takeoff when the flight microcontroller has finished POST's, the initialization procedure will simply wait for a specific takeoff control code to be passed from the optical microcontroller, signaling to the flight microcontroller that the optical system has found a target and takeoff procedure should commence.

Once the takeoff control code is passed into the flight microcontroller, the aircraft will be taken into the takeoff procedure.

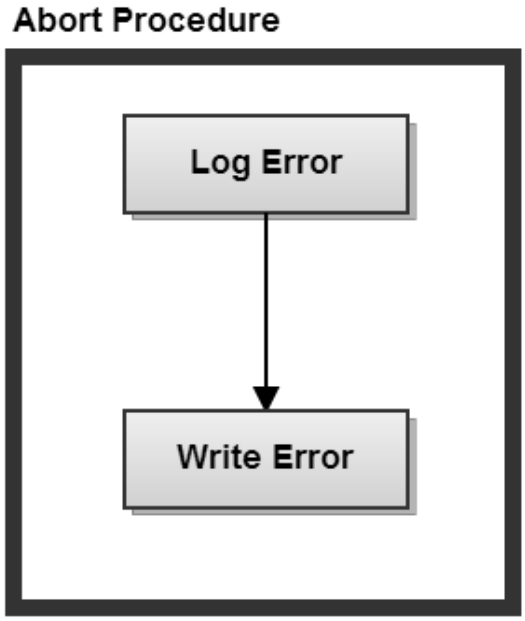

Fig. 6.5.1B: Flight Microcontroller Abort Procedure

The abort procedure will be a simple procedure in which if any initialization POST's had failed, the abort procedure will then log errors and output the error to the system LED's for debugging purposes. This procedure will be a last step procedure, so the aircraft will need to have a power reset performed and initialization procedure performed again before the aircraft may be eligible for takeoff.

The first step in the abort procedure is the log error step. This step will simply handle the logging of the error into an error log file stored in flash memory for use in debugging and testing purposes. Once the error logging has completed, the abort procedure will proceed to the write error step.

The last step in the abort procedure is the write error step. This step will handle the writing of the error out to flight system LED’s. These LED’s will be used to transmit visual representations of the error that was thrown. This will help the team in debugging and testing of the flight microcontroller system.

Since the abort procedure is a last step procedure, the aircraft will stay in this procedure until a power reset has been performed, at which point the aircraft would be taken back into an initialization procedure.

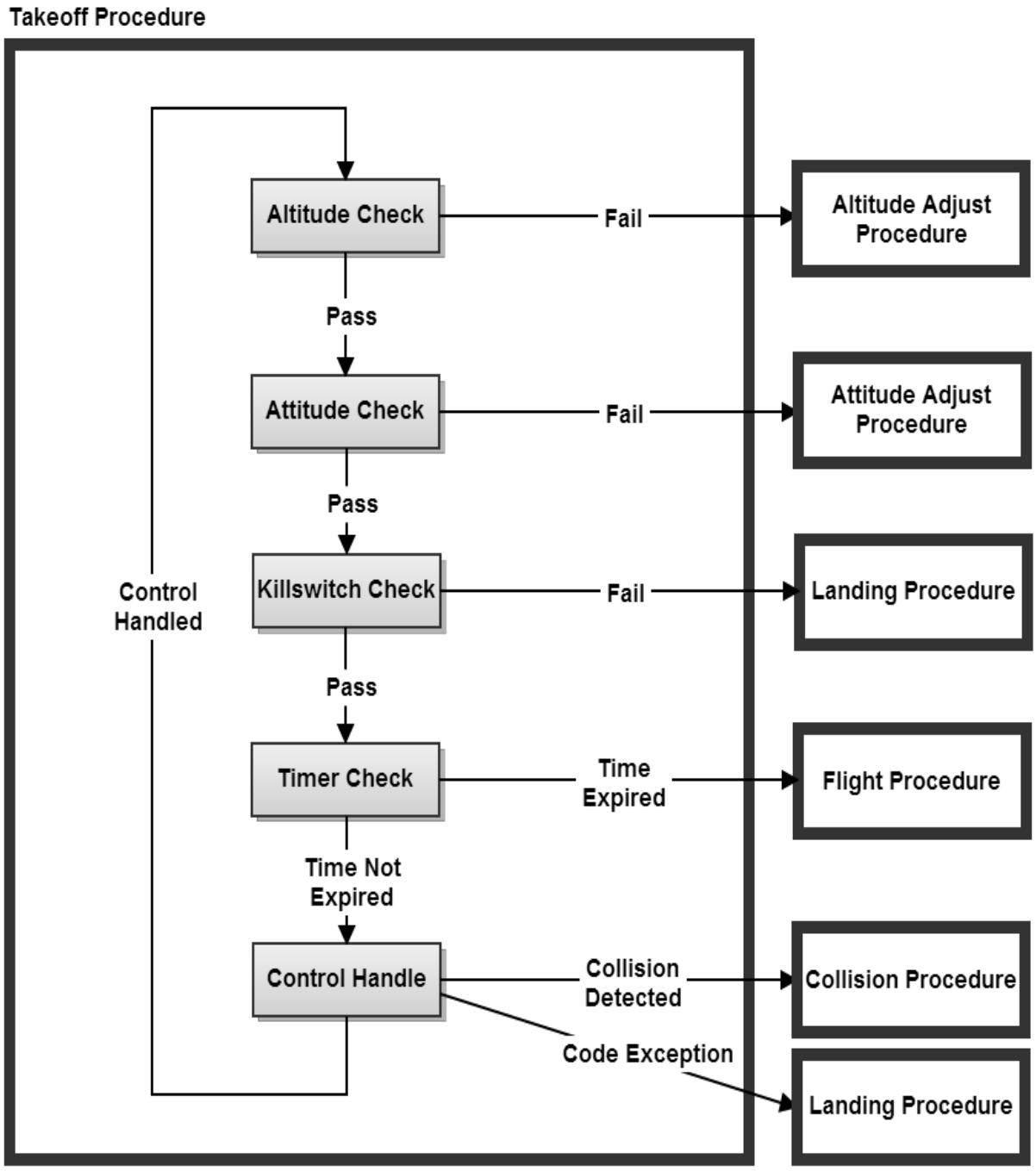

Fig. 6.5.1C: Flight Microcontroller Takeoff Procedure

Takeoff procedure will be active during the first few seconds of flight to ensure that the aircraft gains sufficient altitude and velocity before moving into flight procedure. The motor will be in full throttle in the takeoff procedure to allow the aircraft to gain velocity as quickly as possible. Steering advice in the takeoff procedure will still be taken from the optical microcontroller control code bus but

will only be acted upon at the lowest magnitude to prevent the aircraft from departing controlled flight. Flight control surfaces will be moved to lowest setting to prevent aircraft from becoming unstable. In addition, if the altitude drops too low during takeoff procedure, the aircraft altitude function will return to allow the takeoff procedure to take the aircraft into the altitude hazard procedure immediately and trigger an alert to the servo pulse function to immediately start increasing the altitude of the aircraft to avoid collision with the ground. If at any time during the takeoff procedure the aircraft is moved into a hazard procedure, that hazard procedure will complete and the aircraft will then be returned to the takeoff procedure and the procedure loop will continue as normal.

The first step of the takeoff procedure will be an altitude check. If the altitude of the aircraft is determined to be too high, or too low to the ground, the altitude function will return and the takeoff procedure will immediately take the aircraft into the hazard altitude procedure and call the servo pulse function to immediately adjust altitude of the aircraft.

The second step of the takeoff procedure will be an aircraft attitude check. If the attitude of the aircraft is determined to be outside the thresholds, the flight procedure will immediately take the aircraft into a hazard attitude procedure and call the servo pulse function to correct the attitude of the aircraft. Unlike the flight procedure, the takeoff procedure will not be acting fully on the control codes coming in from the optical microcontroller. The takeoff procedure will still consider the control codes, but will only minimally follow the attitude of the aircraft is determined to be within the thresholds, the takeoff procedure will proceed to the kill switch signal check step of the procedure loop.

The third step of the takeoff procedure will be the kill switch signal check. If the kill switch signal is equal to 1 during descent, indicating that the kill switch has been triggered, the aircraft will go into landing procedure as quickly as possible. If the kill switch signal is read in and is equal to 1, the takeoff procedure will push the aircraft into a kill procedure. If the kill switch signal is read in and is equal to 0, the takeoff procedure will continue to proceed through to the control handle step in the procedure loop.

The fourth step of the takeoff procedure will be the timer check. Since the takeoff procedure is only intended to last for the initial few seconds of flight, a timer needs to be implemented and referenced by the takeoff procedure so it knows when to take the aircraft into the flight procedure. If during the timer check, the procedure finds the takeoff time to still be too low, the takeoff procedure will then move on to the control handle step of the procedure loop, and will not perform a

timer check until the loop iterates around again. If during the timer check, the procedure finds the takeoff time to be greater than the statically set takeoff time limit, the takeoff procedure will simply take the aircraft into the flight procedure.

The final step of the takeoff procedure will be the control handle. This control code handler will differ from the flight procedure control code handler in that within the takeoff procedure it is not desirable to fully act upon the movement commands sent over by the optical microcontroller control code bus. The takeoff procedure will still be acting upon these movement commands, however they will only be minimally acted upon so as to allow hazard procedures, which are more common to be called during takeoff, to have priority in order to keep the aircraft stable and airborne. If a control code comes in from the optical microcontroller control code bus with movement commands for the aircraft, the takeoff procedure will still send this control code over to the control_servoPulse function, but will also send over a unique integer mode value of 1 to signify that the request is being made by the takeoff procedure. This mode value will be used by the control_servoPulse function in order to limit the movement corrections made while in the takeoff procedure.

Once the takeoff procedure has completed, the aircraft will be taken into the flight procedure.

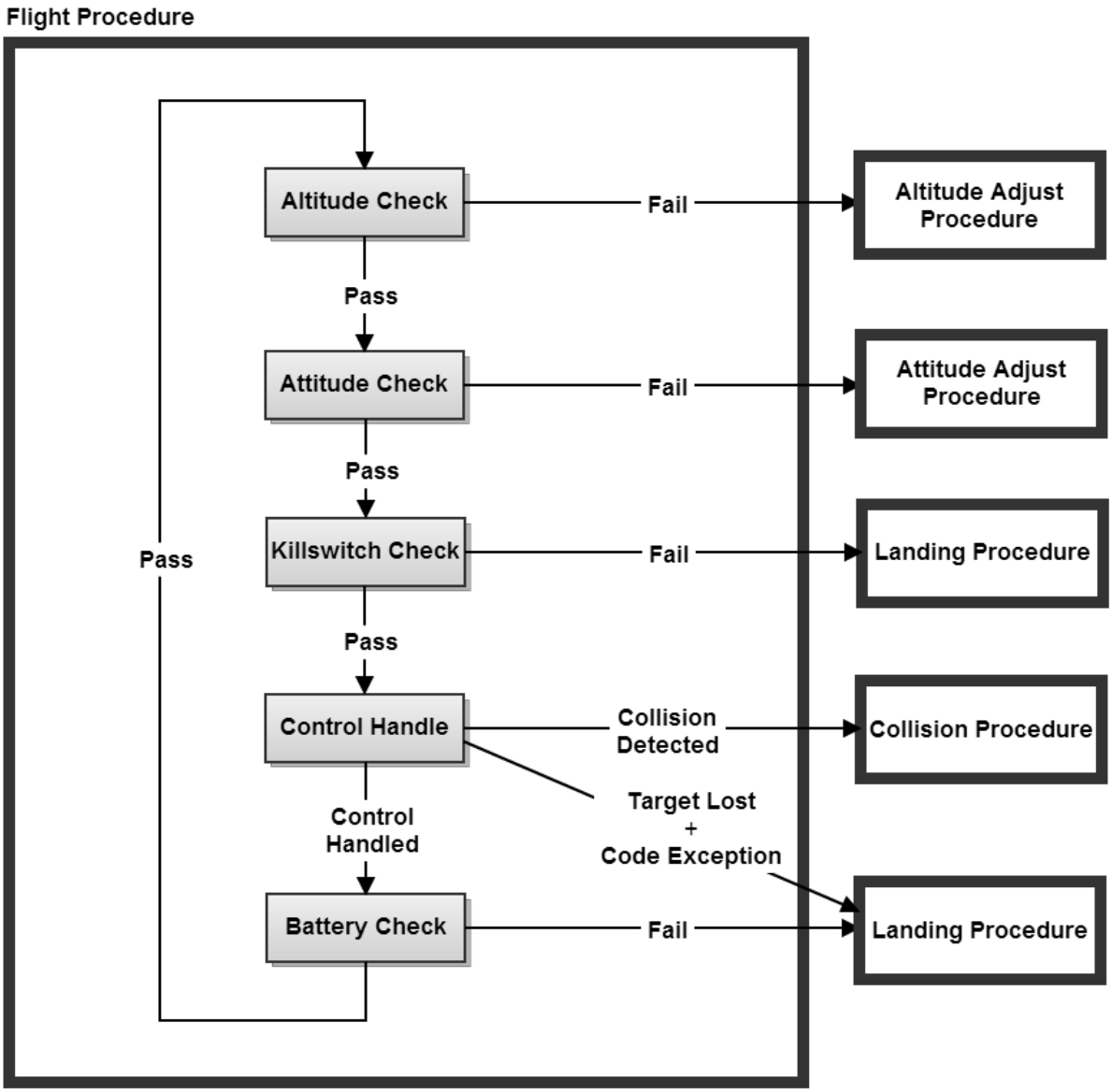

Fig. 6.5.1D Flight Microcontroller Flight Procedure

Flight procedure will be active during the remainder of the flight to the target destination, unless a safety or hazard alert is triggered to move the aircraft into that hazard procedure. Flight procedure will involve looping through safety/hazard check functions as well as reading in control codes from optical microcontroller when available.

The first step of the flight procedure will be an aircraft altitude check. If the altitude of the aircraft is determined to be too high, or too low to the ground, the altitude function will return and the flight procedure will immediately take the

aircraft into the hazard altitude procedure and call the servo pulse function to immediately adjust altitude of the aircraft.

The second step of the flight procedure will be an aircraft attitude check. If the attitude of the aircraft is determined to be outside the thresholds, the flight procedure will immediately take the aircraft into a hazard attitude procedure and call the servo pulse function to correct the attitude of the aircraft. If the attitude of the aircraft is determined to be within the thresholds, the flight procedure will proceed to the kill switch signal check step of the procedure loop.

The third step of the flight procedure will be the kill switch signal check. If the kill switch signal is equal to 1 which indicates that the kill switch has been triggered, the aircraft will need to execute landing procedure as quickly as possible especially during flight. If the kill switch signal is read in and is equal to 1, the flight procedure will push the aircraft into a kill procedure. If the kill switch signal is read in and is equal to 0, the flight procedure will continue to proceed through to the control handle step in the procedure loop.

The fourth step of the flight procedure will be the control handle. Whenever a control code comes in from the optical microcontroller, the servo pulse function will take that control code and alter the system servos as necessary. If the aircraft is currently in a hazard procedure, the servo pulse function will only write the control code to a buffer and wait to act upon the code until after the hazard procedure has completed. When no new control code exists, the servos will be kept in their current state and the aircraft's path will remain unchanged. Once the control code has been handled, the microcontroller will move on to system battery level checks.

The last step of the flight procedure will be the battery level check. If the battery level check function returns that any of the battery levels have reached the minimum threshold, the battery level check function will return to allow the flight procedure to take the aircraft into landing procedure immediately.

Once battery level checks have been completed and have been found to be above the minimum threshold, the flight microcontroller will start back through the flight procedure at the first step and continue to loop while in the flight procedure.

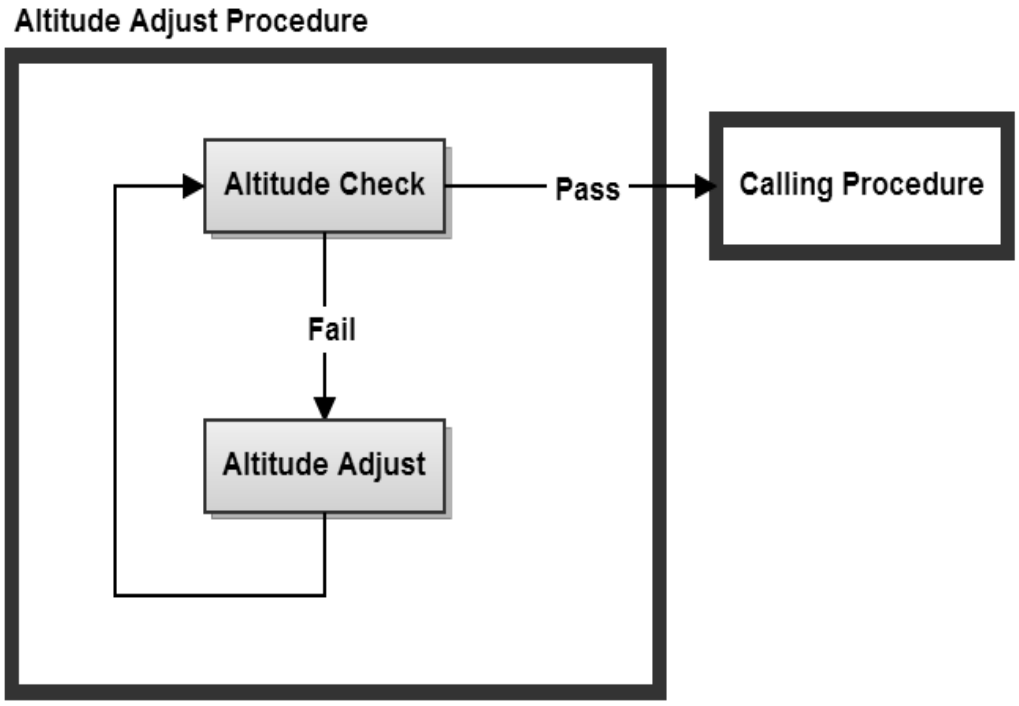

Fig. 6.5.1E Flight Microcontroller Altitude Adjust Procedure

The altitude adjust procedure will be called from either the takeoff or flight procedures when a low or high altitude reading has been confirmed by the calling procedure. This procedure is a hazard procedure, and due to the critical status of the aircraft in which this procedure would be called, the procedure will purely work to increase or decrease the altitude of the aircraft to bring it back to an acceptable altitude.

The first step of the altitude adjust procedure is the altitude check. This step will simply perform a current altitude check to get the correction data needed by the altitude adjust step. If the altitude check finds that the aircraft is indeed outside acceptable altitude ranges, the check will fail, and the altitude adjust procedure will move on to the altitude adjust step in order to correct the altitude of the aircraft. If the altitude check finds that the aircraft is however within acceptable altitude ranges, the check will pass and the altitude adjust procedure will break and return the aircraft back to the calling procedure.

The last step of the altitude adjust procedure is the altitude adjust step. This step will use data retrieved from the altitude check to correctly call the servo pulse function to actually start the leveling and altitude adjustments required. Once adjustments have been made, the altitude adjust function will simply loop back to the altitude check to ensure the aircraft is now within acceptable altitude ranges.

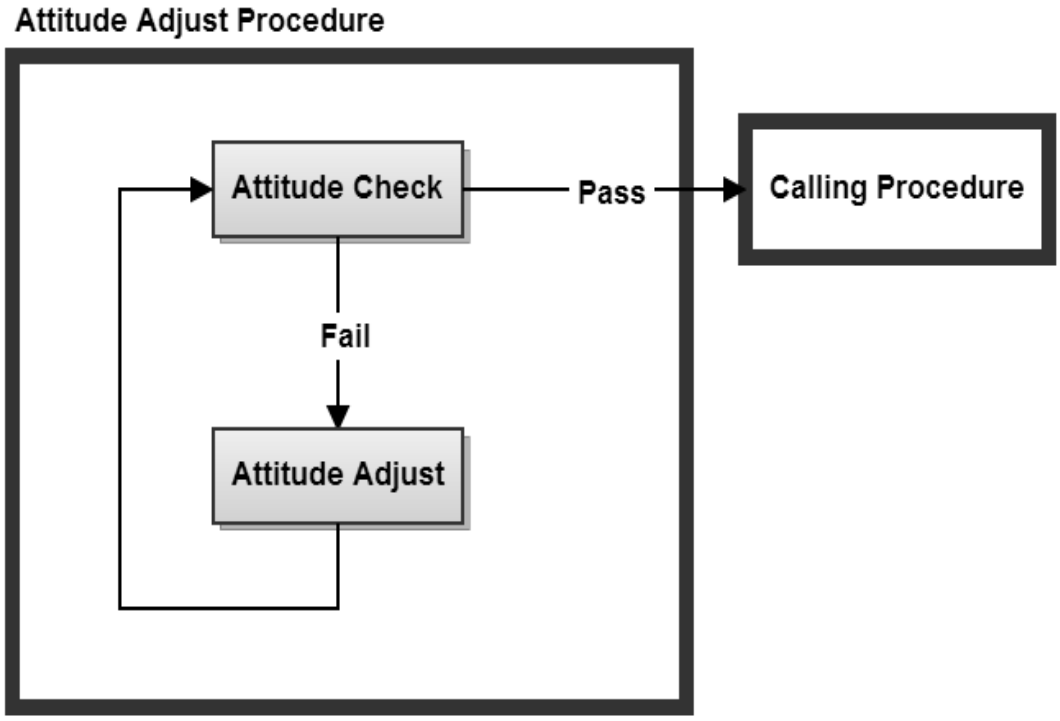

Fig. 6.5.1F Flight Microcontroller Attitude Adjust Procedure

The attitude adjust procedure will be called from either the takeoff or flight procedures when an unsafe attitude reading has been confirmed by the calling procedure. This procedure is a hazard procedure, and due to the critical status of the aircraft in which this procedure would be called, the procedure will immediately work to correct excessive roll, pitch, and yaw readings. Unlike the altitude adjust procedure however, the attitude adjust procedure will not be required to level the aircraft, instead the roll, pitch, and yaw states will each be brought back to acceptable values that should still quite accurately hold the current course of the aircraft.

The first step of the attitude adjust procedure is the attitude check. This step will simply perform a current attitude check to get the correction data needed by the attitude adjust step. If the attitude check finds that the aircraft is indeed outside acceptable attitude ranges, the check will fail, and the attitude adjust procedure will move on to the attitude adjust step in order to correct the attitude of the aircraft. If the attitude check finds that the aircraft is however within acceptable attitude ranges, the check will pass and the attitude adjust procedure will break and return the aircraft back to the calling procedure.

The last step of the attitude adjust procedure is the attitude adjust step. This step will use data retrieved from the attitude check to correctly call the servo pulse function and start bringing the roll, pitch, and yaw values back within acceptable ranges. Once adjustments have been made, the attitude adjust function will simply loop back to the attitude check to ensure the aircraft is now within acceptable attitude ranges.

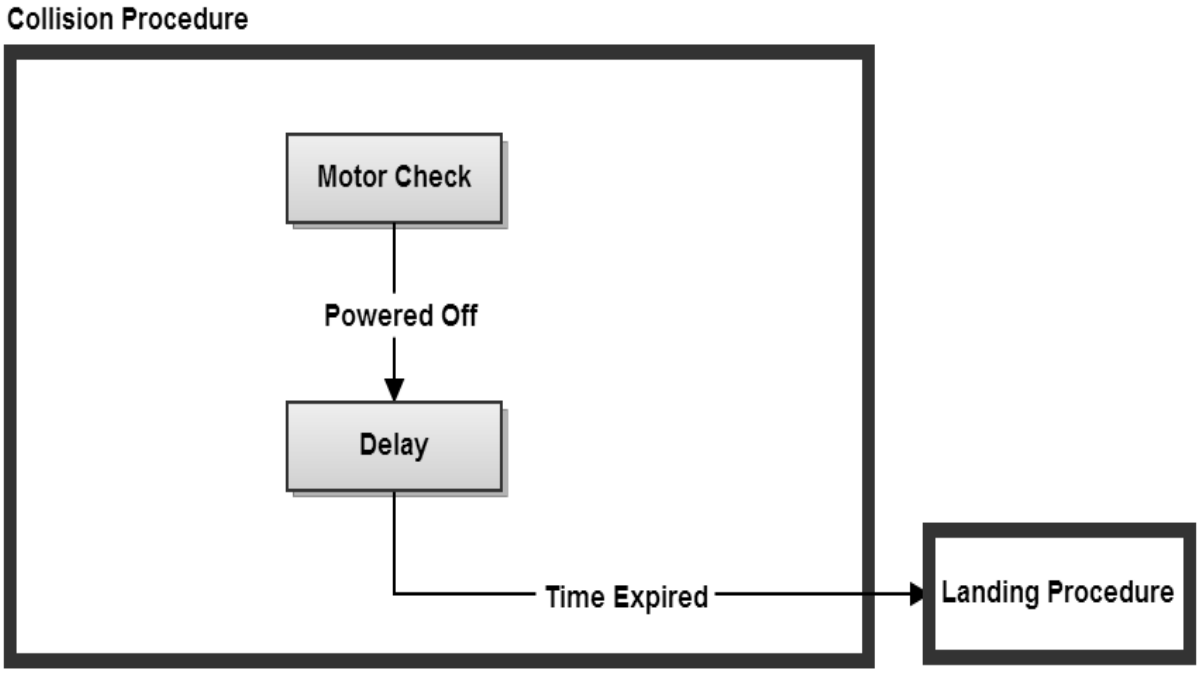

Fig. 6.5.1G Flight Microcontroller Collision Procedure

The collision procedure will be called when the optical microcontroller sends a control code over the control code bus indicating to the flight microcontroller that it has predicted a collision is imminent. The collision procedure will handle additional requirements needed in order to ensure that the aircraft is able to intercept the destination object before being taken into landing procedure. The aircraft will hold its current course for a specified amount of time in order for it to intercept the target. Immediately after interception has occurred, the collision procedure will move the aircraft into the landing procedure. If the aircraft were to simply go into a landing procedure without first completing the collision procedure, the aircraft would immediately start to be leveled out for landing, which would take it off its current course and would risk the aircraft not intercepting the target. Therefore a collision procedure is required to ensure the best possible chance of interception before continuing to level and land the aircraft.

The first step in the collision procedure loop will be a propulsion motor check. This check will simply involve killing the power to the propulsion motor. Since the aircraft will be on course for an interception with the destination, it will be critical to stop the motor before impact to ensure the least amount of damage to the aircraft. Once the motor has been powered off, the collision procedure will proceed on to the timer delay step.

The second step in the collision procedure loop will be a timer delay. This step will involve a simple delay of 2 seconds to allow the aircraft to stay on its current course to the target before the collision procedure sends the aircraft into the landing procedure. This delay is essential to ensure the greatest possible chance of intercepting the target before landing procedure levels out the flight of the aircraft. While the team believes that a 2 second delay will be adequate to allow for the aircraft to intercept the target, collision detection tests will need to be performed to better decide this delay value. Once the timer detects that the delay has surpassed the specified delay time, this will tell the collision procedure that it is now appropriate to take the aircraft into the landing procedure. At this point the aircraft should have intercepted the target and the aircraft should now be leveled out immediately so it can safely land.

Once the collision procedure has confirmation that the delay has completed successfully, the aircraft will be taken into the landing procedure.

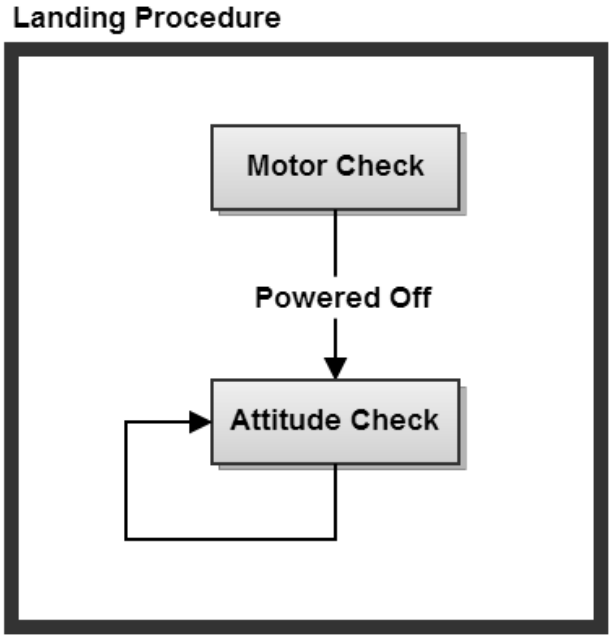

Fig. 6.5.1H Flight Microcontroller Landing Procedure

The landing procedure can be called due to a Safety/Hazard condition or after a collision condition. At this point the servo pulse function will be called to straighten the flight of the aircraft. Once flight is stabilized, the servo pulse function will tell the motor controller to cut power. Once power to the motor has been turned off, the aircraft will then begin a stabilized glide to the ground. A simple glide procedure loop will continue to run to adjust servos to attempt to stabilize flight while gliding to the ground, however adjustments to the servos will be limited since power to the motor has been turned off.

The first step in the landing procedure loop will be a propulsion motor check. This check will simply involve killing power to the motor. A call to the servo pulse function will be made with this instruction. Once power to the propulsion motor has been turned off, the landing procedure loop will proceed on to the attitude check.

The last step in the landing procedure loop will be an aircraft attitude check. While in the landing procedure, the destination attitude of the aircraft will be a stable descent to the ground. This means no control code movement commands are taken into account in the landing procedure. The attitude check function will still be used to pass in movement commands to the servo pulse function, however since power to the propulsion motor is off, the landing procedure will pass in a unique landing mode key value of 2, signifying that the system is in landing mode and that the servo pulse function should not act fully on the movement commands given to it. Since the propulsion motor is not on, if the system was to act fully on the movement commands, it is possible that aircraft attitude adjustments could occur too drastically, which could cause excess drag and possibly result in the crashing of the aircraft.

Since this is the landing procedure, altitude checks will not be necessary as in other procedures. No consideration of kill or battery level signals will be necessary either. This procedure will now proceed to simply loop through the aircraft attitude check function to attempt to keep the aircraft stabilized on its way to the ground.

The landing procedure is also a last step procedure, so the aircraft will stay in this procedure until a system power reset has been performed, at which point the aircraft would be taken back into the initialization procedure.

6.5.2 Flight Microcontroller Class Structure

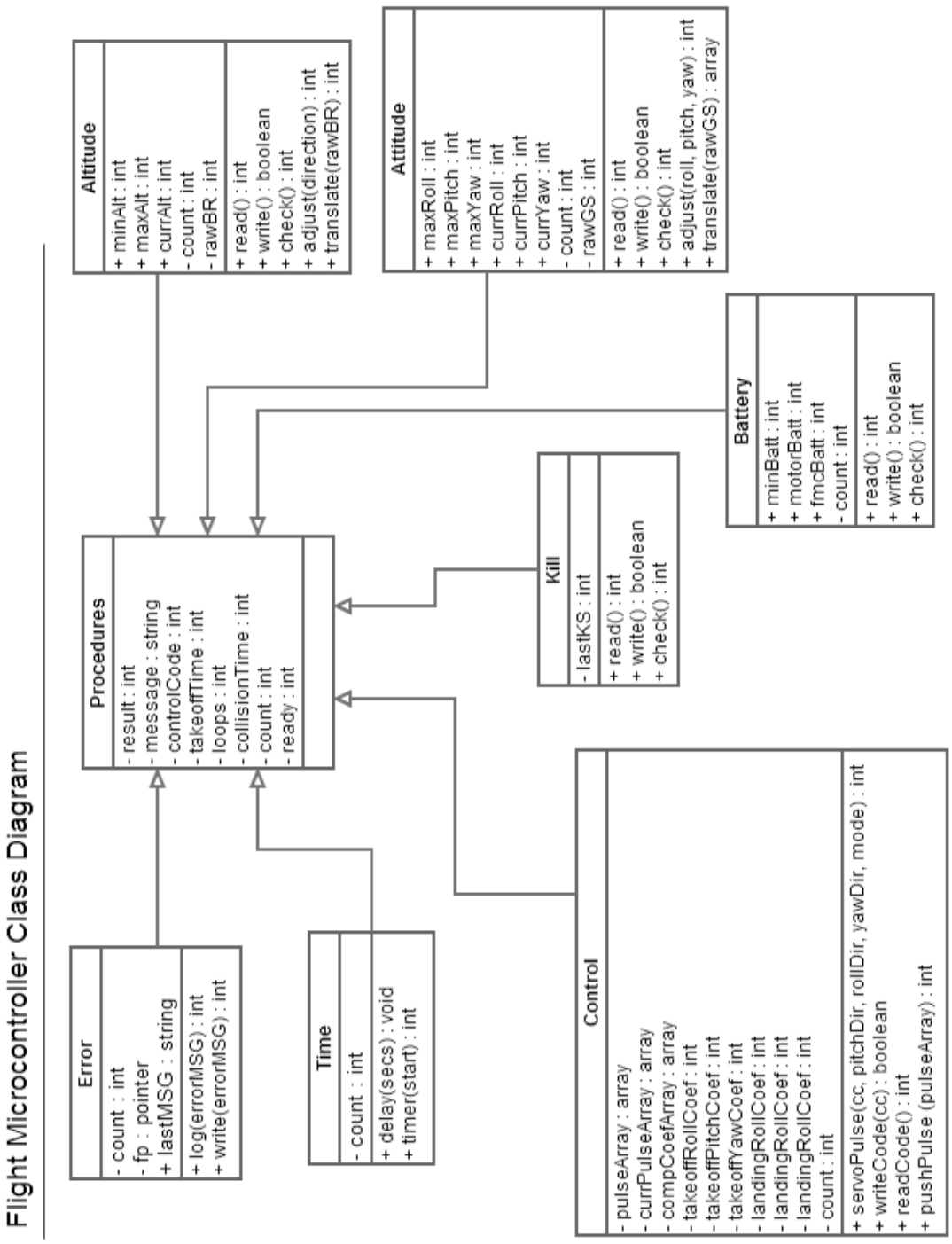

Fig. 6.5.2A: Flight Microcontroller Class Diagram

6.5.3 Altitude Class

The flight microcontroller altitude class consists of functions relating to the altitude of the aircraft as observed by the barometer. The functions allow control over and access to the data gathered by the barometer. Procedures will call

functions in this class in order to monitor the altitude of the aircraft to ensure that the aircraft is within acceptable altitude ranges. The barometric sensor reading could momentarily exhibit excessive inaccuracy due to transients. The main code the flight microcontroller executes will include the storing of many barometric sensor values and average them at a regular rate to help prevent inaccurate readings.

Altitude_write(altitude): The flight microcontroller altitude_write(altitude) function will take in an altitude reading as an input parameter. The function will then take this altitude value and store it in the altitude register. However if the altitude value passed into the function is equal to zero, this will signify to the function that it should initiate the capturing of multiple barometric readings. This barometer data retrieval process will then gather multiple readings from the barometer and calculate the median reading. The median reading will then be compared to rough altitude low and high limits statically set in the code to help ensure the median reading found is within reasonable ranges. If the median reading is found to be outside the acceptable ranges, the function will perform the barometer data retrieval process again. These steps will be done to attempt to prevent erroneous readings read from the barometer from being used in the assessment of the aircraft's current altitude. Once the function has successfully found an acceptable median barometer reading and has finished writing the altitude value to the register, the function will return a status Boolean indicating whether the altitude reading was stored successfully or not. A return Boolean with a value of true will indicate that the function successfully wrote the altitude value to the register, while a return Boolean with a value of false will indicate that the function did not successfully write the altitude value to the register.

read(): The flight microcontroller altitude_read() function will take no input parameters. The function will simply read the current altitude reading value that is stored in the altitude register. Once the function reads the value in from the register, it will then compare this value to the rough low and high altitude limits statically set in the code. If the value is found to be within acceptable ranges, this value will be returned from the function. If the value is not found to be within acceptable ranges, the function will return a value of -1 to signify that an error has occurred in attempting to read the altitude value from the register.

check(): The flight microcontroller altitude_check() function will take no input parameters. This function will define the actual altitude check functionality used by the takeoff and flight procedures during the altitude check step of their procedure loops. The function will start by making a call to the altitude_write function with an input parameter of zero, signifying that the altitude_check

function wants it to initiate barometric readings and store the median reading into the altitude register. The return value of the altitude_write function will be written to a temporary Boolean variable within the altitude_check function. This temporary Boolean variable will then be used as a check before the function proceeds to call the altitude_read function. If the temporary Boolean variable is equal to false, the altitude_check function will not proceed with the altitude_read function call, but instead will enter a critical attempt loop where the function will simply make additional calls to the altitude_write function in an attempt to get a successful barometric reading written to the altitude register. If after a certain number of attempts, which will be statically set within the code file, the function is unable to get a successful response back from the altitude_write function, the altitude_check function will make a call to the error_log function declaring the error and then immediately break and return -1 back to the procedure so that the procedure can immediately initiate the landing procedure. If the temporary Boolean variable is equal to true, the altitude_check function will proceed next with an altitude_read function call. The purpose of this call will simply be to retrieve the altitude value stored in the altitude register. The return value of the altitude_read function will be written to a temporary integer variable within the altitude_check function. This temporary integer variable will then be used as a check before the function proceeds to threshold comparison. If the temporary variable is equal to -1, the altitude_check function will not proceed with threshold comparisons, but instead will enter a critical attempt loop where the function will simply make additional calls to the altitude_read function in an attempt to get a successful barometric reading read in from the altitude register. If after a certain number of attempts, which will be statically set within the code file, the function is unable to get a successful response back from the altitude_read function, the altitude_check function will make a call to the error_log function declaring the error and then immediately break and return -1 back to the procedure so that the procedure can immediately initiate the landing procedure. If the temporary Boolean variable is not equal to -1, the altitude_check function will proceed next with threshold comparisons. In this step, the barometric reading read in from the altitude_read function will be compared to initial barometric thresholds that were set during the initialization procedure. If the barometric reading is not within the thresholds set during initialization, the altitude_check function will return an outside threshold specific warning value. If the return value is 2, this indicates that the barometric reading is below the low threshold. If the return value is 3, this indicates that the barometric reading is above the high threshold. If the return value is 1, this indicates that the barometric reading is within the thresholds, and therefore the aircraft is at an acceptable altitude.

adjust(direction): The flight microcontroller altitude_adjust(direction) function will take in a direction value as an input parameter. The function will be responsible for sending the appropriate signals to various functions to appropriately adjust the altitude of the aircraft during an altitude adjustment hazard procedure. The function will start by comparing the direction value input to the two possible IF conditions. If the direction input is 2, the function will proceed to call the control_servoPulse function with a control code input parameter of 40, indicating a low altitude adjustment needed. If the direction input is 3, the function will proceed to call the control_servoPulse function with a control code input parameter of 44, indicating a high altitude adjustment needed. A temporary integer variable will be used to store the return value of the control_servoPulse function. If the return value from the control_servoPulse function is -1, the altitude_adjust function will proceed into a critical attempt loop in which the function will again attempt to call the control_servoPulse function. If after a specific number of attempts, which will be statically set within the code file, the function is unable to get a successful response back from the control_servoPulse function, the altitude_adjust function will make a call to the error_log function declaring the error and then immediately break and return -1 back to the calling procedure to allow the procedure to immediately initiate the landing procedure. If the return value from the control_servoPulse function is 1, the altitude_adjust function will simply return 1 back to the calling procedure.

translate(attitude): The flight microcontroller altitude_translate(altitude) function will take in a raw barometric altitude value as an input parameter. The function will simply be used to translate the raw barometric altitude values retrieved into more easily understood altitude units for testing/debugging and possibly occasional comparator operations.

6.5.4 Attitude Class

The flight microcontroller attitude class consists of functions relating to the attitude of the aircraft as observed by the gyroscope. The functions allow control over and access to the data gathered by the gyroscope. Procedures will make calls to the functions of this class in order to monitor the current attitude of the aircraft. These attitude checks are common in various procedures and it is critical for the safety of the aircraft that these functions be available to these procedures. Malfunctions of the gyroscope could cause the flight microcontroller to maneuver the aircraft in a manner that causes structural failure or departure from stable and controlled flight. The system will not be sophisticated enough to detect the difference between most executed flight maneuvers and a gyroscope failure. If during the power on self tests in the initialization procedure the gyroscope reports

multiple rotations in any axis, or the pitch or yaw axis report a roll rate of more than 180 degrees per second, the aircraft will fail POST's and be taken into the abort procedure.

write(roll, pitch, yaw): The flight microcontroller attitude_write(roll, pitch, yaw) function will take in a roll, pitch, and yaw value as input parameters. The function will then take these values and store them in their respective registers. However if the function is called with roll, pitch, and yaw values all equal to zero, this will signify to the function that it should initiate a gyroscope data retrieval process. This gyroscope data retrieval process will then gather multiple readings from the gyroscope and calculate the median reading. The median reading will then be compared to rough attitude thresholds statically set in the code to help ensure the median reading found is within reasonable ranges. If the median reading is found to be outside the acceptable ranges, the function will perform the gyroscope data retrieval process again. These steps will be done to attempt to prevent erroneous readings read from the gyroscope from being used in the assessment of the aircraft's current attitude. Once the function has successfully found an acceptable median gyroscope reading and has finished writing the attitude values to the registers, the function will return a status Boolean indicating whether the attitude readings were stored successfully or not. A return Boolean with a value of true will indicate that the function successfully wrote the attitude values to the registers, while a return Boolean with a value of false will indicate that the function did not successfully write the altitude values to the registers.

read(): The flight microcontroller attitude_read() function will take no input parameters. The function will simply read the current attitude values that are stored in the attitude registers. Once the function reads the values in from the registers, it will then compare these values to the rough attitude limits statically set in the code. If the values are found to be within acceptable ranges, these values will be returned from the function in the form of an array. If the values are not found to be within acceptable ranges, the function will return a value of -1 to signify that an error has occurred in attempting to read the attitude values from the registers.

check(): The flight microcontroller attitude_check() function will take no input parameters. This function will define the actual attitude check functionality used by the takeoff and flight procedures during the attitude check step of their procedure loops. The function will start by making a call to the attitude_write function with input parameters of zero, signifying that the attitude_check function wants it to initiate gyroscope readings and store the median reading into the attitude registers. The return value of the attitude_write function will be written to

a temporary Boolean variable within the altitude_check function. This temporary Boolean variable will then be used as a check before the function proceeds to call the attitude_read function. If the temporary Boolean variable is equal to false, the attitude_check function will not proceed with the attitude_read function call, but instead will enter a critical attempt loop where the function will simply make additional calls to the attitude_write function in an attempt to get a successful gyroscopic reading written to the attitude registers. If after a certain number of attempts, which will be statically set within the code file, the function is unable to get a successful response back from the attitude_write function, the attitude_check function will make a call to the error_log function declaring the error and then immediately break and return -1 back to the procedure so that the procedure can immediately initiate the landing procedure. If the temporary Boolean variable is equal to true, the attitude_check function will proceed next with an attitude_read function call. The purpose of this call will simply be to retrieve the attitude values stored in the attitude registers. The return value of the attitude_read function will be written to a temporary integer variable within the attitude_check function. This temporary integer variable will then be used as a check before the function proceeds to threshold comparison. If the temporary variable is equal to -1, the attitude_check function will not proceed with threshold comparisons, but instead will enter a critical attempt loop where the function will simply make additional calls to the attitude_read function in an attempt to get a successful gyroscopic reading read in from the attitude registers. If after a certain number of attempts, which will be statically set within the code file, the function is unable to get a successful response back from the attitude_read function, the attitude_check function will make a call to the error_log function declaring the error and then immediately break and return -1 back to the procedure so that the procedure can immediately initiate the landing procedure. If the temporary integer variable is not equal to -1, the attitude_check function will proceed next with threshold comparisons. In this step, the gyroscopic reading read in from the attitude_read function will be compared to statically set gyroscopic thresholds that were set during system testing. If the gyroscopic reading is not within the thresholds set, the attitude_check function will return outside threshold specific warning values for any parameter outside the threshold. If the roll return value is 2, this indicates that the gyroscopic roll reading is over the left roll threshold. If the roll return value is 3, this indicates that the gyroscopic roll reading is over the right roll threshold. If the pitch return value is 2, this indicates that the gyroscopic pitch reading is over the pitch down threshold. If the pitch return value is 3, this indicates that the gyroscopic pitch reading is over the pitch up threshold. If the yaw return value is 2, this indicates that the gyroscopic yaw reading is over the left yaw threshold. If the yaw return value is 3, this indicates that the gyroscopic

yaw reading is over the right yaw threshold. If the return value is 1 for any of the gyroscopic reading values, this indicates that that particular gyroscopic reading value is within the thresholds, and therefore an acceptable value.

adjust(roll direction, pitch direction, yaw direction): The flight microcontroller attitude_adjust(roll direction, pitch direction, yaw direction) function will take in direction values as an input parameters. The function will be responsible for sending the appropriate signal to the control_servoPulse function to appropriately adjust the attitude of the aircraft during an attitude adjustment hazard procedure. The function will call the control_servoPulse function with a control code of 48, which signals to the function that an attitude adjustment is required. Along with the control code of 48, the roll direction, pitch direction, and yaw direction values will be passed to the control_servoPulse function. A temporary integer variable will be used to store the return value of the control_servoPulse function. If the return value from the control_servoPulse function is -1, the attitude_adjust function will proceed into a critical attempt loop in which the function will again attempt to call the control_servoPulse function. If after a specific number of attempts, which will be statically set within the code file, the function is unable to get a successful response back from the control_servoPulse function, the attitude_adjust function will make a call to the error_log function declaring the error and then immediately break and return -1 back to the calling procedure to allow the procedure to immediately initiate the landing procedure. If the return value from the control_servoPulse function is 1, the attitude_adjust function will simply return 1 back to the calling procedure.

translate(attitude): The flight microcontroller attitude_translate(attitude) function will take in a raw gyroscopic attitude value as an input parameter. The function will be used to translate the raw gyroscopic attitude values retrieved into separate roll, pitch, and yaw values. This operation will be critical for most comparator operations as well as for testing/debugging in the attitude class functions.

6.5.5 Battery Class

The flight microcontroller battery class will allow battery level readings and checks to be made. Procedures will call functions in this class in order to monitor current battery levels in the system. These checks can be used to ensure that the aircraft has enough battery life before initiating flight. The flight microcontroller will pulse battery on an analog input pin and use an analog to digital converter to read the battery level. If for any reason the main battery system fails to provide an adequate voltage level above the set threshold value, the aircraft will be

moved into a landing state which will shut down any unnecessary components such as the propulsion motor. The procedure it will fault into is the landing procedure which in this procedure the system is completely independent of the optical microcontroller systems. The battery level check is necessary to protect the integrity of the design. If unstable voltage levels are unmonitored the aircraft can go into a hazard condition and potentially risk the safety of the aircraft.

write(batteryLevel): The flight microcontroller battery_write(batteryLevel) function will take in a batteryLevel value as an input parameter. The function will then take this value and store it in a register. However if the function is called with a batteryLevel value equal to zero, this will signify to the function that it should initiate a battery level retrieval process. This battery level retrieval process will then gather readings from both the microcontroller and motor batteries and compare them to rough battery level ranges statically set in the code to help prevent an erroneous battery level reading from being used in the assessment of the aircraft's battery systems status. Once the function has successfully identified the battery levels are within acceptable ranges, the battery levels are written to the registers. Once the function has successfully written the battery levels to the registers, the function will return a status Boolean indicating whether the battery level readings were stored successfully or not. A return Boolean with a value of true will indicate that the function successfully wrote the battery level values to the registers, while a return Boolean with a value of false will indicate that the function did not successfully write the battery level values to the registers.

read(): The flight microcontroller battery_read() function will take no input parameters. The function will simply read the current battery level values that are stored in the battery level registers. Once the function reads the values in from the registers, it will then compare these values to the rough battery level ranges statically set in the code. If the values are found to be within the acceptable range, these values will be returned from the function in the form of an array. If the values are not found to be within acceptable ranges, the function will return a value of -1 to signify that an error has occurred in attempting to read the battery level values from the registers.

check(): The flight microcontroller battery_check() function will take no input parameters. This function will define the actual battery level check functionality used by the takeoff and flight procedures during the battery level check step of their procedure loops. The function will start by making a call to the battery_write function with an input parameter of zero, signifying that the battery_check function wants it to initiate battery level readings and store the readings into the battery level registers. The return value of the battery_write function will be

written to a temporary Boolean variable within the battery_check function. This temporary Boolean variable will then be used as a check before the function proceeds to call the battery_read function. If the temporary Boolean variable is equal to false, the battery_check function will not proceed with the battery_read function call, but instead will enter a critical attempt loop where the function will simply make additional calls to the battery_write function in an attempt to get successful battery level readings written to the battery level registers. If after a certain number of attempts, which will be statically set within the code file, the function is unable to get a successful response back from the battery_write function, the battery_check function will make a call to the error_log function declaring the error and then immediately break and return -1 back to the procedure so that the procedure can immediately initiate the landing procedure. If the temporary Boolean variable is equal to true, the battery_check function will proceed next with a battery_read function call. The purpose of this call will simply be to retrieve the battery level values stored in the battery level registers. The return value of the battery_read function will be written to a temporary integer variable within the battery_check function. This temporary integer variable will then be used as a check before the function proceeds to threshold comparison. If the temporary variable is equal to -1, the battery_check function will not proceed with threshold comparisons, but instead will enter a critical attempt loop where the function will simply make additional calls to the battery_read function in an attempt to get successful battery level readings read in from the battery level registers. If after a certain number of attempts, which will be statically set within the code file, the function is unable to get a successful response back from the battery_read function, the battery_check function will make a call to the error_log function declaring the error and then immediately break and return -1 back to the procedure so that the procedure can immediately initiate the landing procedure. If the temporary integer variable is not equal to -1, the battery_check function will proceed next with threshold comparisons. In this step, the battery level readings read in from the battery_read function will be compared to a minimum battery level threshold that was set during the system tests. If the battery level reading is under the minimum threshold set during system testing, the battery_check function will return an under minimum threshold specific warning value. If the return value is 2, this indicates that the battery level reading is below the minimum threshold. If the return value is 1, this indicates that the battery level reading is above the minimum threshold, and therefore the battery systems on the aircraft are at an acceptable level.

6.5.6 Kill Class

The flight procedure kill class will contain functions to read and write the kill switch signal, as well as check the current kill switch signal. These kill class functions will be used by various procedures mainly when a kill switch check is needed. In these instances the procedure would simply be looking to see if the kill switch has physically been enabled, and therefore the kill switch signal is equal to 1. Aside from this main purpose, these class functions will be useful in testing of the kill switch signal, as the team will be able to use the kill_write function to actually write in the kill switch signal which can be directly tested against.

write(killSignal): The flight microcontroller kill_write(killSignal) function will take in a killSignal value as an input parameter. The function will then take this value and store it in a register. However if the function is called with a killSignal value equal to zero, this will signify to the function that it should initiate a kill signal retrieval process. This kill signal retrieval process will then read the kill signal coming into the flight microcontroller from the kill switch data line and store this value in a register. This function will not return any values out of the function.

read(): The flight microcontroller kill_read() function will take no input parameters. The function will simply read the current kill signal value that is stored in the kill signal register. Once the function reads the value in from the register, it will compare this value to ensure it is either 1 or 0. If the value read in from the register is neither 1 nor 0, the function will return -1, signifying that the kill signal read was not a correct signal. If the value read in from the register is either 1 or 0, the function will return that value, signifying that the kill signal read was a correct signal.

check(): The flight microcontroller kill_check() function will take no input parameters. This function will define the actual kill signal check functionality used by the takeoff and flight procedures during the kill signal check step of their procedure loops. The function will start by making a call to the kill_write function with an input parameter of zero, signifying that the kill_check function wants it to initiate a kill signal reading and store the reading into the kill signal register. Once the kill_write function has completed, the kill_check function will proceed next to a kill_read function call. The purpose of this call will simply be to retrieve the kill signal value stored in the kill signal register. The return value of the kill_read function will be written to a temporary integer variable within the kill_check function. This temporary integer variable will then be used as a check before the function proceeds to signal analysis. If the temporary variable is equal to -1, the kill_check function will not proceed with signal analysis, but instead will enter a critical attempt loop where the function will simply make additional calls to the

kill_read function in an attempt to get a successful kill signal reading read in from the kill signal register. If after a certain number of attempts, which will be statically set within the code file, the function is unable to get a successful response back from the kill_read function, the kill_check function will make a call to the error_log function declaring the error and then immediately break and return -1 back to the procedure so that the procedure can immediately initiate the landing procedure. If the temporary integer variable is not equal to -1, the kill_check function will proceed next with signal analysis. In this step, the kill signal reading read in from the kill_read function will be compared to 1. If the kill signal reading is equal to 1, the kill_check function will return kill switch enabled warning value. If the return value is 2, this indicates that the kill signal reading is equal to 1. If the return value is 1, this indicates that the kill signal reading is equal to 0, and therefore the kill switch has not been enabled and the aircraft is safe to resume takeoff or flight procedure.

6.5.7 Control Class

The flight microcontroller control class contains functions to physically manipulate the servos of the aircraft, as well as handling of control codes upon arrival from the optical microcontroller control code bus. Many procedures will call functions from this class to update the movement patterns of the aircraft, adjust the motor throttle, as well as simply read in a control code value that was stored in a register upon arrival. Aside from these purposes, the structure of these functions in the class will allow for very low level testing which will be quite an extensive process.

servoPulse(controlCode, roll, pitch, yaw, mode): The flight microcontroller control_servoPulse(controlCode, roll, pitch, yaw) function will take in a controlCode value, as well as optional roll, pitch, and yaw direction values as input parameters. This function will be responsible for all movement and throttle corrections needing made to the aircraft. These changes may be requested by various procedure functions, including hazard procedure functions, so unique control codes are assigned to the functions to allow easy calls to the control_servoPulse function when specific movement corrections are needed. The function will first compare the control code it received as input against the different control code cases programmed into the function. If a match is found, the function will then proceed into this case and begin to calculate the specific servo pulse widths needed to make this movement correction. Many cases in the function will have statically set gyroscopic data that tells the function at which gyroscopic values the aircraft needs to be for that particular case to succeed. These statically set gyroscopic values will be set during system tests. The

function would then need to make a call to the attitude_write and attitude_read functions to pull in the most recent gyroscopic data. This data will both be used to calculate the difference between the destination and current attitude, as well as for calculating whether a movement correction request will put the aircraft past any attitude thresholds, in which case the function will need to limit the amount of correction in that particular direction. If either of these calls to the attitude class return -1 or false, the respective call will then be brought into a critical attempt loop where the function will simply make additional calls to the respective attitude class function in an attempt to get a successful gyroscopic data reading. If after a certain number of attempts, which will be statically set within the code file, the function is unable to get a successful response back from either attitude class function, the control_servoPulse function will make a call to the error_log function declaring the error and then immediately break and return -1 back to the procedure or function so that a landing procedure can be initiated as quickly as possible. Once the gyroscopic data has been read into the function, it will then begin to dynamically calculate the difference between the current gyroscopic reading and the statically set destination reading. This difference will then be used to calculate the new servo pulse widths to be passed into the control_pushPulse function. The control_servoPulse function will however have to consider maximum change thresholds in servo pulse widths to avoid overly drastic movement corrections in a short amount of time. If it is found that the change required is over the maximum change threshold, a loop will be initiated to incrementally make the needed movement corrections over a longer time so as to avoid possibly putting the aircraft into a hazard condition. Finally, the control_servoPulse function will need to pass the new servo pulse width values over to the actual servos, and to do this, it will send these values into the control_pushPulse function. There are certain cases in the control_servoPulse function in which additional roll, pitch, and yaw direction parameters passed into the function are needed to accurately calculate these servo pulse width values. However once these inputs have been used to calculate the servo pulse width values, the function will simply then proceed as it otherwise would and pass these servo pulse width values into the control_pushPulse function in order to update the aircraft's surfaces. The control codes received from the optical microcontroller control code bus will include standard movement correction control codes. The control_servoPulse function will need to store movement coefficients in the function to account for different movement patterns and requirements as observed in system testing. The control_servoPulse function will return one of two possible values. If the control_servoPulse function returns -1, this indicates that the function was unsuccessful in calculating and pushing the new servo pulse width values over to the control_pushPulse function. If the

control_servoPulse function returns 1, this indicates that the function was successful in calculating and pushing the new servo pulse width values over to the control_pushPulse function.

The control_servoPulse function will have cases for many different unique control codes that will each be responsible for a different aircraft movement change. These control codes are listed below in Table 6.5.7A. The first 32 control codes, 0-31 are reserved for control codes coming over from the optical microcontroller on the 5-bit control code bus. These control codes represent standard movement correction requests made by the optical microcontroller, as well as any additional optical microcontroller requests needing to be added.

Control Code	Called By	Description
0	Any	Shutdown
1	Flight Procedure	Up 1
2	Flight Procedure	Right + Up 1
3	Flight Procedure	Right 1
4	Flight Procedure	Right + Down 1
5	Flight Procedure	Down 1
6	Flight Procedure	Left + Down 1
7	Flight Procedure	Left 1
8	Flight Procedure	Left + Up 1
9	Flight Procedure	Up 2
10	Flight Procedure	Right + Up 2
11	Flight Procedure	Right 2
12	Flight Procedure	Right + Down 2
13	Flight Procedure	Down 2
14	Flight Procedure	Left + Down 2
15	Flight Procedure	Left 2
16	Flight Procedure	Left + Up 2
17	Initialization Procedure	Takeoff Ready
18-30	Flight Procedure	Optical MC Reserved

31	Flight Procedure	No Correction
40	altitude_adjust Function	Low Altitude
44	altitude_adjust Function	High Altitude
48	attitude_adjust Function	Attitude Adjustment

Table 6.5.7A: *A list of flight microcontroller control codes which are called by procedures and functions in code. Includes identification of calling procedure or function, as well as a description of the function of each control code upon being handled by the control_servoPulse function.*

pushPulse(pulse1, pulse2, pulse3, pulse4, pulse5): The flight microcontroller control_pushPulse(pulse1, pulse2, pulse3, pulse4, pulse5) function will take in servo pulse width values as input parameters. This function will simply serve as a setter mechanism for the actual servo pulse widths. The control_servoPulse function will be entirely in charge of ensuring that the pulse width values passed over to the control_pushPulse function are acceptable changes to the current pulse width values being used. This function will not contain complex logical checks or comparators that check these values, therefore this function will only be called from the control_servoPulse function that can ensure that the values being passed over are acceptable change values, since the logical checks and comparators have been performed in that function already. The control_pushPulse function will then simply proceed to adjust each servo pulse width to the new respective pulse width passed into the function. The function will return one of two possible values. If the function returns -1, this indicates that there was a problem in updating the actual servo pulse widths. If the function returns 1, this indicates that the servo pulse width updates were made successfully.

writeCode(controlCode): The flight microcontroller control_writeCode(controlCode) function will take in a control code as an input parameter. This function will simply serve to set the incoming control code from the optical microcontroller control code bus to the appropriate register. This function will be triggered by an interrupt that can occur in any procedure, and will occur whenever a control code comes over from the optical microcontroller control code bus. This simple function will be used to ensure the smallest interrupt time to ensure that the procedure that was interrupted can continue safety and hazard checks as quickly as possible. Since it is possible that this interrupt could happen during a hazard procedure, it is critical that the system does not delay the functioning of that procedure too long or the safety of the aircraft is risked. This function will return one of two possible values. If the

function returns -1, this indicates that there was a problem in writing the control code to the register. If the function returns 1, this indicates that the control code was written to the register successfully.

readCode(): The flight microcontroller control_readCode() function will take no input parameters. This function will simply serve to read in the control code that was stored in register by the control_writeCode function. This function will be triggered by the takeoff and flight procedures during the control handling step. Since it cannot be guarantee that the current procedure running is read to handle a control code, the readCode() function will be required. A read operation that allows the takeoff and flight procedures to retrieve the control code when they enter their control handling step is required since at the time it is possible to be in a hazard procedure. This function will return one of two possible values. If the function returns -1, this indicates that there was a problem in reading the control code from the register. If the function returns 1, this indicates that the control code was successfully read in from the register.

6.5.8 Time Class

The flight microcontroller time class will allow for delays and timers to be used in various procedures. These procedures that require timers and delays will be able to call the functions from this class in order to allow them to dynamically adjust the procedure's timings. One primary purpose will be within the collision and takeoff procedures where these particular procedures are required to last a specified amount of time.

delay(secs): The flight microcontroller time_delay(secs) function will take a time in seconds as an input parameter. This function will then provide a delay of that time interval and then simply return. This function will be helpful in certain class functions where movement change requests are made and the requested change is too drastic of a change, so a delay must be used to incrementally make changes over a slightly longer time period to avoid crashing the aircraft.

timer(start): The flight microcontroller time_timer(start) function will take a Boolean start value in as an input parameter. If the Boolean start value is true, this will indicate to the function to start the timer. If the Boolean start value is false, this will indicate to the function that a timer read request has been made.

6.5.9 Exception Class

The flight microcontroller exception class will allow for the logging and writing of exceptions. Many procedures will call functions in this class in order to log particular errors for debugging purposes, as well as display these errors by writing them to system LED's.

log(errorMessage): The flight microcontroller exception_log(errorMessage) function will take an integer error message value in as an input parameter. This function will be used when errors occur in the code. These errors will help diagnose the problem the aircraft components may have had during flight. This function will hold statically set error message codes signifying different errors. This will allow control over each specific error in terms of how this error will be handled.

write(errorMessage): The flight microcontroller exception_write(errorMessage) function will take an integer error message value in as an input parameter. This function will be used to write the error message out to LED's for a visual representation of the error during testing. This function will be called by the exception_log function if a particular error code being handled needs to write to the LED's.

7.0 Project Prototype Testing

7.1 Hardware Test Environment

The system will initially be tested in the laboratory. Under stringent test and measurement conditions, all components and sub-circuits will be evaluated to ensure they exceed performance specifications and expectations. Criteria for various values will include signal integrity, voltage and frequency tolerances, operating temperatures, angular position, and image quality will be evaluated. Field testing will include the aforementioned values and criteria while the system is operating on battery power. The system will be interfaced with the test aircraft in static and dynamic testing.

7.2 Hardware Specific Testing

The hardware specific includes tests for all hardware components. Each section will discuss the testing and results criteria. Since testing is crucial for this project, each test will be fully detailed. If the desired results are not met for specific sections, the project will have a higher chance of failure.

7.2.1 Camera Basic Test

In attempt to test subsystems in a progressive manner, the camera will be supplied with power and clock signals and its output synchronization signals will be tested to verify understand of the datasheet. In addition, testing images printed with the primary colors will be placed before the camera and the data content of the parallel bus will be probed to verify appropriate bit patterns. For testing purposes, I2C connections from a microcontroller will be routed from the camera to read and write the configuration registers this will allow the team to verify its ability to set camera modes as appropriate.

The camera will be connected to the I2C and parallel busses of a STM32, power and ground, and a frequency generator. Numerous registers will be written to including frequency scaling, resolution, frame rate, camera reset, and color encoding.

All registers will then be sequentially read and verified to verify understanding of the data sheet for the image sensor and ensure I2C communications are occurring as expected and that camera operation can be controlled.

All camera pins used will be probed using an oscilloscope to verify signal integrity and again ensure understanding of the data sheet.

Finally, printed cards with the primary colors will be placed in front of the camera and the bit patterns will be evaluated to verify proper operation of the camera's visible field as well as the black borders that are transmitted by the camera.

7.2.2 Optical Microcontroller to Memory Test

The selected microcontrollers will have their memory handling bus connected to the selected memory modules. The microcontroller will execute code which operates on mathematical equations that produces predictable binary outputs allowing easy verification that numerous byte patterns can be written to all locations of the memory modules. In addition, signal integrity of all memory timing and data lines will be verified with an oscilloscope to verify the absence of impedance mismatches.

7.2.3 Optical Microcontroller to Memory and Display Test

Code will be executed that sends large amounts of data words to memory then back out of memory into the display to produce large areas of predictable colors to the display. The test is for verifying the ability to transfer video data without significant latency. In addition, the team will verify their understanding of the applicable datasheets. Finally, the team will ensure that the display is capable of displaying all available colors and color depths in every area of the display in a manner that is easy to verify by observing the display.

The optical microcontroller, memory, and LCD will be mounted on prototyping boards. Facilities for the appropriate temporary power and oscillator connections will also be created. These boards will facilitate bread boarding connections for testing, verification, and possible reconfiguration.

The microcontroller will be programmed to perform basic power on self tests. Once these are completed, code will be executed that writes then verifies data words to and from the memory. Every section of the LCD will be written to in visually distinguishable color patterns. The microcontroller will then write simulation data to the memory then back out and to the display at a rate that stress tests the data throughput rate.

7.2.4 Full Optical Test

Live images will be transferred from the camera to the memory and then to the display. Camera focus will be evaluated at the expected range. Practically frame rate and its effect on display image will be evaluated and interpreted as a figure of merit for optical system performance. The camera will be pointed at bright light sources to evaluate the effects of image blooming as well as contrast performance. These tests will also verify the continuous real-time capability of the optical microcontroller to act on camera synchronization signals and produce display synchronization signals as appropriate.

7.2.5 Optical Microcontroller to Flight Microcontroller

The optical microcontroller will run image recognition software and communicate steering data in the same manner as it will in operation to the flight microcontroller. Servos will be attached to the flight microcontroller circuit changes in angle in all three axes as well as altitude, should trigger changes in servo position allowing the team to verify correct operation.

7.2.6 Flight Microcontroller to Gyroscope and Barometric Sensor

Microcontroller will be connected to the flight control sensors. I2C signal integrity will be verified. The system will execute temporary sensor interpretation code and test circuit will be tilt to precise angles in all three axes. In addition, the system will be run on battery power and taken into different floors of the building and the changes in barometric pressure relative to the new altitude will allow the team to verify the flight microcontroller is correctly interpreting signals from the barometric sensor.

7.2.7 Flight Microcontroller System Test

The flight microcontroller, 3-axis gyroscope, barometric sensor, and a test servo will be connected to power and oscillator testing facilities. Simulation of steering advisory words that would normally come from the optical microcontroller will be fed to the flight control microcontroller to verify proper operation.

Physical stimuli will be applied and/or verified. The gyroscope will be tilted in each axis to specific angles referenced to the horizon and the encoded signals will be verified. The gyroscope performance criteria include linearity throughout its range and output stability. The local reported barometric pressure and relocation to various heights will be used to verify barometric sensor operation and interfacing.

7.2.8 Power Supplies Test

The power supplies will be statically tested in the lab for rated current and voltage before they are connected to the more sensitive parts of the circuit. Temperature readings will be taken on active components to verify thermal effects are well below maximum ratings. If temperature approaches maximum ratings, simulation of airflow across heat sinks and the effects on observed temperatures will be perform in laboratory.

7.2.9 Battery Level Display Test

The flight control microcontroller will be interfaced with the propulsion battery to sense its voltage. The battery will be loaded with the aircraft’s propulsion motor and propeller for several minutes.

During the course of this loading condition, voltage reports from the flight control microcontroller will be observed. This will facilitate logging of typical real world performance data of the propulsion system to properly characterize propulsion system operation to ensure system compatibility. During this test changes to the relative positions of the motor power connections, motor, and microcontroller will

be made and probing of all microcontroller lines will be performed for the purpose of evaluating the presence of electrical interference caused by the propulsion motor.

7.2.10 Initial tracking experiments

LEDs will be attached to the pins for guidance advice emanating from the optical microcontroller. Image recognition code will be run on the optical microcontroller. A target will be selected during this test setup will be moved up, down, left and right, then steering advice LEDs will be observed as an indication of the ability of the optical microcontroller to track a target.

7.2.11 Installed System Static Test

The entire system will be attached to the selected aircraft. Center of gravity of the aircraft will be re evaluated and corrected as needed. The facilities for semi-permanent mounting of the system on the aircraft will be assessed in terms of ease of mounting, estimated crash resilience, and airframe stresses. Operation of the entire system will be verified to ensure unforeseen packaging and environmental stressors do not impede system operation. A test target will be selected and the aircraft attitude will be varied to verify the guidance system changes the position of the flight control surfaces appropriately.

7.3 Software Development Environment

When deciding on an integrated development environment there are many options in the market. The tool chain selected to develop in is called Atollic TrueSTUDIO. There are a few reasons why this toolchain was selected but the primary reason is previous experience. Although it is just a toolchain it can be very powerful to have previous knowledge and an understanding of the features within it. Aside from having previous experience working with this IDE, Atollic TrueSTUDIO has many other great features that make it a beneficial choose for developing.

Atollic TrueSTUDIO is one if the top embedded systems development tools for ARM microcontrollers. With its optional add-on tools such as its inspector and analyzer, debugging problems in your embedded systems becomes a much simpler task. The toolchain has a wide range of microcontrollers it supports which will be beneficial if the current microcontroller is insufficient enough. Atollic offers a free version of TrueSTUDIO called their lite version which has a code size

limitation of 32 kilobytes but apart from that it shares many of the features included in the Pro version.

What makes TrueSTUDIO such a great toolchain is its debugger. Its debugger is capable of debugging single-core and multi-core systems, as well as debugging both embedded microcontroller code and Windows PC applications. The debug function is a simple one click startup that provides easy to use data inspection windows. These windows are used to monitor various variables and watch expression, CPU core information and peripheral register values, as well as memory monitoring. The ability to monitor all of these components in real time makes this development tool a perfect choice for group development because the system will run as state machines and there will be multiple communication devices to compare stored images. The TrueSTUDIO debugger will be able to monitor all of these things.

7.4 Optical Processing Software Specific Testing

Section 7.4 covers the optical processing testing in detail. The testing will ensure each software component is functioning correctly. If any section is improperly functioning, the chances for errors or malfunction increase. The collision, target center, camera switching, tracking, and false positive sections need to function or the system will fail. Since some sections require another section to work properly, they will need to be tested first.

7.4.1 Optical Fail Safe Test

In order to verify the optical fail safe code is properly functioning, multiple tests scenarios will be conducted. The first will be to check if the flight code will transition properly into the optical fail safe code. The next two will check if the optical fail safe code will trigger and cause the image processing to send landing signals. The next test will see if the optical fail safe activates when necessary but then exits when the criteria is not met. The final test will check only the values supposed to be checked.

Flight mode to optical fail safe: This test will test will be conducted to ensure the code will trigger optical fail safe mode. Since the requirement to enter optical fail safe mode is checking the initial memory location, the memory location will be set to the bit equivalent of black. The next step will be to execute the flight code which will have a case statement checking the first bit. If the code is properly working, the optical fail safe test should execute as well as fail since the rest of

the values should not be black. From this point, the remaining flight code should execute.

All black test: This test will be to store bit values for black into all memory locations read for image processing. If the code is working properly, the code value to enter landing mode should be stored in the register of the optical microcontroller for the flight microcontroller to read.

Camera fail test through image processing: This test conducted will be a variation of storing all black values since the camera will be disconnected to mimic the camera failing during flight. While the camera is disconnected, there should be no data flowing across the bus which will either trigger the camera test for the flight microcontroller or have all bit values of black stored into memory.

Optical fail safe escape test: This test will check the optical fail safe code runs through the majority of tests but still escaping and returning to flight mode. This test is to ensure the spots are correctly being checked as well as ensuring there was not an enigma transferring data to memory. For this test, half of the memory locations read will be set to the bit equivalent of black including the first bit for the optical fail safe to trigger. Even if half of the screen or more is set to black, optical fail safe mode should still terminate to flight mode if any area check is not black.

Optical fail safe exact location test: This test will verify the locations checked are the exact locations expected. The only values in memory to be set to black are the locations set to be checked by the optical software. This will ensure the specific values to be tested are the ones set to be checked.

7.4.2 Collision Mode Test

The first collision test conducted will check if the case statement in the flight procedure will activate. In order for collision mode to be activated, the first pixel value must either equal the target pixel value or be within an acceptable threshold. The first memory location checked will be set to target value while the rest of the memory locations will be set to values outside the target color and threshold. The result of this test should show collision mode being entered as well as returning to flight mode since the screen is not filled.

The second collision test conducted will check the result of a valid case. In order for a valid collision case, all four corners of the frame must either equal the target color or be within an acceptable threshold. There should be two possible outcomes of a valid collision mode which is either the camera being switched or

landing mode being sent. This test will be repeated twice in order to check each scenario executes properly. If the collision is valid and the selected camera is not zoomed, the optical microcontroller will have the landing sequence placed in a specific register which can be verified by checking the register. If the collision is valid and the selected camera is zoomed, then the code should switch the camera and return to flight mode.

7.4.3 Camera Switch Test

This test is intended for any testing that occurs with the dual camera system. The system will be powered up with the zoomed camera selected. It will then be moved toward the selected target. Proper operation will be signified by the system by selecting the non-zoomed camera as soon as the target fills the field of the zoomed camera. In order to tell when the target will fill the screen, code that does not turn off the display will be utilized during these tests. The LCD should then show the object repositioned and not filling the screen.

7.4.4 Target Center Test

In order to verify the correct codes are being set to the flight microcontroller during flight mode, the target center test is required. This test will check that both the calculation with respect to the center is correct as well as the correct adjustment is being transferred.

Centering test: This test will be done by selecting the target and then moving the camera around the target. The code will be adjusted to execute the flight mode loop once as well as to display the image onto the LCD. If the flight mode loop was not restricted, the information displayed will be constantly updated and difficult to verify visually. Once the loop has executed successfully, the new frame should be shown on the screen where the center of the target is away from the center of the screen. The frame will be split into equal sized sections which will be hard coded with respect to the resolution. Depending on the section the center falls into, the appropriate adjustment should be placed into the register of optical microcontroller.

Correction adjustment test: This test will be executed to ensure the correct code sequence is being transferred for the specific area. This test will be fairly simple since the center of the target will be specified manually. Since the specific sections of frame will be known along with the adjustment code, predetermined values could be input to test. For example, manually inputting a target center of 0,25 pixels should result in a level 1 right code sequence and a -300, 400 should

result in a level 2 left and up code sequence. Through this process, every possible code sequence will be checked using predetermined center coordinates.

7.4.5 Seek Mode Test

This test will be conducted to verify the correct execution of seek mode. This will have to check four areas which include the entering seek mode, changing camera, re-finding target, and continuing seek mode.

Entering seek mode: This test will be executed to test the correct function of seek mode. The target will be acquired and then purposely lost. If the code enters seek mode, then execute of seek mode is confirmed.

Camera test: This test will be conducted by checking if the execution regarding the camera is accurate. Once seek mode is entered, the first procedure is to check if the camera is zoomed or not zoomed. This will be checked both ways with initial camera entry being zoomed and not zoomed. If the zoomed camera is selected while entering seek mode, then it will go directly into the next procedure. If the non-zoomed camera is selected during seek mode entry, it will select the zoomed camera. Each scenario will be checked in order to determine proper function.

Finding target test: This test will determine if the appropriate procedure is followed if the target returns to the sight of the camera. Once seek mode is entered, the only way to exit is to find a valid target. Two possibilities exist once a pixel that matches the target color is found: a valid target or a false positive. A validation test is conducted once a possible target is found which will also be tested during this process. The validation test, which will be tested during this test, must be passed in order to return to flight mode. During the testing phase, both a false positive and a valid target will be displayed while in seek mode to determine both sections are working properly. If a false positive is displayed, the validation test will fail and return to seek mode looking for a pixel with the target color value. If a valid target is displayed, the validation test should succeed and return to flight mode. The validation test is tested by reversing the previous test. If a valid target is displayed and seek mode is continued, then the validation test is invalid. Also, if an invalid target is displayed and flight mode is entered, then the validation test is invalid.

Continuation test: The continuation test will verify that control from seek mode into flight mode was completed appropriately. Once seek mode has been entered, a valid target will displayed in order to end seek mode. Since the LCD will be

activated during this test, a visual check will ensure a proper transition. If movement of the target is detected and valid sequence codes are stored in the optical microcontroller's register, then the transition from seek mode to flight mode has been confirmed.

7.4.6 Tracking Test

A target will be locked by the user. The powered system will then be rotated in three axes to verify that the servo outputs change in such a way as to point the airplane in the correct direction. This testing will be performed at various distances to ensure the tracking code and optical hardware work optimally. If the target is lost while still in camera range, then tracking test will fail. If the target stays on the LCD screen and continues to produce valid sequence codes, then the tracking test will be successful.

7.4.7 False Positive Test

Camera environment testing will need to be performed. Testing plans include taking the camera for field testing in specific environment and take various images to test for false positive readings. A false positive reading would be considered any other pixels matching the test object's color that could pose problems when analyzing the images during flight. This testing would help conclude whether or not false positives will be a major problem that must accounted for as well as help determine the ideal color to use for the target object.

7.5 Optical microcontroller Specific Testing

The optical microcontroller is the main component for handling data flow and processing visual information. This microcontroller will handle the information from the camera and transfer it into memory where it can be analyzed by the optical processing algorithms. In this microcontroller specific testing each peripheral attached to it will undergo data flow test. This includes verifying that data has been read from the camera and correctly placed into memory and that the data from the memory to the LCD is displayed properly. To analyze these tests, lab tools such as oscilloscopes, multimeters and function generators will be used to verify correct values for specific situations throughout the systems process.

7.5.1 LCD Check

In the lab, the optical microcontroller will send every one of the 65,536 color bit patterns to every single pixel of the display, in a slow sequence, allowing the design team to visually verify the proper operation of the display associated code. Once this check confirms valid communication to the LCD, additional post test which include sending a specific image from the camera and sending images will be modified by the image processing. For example, a frame will be sent an image that has been interpreted by the image processing where a target has been selected.

7.5.2 Camera Check

In the lab the optical microcontroller will send register configuration commands via I2C bus to the camera. It will then read every single camera register back to verify camera operation and proper software implementation. Cards printed with the primary colors will be placed in front of the camera, causing it to send bus values to the integrated development environment on a regular basis to verify camera operation in RGB565 mode. After the remainder of the optical half is working proper operation will involve sending captured image frames to the display.

7.5.3 Optical Microcontroller Check

Proper operation of both microcontrollers will be indicated by the appropriate operation of the peripherals attached to them. For instance, when the power on self test cases displayed color patterns to occur in the correct manner, the microcontroller will have correctly provided the appropriate control signals and transferred data.

7.5.4 Class Specific Function Testing

Class specific testing for the optical microcontroller will include monitoring the systems state machine. By simulating the signals required for state transitions and using the TrueSTUDIO built in debugger, monitoring the systems state can be done with just a computer. This will allow for development and testing to be done outside the lab with no restrictions.

Among other class specific testing includes communication buffering. Since message can potentially occur at the same time it is important store the messages in temporary buffers for later handling. Testing to make sure the buffer sizes are large enough and can be accessed correctly will be done through the

debugger. To verify the messages are being read correctly and sent correctly the team can hard code messages into the buffers and read them on an oscilloscope as they are being sent over the communication lines. Testing for each time of communication protocol used will all require the same type of testing so these procedures will be repeated several times.

Since the microcontroller to microcontroller bus will be a 5-bit bus designed specifically for this system extensive testing will be done to verify the registers used are changing correctly and can be captured by each of the microcontrollers.

Additionally, memory management functions must also be tested in the class specific function testing. Since the memory management class will be controlling the flow of such large amounts of data, speed tests as well as quality test will done to ensure that bottlenecking won't occur during this process. Because the external memory can only hold so many images the memory bank will eventually need to be cleared and overwritten. To do this, class functions will be designed to monitor when an image is too large for the remaining space or when there is enough room for it to fit. Testing the functionality of these functions can be done by hard coding starting points and image sizes and testing to see if the function can correctly detect when an invalid amount of space is left.

The last class specific function testing for the optical microcontroller is the interrupt class. Since the frames will be captured at specific intervals, an interrupt driven function that uses the system clock will be needed to calculate correct time. For example if the system needs to take 5 frames per a second to provide sufficient data to calculate a flight path then the timer interrupt should occur every 200 milliseconds. This interrupt just sets an output pin high enabling the flow of data to the memory bus. Not only does an interrupt need to occur to start the flow of data but it also needs to occur to stop the flow of data. This can be done by reading the sync line from the camera which signals when the end of each frame is. To test these signals and to make sure their timings are correct an oscilloscope can be hooked up to the lines which measure over an interval of time. This test combined with the memory test will verify that the data flow from the camera to memory is working correctly.

7.6 Flight Microcontroller Specific Testing

The flight microcontroller testing will involve the testing of the individual power on self tests that will need to be performed during the initialization procedure prior to takeoff. These POST’s will include specific class function tests and return value

analysis, physical component testing, as well as interrupt handling and procedure change analysis.

7.6.1 Flight Microcontroller Check

Proper operation of both microcontrollers will be indicated by the appropriate operation of the peripherals attached to them. For instance, when the power on self test cases displayed color patterns to occur in the correct manner, the microcontroller will have correctly provided the appropriate control signals and transferred data.

7.6.2 POST Function Testing

POST functions will all need to be tested together in a formal POST to ensure that the procedure conditions are being followed correctly and that the procedure is moving through the steps in the correct order. These tests will be important as the POST's will occur upon aircraft power being turned on and initialization procedure started. Tests will also be performed to ensure that manual use of the kill switch will enable the user to send the aircraft into the abort procedure properly if the user visually detected problems during the POST.

7.6.3 Class Specific Function Testing

In laboratory testing, the classes that contain read and write functions will need to undergo basic tests in which will simply read in values and write this data to registers which will be used for data analysis to ensure accurate readings. In another basic test, write functions will be provided with values to store in the registers which will be confirmed by reading the data in these register locations by checking if the values were correctly stored in the registers after receiving input.

The team will verify the flight microcontroller is accurately and continuously monitoring system battery level by comparing microcontroller output with physical measurements in the laboratory. A low battery condition will be created and flight microcontroller response will be verified to ensure all servos are centered and motor controller is shut off.

Testing will need to be performed on the kill switch mechanism to ensure all safety requirements for aircraft flight are met. This will involve testing of the physical mechanism that will turn power to the motor off when the kill switch is

tripped via the kill switch button, as well as if the aircraft RF receiver goes out of range.

In laboratory testing, the control_servoPulse function will need to undergo a control code test which will test that the control codes provided to it as input are being mapped to specific servo pulse widths correctly. The function will also need to undergo a servo motion test in which certain pulse widths will be sent out to the servos, during which measurements can be taken of the physical movement of the aircraft surfaces, including the ailerons, elevators, and rudder to ensure proper movement of these elements.

Gyroscope data readings post-launch will be used to identify the point at which the aircraft is turning at the maximum desired rate for the magnitude of directional error. Aircraft structural integrity and dynamic stability are very important to avoid airframe damage. Additional gyroscope tests will involve gyroscope error reading tests to check at what rate the gyroscope may send faulty data to a procedure. The gyroscope will also need to be tested in lab to get an accurate level reading.

Barometric readings will be taken at ground level and the value will be stored in memory during every initialization procedure. The decision to go with a dynamic initialization of reading gyroscope data during each initialization procedure instead of a static value was to help eliminate possible variations with gyroscope output over time. Testing will need to be conducted to ensure these base level readings are accurate for different weather and locations. Testing will involve reading various barometric values at different altitudes. This data will then be accessed by the altitude check function during flight to ensure the aircraft is not approaching too close to the ground. Software testing will include analyzing the output of the altitude check function with the aircraft at various altitudes to check movement correction. Testing of the barometric sensor provisions will include test code that will relay the sensor readings, in easy to understand units, to the integrated development environment. Sensor readings will be taken on different floors of the ENG3 building to verify the accuracy of the hardware and software implementation. The barometer testing will also include error testing in order to get an idea of how likely it will be that the barometer will send over inaccurate readings to a procedure.

Movement correction will be handled by the servo pulse adjust function, which upon getting movement adjustment data values from the optical microcontroller, will properly adjust the signals out to the appropriate servos to adjust airplane flight toward the object. Testing will facilitate the input of specific static control

codes as if coming from the optical microcontroller to test response of the servos. This test will also be used to verify landing and hazard procedures in the event certain control codes are received.

Altitude correction testing will be performed to ensure proper correction of aircraft movement, as well as allow the team to analyze the response time of the system to the low altitude alert. The specific corrections observed in this test will be how quickly and precisely the ailerons can level the roll of the aircraft, how quickly the elevators can be lifted, as well as the time to full throttle power. These tests empower the team to monitor the stability in these changes all happening together in a controlled lab environment. The timing data gathered from this testing will also help establish the minimum altitude threshold needed to regain altitude before collision with the ground occurs. This test will help greatly prevent ground collisions better during the flight procedure.

Attitude correction testing will be performed to ensure proper correction of aircraft movement in the event of an attitude correction alert. The specific corrections which will be looked for in this test will include bringing back the roll axis magnitude to an acceptable level if the alert signaled that the roll threshold was exceeded, as well as leveling the pitch of the aircraft and bringing the engine to full throttle if the alert signaled that the pitch threshold was exceeded. The group will be able to monitor the stability of the aircraft during these changes occurring in a controlled lab environment. The timing data gathered from this testing will also help establish the roll and pitch thresholds needed to regain control of the aircraft and resume flight procedures. This test will greatly help to prevent stalls and departure from controlled flight better during the system demonstration.

7.6.4 Miscellaneous Testing

Optical microcontroller testing in lab including LED’s off the optical microcontroller to display control codes apart from reading them from the flight microcontroller.

Flight microcontroller testing will also need to consider test cases where there is a problem with the optical microcontroller, or the control code bus. These tests would help test the catch of various exceptions which could be faced in unexpected events.

7.7 Field Testing

Field testing is an essential part of the project. It will provide real situational data that will not only facilitate design and implementation of the system but will also verify that it is fully functional. Since the airplane will be performing outside under uncontrollable conditions, it is vital to run extensive field testing to verify every condition is handled correctly.

7.7.1 Barometric Sensor Field Testing

The barometric sensor in the flight microcontroller system will need to be field tested on top of lab testing to ensure accurate barometric readings in different weather conditions as well as different locations. Evaluation of the effects of airflow and humidity

7.7.2 Optical Distance Test

The optical distance field test will be executed to determine exact system specifications which will relate a distance versus target size. In order to determine the exact distance for the autonomous airplane to track, various distances will be marked and the plane will be launched and observed. From this test, a specific distance will be determined for both launching range and the effective distance of seek mode.

7.7.3 Stock Airplane Field Testing

Before building and implementing the design the entire team must first learn the characteristics of a remote controlled airplane. In the early stages of design test flights using the stock commercially available radio system must be conducted. These flight tests will be performed to identify the control surface positions that cause the aircraft to fly straight and level. These positions will be then duplicated as the servo null positions created in the guidance system code. Based on observed behaviors, decisions will be made regarding reasonable extremes of servo positions based on the behavior of the aircraft in flight.

Apart from determining these null and extreme positions, the airplane will be monitored while in the air. Since the group has limited knowledge of remote controlled airplanes, this will be a very key test to solving the task of controlling the airplane autonomously.

Another important factor to monitor during these test runs will be how the remote controlled plane handles natural variable, wind primarily. Because of its size even

the smallest gust of wind can play a huge factor on steering the aircraft to its target.

7.7.4 Autonomous Airplane Field Testing

In this phase of field testing the autonomous guidance system will be integrated into the selected airplane. The system will be tested in its entirety, including the power on self test to its motor functions to its optical recognition and tracking. Complete testing and debugging of the system during this phase is critical to facilitate efficient follow on testing in the field. This means very close attention must be paid to the conditions of the weather and what event occurred prior to any errors or faults. Hopefully by this point in testing all of the larger issues will have been resolved and the focus will be working on tweaking the minor unexpected issues.

There are a few main components that will be receiving the most attention in during this second phase of field testing, the first being the optical microcontroller. Up until this point all tests run on the optical detecting and tracking algorithm will be in the lab with very specific images to test. These background images and objects that will be tracked will only ensure that the algorithm is working correctly but will not be putting any kind of stress on it. In the field testing the camera will be taking very detailed and unique images that will have to be handled very carefully by the microcontroller. They will essentially be a stress test to see how accurate the image processing is or how effective it’s working. This stage of testing is very important for the image processing component of the project because it will help determine what needs to be corrected and or adjusted in the algorithm.

The next main component that will be under extensive monitoring is the flight controller. Once again, at this point of testing all tests will have been performed in the lab. Laboratory testing will be designed to simulate real world conditions as closely as possible. Careful design of the gyroscope will ensure the plane won't flip or dive. Thorough testing of the barometric pressure sensor will minimize unacceptable altitude deviations while the system is in operation. In the field test the flight controller will have its first go at autonomously flying a remote controlled plane. This task is one of the most difficult parts of the entire project and will make or break the success of it. The flight controller on top of such a difficult task will also be put up against natural variables such as wind that it will have to balance against while maintaining its primary goal. Gusts of wind will attempt to steer the airplane off course or even flip it over during its flight, a condition which the flight controller will have to correct for.

Static propulsion tests will also be performed during the field testing to identify typical run times for the propulsion motor's battery. Other tests will include battery fail safe tests, and measure electromagnetic interference in the guidance system created by the propulsion motor.
With the guidance system installed, the center of gravity of the entire aircraft must be evaluated and tailored again which is paramount to maximize flight stability. The static propulsion test is also where the calculations of the null and extreme positions of the servos will be tested and verified that they can handle any situation.

8.0 Administrative Content

The administrative content section breaks down the project from an overall view of the project plan. It discusses how the group planned out the budgeting, due dates and responsibilities. The meetings which formulated this content, though not as important in terms of actual progress, are a good starting point in early stages of design. By laying out the framework and assigning roles to the project, group members can start working on parts of the project as soon as possible. With budgeting planned out, it can be used as a reference to determine if whether the approach is cost efficient. Setting milestones and due dates is an excellent way to make sure the project is moving at the correct pace. If it starts to fall behind around a specific due date, it can be caught early and correct by putting in extra time. These administrative procedures form the backbone and game plan which must be followed in order to complete the project on time.

8.1 Roles and Responsibilities

Division of project duties among team members is facilitated by the varied interests that each member possesses. System development duties can be divided among the constituent subsystems. These subsystems can be designed and developed independently of each other. For instance, the optics development does not require the hardware design for the flight controller to be developed and tested. While the flight might need some hardware for testing, it too can be developed without any other sections. This can be applied to each section of the project. The ability to have each member of the group design their part of the system independently will means the team can achieve great strides in development and not be held back or have to wait for a separate part to be completed.

The assignment of sections does not mean that a specific person is responsible for that part only and nobody else can work on it. It means that person is going to be the lead on that part and is to make sure that it works successfully. Every member has the opportunity to work on any section they would like. This technique for development was agreed upon and selected by all members of the group for reasons mentioned above. Although this method was chosen for primary development strategy, it is subject to change if problems arise. The roles and responsibilities for each of the four sub parts are broken down and described below.

The image recognition hardware and software will continue to be developed, researched, and tested by at least two members. The associated responsibilities include providing hardware suitable to robustly identify targets and objects and provide steering advice data. It's worth noting that that the image recognition hardware must be shared with the image data handling hardware.

The image handling hardware and software will be monitored by two members. These duties include hardware and software for moving image frame data between the optical microcontroller, memory, camera, and display. The responsibilities for power on self tests are included in this area.

The flight microcontroller, sensors, and servo interfacing duties will be handled by the electrical engineer and the flight microcontroller programmer. Duties associated with this part of the system include the hardware and software to read sensor values and steering advice, and maintain the stability and direction of the airplane to prevent crashes and strike the target whenever possible.

The remainder of the hardware, including the airplane hardware, is the responsibility of the electrical engineer. Guidance system support circuitry, airplane equipage, and airplane flight trim and general maintenance are included in this area.

8.2 Milestone Discussion

Milestones and due dates are essential components in designing large scale projects. They help break down the project into smaller components and allow development to happen in phases. By strategically setting milestones based on a thorough examination of the task, a well designed procedure can ensure that the project will be completed on time. The following dates for each hardware and software test, acquiring all hardware and supplies, assembling all hardware

prototyping and final assembly are listed below. Each date selected will be explained following the list.

15 November - completed system schematic
30 November - completed system document
15 January - complete system BOM including associated construction supplies
31 January - surface mount components mounted to prototyping boards
15 February - hardware and software testing of picture frame data path
28 February - hardware and software testing of flight control system
15 March - hardware and software testing of full system (in lab)
31 March - fully assembled final system
15 April - flight testing begins

The completion of the system schematics is an early stage milestone that must be completed in order to determine what components will be needed in the design. Finding compatible component parts that not only work together but also can handle the purpose they are intended for is a very difficult assignment. By taking care of this first it allows for a visual idea of how the system will work together. Because problems may occur after the software development begins, these system schematics can be easily altered and modified if designed correctly. With this in mind the date of November 15th was set in order to leave time to complete the system document.

Although the system document has an official due date of December 6th, it was decided as a group to push this date forward. The reason this was decided is to allow a soft finalization of the document with the remaining days. This time will allow for review all sections and confirm that everything is correctly stated.

A complete bill of materials for the system is associated with the hardware design. Since the initial design was done before software development, alterations in the hardware design and or component addition may be necessary. By January 15th a complete list must be finalized because prototype board will be needed for software development. Once these prototype boards have been finalized and printed all surface mount parts must be added and verified to be installed correctly. By the end of January all prototype boards must be complete and hardware tested. It is important to make sure this board works because software will soon be installed on them. Up to this point all software will have been designed on development boards.

The milestone date for software testing on the prototype board is set to begin on February 15th for picture frame data path flow. The reason this is set before the

flight control testing is because verification of correct image processing is necessary before it can be handled by the flight controller. Though certain test can be done on the flight controller independent of the rest of the system the final test must be done after. There is a two week testing period between verifying the data path flow and the flight control testing. In this time period corrections will be made if necessary. It was determined that two weeks would suffice for any difficulties that have gone unforeseen. Starting February 28th milestone testing for the flight controller will begin which is where tweaking and adjusting of motor control signals will take place. These two milestones are set accordingly to allow for any major modifications that may need to occur. The 28th of February give the team roughly 2 months to handle these issues if need be.

By March 15th the A.O.G. System should be fully tested and with all prototypes working together. At this point the system will neither be installed into the aircraft nor will it be hooked up to any motor but all signals should be evaluated by lab tools and verified to be working correctly. Shortly after this testing the entire system should be installed in the aircraft and be controlling the vehicles steering components as well as its prop. Milestone marked on March 30th will be final lab testing. For two weeks testing for every scenario will be performed and the results verified. All possibilities for faults will be tested to see how the plan will handle. This date was set as a guideline for the final system. With a month left and a complete system there shouldn’t be any problems that can be resolved at this point.

The final milestone set is the field flight testing which is to begin on April 15th. This is the most important testing that will be done throughout the whole project and the most unpredictable. With many variables playing a factor in this testing phase and being that it will be the first time the plane will attempt to fly, it’s expected that there will be difficulties experienced.

8.3 Budget and Finance Discussion

The Autonomous Optical Guidance System was completely funded by the members of the group. With no funding being provided from an external source or sponsor the group must ensure that everything purchased throughout the course of this project is a necessity and chosen carefully. Much research was spent making sure that components would be not only adequate for its task but would work well with the rest of the components. For the development stages of the project, the group members will be purchasing any additional parts deemed necessary to complete the task.

The estimated cost for all parts and supplies is $2000.00. This number is a slight overestimate in case of any major changes that occur unexpectedly.

Appendices

Datasheets

http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATA_BRIEF/DM00037955.pdf

http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00278141.pdf

http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00200423.pdf

http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00269027.pdf

http://www.issi.com/pdf/61-64WV51216.pdf

http://www.sparkfun.com/datasheets/LCD/LCD-08335-1.pdf

http://www.ti.com/lit/ds/symlink/74ac11245.pdf

http://www.ti.com/lit/ds/symlink/reg1117a.pdf

http://cds.linear.com/docs/Datasheet/690812fa.pdf

http://www.fairchildsemi.com/ds/LM/LM7805.pdf

http://www.ti.com/lit/ds/slvs413f/slvs413f.pdf

http://www.bosch-sensortec.com/content/language1/downloads/bmp085_datasheet_rev.1.0_01july2008.pdf

http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/DM00035129.pdf