6-13-2016

CuriPilot

Gregory Cusack
Santa Clara University

Karan Daryanani
Santa Clara University

Nathaniel Tucker
Santa Clara University

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING

Date: June 8, 2016

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Gregory Cusack
Karan Daryanani
Nathaniel Tucker

ENTITLED

CuriPilot

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING
BACHELOR OF SCIENCE IN ELECTRICAL ENGINEERING

[illegible] 6/13/16
Thesis Advisor

[illegible]
Thesis Advisor

[illegible]
Department Chair

[illegible]
Department Chair

CuriPilot

by

Gregory Cusack
Karan Daryanani
Nathaniel Tucker

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
Bachelor of Science in Electrical Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 8, 2016

CuriPilot

Gregory Cusack
Karan Daryanani
Nathaniel Tucker

Santa Clara University
June 8, 2016

ABSTRACT

The drone industry has rapidly grown over the last three years and many large companies have determined that the phenomenon is worth investing in. Recently, Intel has shown interest in this growing industry. Intel developed a next generation processor to be used in drones in addition to investing in the Chinese drone hardware company, Yuneec. Contracted by Intel, the CuriPilot team laid the foundation for linking the hardware and software layers of Intel's drone technology by designing the stabilization control system for the next generation of processors and drones. The CuriPilot team built a foundation for future drone development and gave Intel valuable feedback on both the hardware and software being used with the Curie processor. Looking forward, future development can easily be started with the new EarHart boards and existing drone frames. The elusive system bugs have been fixed and the EarHart board provides a stable environment to continue autopilot development.

Acknowledgements

The CuriPilot team would like to thank the following people for helping development throughout the year:

- Dr. Christopher Kitts for his guidance as faculty advisor.
- Dr. Sally Wood for her guidance as faculty advisor.
- Jeff Ota and the Intel Corporation for sponsoring the team with Curie development boards and Yuneec drone platforms.
- Santa Clara University's Robotics Systems Lab for allowing the CuriPilot team to work on development in their facilities.
- Steven Xing and Michael Rosen for their time and support with the hardware and software.
- Dr. Jacquelyn Hendricks for her assistance with our thesis and presentation.
- All the friends and family who supported CuriPilot at one time or another.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Drone Background

The demand for drones, also known as quadcopters or quadrotors, has risen dramatically in the last few years. Initially drone use was restricted to mainly the military sector due to their high costs; however, hobbyists and engineers around the world have found ways to build small-scale drones to perform specific tasks at a reasonable cost. Buying into the drone market means lower costs on various surveillance, maintenance, and even film projects for not only commercial entities, but also for individuals. People all over the world are realizing the cost benefits and purchasing over 200,000 drones in total each year [1]. However, many consumers are still undecided on drones in the everyday world due to valid concerns regarding surveillance, privacy, and physical safety. In the end, when used in a responsible way, unmanned drones open up doors to a massive number of opportunities that would otherwise be expensive or impossible.

One industry that could benefit greatly from the use of drones is the firefighting industry. In fact, drones are already being used to monitor fires for first res ponders [2]. It is not always safe or practical to fly a plane over a fire, so by flying an unmanned drone with an infrared camera, firefighters can avoid danger and properly monitor the fire. Furthermore, police departments have benefited from the widespread development of drones. While a typical police helicopter costs about $1 million, an unmanned drone can provide essentially the same services but at a fraction of the cost and risk [3]. However, law enforcement use of drones causes a fourth amendment issue if used unwisely; therefore, the FAA requires police officers to produce a search warrant before using a drone for surveillance [4].

Unfortunately, drone pilots, whether they be hobbyists or trained professionals, are not always responsible and have caused the FAA trouble. In fact, last year the FAA reported that 700 drones have had ”near-misses” with commercial airliners due to reckless drone flight [5]. These close encounters are just one of the many reasons why the FAA has rolled out a new law requiring all drones between 0.5lbs and 55lbs to be registered FAA in order to ensure accountability and safety [6]. While there are some regulations that need to be worked out in order to ensure safe and proper flight, drones provide a wide variety of ways to cut costs, reduce safety risks, and increase efficiency in many different industries.

1.2 Technical And Theoretical Background

A quadcopter (also referred to as a quadrotor) is a helicopter-like vehicle that has four equally spaced rotors around a central body. The quadcopter utilizes the multiple rotors simultaneously to create a stable platform for itself. As a result of their high level of stability, multiple rotor drones have become popular among hobbyists and professionals for use in aerial photography, search, and surveillance. For a quadcopter, each rotor drives a propeller which spins in the same direction as the one directly across from it. However, the propeller to its immediate left and right spin the opposite direction. By having the propellers spin in this particular way, the rotors counteract the torque introduced into the system. In other words, if all of the propellers spun the same way, the drone would also spin, rendering it useless in many practical applications. To solve this problem, quadcopters use autopilot software routines and hardware control systems to maintain stable flight.

When dealing with aircraft control systems, there are three main degrees of flight that need to be properly controlled in order to produce a stable flight. These four degrees of freedom include roll, pitch, yaw. To be more specific, roll, pitch, and yaw represent the drone's rotation around the x, y, and z axes, respectively. The stabilization control system allows the user to fly the drone from a remote control without having to worry about stabilizing the drone when flying.

When the ground operator sends a signal to the drone to fly forward, for example, two of the propellers will increase their rotations per minute (RPM) while the two propeller directly across will decrease their RPM; therefore, causing the drone to tilt forward slightly. The two sets of rotors will then increase their RPMs proportional each set based on the desired tilt causing the drone to fly forward.

1.3 Autopilot Systems

When controlling unmanned aerial vehicles, also known as UAVs, pilots need the aid of an autopilot system to ensure safe and accurate operation. A drone can be controlled manually via standard radio controllers but still needs stabilization and routing assistance through an autopilot system. There are various UAV autopilot systems available, both open-source and closed-source. In general, autopilot systems create a layer of abstraction for the communication between user input, sensor data, and the drone's motors for precise and accurate flight.

Currently there are several autopilot systems available. These range from simple hobbyist systems to complex proprietary systems. Most UAV projects currently use some version of ArduPilot. ArduPilot is an autopilot system that is based on the Arduino, an open-source, electronic, prototyping platform. ArduPilot has a strong development community behind it, resulting in a large forum dedicated to helping others get their drones flying, modifying current autopilot functions, and even developing new autopilot applications. Open source allows for all people of various skill levels to take ArduPilot code, modify it, and post it back online. As a result, much of the code that is available is not optimized and incomplete which hinders the implementation of reliable software. On the other hand, proprietary autopilot systems may boast the same functionality as ArduPilot or other open-source systems, but they do not have the support community behind them. The lack of support results in unsolved bugs, lengthy waits for technician responses, and ultimately the inability to reprogram the UAV at any given moment. Furthermore, many of these autopilot systems provide high levels of abstraction which improve open source development but slow down response time.

Although there are many autopilot systems for UAVs, most of them are either proprietary or completely open-source. Intel would like to enter the UAV market, but to do so they cannot use another company's autopilot or the open-source autopilots for legal reasons. Another issue is that Intel wants to control their drones with their new Curie processor and select peripheral sensors. None of the current autopilot systems are optimized for this processor's architecture or the sensors Intel wants to integrate into their UAV.

In the end, one of Intel's main competitors will be MicroPilot, a company that produces both autopilot hardware and software for drones. MicroPilot focuses on all areas of autonomous robotics including fixed-wing aircraft, land rovers, and multicopters. MicroPilot will be a big competitor for Intel since they also develop their own software for their house designed boards.

1.4 Intel's Curie Microprocessor as a Strategic Choice

CuriPilot is a completely new autopilot system built from the ground up. We are starting with a processor that has never been implemented on a drone before. The Intel Curie processor was developed for situations that need a strong computational device and a small form factor. Intel has showcased the Curie with multiple wearable technologies and robotics, and now they want to use it on a drone. The Curie module itself contains a gyroscope and accelerometer, two things necessary for stable UAV flight. CuriPilot is an autopilot and stabilization system that will utilize the Curie's small form factor and built in sensors. Since the project is based on the Curie processor, it is desired that all flight calculations be performed solely on the Curie. That being said, in the future the UAV might also have a supplementary Intel Edison processor to handle a waypoint autopilot system and other computationally heavy tasks such as automatic landing procedures.

The long term goal of CuriPilot is to develop a brand new autopilot system developed solely for Intel.

The autopilot system will be built from the ground up to be fully optimized for Intel's new Curie processor and selected peripheral sensors. The autopilot system will take in sensor data as well as commands from the radio controller to allow for stable controlled flight. By developing the system specifically for Intel's Curie processor and peripheral sensors, CuriPilot will greatly optimize the data flow and processor usage allowing for other processes to run simultaneously and execute more advanced operations.

1.5 About the Curie Processor

Intel's Curie microprocessor is a low-power, 32-bit, system-on-chip (SoC) device that was initially designed for wearable technologies.[1] SoC refers to a low-power, integrated circuit device that contains all of the processors and memory on one chip. The Curie is run on Intel's x86-based Quark processor that contains 386kB of flash memory and 80kB of SRAM. Furthermore, the Curie has a 6 axis IMU (Inertial Measurement Unit) containing the on-board gyroscope and accelerometer. The microprocessor runs on a real-time operating system (RTOS), meaning that it is optimized for reading data, performing calculations, and then writing data with minimal delays. As a result of the RTOS, the Curie serves as a valid option for a quadcopter's control system since the drone will need to read data from the on-board gyroscope and accelerometer, perform stabilization calculations, and then output signals to the four motors in real-time.

As a result of its small size, the Curie is not the most powerful processor. Therefore, data reads and writes from the various sensors need to be properly timed in order to maintain stable flight while also reading data from an RC controller. Also, when adding new software and fixing bugs, the Curie's RTOS needs to be loaded on the Curie each time. The process of updating the RTOS is called "flashing" and requires the team to load the OS onto the Curie's limited RAM each software update iteration. This process usually takes about two minutes to complete.

By utilizing Intel's Curie microprocessor, effective and efficient software and hardware interfaces can be developed to produce a reliable autopilot system. Using Intel's own chips and software development platform, CuriPilot will interface seamlessly with the hardware; therefore, the system will provide the most powerful and effective solution for drone autopilot.

[1]Side Note: The same Curie architecture that is being used for drone stabilization was also used on snowboards at the 2016 Winter X Games.

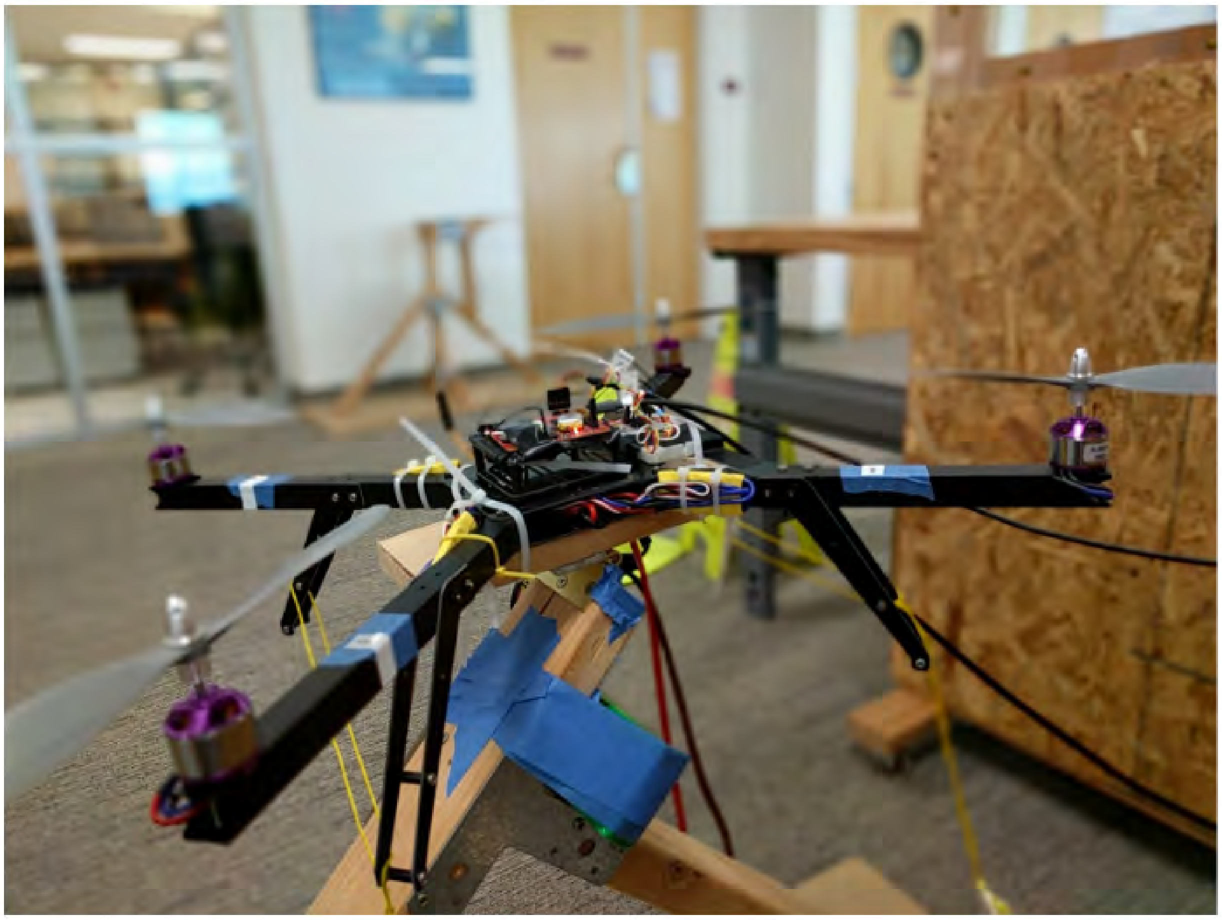


Figure 1.1: UAV Development Drone

1.6 Project Objectives

The goal of the project was to start developing a stabilization system on a quadcopter running on the Curie. With only a development time-line of a year, the CuriPilot team focused on four main areas of development. First, the team began with stabilization using PID rate control, modifying and adding to the current control code. The team also began development on GPS waypoint navigation by integrating MAVLink and QGroundControl together to talk to the Curie. Next, the CuriPilot team looked into various communication methods in order to manually control the drone. Finally, since the Curie is still in the development stage, the team looked to provide valuable board and firmware documentation and feedback to Intel.

In the end, the stabilization process was not fully completed; however, sensor polling rates were adjusted and the beginning stages of angle control were designed. As for GPS navigation, a wired communication protocol was set up that allowed QGroundControl to communicate with the Curie. Furthermore, an RC controller was selected for use in manual control flight over the on-board XBee Radio for reasons that are discussed in Chapter 5. Finally, the entirety of the development process left the CuriPilot team with valuable feedback regarding the board design and the firmware on the Curie. This feedback was then passed on to Intel engineers. After a year of development, ground floor has been laid out for future development teams to continue work on the Curie controlled quadcopter.

Chapter 2

Design and Implementation

2.1 System Diagram

Figure 2.2 shows the high-level architecture of the Curie breakout board and and the ground controller. We decided to break up the architecture into the components that are on the quadcopter and the components that are off the quadcopter. The gyroscope and accelerometer are integrated into the Curie itself, while the barometer and magnetometer are off chip. The Curie communicates with the motors via a voltage step-up converter. The GPS, flight battery, and RC receiver are also on the quadcopter, while the ground control station and the RC transmitter are off. Figure 1.1 shows the physical development drone.

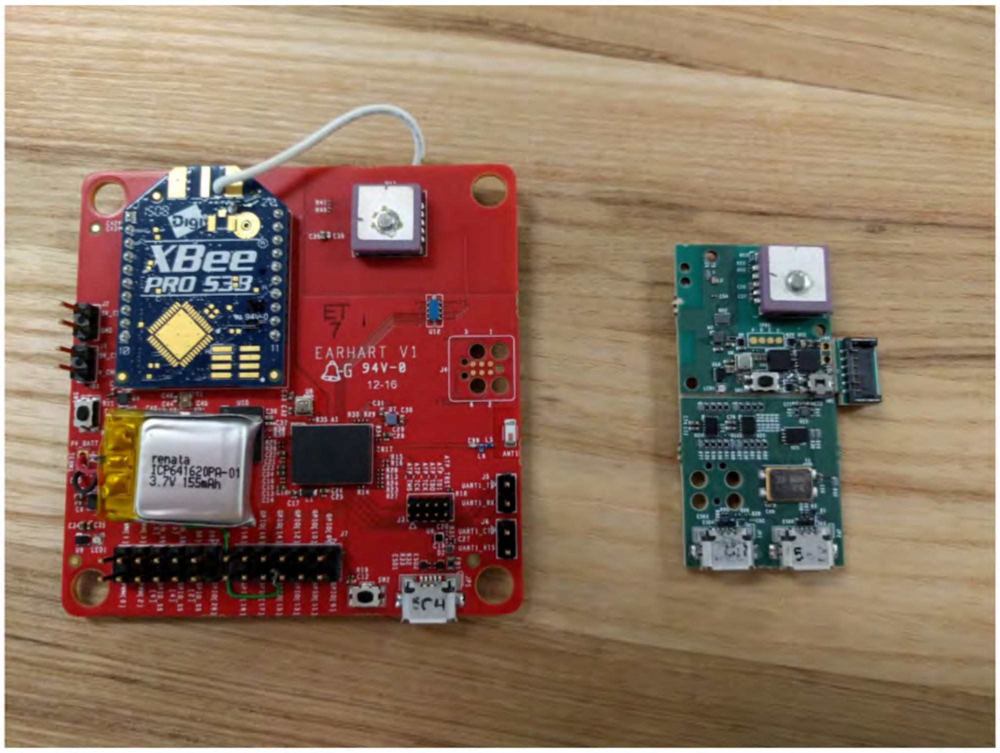


Figure 2.1: Revision 3: EarHart (Left) and Revision 2: Sharkjumper (Right)

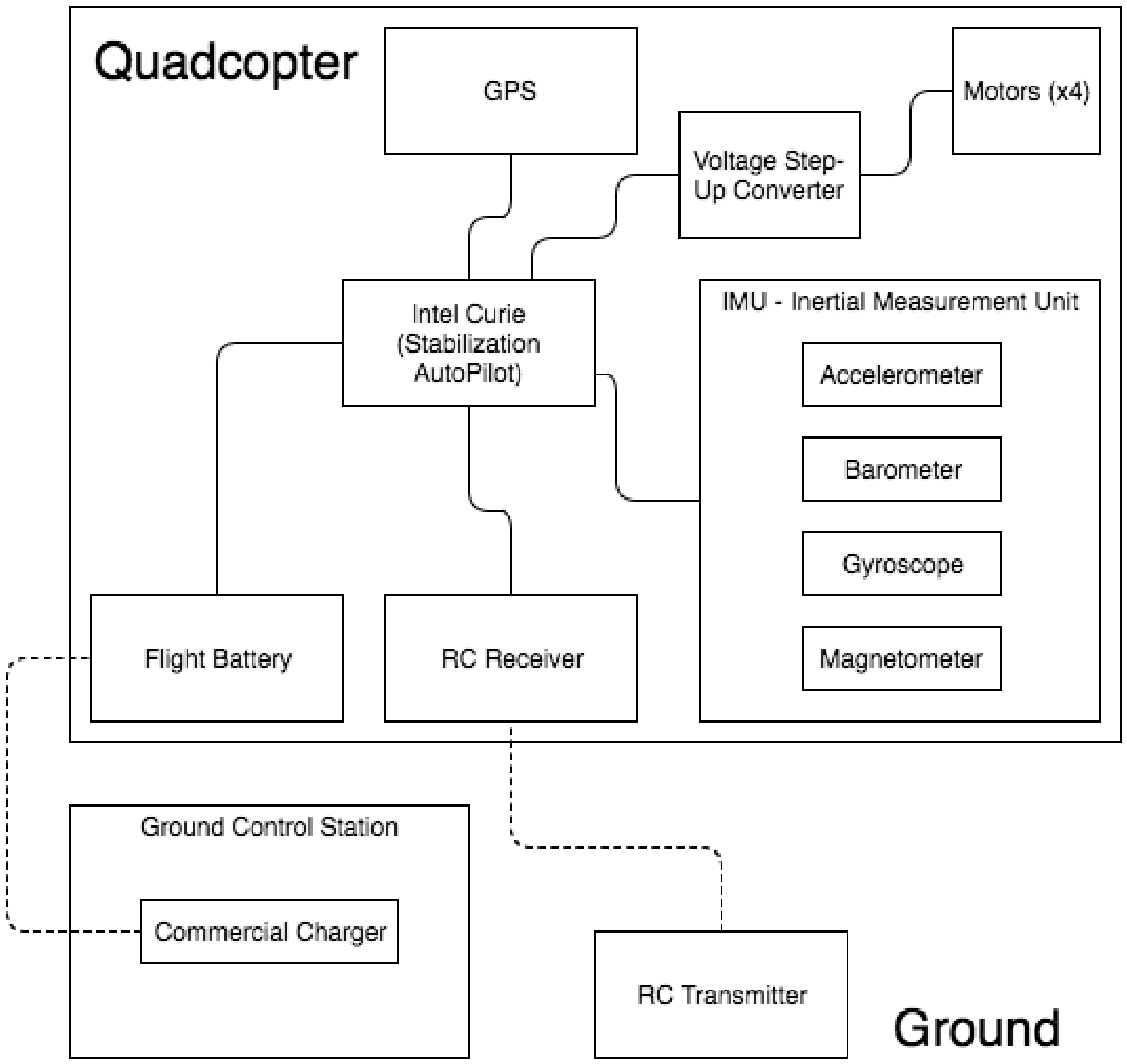


Figure 2.2: System diagram explicitly stating the hardware connections and data streams

2.2 Requirements

The requirements for the UAV Autopilot system system described in the introduction can be broken down into two categories: functional and non-functional requirements. The former outlines requirements regarding what the system does, while the latter outlines the manner in which the functional requirements will be accomplished. The restricting factors that limit the system in regards to sustainability, scalability, deployment, and implementation are outlined in the Design Constraints section.

Table 2.1: Requirements and Constraints

Functional Requirements (The system will...)	Nonfunctional Requirements (The system will be...)	Constraints
Maintain stability while hovering.	Modular	The system must be memory conscious.
Allow for stable user controlled flight	Maintainable	The system must run on the Intel Curie processor.
System will allow for altitude-locked flight	Robust	The system must be be implemented on a Yuneec drone.
	Reliable	The stabilization control system must be developed in Embedded C.
	Safe	

2.3 Use-Cases

A use case defines the necessary steps for an actor (or actors) to achieve a specific goal in the system. Also included are preconditions (conditions that have to be met before steps are performed), post-conditions (conditions that will be true after use case is performed) and exceptions (possible hindrances to use case being performed successfully).

2.3.1 Use-Case Actors

In this case, the use-case diagram seen in Figure 2.3 shows how the actor (the pilot) interacts with the CuriPilot drone.

2.3.2 Use Case 1: Manual Flight

Actors: Pilot

Goal: Manually fly quadcopter using remote control

Pre-Conditions: Turn on drone and controller

Post-Conditions: Actor is able to fly drone

Steps:

1. Turn on drone and RC controller

2. Fly drone via joystick input on RC controller

Exceptions:

1. Batteries out of charge, charge batteries

2.3.3 Use Case 2: Altitude-Locked Flight

Actors: Pilot

Goal: Manually fly drone at locked altitude

Pre-Conditions: Drone is currently flying

Post-Conditions: Actor is able to fly drone at fixed altitude

Steps:

1. Manually fly drone to desired altitude
2. Set to altitude-locked flight mode
3. Manually control drone in X-Y plane

Exceptions:

None

2.3.4 Use Case 3: Way-Point Navigation

Actors: Pilot

Goal: Autonomous flight based off of user selected waypoints

Pre-Conditions: Set controller/drone to waypoint navigation mode

Post-Conditions: Actor is able to fly drone by simply selecting waypoints on a map

Steps:

1. Set to waypoint navigation flight mode
2. Select desired flight path via QGroundcontrol on ground station
3. Drone will begin navigating desired route

Exceptions:

1. Flying indoors, GPS is unable to lock onto satellite. Fly outside instead.

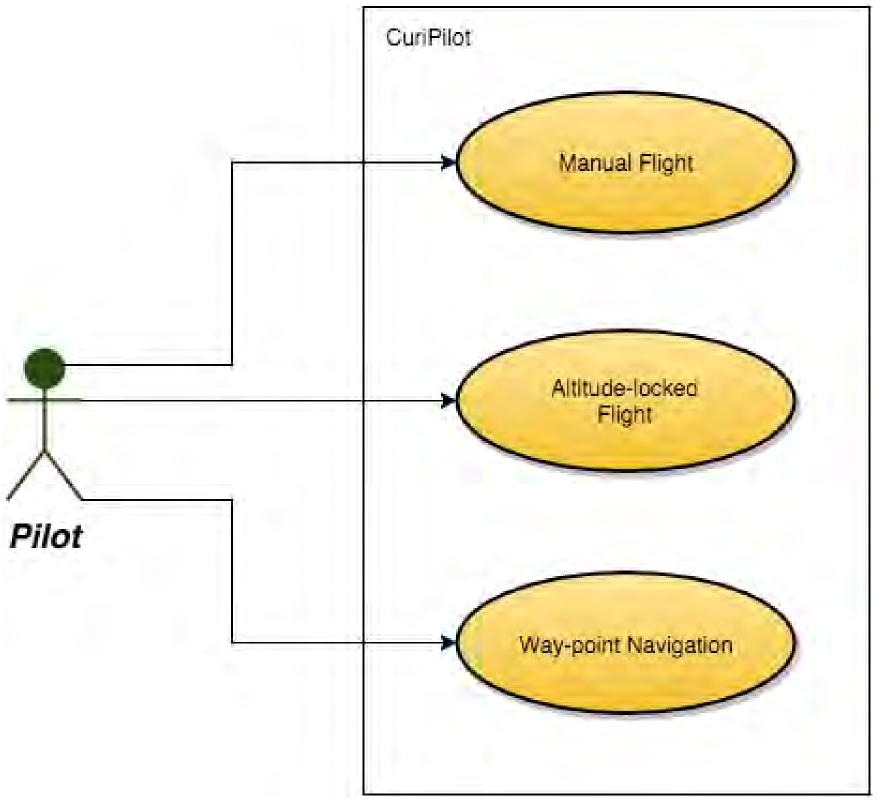


Figure 2.3: Use Cases and Actors

2.4 Software Layers

The most important part of designing an autopilot system is scalability and optimization. As more features were implemented, it seemed pointless to have to completely restructure the code each time. To avoid the negative effects of high coupling within the software system, Curipilot utilizes a two-point modular code base. The system is comprised of two layers: application and hardware. The application layer holds the logic for how the autopilot system should function. Some of these features include pitch, roll, and yaw stabilization as well as altitude lock and waypoint navigation. The hardware layer acts as a communication platform with each specific piece of hardware, such as the gyroscope and accelerometer. The goal of the hardware layer is to streamline communication with various hardware components. If one piece of hardware is changed, most of the time only one function needed to be adapted rather than re-factoring the entire code base.

The way the two layers are linked is through a read/write protocol. This protocol is the only point of communication between the two layers. The hardware layer passes up values from specific sensors needed in the application logic, and the application layer passes down values that need to be altered on the hardware. The reason for a specific I/O protocol is efficiency and optimization. By having one pipe where all the communication is funneled through, each I/O command can be prioritized; therefore, increasing the response time of the system. If the two layers were to communicate at-will, every execution would prioritize itself and could lead to a possible deadlock or run-time failure, which is unacceptable. In summary, by utilizing modularity in the code base, new features are easily implemented; thereby, altering the existing functionality. Moreover, by enforcing a read/write protocol, efficiency is maximized with little to no room for run-time failures. Table 2.2 shows a condensed overview of the control layers of the Curie.

Table 2.2: The considerations taken into account when implementing software layers

Control Layer	Purpose (Controls the...)	Design Considerations
Application	1. Pitch and roll stabilization loop 2. Waypoint navigation logic	1. Drones need a base loop to ensure constant stability 2. The logic for navigation will be done in this layer using the Intel Edison
I/O (Read and Write)	1. Communication between layers 2. Operating system uses the protocol 3. Processor interrupt loop	1. There is no streamlined way for layers to communicate efficiently, so a new protocol has been designed
Hardware	1. Way to communicate with the hardware 2. Signal provided to each motor	1. The sensors used in the system may be switched out at any time. Need to be able to exchange hardware without rewriting all of the code

2.5 Technologies Used

The technologies used are a combination of coding languages and web frameworks that have helped us to create a stable yet flexible system. Table 2.3 shows the technology and in what manner that technology was used.

2.6 Initial Implementation Methods

Before immediately designing the autopilot system, a systematic method was put together that allows the team to take the appropriate, logical, and sequential steps needed to successfully build the autopilot.

The first step involved making sure that the Curie was able to operate as its specifications suggest. The Curie will be dealing with the internal control system of the drone, including the roll, pitch, and yaw stabilization. For the safety of the people around the drone, this control system must function without any lag from the processor. Balancing the drone properly requires the Curie to consistently send signals to the motors, adjusting each one's RPM as desired. This has been one of the main steps in the project's

Table 2.3: Technologies Used

Item	Specifications/Use
Intel Curie, Intel Sharkjumper, Intel Earhart	- Stabilization control and altitude lock computations - Peripheral sensors, communications, GPIO, and PWM pins
Yuneec Quadcopter	- Frame, motors, batteries, and radio controller - For future integration with the Intel Curie
Programming Languages: C and Python	- Embedded C - Used for control system implementation (running on Curie) - Python - Used for MAVLink and QGroundControl communication
Vim and Gedit	- Preferred development environments and file editors
QGroundControl	- Open source Graphical User Interface used for drone navigation
MAVLink	- Air Vehicle Link Protocol for communication to drone

timeline. Keeping the drone stable in the air, even if it is just hovering, is vital before the team moves on to maneuvering the drone through the air.

Another large item was preparing our testing setup. In the Robotics Systems Lab, CuriPilot is utilizing a large wood and plastic cage to test the drone inside. This prevents any team member or bystander from being injured during development. Safety has always been a top priority. Inside the cage, the drone is connected to a wooden mount with one degree of free motion. This degree of free motion was used to develop roll and pitch control algorithms. Once roll and pitch are fully stabilized, another mount with two degrees of freedom can then be used to test roll and pitch simultaneously as well as yaw.

In the future, once the drone is able to stabilize itself, the Edison will be hooked up to the Curie in order to control waypoint navigation, a more computation-intensive system. The Edison will also be responsible for receiving commands from the ground operator's remote control. This step is likely to be the most complex step in the design process. The Edison will have to receive signals from an RC controller and the Curie at the same time. The Edison will also need to work with the Curie microcontroller in order to autonomously direct the drone on a user-selected path when necessary.

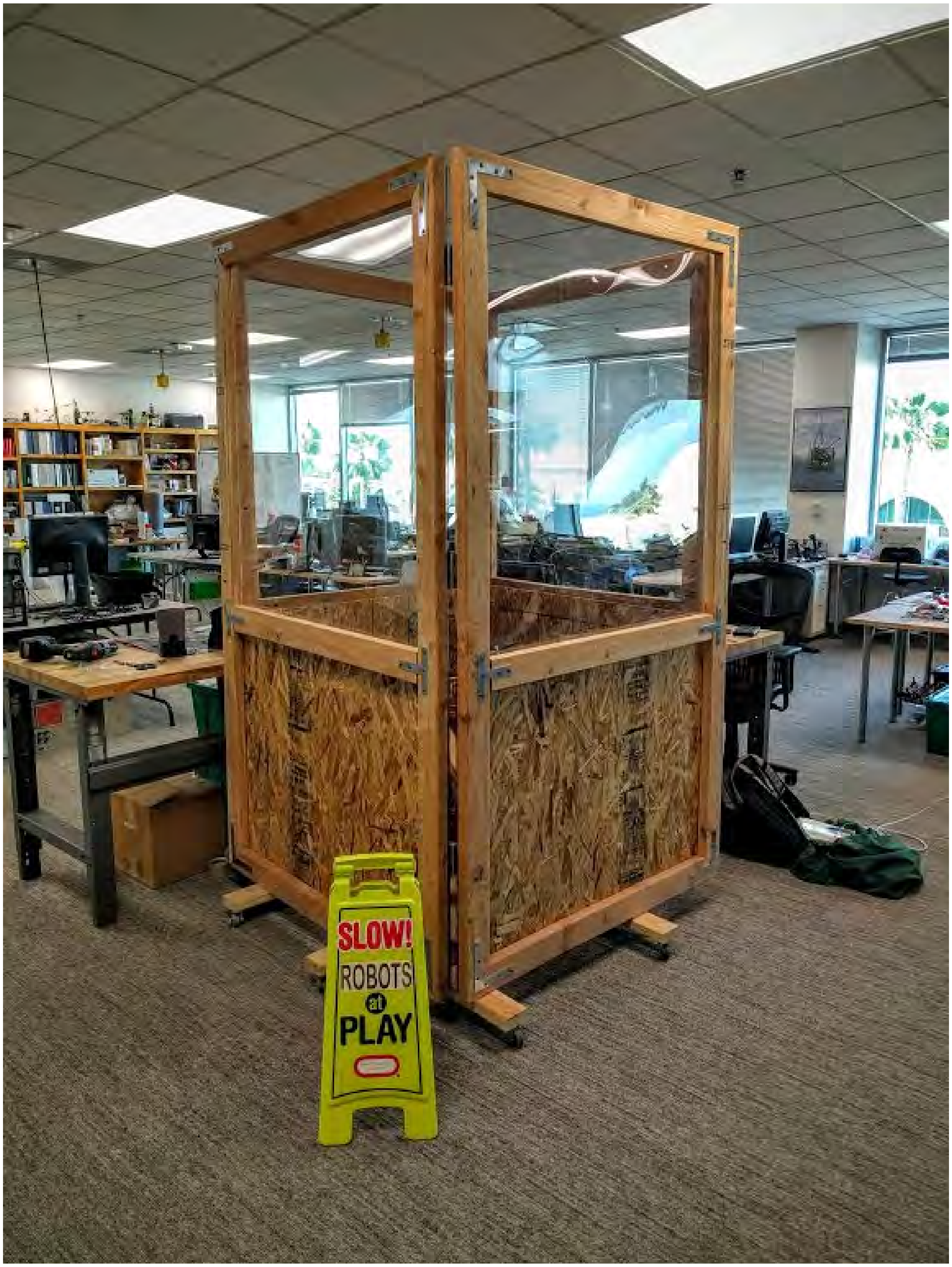


Figure 2.4: Drone Development Cage

2.7 Risk Analysis

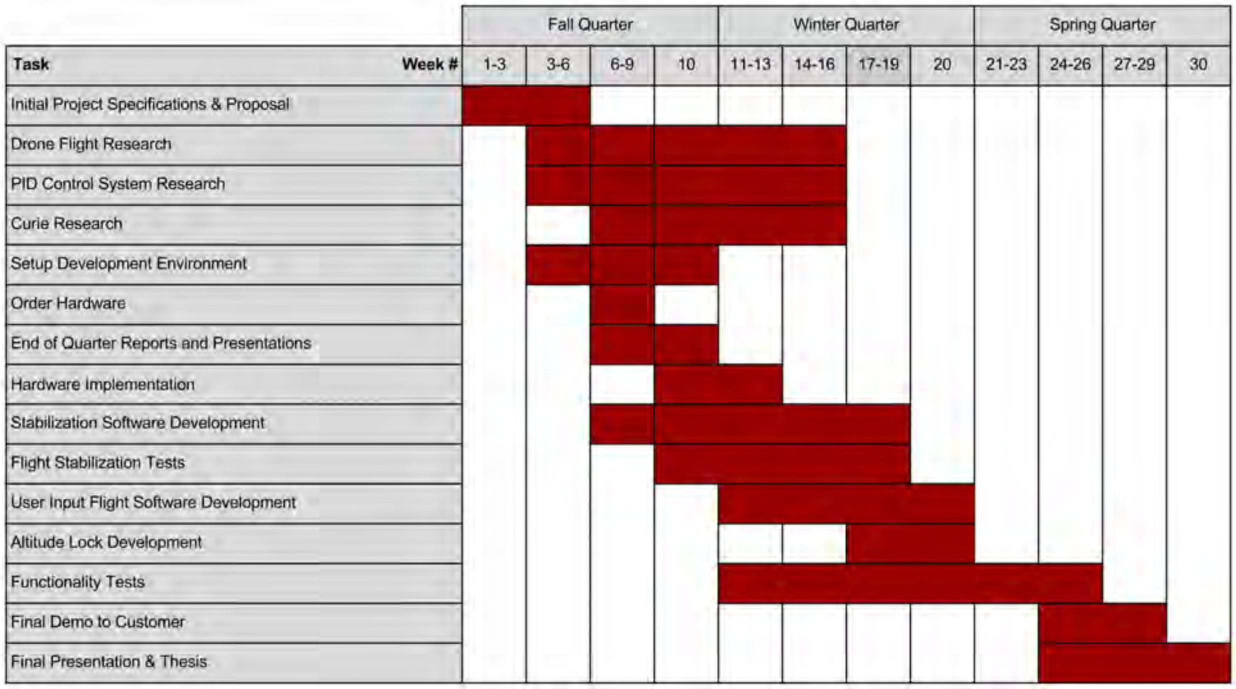
During the CuriPilot design stage, there were certain risks that needed to be addressed during the development process. The following table, Table 2.7, lays out the five most important risks that our team faced during the course of the project. Each risk has an associated consequence, probability of occurrence, severity of risk, the impact (probability*severity) the event will have on the development process, and two mitigation strategies. The mitigation strategies describe possible steps to take in order to prevent the risk from happening and/or steps to follow in order to mitigate the fallout if the risk occurs.

Table 2.4: Risk Analysis Table with Risk Mitigation Strategies

Risk	Consequence	Probability	Severity	Impact	Mitigation Strategy
Drone Legislation Issues	Permanently grounded and unable to have frequent test flights	1.0	4	4.0	1. Stayed updated on current legislation 2. Planned ahead for indoor flight tests
Running out of Development Time	Forced to work long hours to finish deliverable on time	0.4	8	3.2	1. Spread work out 2. Followed development timeline as closely as possible
Lack of Technical Ability	Extra time spent learning how to implement a feature	0.75	4	3.0	1. Tried to identify necessary technologies early on 2. Used technologies team is already familiar with
Requirements Change/ Miscommunication	Possible wrong implementation and lost time	0.2	8	1.6	1. Remained in contact with customer on what they wanted 2. Tried to finish deliverables ahead of schedule to get feedback early
Drone Related Injury	Medical aid required, possible legal action, loss of flight privileges	0.05	10	0.5	1. Followed established safety protocols 2. Never flew in unapproved surroundings

2.8 Development Timeline

The development timeline below was designed to provide a structure for the team's development process. Figure 2.5 represents the development timeline for the thirty week school year. The CuriPilot team was unable to follow this timeline throughout the entire year. Specifically, development halted during December and January due to broken hardware. At this point, the team was forced to reevaluate goals which altered the last half of the development timeline.



	Fall Quarter				Winter Quarter				Spring Quarter			
Task **Week #**	1-3	3-6	6-9	10	11-13	14-16	17-19	20	21-23	24-26	27-29	30
Initial Project Specifications & Proposal	🟥	🟥										
Drone Flight Research		🟥	🟥	🟥	🟥	🟥						
PID Control System Research		🟥	🟥	🟥	🟥	🟥						
Curie Research			🟥	🟥	🟥	🟥						
Setup Development Environment		🟥	🟥	🟥								
Order Hardware			🟥									
End of Quarter Reports and Presentations			🟥	🟥								
Hardware Implementation				🟥	🟥							
Stabilization Software Development			🟥	🟥	🟥	🟥	🟥					
Flight Stabilization Tests				🟥	🟥	🟥	🟥					
User Input Flight Software Development					🟥	🟥	🟥	🟥				
Altitude Lock Development							🟥	🟥				
Functionality Tests					🟥	🟥	🟥	🟥	🟥	🟥		
Final Demo to Customer										🟥	🟥	
Final Presentation & Thesis										🟥	🟥	🟥

Figure 2.5: Year-long Development Timeline.

Chapter 3

Roadblocks and Earhart Development

3.1 Initial Roadblocks

The Curie microprocessor is still in development which provided a very unique set of challenges. Many of the challenges we ran into were at the hardware layer of the Curie control layer scheme. When we initially started this project, our code base was about half a year old. In a development cycle for upbringing a new microprocessor architecture such as the Curie, half of a year can seem more like a lifetime. After a few weeks of development, the board broke and was rendered useless.

There could be several causes as to why the board ceased functioning, but we, along with Intel engineers, never found out the true reason. The leading theory is a memory management issue within the operating system that resulted in corrupt memory which blocked the flashing process.

3.1.1 Dealing With a Bricked Board

Instead of halting development, the bricked board prompted a restructuring of the scope of the project. The CuriPilot team increased the breadth of the project and invested development time into other areas rather than purely stabilization. The team began development on a navigation system by using MAVLink to send position data to the drone as well as development on an RC controller. Integrating angle control into the stabilization algorithm was another goal.

The bricked board provided a unique opportunity; CuriPilot was in a position to provide Intel with the requirements and design constraints of a Curie breakout board specific for drone flight. The team outlined the base requirements required to fly a drone, as well as a feature set which would aid the development process. Intel had already fabricated a second revision of the Curie board for another endeavor named Sharkjumper. Unfortunately, Sharkjumper was not ideal for controlling electronic speed controllers since it had only two USB ports rather than the desired four PWM pins. However, Shakjumper did have an onboard GPS module which allowed for preliminary development on GPS waypoint navigation.

3.1.2 Minimum Requirements for Drone-Specific Board

The CuriPilot team, after bricking the last board, sat down with Intel and laid out the components and features needed on a new Curie breakout board in order to properly fly a drone. Table 3.1 shows the agreed upon parts that were to be on the next board revision.

Table 3.1: Desired Components

Component/Features Needed
- Updated Software
- Accelerometer/Gyroscope/Barometer (on Curie)
- Minimum of 4 PWM output pins for rotors
- UART (Universal Asynchronous Receiver/Transmitter)/USB debugging capabilities
- Wireless communication capability
- 4 additional PWM pins
- 4 analog to digital converter (ADC) pins for adjusting gains in real-time
- Barometer (Altitude-lock)
- On-board rechargeable battery
- GPS module for waypoint navigation

3.2 Curie Breakout Board Revision 3: Earhart

After providing multiple rounds of feedback to Intel, we were provided a new board named Earhart. It met all of our requirements and feature requests except for the ADC pins and additional PWM pins. After flashing a bootloader onto our three Earhart boards, it was time to port the code. The code from the original breakout board had many deprecated methods as well as a different architecture than the code that the Earhart utilized. This required a large amount of collaboration between the CuriPilot team and Intel in order to ensure the code still functioned in the manner desired. After the port, the way sensor data was read had to be altered due to the different orientation of the sensors on the board. At the point of testing the PWM signal output to the electronic speed controllers, it was discovered that only three of the four PWM pins were configured for output. After restructuring the output of the final PWM pin, all of the motors were spinning. After this process, the team was able to resume stabilization development.

3.3 Finalized Earhart Board

Below, in Figure 3.1, is the finalized Earhart board. This is a board powered by the Curie processor specifically for drone flight. Special features to note are the board has an onboard XBee 900Mhz radio for preliminary communication systems development and a small rechargeable battery just for the Curie module.

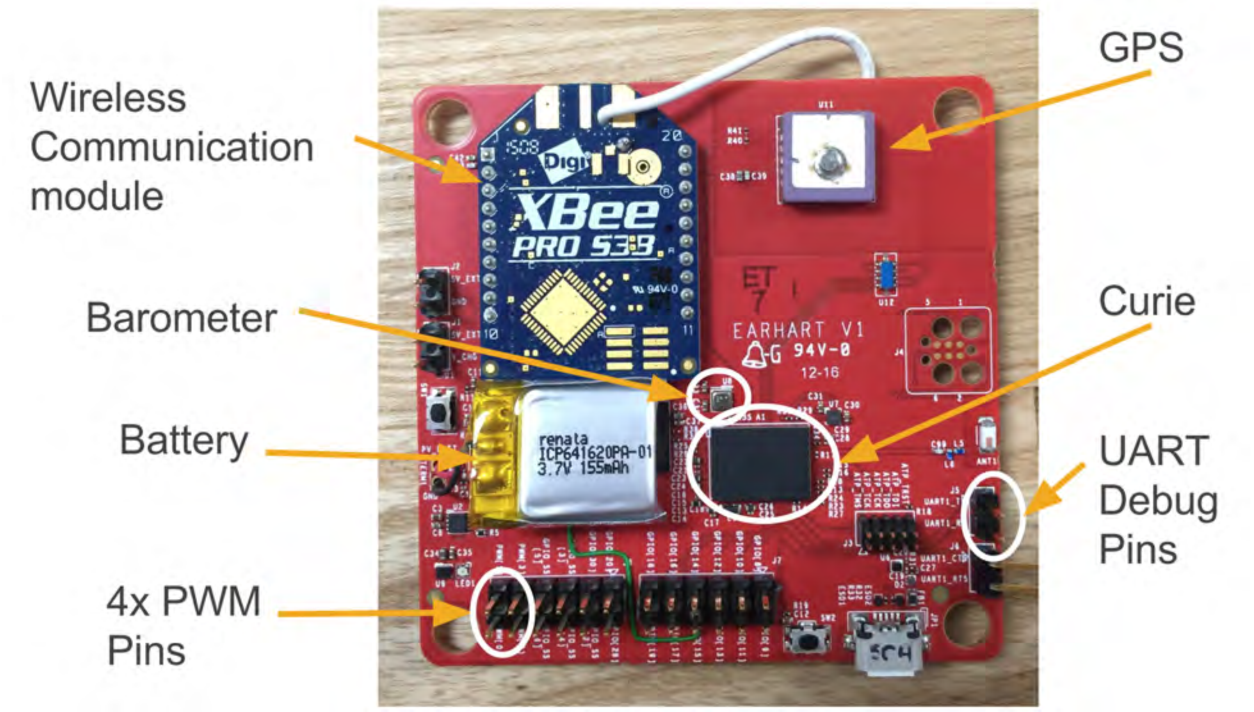


Figure 3.1: Curie Breakout Board v3 (Earhart) with labels identifying various components necessary for drone stabilization, navigation, and user-control

Chapter 4

Drone Stabilization

4.1 Real-Time Data Acquisition

For quadrotor stabilization, the Curipilot team dealt mostly at the application layer since the stabilization algorithms were written in the software. There were two main sensors needed for stabilization. First, a gyroscope is used in order to control the rate at which the drone rotates or turns. The second is the accelerometer which is used to determine the position of the drone in space. As for altitude lock control, the main sensor needed is the barometer, which measures the altitude of the drone with respect to sea level. In order to fully implement waypoint navigation, a drone needs the aid of a magnetometer and a GPS for routing assistance. The above sensors will be discussed in more depth in the following subsections.

4.1.1 Gyroscope

As mentioned above, the gyroscope is used to measure the rate at which the drone is rotating. This is very helpful when calculating the current motion of the quadcopter for stabilization. However, gyroscope data tends to drift away from its initial correct values due to the nature of how it functions. This means that it needs to be offset periodically to ensure it is generating useful data. Also, a gyroscope does not measure angles, so an accelerometer must be used if one intends to implement any sort of angle detection or angle control algorithm.

4.1.2 Accelerometer

In order to measure the angle the drone made with the surrounding axes, the accelerometer was calibrated in order to find what sensor values corresponded to -1G and 1G respectively. All units were converted to Gs in order to maintain a consistent and easy to understand measurement unit. For example, in order to calculate the angle the drone makes with the Z-axis, as seen in in Figure 4.1, the Y-force and Z-force sensor values are read and converted into Gs. The equations to find the Y and Z forces are as follows:

$$yG = \frac{2(accel_Y - min_Y)}{max_Y - min_Y} - 1 \tag{4.1}$$

$$zG = \frac{2(accel_Z - min_Z)}{max_Z - min_Z} - 1 \tag{4.2}$$

Note that the values $accel_Y$ and $accel_Z$ are the current sensor values from the accelerometer. max_Y, min_Y, max_Z, and min_Z are the calibrated values that correspond to 1G and -1G respectively. Linear interpolation can then be used to estimate the intermediate G values using equations 4.1 and 4.2. The angle the quadcopter makes with the Z-axis can then be calculated using equation 4.3.

$$\theta = \arctan\left(\frac{1 - yG}{1 - zG}\right)\frac{180}{\pi} \tag{4.3}$$

Unfortunately, the current state of the Curie's architecture does not support the arctan() function. As a result, the CuriPilot team is looking into a workaround by using a look-up table or including only the necessary source code for the inverse tangent function from the common 'math.h' library. This issue has also been brought to the attention of Intel for future Curie development.

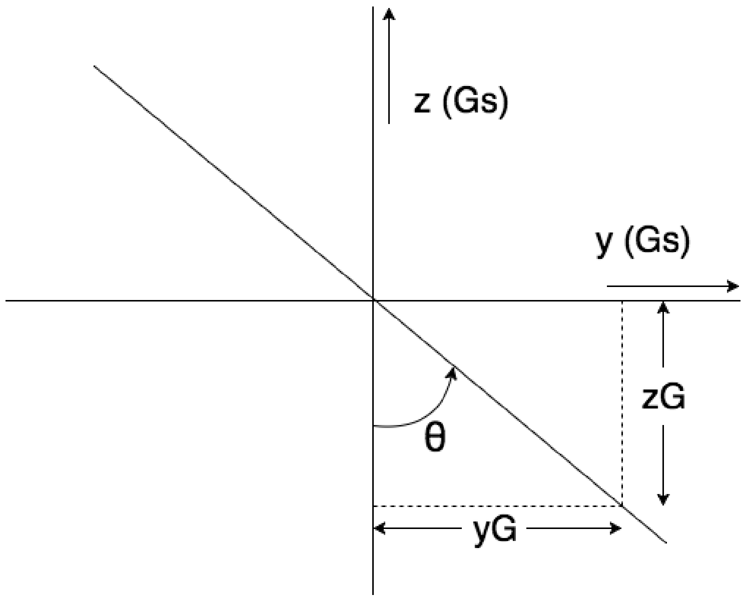


Figure 4.1: The figure above shows a visual of what is needed to find the position of the quadcopter with respect to the Z axis.

4.1.3 Barometer

The barometer on the Curie measures the atmospheric pressure of the surrounding area. This pressure can estimate the altitude of the drone with respect to sea level. Based on the elevation of the ground controller, the barometer can be offset to provide the altitude of the drone with respect to the ground. As a result, the Earhart can use the available sensor data to implement altitude-locked flight. Currently, the Earhart board can read and output barometer data. In the future, the Curie will need to take that data and feed it into a control loop which will lock the drone at a specified altitude.

4.1.4 Magnetometer

While the CuriPilot team has not made an effort to implement magnetometer readings into the drone's functionality, it will be used for routing assistance in the future. The magnetometer, along with the GPS, will help the waypoint navigation system in directing the drone in an accurate and efficient manner.

4.1.5 GPS

The GPS is a necessary component to waypoint navigation. By selecting specific points on a map, the Curie can take the GPS values to navigate the drone to the proper GPS coordinates. GPS integration into the waypoint control will be discussed further in Chapter 5.

4.2 Stabilization Methods

In order to maintain a stable system, both the roll (left and right rotation), pitch (forward and backward rotation), and yaw (left and right spin) of the system need to be controlled within safe operational bounds.

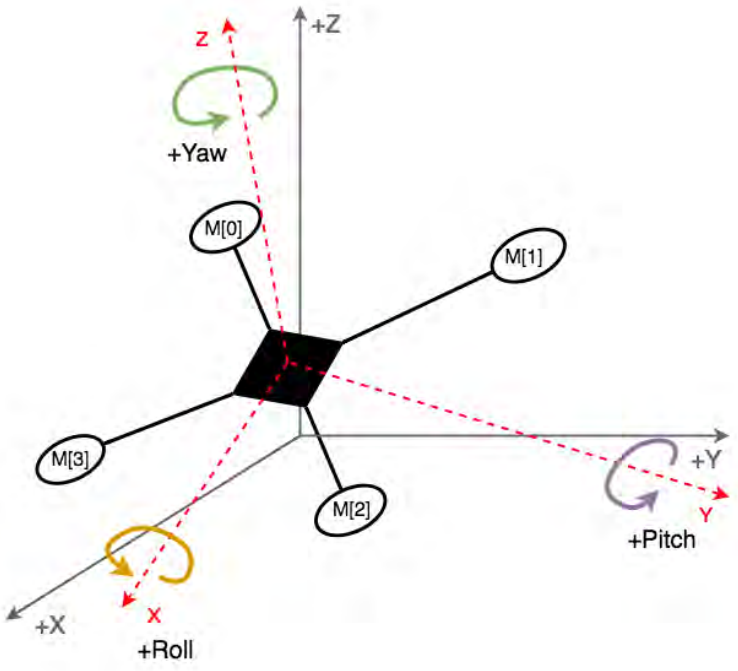


Figure 4.2: The three degrees of freedom (roll, pitch, yaw) with respect to the Cartesian Coordinate System

4.2.1 PID Control Overview

Figure 4.3 below outlines a simple implementation of a PID control loop. PID stands for Proportional Integral Derivative. PID is one of the easiest types of control to implement and is ideal for real-time systems such as a quadcopter. PID control is also pretty easy to tune. Tuning will be discussed in more detail in Section 4.2.3.

The basics of PID are pretty simple. There is a desired setpoint in the algorithm, determined by the developer. This desired setpoint is how the system should behave. In the case of a quadcopter, this can either be a desired rate of rotation around an axis or a desired angle between the quadcopter and the ground. In the average case, the desired rate of rotation is zero, and the desired angle with the ground is 180, or parallel.

Sensors are then used to calculate an error factor. The error tells the system how far from the desired setpoint it currently is. This error is manipulated and then fed back into the system via a negative feedback loop to correct the error factor. Three equations make up this feedback. The first is the P equation:

$$(Proportional\_Gain) * (Error\_Value(t)) \tag{4.4}$$

This is the Proportional or Primary gain of the system. This factor determines if the system will have enough power to correct itself. The second equation is the integral equation:

$$(Integral\_Gain) * \sum_{t=0}^{T}(Error\_Value(t)) \tag{4.5}$$

This is a summation of all the previous errors scaled by another gain factor. This smooths out the response of the system over long periods of time and ensures that the system will settle with minimal steady state error. The third equation is the derivative equation:

$$(Derivative\_Gain) * (\frac{dError\_Value(t)}{dt}) \tag{4.6}$$

This looks at how fast the data is changing and helps smooth it out in the present and near future. This helps limit the effects of sudden transient inputs. All three of these factors are then added up and sent back into the system to correct the error.

4.2.2 Rate Control and Angle Control

The CuriPilot team decided to use a PID control loop to stabilize the system. The process began with rate control; however, the stabilization results were not as positive as expected. The drone would tilt to one side and then remain there occasionally. This is due to the gyroscope only picking up data when the quadcopter is in motion which can result in incorrect orientations. Angle control would fix this by looking at the angle the quadcopter makes with the ground and correcting it back to being flat. The accelorometer data was needed for angle control. The combination of rate control and angle control resulted in a much more stable system.

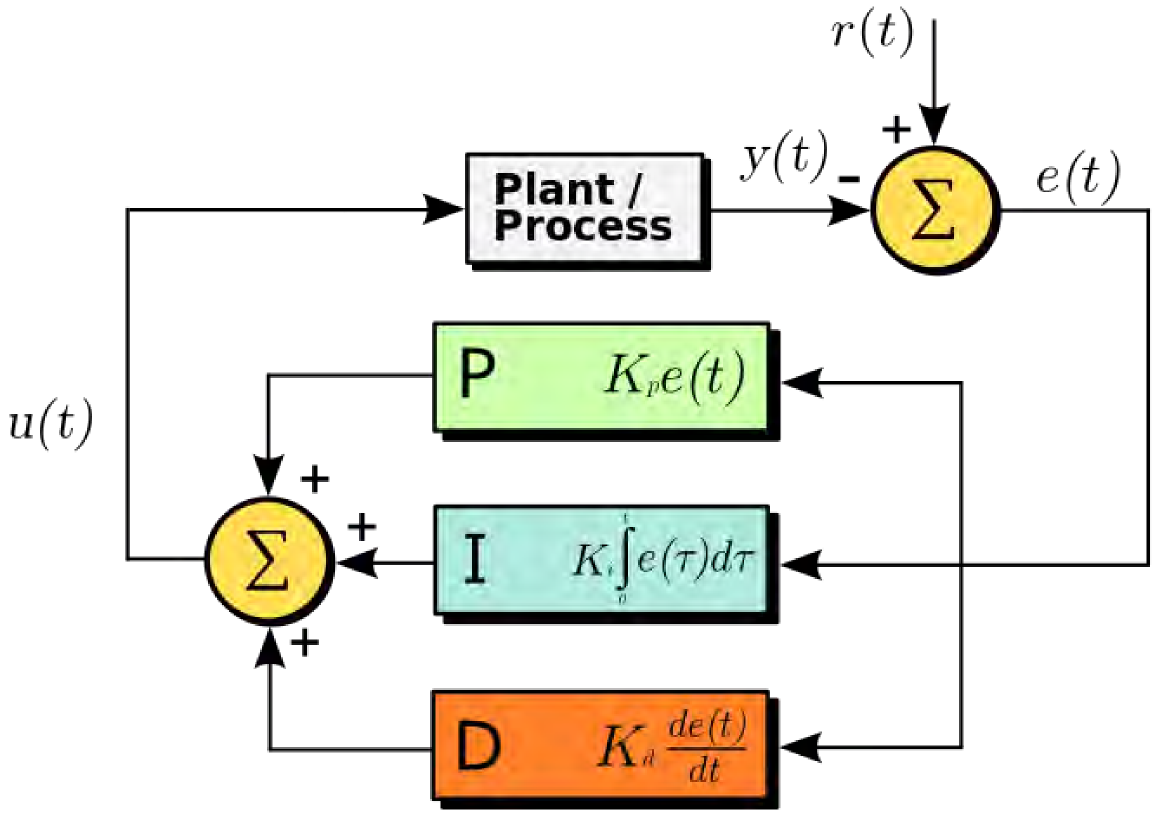


Figure 4.3: PID Control Loop. Courtesy of Wikipedia

4.2.3 Tuning

In order to stabilize the drone, the drone's gains were tuned starting with P, moving on to D, and then finishing with I. Each gain value was initially tested at a relatively low value (around 1), and then was increased based on how the drone worked to stabilize. For each test, the gain value was written down in an Excel table with notes describing what was seen in order to not repeat already tested gain values. Once the team felt the P value was set at the appropriate level, the D gain was adjusted at an increasing increments. There are nine separate gain values that need to be tested in order to fully stabilize the drone. Each degree of freedom has its own PID control. However, if the quadrotor is relatively balanced, once the gain values for roll are obtained, then the gain values for pitch will be similar.

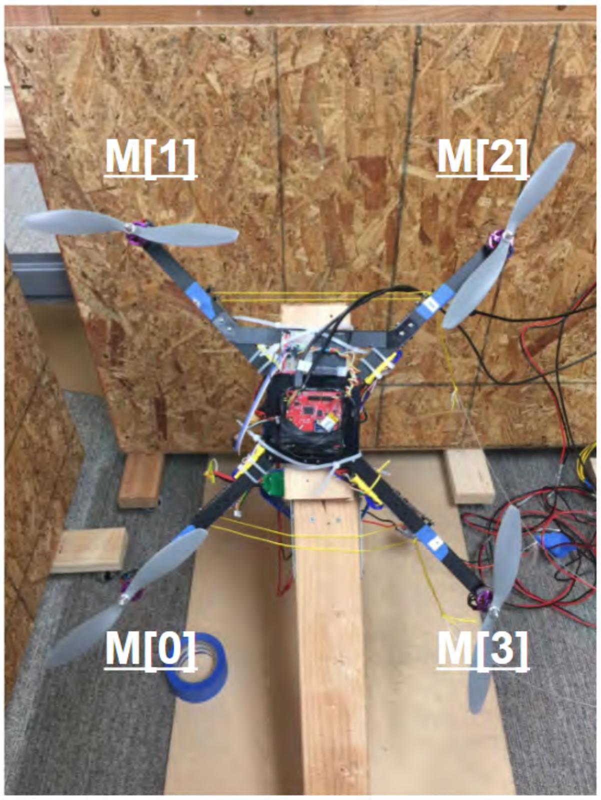


Figure 4.4: Quadcopter with motors labeled 1 through 4

4.3 Pulse Width Modulation

Pulse width modulation (PWM) is the manner in which electronic high and low pulses (digital) are used to produce a single output value (analog). The Earhart board has 4 pulse width modulation pins which were used to control the rotor speed. The rotors were able to spin at a desired speed with adjusting the PWM duty cycle between 50% and 100%. These percentages correspond to a 10ms and 20ms period respectively.

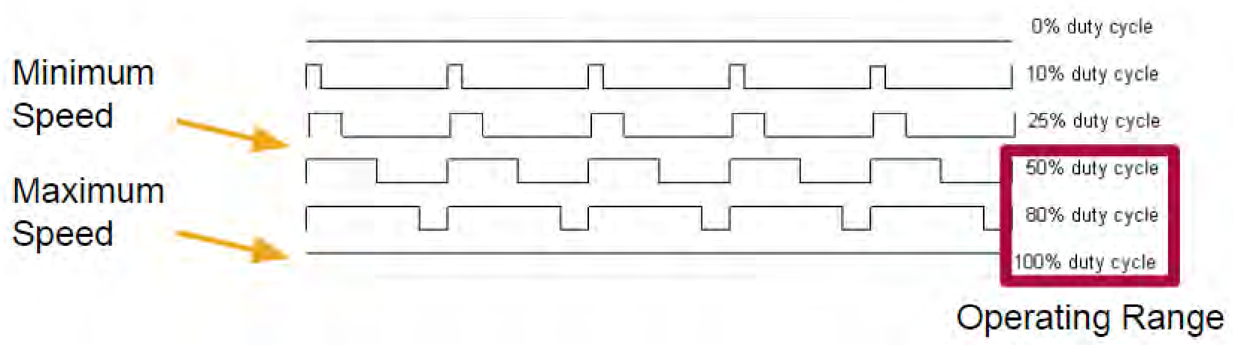


Figure 4.5: Diagram of PWM Signals

One thing to note is that the Curie performs all of its logic using 3.3V. Unfortunately, the electronic speed controllers (ESCs) require 5V logic in order to control the rotors. In order to fix this compatibility issue, a step-up converter was used with a comparator as the main chip.

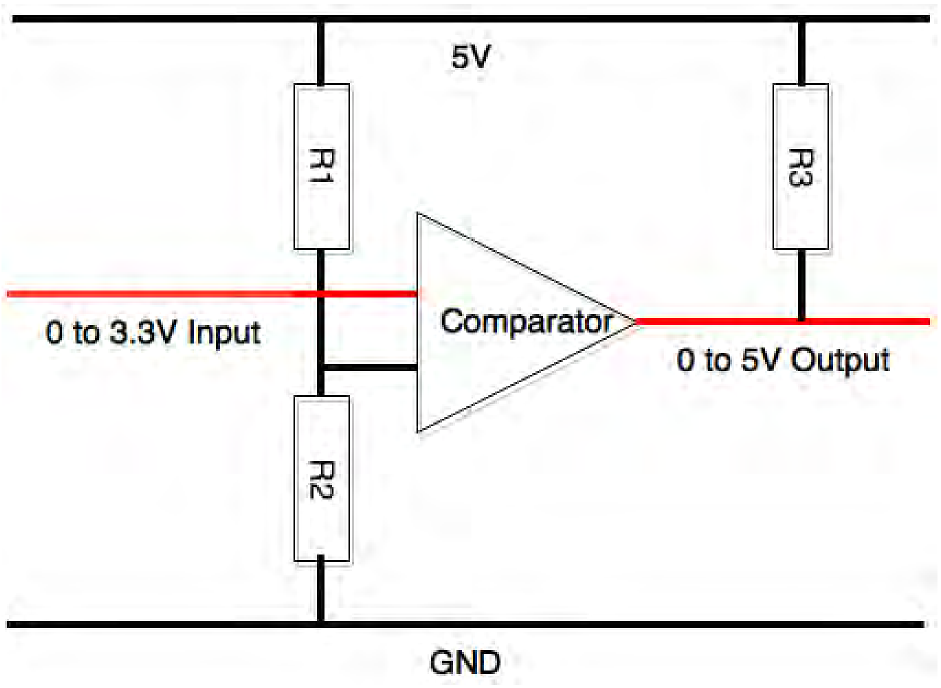


Figure 4.6: Voltage Step-up Circuit

Resistors R1 and R2 were matched to divide 5V down to 2.5V for the reference voltage. The PWM signal from the Curie, at 0-3.3V, was the other input. The comparator combined with resistor R3, the pull-up resistor, would boost the voltage up to 5V or leave it at 0V accordingly.

Chapter 5

Waypoint Navigation Development

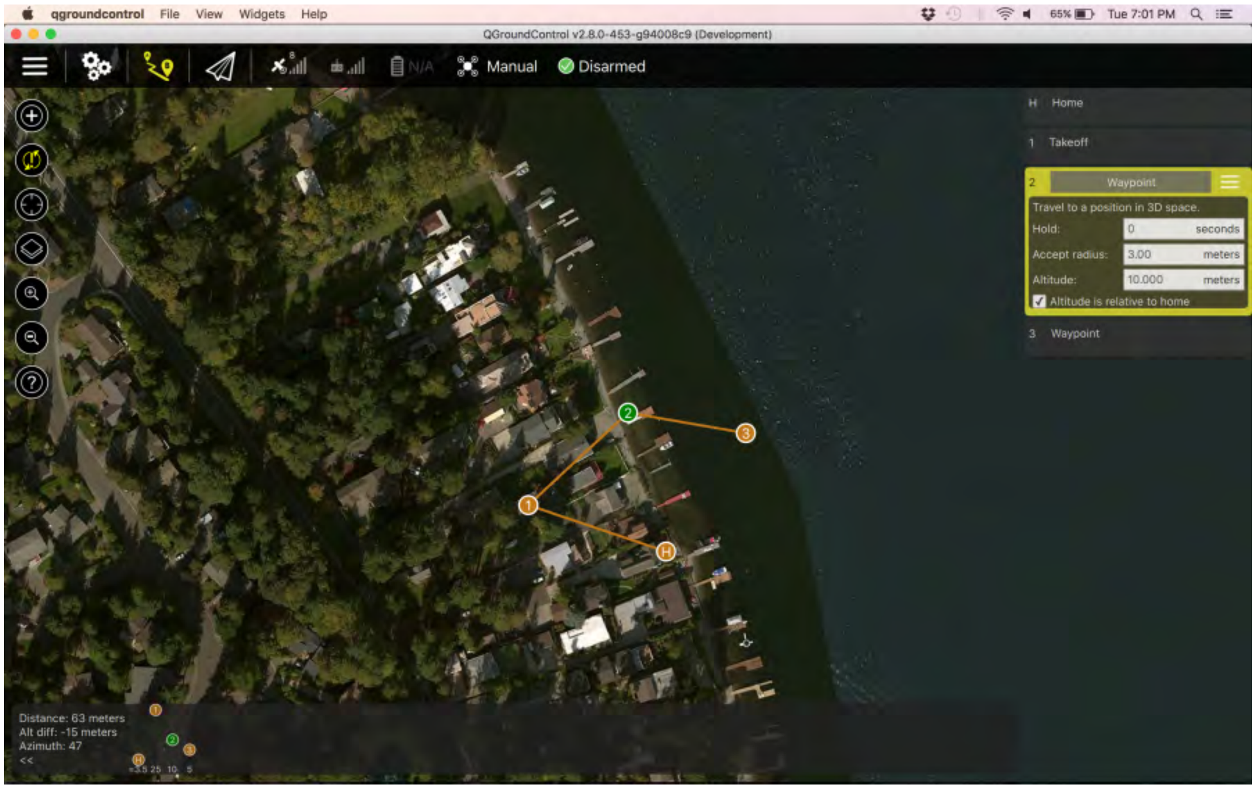
To implement a waypoint navigation system there must be stable communication to the drone, a procedure to input desired coordinates to send the drone to, and a method to manually fly the drone. To communicate with the drone we utilized MAVlink. For sending coordinates to the drone we utilized QGroundControl, and to manually control the drone we utilized radio communication. Note, that the waypoint navigation development was being developed in the application layer.

5.1 MAVLink

The first step to having a drone automatically navigate to a certain location is sending the drone the coordinates it needs to navigate to. In order to do that, a communication protocol must be implemented to establish constant communication between an interface that controls where the drone will go, and the drone itself. The protocol used is MAVLink. Micro Air Vehicle Link is an open sourced protocol used strictly for communicating with unmanned vehicles. Mavlink is to be integrated with the graphical user interface QGroundControl for sending and receiving position data. Mavlink operates at a heartbeat stabilization loop of 1 Hz or once per second. This means the protocol is expecting a message every second to ensure there is stable communication between the two ends. A Mavlink packet contains 17 bytes broken down as follows: a 6-byte header for identifying the message, a 9-byte payload containing position data, and a 2-byte checksum for verifying the data arrived in its entirety. The position data contains latitude and longitude coordinates. Unfortunately, the onboard GPS sensor on the Curie was never able to catch a signal; however, if it were to catch a signal, it would be able to compare its own coordinates with the data from the Mavlink payload to ensure it is constantly moving in the right direction. The CuriPilot team has made strides looking to integrate another GPS sensor to continue development.

5.2 QGroundControl

QGroundControl is an open source graphic user interface for setting waypoints on a map. It is compatible with Mac, Windows, and Linux. It has built in support for Mavlink and due to the fact that it is open source, we modified it to be compatible with the Curie. After realizing the GPS module on the Earhart was unable to receive a signal, the scope of our navigation work shifted. It was imperative the Curie and QGroundControl had stable communication so the CuriPilot team focused on ensuring this.



Travel to a position in 3D space.
Hold: 0 seconds
Accept radius: 3.00 meters
Altitude: 10.000 meters
☑ Altitude is relative to home

Distance: 63 meters
Alt diff: -15 meters
Azimuth: 47

Figure 5.1: The Graphical User Interface that accepts parameters such as Altitude and Position and uses MAVLink to send parameters to the UAV

To send data from a ground station to the drone over the MAVLink protocol a special packet must be constructed. Once the packet is sent to Sharkjumper, a function has to receive the data and handle the message. The code in Appendix C takes the first step of establishing communication between the two platforms. Once the data is received the message has to be handled. As stated before, MAVLink runs on a heartbeat loop. Due to this, our receiving function must also account for a disruption in stable communication. To test for stable communication in a visual respect, a boolean variable is passed in the message which lights up an LED for one second out of every three seconds. The testing proved successful as the LED lit up once every three seconds. In the end, stable communication between QGroundControl and Sharkjumper was successful, but still needs further development before implementation on the Earhart board.

5.3 Radio Communication

While the CuriPilot team began waypoint navigation development, the drone still needed a way to be controlled by a user. The team looked at two main options for controlling the drone. The first option was to use the on-board XBee, which is a 900MHz radio commonly used for wireless transmission. On the surface, the XBee is viable option since it would not require any extra hardware on the drone's frame. Unfortunately, it is not suitable for flying drones, since communication with the ground controller is easily lost. The other radio controlled that was looked into was the Spektrum AR8000. The AR8000 is a hobbyist RC controller that is widely used, well documented, and relatively easy to implement. Unfortunately, the AR8000 requires extra hardware on the drone's frame. It also requires an extra PWM input pin on the Earhart. As of our current state, the Earhart does not have an extra PWM pin. As a result, the CuriPilot team has provided feedback to Intel regarding this issue. The team hopes to get an extra PWM pin on the next revision of the Earhart board. Another option, in the meantime, is to "bit-bang" one of the available GPIO pins and turn it into a makeshift PWM pin. In other words, use software interrupts on the GPIO pin to make it act like a PWM pin. Due to the scope of the project, in-depth development has not begun on the bit-bang process. In the end, the CuriPilot team recommends using the Spektrum AR8000 as an RC controller to future teams because the extra documentation will be vital during development and trouble shooting.

Chapter 6

Test Results

When diving into the testing stage of the project, the team needed to test the control system to make sure that the drone even had an opportunity to stabilize. Furthermore, tests on the gyroscope and accelerometer were run in order to properly bias the sensor data.

6.1 Gyroscope and Accelerometer Drift and Calibration

When conducting stabilization testing, the CuriPilot team realized that the drone's rotors would not spin at the same rotational speed when the drone was not moving or rotating. After printing out the gyroscope and accelerometer readings, it was discovered that an offset was needed in order to spin the rotors at the same speed during stationary flight. In other words, when the drone was not moving, the sensor readings were nonzero. In order to find a suitable offset, five tests were run with the quadrotor locked into a horizontal position with respect to the Z axis. Both the gyroscope and accelerometer values were printed to the screen. These values were averaged over the five tests and the resulting value was subtracted from the sensor reads within the code in order to achieve a bias of 0 during stationary flight. The gyroscope and accelerometer needed to be recalibrated every couple of testing cycles in order to combat the inevitable drift known to affect these sensors. As mentioned in Section 4.1.2, the acclerometer also required extra calibration in order to calculate the drones position with respect to its surrounding axes.

6.2 Packet Rate Issues

When testing the drone's ability to stabilize, the team noticed that there was a lag between the drone rotating and the rotors adjusting their speeds to maintain stabilization. It turns out that the Curie used only 1% of the available sensor data to control the rotor speed. On the initial implementation of the Curie, Intel used it to print out data to a screen, but they only needed to print to the screen every half a second. The CuriPilot team adjusted the packet rate to fully utilize the sensor data to control the drone. The increased packet rate has had a direct impact on the drones ability to stabilize.

6.3 Kernel Panic

While testing the stabilization system, Earhart constantly went into a fail-safe method. This fail-safe mode is called a kernel panic. The Curie entered this state due to a miscommunication between the two cores of the Curie processor. To understand the issue, some background information is required. The Curie has multiple cores and each one specializes in something different. The two cores used were ARC and QUARK. The ARC core was used to interface directly with the hardware and read sensor data, as well as allow other cores to subscribe to sensor messages. The QUARK core is where most of the stabilization and waypoint code lived. This logic requires constant sensor data being passed to it from the ARC core. To initialize the motors, the electronic speed controllers require a low signal for 5 seconds. To accomplish this, a low signal is sent and a built in Curie method "task.sleep" is called for a specific amount of time. This locks the processor's current task and holds for the specified amount of time. There was a fault within this method that caused invalid parameters to be passed between the ARC and QUARK core, which resulted in a kernel panic.The "task.sleep" method should not cause a miscommunication between the cores; this error was successfully solved by replacing the faulty method with a different implementation, a semaphore lock. Most importantly, the CuriPilot team reported this fault in the code to Intel to be changed for future iterations. The kernel panic issue arose as a result of code that was written at the application level; however, it caused an error all the way down at the hardware layer, preventing the two cores to communicate properly.

Chapter 7

Professional Issues and Constraints

7.1 Ethics

Ethics and Engineering have a closer relationship than most might think. Engineering is a vast field and contains a wide variety of jobs, processes, and mindsets. Most of the time, the goal of engineering is to advance technology for the betterment of human life. This is where ethics come into play. Since engineering is bringing brand new ideas, processes, and products into the world, the effects are unknown. Something that was created with good intentions could end up causing horrible unseen consequences. Because of this, ethics should always be discussed and analyzed when any type of engineering takes place.

As the consumer drone market begins to expand rapidly, the question of ethics surrounding drones has forced its way to the forefront of ethical issues within the United States. As a result, the CuriPilot team has looked into how our project is going to deal with all of the ethical concerns that come with any drone project.

We divided the ethical breakdown into three main areas. First, we have the team and organizational ethics, which is more general and deals with the ethics within our own team structure. The second area deals with social ethics which will serve as the meat of the ethical discussion and will talk about various issues that come up with commercial drone technologies. Finally, we looked at our product development ethics, which looks at how our design process works and how we make sure our research is ethically sound. By breaking down the ethical issues that may be faced within our project, we have ensured that as many circumstances that relate to ethics are acknowledged.

7.1.1 Ethical Justification for CuriPilot

While CuriPilot is not energizing a small town in a third-world country, it is paving the way for drone powered automation. Automated drones can provide a number of benefits including 3D structure modeling and surveying buildings in need maintenance. With the growing demand for automated services, autonomous drones can help pave the way to saving companies money and keeping people safe. However, there are still some ethical issues that need to be addressed.

7.1.2 Team and Organizational Ethics

The team behind any engineering project should be organized and have a shared ethical code of conduct. Sometimes this is forgotten and the end product becomes the focal point of an ethics analysis. CuriPilot is no exception to this and has many organizational components that should be looked at.

First, we needed to act ethically in relation to each other as teammates, our university, and the city of San Jose. As developers, teammates, and friends, we had the responsibility of creating a successful product and keeping a professional development environment. When issues arose, we stepped back and discussed them. We then decided the best course of action. While we never ran into the following issue, we had decided that if we could not solve an argument within ourselves, we would immediately go to our faculty adviser to help us solve the problem. The end product was very important; however, our integrity and relationships with our teammates was important as well. In other words, the students working on CuriPilot needed to work together and communicate about each others individual work in order to avoid situations which could have proven costly to successful completion of the project.

One of the main ethical issues we could have run into as a group is if one or more of the group members had slacked off or lied about the work that he was doing as an individual. Much of the project needed to be broken up into various modules which allowed us to not only test different parts of the drone separately, but also helped in debugging the system along the way. As a result, a lot of work fell into the hands of individual group members. It was possible that one member could have lied about how much work he had gotten done for the drone. If this was the case, the timeline for the implementation of the drone system could be delayed dramatically. In order to avoid this situation, our team had put together a weekly plan that required each team member to demo his own part of the project he worked on that past week. By implementing a system that required team members to demonstrate their progress for the week, we ensured that all members were acting ethically in relation to their teammates.

Furthermore, our project had the possibility of injuring people, and we rarely messed around and never skipped safety measures. If there was something unsafe or illegal happening, all of us decided that we would speak up. As individuals, we objected to any event that might harm someone. If our team members disagreed with each other, we either removed the dangerous hardware and leave or had the option of calling campus security to mitigate the issue.

7.1.3 Social Ethics

Since we began development on a new product, we had to analyze our ethical duty to society and the future users of our drone system. Drones are very dangerous vehicles if they are not programmed and used correctly. The rotors spin fast enough to cut deep into flesh and the drone flies at high speeds. If our system is not completely stable, the user or a bystander could be severely injured. We readjusted our scope and focused mainly on developing the software system keeping the drone stable. If our system were to fail, the drone will undoubtedly fall out of the air and crash. The following paragraph touches on the safety requirements of our system.

7.1.4 Safety Requirements of Autopilot System

The safety requirements of our control system were very strict. The drone will have to be in stable flight constantly or else it will crash. To ensure this, we developed our system around two major concepts.

The first main concept was to never overburden the Curie processor. If at any point the processor becomes overburdened, core dumps, or locks up, the drone will become unstable and crash. The Curie is not the most powerful processor and we needed to consider our resource usage when developing our stabilization system. We did this by minimizing our memory usage and relying on as few computations as possible.

The second major concept was what to do if/when the system does fail. This could be the processor freezing, memory overflowing, or the battery running low. We decided the best course of action was to simply shut off the rotors and let the drone fall. The damage from falling would be less than the drone flying around uncontrollably and possibly hitting someone with the rotors still spinning.

7.1.5 Drones in Society

Currently, drone safety is a very popular topic in the news. Drones can be used for a large variety of things. From hobbyists racing them, aerial photography, surveying, searching for survivors, and even delivering packages, drones can be extremely helpful in society. However, in an analysis of ethics, the possible bad uses of the technology must be examined as well.

First and foremost, drones can be used as weapons. They can be crashed into buildings, people, cars, and even planes. They can be modified or constructed to carry explosives and other weapons. They provide a remote way to injure people without the pilot being discovered.

Drones also are great for photography. They can be used to illegally spy on people from above, peek through windows, and record footage over fenced areas. They allow for a huge invasion of privacy. Because of these malicious uses, the U.S. government has enforced laws to prohibit these problems; however, laws will not stop people from using drones with the intent to do harm or wrongfully spy on others.

7.1.6 Operational Ethics

The operation of our system during development also has strict guidelines. First, Santa Clara University banned drone flight on campus. Second, the cities of San Jose and Santa Clara banned drone flight near airports. Third, the U.S. government implemented legislature to make it illegal to fly a drone without registering it first.

CuriPilot is severely limited due to the current legal situation of drone flight. Currently, to avoid legal and ethical issues at Santa Clara University, we only tested the drone firmly attached to a stationary testing platform with protective plastic walls surrounding it. We know that our system can cause great harm, and we proceeded as conservatively as possible to avoid legal and ethical issues.

7.2 Sustainability

7.2.1 Environmental Sustainability

The first topic of sustainability that will be examined is how the CuriPilot project affects the planet. There are many aspects of CuriPilot that affect our world and therefore need to be analyzed in depth. First, there are several resources needed to produce our product. Drones are highly tech-heavy and require many different types of metals and plastics to be manufactured. Drone frames can be made out of plastics or carbon-fiber, both of which require extensive manufacturing. The CuriPilot drone uses an Intel Curie processor and breakout board which uses a substantial amount of silicon, solder, and rare metals. The processor and board are also the products of extensive manufacturing processes. So, when looking at the environmental impact of our project, we have to look into the acquisition of the materials as well as the manufacturing processes of our components. To keep these footprints as small as possible, we need to limit ourselves to as few parts as possible. We do not want to order excess and waste these valuable resources. The last major part of our system is the lithium polymer battery, or LiPo for short. These batteries are resource intensive, expensive, and dangerous if handled incorrectly. To avoid issues wasting batteries, we need to handle them with the utmost care. They must always be charged and discharged at specified rates in protected/enclosed containers to avoid explosions and/or fires. When our products lifespan comes to an end, the materials will have to be disposed of as well. The drone frame should not be thrown away. The frame should be taken apart and all the parts either recycled or re-purposed for other drone projects. The Curie processor and breakout board should be recycled correctly as electronic goods. Lastly, the LiPo batteries should be discharged completely and taken to a specialized battery recycling center to avoid harming the earth by ending up in landfills.

7.2.2 Social Sustainability

The second issue that needs to be taken into account when looking at the sustainability of a product is the social aspect of sustainability. As the demand for drones grows across the country, companies are looking to utilize these new devices to perform tasks which were previously too difficult or expensive to perform. For example, at Santa Clara University, the Facilities Department has already expressed interest in the CuriPilot drone. Facilities currently spends tens of thousands of dollars each year in building inspection costs. By attaching a camera to a CuriPilot drone, Facilities could quickly perform building inspections without hiring people to manually inspect the building. Our project has the potential to save SCUs facilities department thousands of dollars each year.

When it comes to possible negative effects on ones health and welfare, there are a couple issues. Drones in general have been known to lose contact with the ground controller resulting in a runaway drone which could fall out of the sky and seriously hurt someone. As a result, a tether can be attached to the drone which will prevent it from losing contact with ground control and flying away. However, even drones on a tether are not free from causing harm to people. Therefore, when flying the drone, people need to make sure that there are not bystanders that could be hit by the tethered drone. Furthermore, drones have been causing headaches for the Federal Aviation Administration, also known as the FAA. A drone flying into a commercial airline turbine could result in a detrimental outcome. In fact, the FAA just released sanctions on drone hobbyists saying that they cannot fly drones within 5 miles of an airport. Once again, tethering a drone may prevent some runaway issues, but will not prevent all of them. When using the CuriPilot drone, operators must be sure to follow all FAA regulations and fly responsibly, taking all precautions to make sure other people are not in harm's way.

7.2.3 Economic Sustainability

The third aspect of sustainability analyzed is CuriPilots economic sustainability. Here we look into the financial investment put into our project and the possible financial return from CuriPilot. First, our project required a large investment upfront in order to begin development. We required the drone frame, motors, wires, breadboard, and the Curie processor. To continue development, all we required was a computer running Ubuntu 12.04. If our project were to be put into production later on, the initial investment would be about the same. One would need all the required components and a computer to program the system. This brings the question, would our system be able to create enough profit or value to customers in order to support continued use and/or purchase of CuriPilot? There are many possible sources of income for the CuriPilot system. The first is from drone manufacturers buying low-cost Curie processors for their drones and utilizing our autopilot system. If Intel decides to sell the Curie processor at a competitive price and the industry likes the processor, CuriPilot would have a strong financial future. If this is the case, in order to support the system in the future, CuriPilot would require technicians and/or software specialists to help repair and troubleshoot issues. This would require a large amount of funding to set up. To summarize, if a maintenance system was setup for CuriPilot, a large amount of financial backing would be required.

7.3 Intellectual Property

Our research and implementation also opened the door to plagiarism and copyright infringement. We looked into both and researched how to avoid breaking any laws. We also discussed the topic with our contact at Intel to further understand our legal situation. This included making sure we avoided infringing on the nondisclosure agreement (NDA) we signed in conjunction with Intel.

The research and development of a system is an area where teams can run into trouble with intellectual property. Our system is not a new idea. Stabilization systems have been implemented on many other microprocessors over the years. Source code for these projects is readily available online. However, no one has implemented these on the Curie processor. The control algorithms are the same across many implementations. They only vary from system to system depending on peripheral hardware and the architecture of the processor. We looked at and learned from other open-source implementations, but we had to develop our own for the Curie's architecture.

We signed an NDA with Intel concerning the architecture of the Curie processor, its peripheral circuit boards, and our flight system implementation. We are legally unable to speak about the technological details of our project without speaking to our contact at Intel first. To avoid any issues with this, we talk to our contact at Intel before putting any undisclosed information out in the public through this thesis and various presentations.

The biggest intellectual property conflict that could arise from the project in the future would most likely come from waypoint navigation implementation. Implementing waypoint navigation from the ground up is an arduous task; however, there are many open source programs that can be used by hobbyists. Furthermore, many other drone companies currently using waypoint navigation in their drones. So, instead of reinventing the wheel, the CuriPilot team could use some of the open source software for waypoint navigation. Unfortunately, the CuriPilot team may run into some IP issues when using and adapting the open source software. Intel cannot claim the software as their own, and the adaption of the open source code may infringe upon one of the many waypoint navigation platforms for quadrotors or fixed-wing aircraft. For our project scope, the best way to avoid this problem is to use a pre-built, open source waypoint program. This way, the CuriPilot team will not be infringing on any companys patents. Down the road, in order for Intel to design a high quality quadcopter, they will have to write their own waypoint system or buy a license to waypoint software from a well-versed company in waypoint navigation for drones.

7.4 Conclusion

In the end, there were many societal concerns that needed to be taken into account when working on a drone development project. The ethics that concern the team dynamic were important to the overall success of the project. The issues that surround drone safety and privacy violations are also a hot topic for debate. However, the good that drones can bring to people may, in fact, outweigh the negatives. Also, when in the actual development process, the CuriPilot team needed to be aware of plagiarism issues with open-source software when being used for a closed source system. Since we worked for Intel and under an NDA, we needed to make sure we didn't break our agreement with Intel. Clearly, the ethics in drone production are a primary cause for debate that could last for hours; however, the main issues are addressed above.

Chapter 8

Conclusion

8.1 Summary

The goal of the project was to begin development on an Intel Curie based quadcopter. The CuriPilot team was tasked with the preliminary research and testing of the capabilities of the Intel Curie processor with the end goal of using it to fly a drone.

In order to do this, CuriPilot focused on four major major areas of development: stabilization, GPS waypoint navigation, wireless communication integration, and board/firmware documentation and feedback. The results for stabilization include the implementation of closed loop PID rate and angle control to try and keep the quadcopter balanced. While the drone is not fully stabilized at this point in time, finer tuned gain values could result in a balanced quadrotor. The stabilization results also include the modification of sensor data flow in the Curie firmware and the filtering of said data. The results from waypoint navigation development include integration of MAVLink and QGoundControl for communication with the Sharkjumper board. As for the wireless communication system for user controlled flight, the Spektrum AR8000 RC module was chosen and the CuriPilot team began research on how to connect this to the Curie board. Lastly, the CuriPilot team gave valuable feedback to the engineering team at Intel corresponding to software and hardware issues.

There are two main contributions from CuriPilot's development. First, CuriPilot developed a strong foundation for future groups to build from. The CuriPilot team debugged numerous critical issues with the code, specified which parts would be best for future implementations, and helped produce a new Curie breakout board with Intel, the EarHart, specifically for drone use. Second, the CuriPilot team encountered several software and hardware issues with the Curie that Intel had not yet seen. Our development and debugging produced valuable information about their processor that they can formally document for their own developers later on. The CuriPilot team helped Intel take the first step in Curie controlled quadcopter flight.

8.2 Future Work

The CuriPilot team has identified some areas of the project that need to be completed before turning over a completed project to Intel.

8.2.1 Stabilization

While one of the goals for the CuriPilot team was to fully stabilize a drone, certain roadblocks in the development process prevented the team from fully stabilizing the drone. There are two main areas in the stabilization area that need to be addressed. First, full integration of angle control is necessary. This means that the arctan() function needs to be built into the code and the corresponding control algorithm using Equations 4.1, 4.2, and 4.3 needs to be written into the Curie code. Furthermore, once the angle control algorithm is fully implemented, the PID gains need to be tuned once again in order to fully finish drone stabilization.

8.2.2 User-Controlled Flight

As mentioned in Section 5.3, the CuriPilot team began looking into implementing user-controlled flight. However, the control system running on the Curie needs to be developed in order to properly fly the drone. For example, the drone needs to maintain a constant tilt when flying forward. This control system will need to take in data from the RC controller and feed the data through a set of algorithms. These algorithms will allow the drone to fly in all directions.

8.2.3 Way-Point Navigation

Due to the hardware issues with GPS module, the Earhart was not able to gather GPS data from the board itself. The CuriPilot team provided feedback to Intel to replace the on-board GPS module on the next Curie board revision which will allow for continued development of the waypoint navigation system. If another revision is not available, a future team should use a off-board GPS module and integrate it with the ARC core.

8.2.4 Curie and Edison Integration and Communication

Another goal is to integrate the Curie with the Edison to help divide up the processing power. The end goal would be to have the main control systems running on the Curie, while the Intel Edison, a more powerful processor, handles the heavier waypoint navigation computations.

8.2.5 Graphical User Interface

QGroundControl has proven to be a viable solution for mapping waypoints on an interface and sending that data to the UAV. It can be modified to support more complex MAVLink messages. Future work for the graphical user interface involves streamlining message formats as well as implementing support for more than four Waypoints.

8.2.6 Yuneec Integration

Intel purchased two Yuneec drones for the CuriPilot team. The end goal is to have the new revision of the Earhart board installed on the Yuneec drone, controlling it. The Yuneec drone integration is likely the last part of the project that needs to be completed before handing the final product over to Intel.

8.3 Lessons Learned

8.3.1 Team Member Accountability

One of the biggest lessons we learned over the course of the project was the importance of team member accountability. During the Fall quarter, our team was relatively consistent with setting up meeting times each week to work on the project and to meet with our advisor. Unfortunately, during the second quarter when our team needed to dive into development, class work began to intrude on our regular meeting times. As a result, the development processes slowed and the number of meetings began to decrease. By keeping each other accountable, we would have kept our regular meetings and improved our development progress.

8.3.2 Communication with Support Team

Another lesson we learned involved simply asking for help. Intel was constantly available to help us with any issues we ran into. However, sometimes the CuriPilot team hesitated in reaching out to Intel for help or advice. When we did reach out, developers from Intel would come out to the Robotics Systems Lab to help us with any issue. In the end, if one does not ask for help, it only delays the development process. Engineering is a team process, not to be done individually. We highly suggest that the next team not be afraid to use Intel as a resource.

8.3.3 Technical Lessons

We learned a couple lessons regarding actual software and hardware development. First, we need to make sure we have backup boards in case one of the boards gets bricked. The first Curie breakout board we used for development stopped working in January. Unfortunately, we did not have a backup board; therefore, we had to get a new one. This issue halted the development process in terms of stabilization progress dramatically. Had we had backup boards, we could continued on the desired development timeline.

The second lesson we learned regarding the actual technology has to do with the gyroscope and accelerometer. We were initially unaware that the gyroscope and accelerometer tend to drift pretty dramatically over time; therefore, requiring regular calibration. Without calibrating the sensors regularly, the drone stabilization process is much more difficult.

Finally, we have provided input to Intel regarding what is needed for the next revision of the Earhart Board. One of the most important components that was lost between the first and third revision of the Curie breakout boards was the analog-to-digital (ADC) input pins. These pins provide a way to adjust the drone's gains in real time. This will save massive amounts of time since the code does not have to be recompiled and reflashed onto the board every time a single PID value is updated. The CuriPilot team should have pushed harder on getting ADC pins on the Earhart Board since the recompile and flash technique cuts deeply into the valuable development time.

References

[1] Dolesh, Richard J. "The Drones Are Coming." Parks & Recreation 50.3 (2015): 48-53. Academic Search Complete. Web. 19 Nov 2015.

[2] Beaugex, Jim. "The Unmanned Revolution." Systems Contractor News 22.7 (2015): 70-74. Applied Science & Technology Source. Web. 31 May 2016.

[3] Koerner, Matthew R. "Drones And The Fourth Amendment: Redefining Expectations Of Privacy." Duke Law Journal 64.6 (2015): 1129-1172. Academic Search Complete. Web. 19 Nov 2015.

[4] Alba, Davey. "Behind The Curve." Popular Mechanics 191.4 (2014): 84. MAS Ultra - School Edition. Web. 31 May 2016.

[5] Burt, Jeffrey. "Qualcomm Takes Aim At The Consumer Drone Market." Eweek (2015): 1. Applied Science & Technology Source. Web. 31 May 2016.

[6] Erb, Kelly Phillips. "FAA Announces Registration Requirements & New 'Drone Tax'." Forbes.Com (2015): 6. Business Source Complete. Web. 31 May 2016.

Appendices

Appendix A

Materials and Equipment

- Curie Microprocessor and breakout board
- Yuneec Drone (x2)
- Laptop running Ubuntu 14.04 LTS
- USB cables (x4)
- Quadcopter frame
- Electronic speed controller (x4)
- 3 Cell 11.1V Lithium Polymer Battery
- Battery charger
- Charge Controller
- Rotors (x4)
- Breadboard (x2)
- Potentiometers for gain tuning (x4)
- Wires
- Jumpers
- Comparator
- Resistors
- Solder and Flux

Appendix B

Main Control Loop Code

```
/*
 * Note: This code is abridged due to NDA restrictions
 * with Intel; and therefore, will not compile.
 * However, it does provide an overview of the code
 * and logic that was used for PID control, motor
 * initialization, and debugging.
 */
int32_t KPX = 50;
int32_t KIX = 30;
int32_t KDX = 10;

int roll_rate_pid()
{
   //int32_t accel_val = dater->ay - 40;
   int32_t roll_rate_error = des_roll_rate - dater->gx + 15;
   //if(accel_val > 1) {
   // roll_rate_error = roll_rate_error + accel_val;
// }
   //else if(accel_val < -1) {
   // roll_rate_error = roll_rate_error - accel_val;
   //}

   //int32_t roll_rate_error = dater->gx - des_roll_rate + 15;

   //filter noise below
/*
   if(roll_rate_error > -2000 && roll_rate_error < 2000) {
      roll_rate_error = 0;
   }
*/
   //if(roll_rate_error > 10000) {
   // roll_rate_error = 10000;
   //}else if(roll_rate_error < -10000){
   // roll_rate_error = -10000;
   //}

   //int32_t error_deriv = roll_rate_error - rate_error_prev->prevx; didnt work for rate error prev
 rate_error_prev->prevx);
  int32_t error_deriv = roll_rate_error -rate_error_prev->prevx;

  rate_error_int->intx = roll_rate_error - rate_error_prev->prevx + rate_error_int->intx;
  rate_error_prev->prevx = roll_rate_error;
```

```
   int ret = (KPX * roll_rate_error) + (KIX * rate_error_int->intx) + (KDX * error_deriv);

   return ret;
}

//#define KPY 20
//#define KIY 0
//#define KDY 0
int32_t KPY;
int32_t KIY;
int32_t KDY;

int32_t pitch_rate_pid()
{
   int32_t pitch_rate_error = des_pitch_rate - dater->gy;
   int32_t error_deriv = pitch_rate_error - rate_error_prev->prevy;
   rate_error_int->inty = pitch_rate_error - rate_error_prev->prevy
           + rate_error_int->inty;
   rate_error_prev->prevy = pitch_rate_error;
   int64_t ret = (KPY * pitch_rate_error)
         + (KIY * rate_error_int->inty)
         + (KDY * error_deriv);
   ret = 0;
   return ret;
}

#define KPZ 20
#define KIZ 0
#define KDZ 0
int32_t yaw_rate_pid()
{
   int32_t yaw_rate_error = des_yaw_rate - dater->gz;
   int32_t error_deriv = yaw_rate_error - rate_error_prev->prevz;
   rate_error_int->intz = yaw_rate_error - rate_error_prev->prevz
           + rate_error_int->intz;
   rate_error_prev->prevz = yaw_rate_error;
   int64_t ret = (KPZ * yaw_rate_error)
         + (KIY * rate_error_int->intz)
         + (KDY * error_deriv);
   ret = 0;
   return ret;
}

void set_pwm_output(uint8_t channel, int64_t pwm_duty_cycle)
{
   struct soc_pwm_channel_config configure;
   configure.mode = PWM_MODE;
   configure.pwm_period_ns = 20000000; //20 ms period
   configure.pwm_duty_cycle_ns = pwm_duty_cycle;
   configure.pwm_enable_interrupts = false;
}

void pwm_motor_output()
{
   int64_t roll_adjust = roll_rate_pid();
   if(roll_adjust > -10 && roll_adjust < 10)
      roll_adjust = 0;
   int64_t pitch_adjust = pitch_rate_pid();
   //int64_t pitch_adjust = 0;
   int64_t yaw_adjust = yaw_rate_pid();
   //int64_t yaw_adjust = 0;
   int64_t motor_0_pwm = throttle+(roll_adjust - pitch_adjust - yaw_adjust) / 3;
   int64_t motor_1_pwm = throttle+(yaw_adjust - roll_adjust - pitch_adjust) / 3; //change roll to
       +?
   int64_t motor_2_pwm = throttle+(roll_adjust + pitch_adjust + yaw_adjust) / 3;
   int64_t motor_3_pwm = throttle+(pitch_adjust - roll_adjust - yaw_adjust) / 3; //change roll to
       +?

   if (motor_0_pwm > 2000000) motor_0_pwm = 2000000;
   if (motor_1_pwm > 2000000) motor_1_pwm = 2000000;
   if (motor_2_pwm > 2000000) motor_2_pwm = 2000000;
   if (motor_3_pwm > 2000000) motor_3_pwm = 2000000;
   if (motor_0_pwm < 1000000) motor_0_pwm = 1000000;
   if (motor_1_pwm < 1000000) motor_1_pwm = 1000000;
   if (motor_2_pwm < 1000000) motor_2_pwm = 1000000;
   if (motor_3_pwm < 1000000) motor_3_pwm = 1000000;
   pwm_0_duty_cycle = motor_0_pwm;
   pwm_1_duty_cycle = motor_1_pwm;
   pwm_2_duty_cycle = motor_2_pwm;
   pwm_3_duty_cycle = motor_3_pwm;
    /*We were able to print out PWM values here to troubleshoot and observe response time*/

   set_pwm_output(0, pwm_0_duty_cycle);  //Green
   set_pwm_output(1, (pwm_1_duty_cycle+50000)); //Yellow
   set_pwm_output(2, pwm_2_duty_cycle);  //White
   set_pwm_output(3, (pwm_3_duty_cycle+50000)); //Orange



   //set_pwm_output(0, 1500000);//GREEN
   //set_pwm_output(1, 1550000);//YELLOW
   //set_pwm_output(2, 1500000);//WHITE
   //set_pwm_output(3, 1550000);//ORANGE
   return;
}

int32_t des_roll_angle;
int32_t des_pitch_angle;
int32_t des_yaw_angle;

struct angle_integral {
   int32_t intx;
   int32_t inty;
   int32_t intz;
};
struct angle_integral *int_angle;

#define KR 1
#define KP 1
#define KY 1

/**/

struct gyro_average{
   int32_t *avg_x;
   int32_t *avg_y;
   int32_t *avg_z;
};
```

```
struct gyro_average *gyro_avg;

int32_t i;
void update_gyro_avg(int32_t x, int32_t y, int32_t z)
{
   gyro_avg->avg_x[i] = x;
   gyro_avg->avg_y[i] = y;
   gyro_avg->avg_z[i] = z;
   i++;
   if(i>4) i = 0;
}
void average_gyro_data()
{
   int64_t x = 0;
   int64_t y = 0;
   int64_t z = 0;
   for (int j=0; j<3; j++) //changed 2 to 5
   {
      x+=gyro_avg->avg_x[j];
      y+=gyro_avg->avg_y[j];
      z+=gyro_avg->avg_z[j];
   }
   dater->gx = x / 3;
   dater->gy = y / 3;
   dater->gz = z / 3;
}

void main_task(void *param)
{
   for (int a=0; a<4; a++)
   {
      struct soc_pwm_channel_config config;
      config.mode = PWM_MODE;
      config.pwm_period_ns = 20000000;
      config.pwm_duty_cycle_ns = 1000000;
      config.pwm_enable_interrupts = false;
   }

   rate_error_prev->prevx = 0;
   rate_error_prev->prevy = 0;
   rate_error_prev->prevz = 0;
   rate_error_int->intx = 0;
   rate_error_int->inty = 0;
   rate_error_int->intz = 0;

   for (int j = 0; j<5; j++)
   {
      gyro_avg->avg_x[j] = 0;
      gyro_avg->avg_y[j] = 0;
      gyro_avg->avg_z[j] = 0;
   }
   i = 0;

   dater->ax = -150;
   dater->ay = -30;
   dater->az = 1160;

   dater->gx = 0;
   dater->gy = 0;
```

```
  dater->gz = 0;

  int_angle->intx = 0;
  int_angle->inty = 0;
  int_angle->intz = 0;

  CHANGE = 0;
  des_roll_rate = 0;
  des_pitch_rate = 0;
  des_yaw_rate = 0;
  des_roll_angle = 0;
  des_pitch_angle = 0;
  des_yaw_angle = 0;
}
```

Appendix C

MAVLink Communication Code

C.0.1 Sending Data

```
mavlink_system_t mavlink_system;

mavlink_system.sysid = 119; //ID of SHKJMP
mavlink_system.compid = MAV_COMP_ID_IMU; //Component/Process ID
// Define the system type, in this case an airplane
uint8_t system_type = MAV_TYPE_QUADROTER;
uint8_t autopilot_type = MAV_AUTOPILOT_GENERIC;

uint8_t system_mode = MAV_MODE_PREFLIGHT; //Initialize Motors
uint8_t system_state = MAV_STATE_STANDBY; //Ready for action

mavlink_message_t msg; //Initiiazlize Message
uint8_t buf[MAVLINK_MAX_PACKET_LEN];

// Pack the message
mavlink_msg_heartbeat_pack(mavlink_system.sysid, mavlink_system.compid, &msg, system_type,
    autopilot_type, system_mode, custom_mode, system_state);

// Copy the message to the send buffer
uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);

//Send Data over UART
uart0_send(buf, len);
```

C.0.2 Receiving Data

```
static void communication_receive(void)
{
   mavlink_message_t msg;
   mavlink_status_t status;

   while(uart0_char_available()) //UART used for communication
   {
      uint8_t c = uart0_get_char();
      // Try to get a new message
      if(mavlink_parse_char(MAVLINK_COMM_0, c, &msg, &status)) {
         GPIO_PIN_2 = &msg;
         //MSG is bool, Pin is LED
         switch(msg.msgid)
         {
             case MAVLINK_MSG_ID_HEARTBEAT:
                 {
              // Ensure Comm
                 }
                 break;
            case MAVLINK_MSG_ID_COMMAND_LONG:
            // ACTION = TRUE
            break;
         default:
            break;
         }
      }
   }

   // Track number of packet drops
   packet_drops += status.packet_rx_drop_count;
}
```

Appendix D

Lithium Polymer Battery Safety Information

E-flite

Lithium Polymer Batteries

Thank you for purchasing E-flite Li-Po batteries featuring the proven technology of ThunderPower cells with Charge Protection Circuitry, which safely monitors the status of each cell to ensure a safe charging process. ThunderPower's match cell technology assures the consistency in performance of our E-flite batteries.

E-flite Li-Po batteries have two separate leads. ***The shorter lead with the WHITE (JST) connector attached is for charging only.*** It is the only lead protected by Charge Protection Circuitry, helping prevent costly charging errors. It is very important to note that the circuit only protects during the charge process. The second longer lead is for discharging or the power source for your airplane. The longer lead may also be supplier with a JST connector or you may need to solder a connector onto the 16-gauge wire before use.

WARNING: Please read before charging or using battery

IMPORTANT SAFETY INSTRUCTIONS AND WARNINGS

• You must read these safety instructions and warnings before using or charging your batteries.
• ***Lithium Polymer batteries are volatile***. Failure to read and follow the below instructions may result in fire, personal injury and damage to property if charged or used improperly.
• Neither Horizon Hobby, Thunder Power, or our retailers assume any liability for failures to comply with these warnings and safety guidelines.
• **By purchasing this battery, the buyer assumes all risks associated with lithium batteries. If you do not agree with these conditions, return the battery immediately before use.**

General Guidelines and Warnings

1) ***Use specific Lithium Polymer charger only. Do not use a NiCd or NiMh charger*** - Failure to do so may cause a fire, which may result in personal injury and property damage.
2) ***Never charge batteries unattended.*** When charging Li-Po batteries you should always remain in constant observation to monitor the charging process and react to potential problems that may occur.
3) Some Li-Po chargers on the market may have technical deficiencies that may cause it to charge the Li-Po batteries incorrectly or at an improper rate. It is your responsibility solely to assure the charger you purchased works properly. Always monitor the charging process to assure batteries are being charged properly. Failure to do so may result in fire.
4) ***If at any time you witness a battery starting to balloon or swell up, discontinue charging process immediately. Disconnect the battery and observe it in a safe place for approximately 15 minutes.*** Continuing to charge a battery that has begun to swell will result in fire. Likewise, never use a battery if you find it swollen or ballooned upon purchase.
5) Since delayed chemical reaction can occur, it is best to observe the battery as a safety precaution. Battery observation should occur in a safe area outside of any building or vehicle and away from any combustible material.
6) ***Wire lead shorts can cause fire!*** If you accidentally short the wires, the battery **must** be placed in a safe area for observation for approximately 15 minutes. Additionally, if a short occurs and contact is made with metal (such as rings on your hand), severe injuries may occur due to the conductibility of electric current.
7) A battery can still ignite even after 10 minutes.
8) In the event of a crash, you must remove battery for observation and place in a safe open area away from any combustible material for approximately 15 minutes.
9) If for any reason you need to cut the terminal wires, it will be necessary to cut each wire separately, ensuring the wires to not touch each other or a short may occur, potentially causing a fire.
10) To solder a connector: Remove insulating tape of Red wire and solder to positive terminal of a connector, then remove insulating tape of Black wire and solder to the negative terminal of connector. Be careful not to short the wire lead. If you accidentally cause the battery to short, place it in a safe open space and observe the battery for approximately 15 minutes. ***A battery may swell or even possibly catch fire after a short time.***
11) Never store or charge a battery pack inside your car in extreme temperatures, since extreme temperature could cause fire.

Before You Charge

1) Make a visual inspection of the pack. Look for any damaged leads, connectors, broken shrink, swelling of cells, or other irregularities. Do not use if you find any of the above issues with your pack.
2) Before installing or changing the connector, check the voltage of the pack using a digital voltmeter (not your charger). All new packs ship at approximately 3.80V per cell.

   Example 2S pack should read approximately 7.60V
   3S pack should read approximately 11.40V
3) If any damage to the pack or leads is found, or the voltage is significantly less for your pack than specified above, do not attempt to charge or fly the pack; contact E-flite (Horizon Hobby) or Thunder Power directly as soon as possible.

Charging Process

1) Never charge batteries unattended.
2) ***Charge in an isolated area, away from other flammable materials on a concrete surface outside of buildings.***
3) Let the battery cool down to ambient temperature before charging.
4) ***Do not charge batteries packs in series.*** Charge each battery pack individually. Failure to do so may result in incorrect battery recognition and charging functions. Overcharging may occur and fire may be the result. ***In order to discharge packs in series, the charged voltage of both packs must be within 0.01V for the same cell count pack***

5) ***Be sure you use the shorter lead with the JST (BEC) connector for all charging.*** This is the lead protected by the Charge Protection Circuitry referenced in the introduction. There are two sets of lead wires on this battery. The shortest one is always used exclusively for charging.
6) ***When selecting the cell count or voltage for charging purposes, select the cell count and voltage as it appears on the battery label.*** Selecting a cell count or voltage other than the one printed on the label can cause fire. As a safety precaution, please confirm the information printed on the battery is correct.
   a. Example: The label on a 2-Cell battery pack in series will read – "Charge as 2-Cell (7.4V), or may cause fire" – You must select 2-Cell for charging.
   b. Example: The label on a 3-Cell battery pack in series will read – "Charge as 3-Cell (11.1V), or may cause fire" – You must select 3-Cell for charging.
7) ***You must check the pack voltage before charging after flight.*** Do not attempt to charge any pack if open voltage per cell is less than 3.3V

Example	Do not charge a 2-cell pack if below 6.6V
	Do not charge a 3 cell pack if below 9.9V

8) ***You must select the charge rate current that does not to exceed 1C (one times the capacity of the battery, unless otherwise noted*).*** A higher setting may cause fire. The below chart is calculated at 1 x capacity of pack.

Example	860 mAh:	Charge at or below 860 mA
	1200 mAh:	Charge at or below 1.2 Amps
	1800 mAh:	Charge at or below 1.8 Amps
	2100 mAh:	Charge at or below 2.1 Amps

First Flights
We recommend 3-5C max average discharge for breaking in new packs. Also be extremely careful not to over discharge new packs (Packs should NEVER be over discharged at any time, but over discharging on the first flight will ruin the battery permanently before you are able to enjoy it. See "Caring for Battery" below).

Storage & Transportation
1) Store battery at room temperature between 40 and 80 degrees F for best results.
2) Do not expose battery pack to direct sunlight (heat) for extended periods.
3) When transporting or temporarily storing in a vehicle, temperature range should be greater than 20 degrees F but no more than 150 degrees F.
4) ***Storing battery at temperatures greater than 170 degrees F for extended periods of time (more than 2 hours) may cause damage to battery and possible fire.***

Caring for Battery
1) Charge battery with good quality Lithium Polymer charger. A poor quality charger can be dangerous (such as the MRC Super Brain 969 which is NOT a proper Lithium Polymer charger).
2) Set voltage and current correctly (failure to do so can cause fire).
3) Please check pack voltage after the first charge.

Example	2-Cell: 8.4V (8.30 to 8.44)
	3-Cell: 12.6V (12.45 to 12.66)

4) ***Do not discharge battery to a level below 3V per cell under load.*** Deep discharge below 3V per cell can deteriorate battery performance. Be sure to set your ESC for the proper cut off voltage (6.0V cut off for 2S packs, 9.0V cut off for 3S packs, etc).
5) Use caution to avoid puncture of the cell. Puncture of cells may cause fire.

Operating Temperature
Charge: 32 to 113 degrees F
Discharge: 32 to 140 degrees F
1) Let battery cool down to ambient temperature before charging.
2) During discharge and handling of batteries, do not exceed 160 degrees F.

Battery Life
Batteries that lose 20% of their capacity must be removed from service and disposed of properly.
Discharge the battery to 3V/Cell, making sure output wires are insulated, then wrap battery in a bag for disposal.

Product Warranty
Product warranty is limited to original defects in material and workmanship. Warranty does not cover collateral damage. Due to the nature and use of this product there is no term warranty. Misuse, abuse, incorrect charging, failure to comply with the above warnings and guidelines, and other inappropriate use of this product are not covered under warranty.

Horizon Service Center
Attention: E-flite Service
4105 Fieldstone Road
Champaign, IL 61822
Phone: (877) 504-0233

Robotic Systems Lab Safety Instructions
LiPo Battery Safety

Overview
This packet covers the requirements for safe handling of Lithium Polymer batteries.

Training
Watch the following videos
https://www.youtube.com/watch?v=K6hGvXxFTFA
https://www.youtube.com/watch?v=gobFcNzGG9I

Optional video
https://www.youtube.com/watch?v=ixIOEPnsgbI

Read and review the following document
http://www.horizonhobby.com/pdf/EFL-LiPoSafetyWarnings.pdf

Charging
Batteries must be charged in the designated fire cabinet.
Do not leave the batteries charging unattended or overnight.
It is important to make sure that the cells are charging evenly. **Never** charge over **4.2 volts** per cell.
Always charge at **1C**.
Make sure you select the right number of cells for the battery you are charging.
Let batteries cool down before charging.

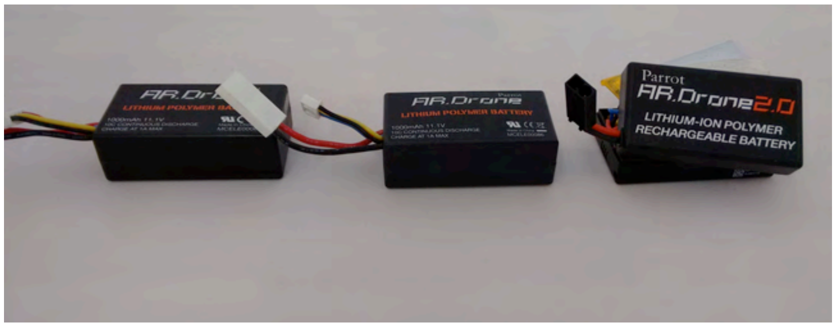
Usage
Never go below **3.3 volts** per cell.
Never discharge over its maximum C rating. Example- If you need to draw 40 amps, the smallest battery you can use is a 2000mAh battery that is rated at 20 C.

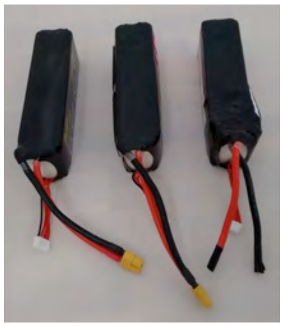
Storage
Batteries should be stored with each cell at **3.7-3.8 volts**.
Store the batteries in designated fire cabinet.

Transport
When transporting the batteries, use LiPo bags and/or metal cases

When to dispose of battery
Do not use the battery if any of the following are met:
At least one of the cells is below 3.0 volts

The battery is bloated
The battery has physical damage





In both photos, the batteries on the left appear to be in good condition; the cells still need to be checked to make sure that the battery is balanced. The middle batteries are started to bloat and should be disposed of and no longer used. The batteries on the right have physical damage and should also be disposed of.

<u>Discharge for disposal</u>
TBD- possibly EHS

<u>In case of fire</u>
TBD-If can be safely done, pour sand over batteries to smother fire or use D rated fire extinguisher

In an Emergency:
Call 4444 from the lab phone (or 408-554-4444 from a cell phone), or 911 from a cell phone.

Appendix E

Spektrum AR8000 Information

SPEKTRUM

The Leader in Spread Spectrum Technology

AR8000 User Guide
AR8000 Bedienungsanleitung
AR8000 Guide de l’utilisateur
AR8000 Guida dell’utente

AR8000 User Guide

The AR8000 full range 8-channel receiver features DSM2™ technology and is compatible with all Spektrum™ and JR® aircraft radios that support DSM2 technology including: JR12X, X9303, Spektrum DX8, DX7, DX7se, DX6i, DX5e and Module Systems.

Note: The AR8000 receiver is not compatible with the Spektrum DX6 parkflyer transmitter.

Features

- **8-channel full range receiver**
- **Patented MultiLink receiver technology**
- **Includes one internal and one remote receiver**
- **SmartSafe™ failsafe system**
- **QuickConnect™ with Brownout Detection**
- **Flight Log compatible (optional)**
- **2048 Resolution**
- **High-speed 11ms operation when used with capable transmitters**
- **Hold indicator (Red LED indicates number of holds incurred during flight).**

Applications

Full range up to 8-channel aircraft including:

- All types and sizes of glow, gas and electric powered airplanes
- All types and sizes of no-powered gliders
- All types and sizes of glow, gas and electric powered helicopters

Note: Not for use in airplanes that have full carbon fuselages. Not for use in airplanes that have significant carbon or conductive structures. It is recommended to use either the AR6250 or AR9300 receivers for carbon fuselage applications.

Specifications:

Type: DSM2 full range receiver
Channels: 8
Modulation: DSM2
Main Receiver Dimensions: 1.27 x 1.35 x 0.45 in (32.3 x 34.3 x 11.4mm)
Remote Receiver Dimensions: 0.80 x 1.10 x 0.27 in (20.3 x 28.0 x 6.9mm)
Main Receiver Weight: .33 oz (9.4 g) Weight with Remote Receiver: .49 oz (13.9 g)
Voltage range: 3.5 to 9.6V
Resolution: 2048
Frame rate: 11ms when paired with the DX7se or DX8
Compatibility: All DSM2 aircraft transmitters and module systems

Patented MultiLink™ Receiver Technology

The AR8000 incorporates dual receivers, offering the security of dual path RF redundancy. An internal receiver is located on the main printed circuit board, while a second external receiver is attached to the main board with a 6-inch extension. By locating these receivers in slightly different locations in the aircraft, each receiver is exposed to its own RF environment, greatly improving path diversity (the ability for the receiver to see the signal in all conditions).

Antenna Polarization

For optimum RF link performance it's important that the antennas be mounted in an orientation that allows for the best possible signal reception when the aircraft is in all possible attitudes and positions. This is known as antenna polarization. The antennas should be oriented perpendicular to each other; typically one vertical and one horizontal (see Receiver Installation). The remote receiver's antenna should be mounted in a position perpendicular at least 2 inches away from the main receiver's antenna using double-sided foam tape.

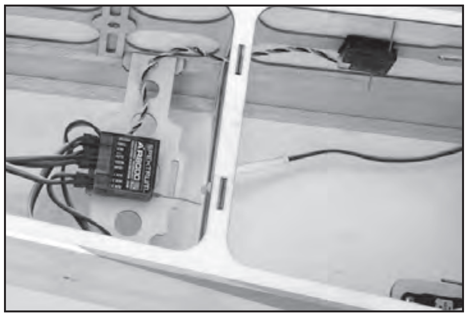
Receiver Installation

In gas and glow aircraft install the main receiver using the same method you would use to install a conventional receiver in your aircraft. Typically, wrap the main receiver in protective foam and fasten it in place using rubber bands or hook and loop straps.

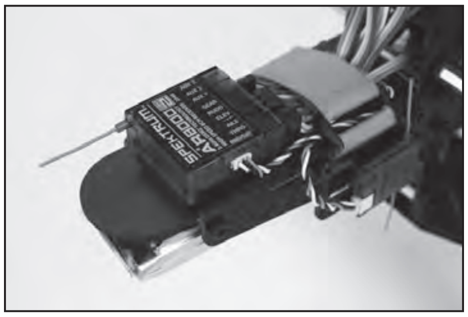
Alternately, in electric airplanes or helicopters, it's acceptable to use thick double-sided foam tape to fasten the main receiver in place.

Mounting the remote receiver in a slightly different location, even just inches away from the primary receiver, gives tremendous improvements in path diversity. Essentially, each receiver sees a different RF environment and this is key to maintaining a solid RF link, even in aircraft that have substantial conductive materials (e.g. larger gas engines, carbon fiber, pipes, etc.), which can weaken the signal.

Using servo tape, mount the remote receiver keeping the remote antennas at least 2 inches away from the primary antenna. Ideally, the antennas will be oriented perpendicularly to each other. In airplanes, we've found it best to mount the primary receiver in the center of the fuselage on the servo tray and to mount the remote receiver to the side of the fuselage or in the turtle deck.



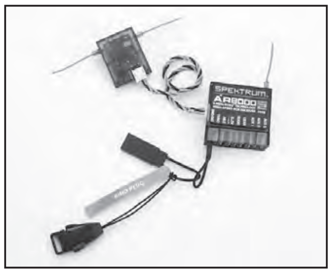
In helicopters, there is generally enough room on the servo tray to achieve the necessary separation. If necessary a mount can be fashioned using clear plastic to mount the external receiver.

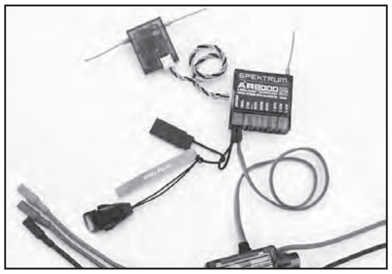


Important: Y-Harnesses and Servo Extensions
When using Y-harnesses or servo extensions it's important to use standard non-amplified Y-harnesses and servo extensions as this can/will cause the servos to operate erratically or not function at all. Amplified Y-harnesses were developed several years ago to boost the signal for some older PCM systems and should not be used with Spektrum equipment. Note that when converting existing models to Spektrum be certain that all amplified Y-harnesses and or servo extensions are replaced with conventional non-amplified versions.

Binding
The receiver must be bound to the transmitter before it will operate. Binding is the process of teaching the receiver the specific code of the transmitter so it will only connect to that specific transmitter.

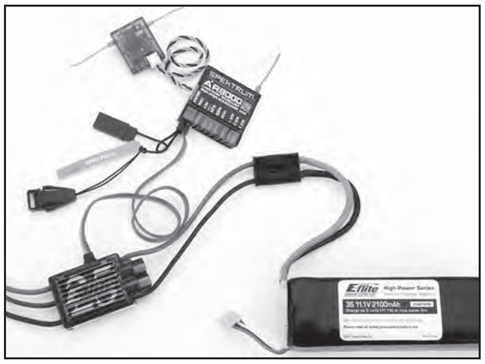
1. To bind an AR8000 to a DSM2 transmitter, insert the bind plug in the BATT/BIND port on the receiver.



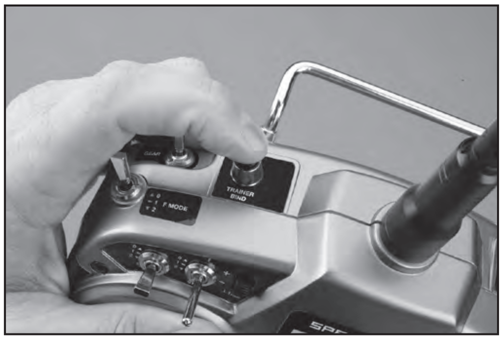


Note: To bind an aircraft with an electronic speed controller that powers the receiver through the throttle channel (ESC/BEC), insert the bind plug into the BATT/BIND port in the receiver and the throttle lead into the throttle (THRO) port. Proceed to Step #2.

2. Power the receiver. Note that the LED on the receiver should be flashing, indicating that the receiver is in bind mode and ready to be bound to the transmitter.



3. Move the sticks and switches on the transmitter to the desired failsafe positions (low throttle and neutral control positions).



4. Follow the procedures of your specific transmitter to enter Bind Mode, the system will connect within a few seconds. Once connected, the LED on the receiver will go solid indicating the system is connected.

5. Remove the bind plug from the BATT/BIND port on the receiver before you power off the transmitter and store it in a convenient place.

6. After you've set up your model, it's important to rebind the system so the true low throttle and neutral control surface positions are set.

IMPORTANT: Remove the bind plug to prevent the system from entering bind mode the next time the power is turned on.

SmartSafe™ Failsafe

The AR8000 features SmartSafe failsafe. SmartSafe is ideal for most types of aircraft. With SmartSafe, when signal is lost the throttle channel only is driven to its preset failsafe position (normally low throttle) while all other channels hold last command.

- Prevents unintentional electric motor response on start-up.
- Eliminates the possibility of over-driving servos on start-up by storing preset failsafe positions.
- Establishes low-throttle failsafe and maintains last-commanded control surface position if the RF signal is lost.

Receiver Power Only

- When the receiver only is turned on (no transmitter signal is present), the throttle channel has no output, to avoid operating or arming the electronic speed control.
- All other channels are driven to their preset failsafe positions set during binding.

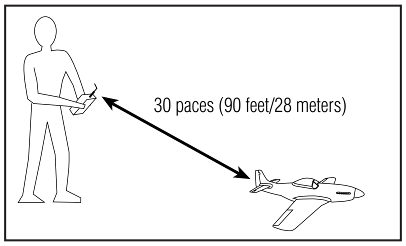
After Connection

- When the transmitter is turned on and after the receiver connects to the transmitter, normal control of all channels occurs.
- After the system makes a connection, if loss of signal occurs SmartSafe drives the throttle servo only to its preset failsafe position (low throttle) that was set during binding.
- All other channels hold their last commanded position. When the signal is regained, the system immediately (less than 4ms) regains control.

Plugging in the Leads
Plug the servo leads into the appropriate servo ports in the receiver noting the polarity of the servo connector.

Red LED Hold Indicator
The AR8000 features a red LED that indicates the number of holds that have occurred since the receiver was last powered on. The LED will flash the number of holds then pause (e.g., flash, flash, flash, pause, flash, flash, flash, pause indicates three holds occurred since the receiver was last turned on). Note that holds are reset to zero when the receiver is turned off. During the first flights of a new airplane, it's recommended to check the red LED hold indicator. If it's flashing, it's important to optimize the installation (move or reposition antennas) until no hold occurs. On later flights, the LED Hold Indicator can be used to confirm RF link performance.

Range Testing
Before each flying session and especially with a new model, it is important to perform a range check. All Spektrum aircraft transmitters incorporate a range testing system which, when activated, reduces the output power, allowing a range check.



1. With the model restrained on the ground, stand 30 paces (approx. 90 feet/28 meters) away from the model.
2. Face the model with the transmitter in your normal flying position and place your transmitter into range check mode.
3. You should have total control of the model with the button depressed at 30 paces (90 feet/28 meters).
4. If control issues exist, call the Product Support Team in the U.S. at 1-877-504-0233 for further assistance. In the UK or Germany use one of the following contacts.

European Union: +44 (0) 1279 641 097 (United Kingdom)
or email sales@horizonhobby.co.uk
+49 4121 46199 66 (Germany)
or email service@horizonhobby.de

Advanced Range Testing
For sophisticated models that have significant conductive material in them, the advanced range test using a flight log is recommended. The advanced range check will confirm that the internal and remote receivers are operating optimally and that the installation (position of the receivers) is optimized for the specific aircraft. This Advanced Range Check allows the RF performance of each receiver to be evaluated and to optimize the locations of the remote receiver.

Advanced Range Test
1 Plug a Flight Log (SPM9540 - optional) into the Batt/Data port on the AR8000 and turn on the system (Tx and Rx).
2. Advance the Flight Log until F-frame losses are displayed by pressing the button on the Flight Log.
3. Have a helper hold your aircraft while observing the Flight Log data.
4. Standing 30 paces away from the model, face the model with the transmitter in your normal flying position and put your transmitter into range test mode. This causes reduced power output from the transmitter.
5. Have your helper position the model in various orientations (nose up, nose down, nose toward the Tx, nose away from the Tx, etc.) while your helper watches the Flight Log noting any correlation between the aircraft's orientation and frame losses. Do this for 1 minute. The timer on the transmitter can be used here.

Receiver Power System Requirements
Inadequate power systems that are unable to provide the necessary minimum voltage to the receiver during flight have become the number one cause of in-flight failures. Some of the power system components that affect the ability to properly deliver adequate power include:

- Receiver battery pack (number of cells, capacity, cell type, state of charge)
- The ESC's capability to deliver current to the receiver in electric aircraft
- The switch harness, battery leads, servo leads, regulators etc.

The AR8000 has a minimum operational voltage of 3.5 volts; it is highly recommended the power system be tested per the guidelines below and in the Flight Log section.

Recommended Power System Test Guidelines
If a questionable power system is being used (e.g. small or old battery, ESC that may not have a BEC that will support high-current draw, etc.), it is recommended that a voltmeter be used to perform the following test.

Note: The Hangar 9 Digital Servo & Rx Current Meter (HAN172) or the Spektrum Flight Log (SPM9540) are the perfect tools to perform the test below.

Plug the voltmeter into an open channel port in the receiver and with the system on, load the control surfaces (apply pressure with your hand) while monitoring the voltage at the receiver. The voltage should remain above 4.8 volts even when all servos are heavily loaded.

Note: The latest generations of Nickel-Metal Hydride batteries incorporate a new chemistry mandated to be more environmentally friendly. These batteries when charged with peak detection fast chargers have tendencies to false peak (not fully charge) repeatedly. These include all brands of NiMH batteries. If using NiMH packs, be especially cautious when charging, making absolutely sure that the battery is fully charged. It is recommended to use a charger that can display total charge capacity. Note the number of mAh put into a discharged pack to verify it has been charged to full capacity.

QuickConnect With Brownout Detection
Your AR8000 features QuickConnect with Brownout Detection.

- Should an interruption of power occur (brownout), the system will reconnect immediately when power is restored (QuickConnect).
- The LED on the receiver will flash slowly indicating a power interruption (brownout) has occurred.
- Brownouts can be caused by an inadequate power supply (weak battery or regulator), a loose connector, a bad switch, an inadequate BEC when using an electronic speed controller, etc.
- Brownouts occur when the receiver voltage drops below 3.5 volts thus interrupting control as the servos and receiver require a minimum of 3.5 volts to operate.

How QuickConnect™ With Brownout Detection Works

- When the receiver voltage drops below 3.5 volts the system drops out (ceases to operate).
- When power is restored the receiver immediately attempts to reconnect to the last two frequencies that it was connected to.
- If the two frequencies are present (the transmitter was left on) the system reconnects typically within one second.

QuickConnect with Brownout Detection is designed to allow you to fly safely through most short duration power interruptions, however, the root cause of these interruptions must be corrected before the next flight to prevent catastrophic safety issues.

Note: If a brownout occurs in flight it is vital that the cause of the brownout be determined and corrected.

Flight Log (SPM9540 Optional)
The Flight Log is compatible with the AR8000. The Flight Log displays overall RF link performance as well as the individual internal and external receiver link data. Additionally it displays receiver voltage.

Using the Flight Log
After a flight and before turning off the receiver or transmitter, plug the Flight Log into the Data port on the AR8000. The screen will automatically display voltage e.g. 6v2= 6.2 volts.

Note: When the voltage reaches 4.8 volts or less, the screen will flash indicating low voltage.

Press the button to display the following information:
A - Antenna fades on the internal antenna
B – Not used
L – Antenna fades on the external antenna
R – Not used
F - Frame loss
H - Holds

Antenna fades—represents the loss of a bit of information on that specific antenna. Typically it's normal to have as many as 50 to 100 antenna fades during a flight. If any single antenna experiences over 500 fades in a single flight, the antenna should be repositioned in the aircraft to optimize the RF link.

Frame loss—represents simultaneous antenna fades on all attached receivers. If the RF link is performing optimally, frame losses per flight should be less than 20. A hold occurs when 45 consecutive frame losses occur. This takes about one second. If a hold occurs during a flight, it's important to evaluate the system, moving the antennas to different locations and/or checking to be sure the transmitter and receivers are all working correctly.

Note: A servo extension can be used to allow the Flight Log to be plugged in more conveniently. On some models, the Flight Log can be plugged in, attached and left on the model using double-sided tape. Mounting the Flight Log conveniently to the side frame is common with helicopters.

ModelMatch
Some Spektrum and JR transmitters offer a patent pending feature called ModelMatch. ModelMatch prevents the possibility of operating a model using the wrong model memory, potentially preventing a crash. With ModelMatch each model memory has its own unique code (GUID) and during the binding process the code is programmed into the receiver. Later, when the system is turned on, the receiver will only connect to the transmitter if the corresponding model memory is programmed on screen.

Note: If at any time you turn on the system and it fails to connect, check to be sure the correct model memory is selected in the transmitter. Please note that the Spektrum Aircraft Modules do not have ModelMatch.

Tips on Using Spektrum 2.4GHz

While your DSM equipped 2.4GHz system is intuitive to operate, functioning nearly identically to 72MHz systems, following are a few common questions from customers.

Q: Which do I turn on first, the transmitter or the receiver?
A: If the receiver is turned off first—all servos except for the throttle will be driven to their preset failsafe positions set during binding. At this time the throttle channel doesn't output a pulse position preventing the arming of electronic speed controllers or in the case of an engine powered aircraft the throttle servo remaining in its current position. When the transmitter is then turned on the transmitter scans the 2.4GHz band and acquires two open channels. Then the receiver that was previously bound to the transmitter scans the band and finds the GUID (Globally Unique Identifier code) stored during binding. The system then connects and operates normally.

If the transmitter is turned on first—the transmitter scans the 2.4GHz band and acquires two open channels. When the receiver is then turned on for a short period (the time it takes to connect) all servos except for the throttle are driven to their preset failsafe positions while the throttle has no output pulse. The receiver scans the 2.4GHz band looking for the previously stored GUID and when it locates the specific GUID code and confirms uncorrupted repeatable packet information, the system connects and normal operation takes place. Typically this takes 2 to 6 seconds.

Q: Sometimes the system takes longer to connect and sometimes it doesn't connect at all?
A: In order for the system to connect (after the receiver is bound) the receiver must receive a large number of consecutive uninterrupted perfect packets from the transmitter in order to connect. This process is purposely critical of the environment ensuring that it's safe to fly when the system does connect. If the transmitter is too close to the receiver (less than 4 feet) or if the transmitter is located near metal objects (metal TX case, the bed of a truck, the top of a metal work bench, etc.) connection will take longer and in some cases connection will not occur as the system is receiving reflected 2.4GHz energy from itself and is interpreting this as unfriendly noise. Moving the system away from metal objects or moving the transmitter away from the receiver and powering the system again will cause a connection to occur. This only happens during the initial connection. Once connected the system is locked in and should a loss of signal occur (failsafe) the system connects immediately (4ms) when signal is regained.

Q: I've heard that the DSM system is less tolerant of low voltage. Is this correct?
A: All DSM receivers have an operational voltage range of 3.5 to 9.6 volts. With most systems this is not a problem as in fact most servos cease to operate at around 3.8 volts. When using multiple high-current draw servos with a single or inadequate battery/power source, heavy momentary loads can cause the voltage to dip below this 3.5-volt threshold thus causing the entire system (servos and receiver) to brown out. When the voltage drops below the low voltage threshold (3.5 volts), the DSM receiver must reboot (go through the start-up process of scanning the band and finding the transmitter) and this can take several seconds. Please read the receiver power requirement section as this explains how to test for and prevent this occurrence.

Q: Sometimes my receiver loses its bind and won't connect requiring rebinding. What happens if the bind is lost in flight?
A: The receiver will never lose its bind unless it's instructed to. It's important to understand that during the binding process the receiver not only learns the GUID (code) of the transmitter but the transmitter learns and stores the type of receiver that it's bound to. If the transmitter is placed into bind mode, the transmitter looks for the binding protocol signal from a receiver. If no signal is present, the transmitter no longer has the correct information to connect to a specific receiver and in essence the transmitter has been "unbound" from the receiver. We've had several DX7 customers that use transmitter stands or trays that unknowingly depress the bind button and the system is then turned on losing the necessary information to allow the connection to take place. We've also had DX7 customers that didn't fully understand the range test process and pushed the bind button before turning on the transmitter also causing the system to "lose its bind."

Warranty Period

Exclusive Warranty- Horizon Hobby, Inc., (Horizon) warranties that the Products purchased (the "Product") will be free from defects in materials and workmanship for a period of 1 year from the date of purchase by the Purchaser.

1-Year Limited Warranty

Horizon reserves the right to change or modify this warranty without notice and disclaims all other warranties, express or implied.

(a) This warranty is limited to the original Purchaser ("Purchaser") and is not transferable. REPAIR OR REPLACEMENT AS PROVIDED UNDER THIS WARRANTY IS THE EXCLUSIVE REMEDY OF THE PURCHASER. This warranty covers only those Products purchased from an authorized Horizon dealer. Third party transactions are not covered by this warranty. Proof of purchase is required for warranty claims. Further, Horizon reserves the right to change or modify this warranty without notice and disclaims all other warranties, express or implied.

(b) Limitations- HORIZON MAKES NO WARRANTY OR REPRESENTATION, EXPRESS OR IMPLIED, ABOUT NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OF THE PRODUCT. THE PURCHASER ACKNOWLEDGES THAT THEY ALONE HAVE DETERMINED THAT THE PRODUCT WILL SUITABLY MEET THE REQUIREMENTS OF THE PURCHASER'S INTENDED USE.

(c) Purchaser Remedy- Horizon's sole obligation hereunder shall be that Horizon will, at its option, (i) repair or (ii) replace, any Product determined by Horizon to be defective. In the event of a defect, these are the Purchaser's exclusive remedies. Horizon reserves the right to inspect any and all equipment involved in a warranty claim. Repair or replacement decisions are at the sole discretion of Horizon. This warranty does not cover cosmetic damage or damage due to acts of God, accident, misuse, abuse, negligence, commercial use, or modification of or to any part of the Product. This warranty does not cover damage due to improper installation, operation, maintenance, or attempted repair by anyone other than Horizon. Return of any goods by Purchaser must be approved in writing by Horizon before shipment.

Damage Limits

HORIZON SHALL NOT BE LIABLE FOR SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOSS OF PROFITS OR PRODUCTION OR COMMERCIAL LOSS IN ANY WAY CONNECTED WITH THE PRODUCT, WHETHER SUCH CLAIM IS BASED IN CONTRACT, WARRANTY, NEGLIGENCE, OR STRICT LIABILITY. Further, in no event shall the liability of Horizon exceed the individual price of the Product on which liability is asserted. As Horizon has no control over use, setup, final assembly, modification or misuse, no liability shall be assumed nor accepted for any resulting damage or injury. By the act of use, setup or assembly, the user accepts all resulting liability.

If you as the Purchaser or user are not prepared to accept the liability associated with the use of this Product, you are advised to return this Product immediately in new and unused condition to the place of purchase.

Law: These Terms are governed by Illinois law (without regard to conflict of law principals).

Safety Precautions

This is a sophisticated hobby Product and not a toy. It must be operated with caution and common sense and requires some basic mechanical ability. Failure to operate this Product in a safe and responsible manner could result in injury or damage to the Product or other property. This Product is not intended for use by children without direct adult supervision. The Product manual contains instructions for safety, operation and maintenance. It is essential to read and follow all the instructions and warnings in the manual, prior to assembly, setup or use, in order to operate correctly and avoid damage or injury.

Questions, Assistance, and Repairs

Your local hobby store and/or place of purchase cannot provide warranty support or repair. Once assembly, setup or use of the Product has been started, you must contact Horizon directly. This will enable Horizon to better answer your questions and service you in the event that you may need any assistance. For questions or assistance, please direct your email to productsupport@horizonhobby.com, or call 877.504.0233 toll free to speak to a Product Support representative.

Inspection or Repairs

If this Product needs to be inspected or repaired, please call for a Return Merchandise Authorization (RMA). Pack the Product securely using a shipping carton. Please note that original boxes may be included, but are not designed to withstand the rigors of shipping without additional protection. Ship via a carrier that provides tracking and insurance for lost or damaged parcels, as Horizon is not responsible for merchandise until it arrives and is accepted at our facility. A Service Repair Request is available at www.horizonhobby.com on the "Support" tab. If you do not have internet access, please include a letter with your complete name, street address, email address and phone number where you can be reached during business days, your RMA number, a list of the included items, method of payment for any non-warranty expenses and a brief summary of the problem. Your original sales receipt must also be included for warranty consideration. Be sure your name, address, and RMA number are clearly written on the outside of the shipping carton.

Warranty Inspection and Repairs

To receive warranty service, you must include your original sales receipt verifying the proof-of-purchase date. Provided warranty conditions have been met, your Product will be repaired or replaced free of charge. Repair or replacement decisions are at the sole discretion of Horizon Hobby.

Non-Warranty Repairs

Should your repair not be covered by warranty the repair will be completed and payment will be required without notification or estimate of the expense unless the expense exceeds 50% of the retail purchase cost. By submitting the item for repair you are agreeing to payment of the repair without notification. Repair estimates are available upon request. You must include this request with your repair. Non-warranty repair estimates will be billed a minimum of ½ hour of labor. In addition you will be billed for return freight. Please advise us of your preferred method of payment. Horizon accepts money orders and cashiers checks, as well as Visa, MasterCard, American Express, and Discover cards. If you choose to pay by credit card, please include your credit card number and expiration date. Any repair left unpaid or unclaimed after 90 days will be considered abandoned and will be disposed of accordingly. Please note: non-warranty repair is only available on electronics and model engines.

Electronics and engines requiring inspection or repair should be shipped to the following address:

Horizon Service Center
4105 Fieldstone Road
Champaign, Illinois 61822, USA

All other Products requiring warranty inspection or repair should be shipped to the following address:

Horizon Product Support
4105 Fieldstone Road
Champaign, Illinois 61822, USA

Please call 877-504-0233 or e-mail us at productsupport@horizonhobby.com with any questions or concerns regarding this product or warranty.

European Union:

Electronics and engines requiring inspection or repair should be shipped to one of the following addresses:

Horizon Hobby Limited
Units 1-4 Ployters Rd
Staple Tye, Harlow
Essex CM18 7NS
United Kingdom

Please call +44 (0) 1279 641 097 or email sales@horizonhobby.co.uk with any questions or concerns regarding this product or warranty.

Horizon Technischer Service
Hamburger Str. 10
25335 Elmshorn
Germany

Please call +49 4121 46199 66 or email service@horizonhobby.de with any questions or concerns regarding this product or warranty.

Horizon Hobby SAS
14 Rue Gustave Eiffel
Zone d'Activité du Réveil Matin
91230 Montgeron
France

Please call +33 (0) 1 60 47 44 47 with any questions or concerns regarding this product or warranty.

Compliance Information for the European Union

Declaration of Conformity

(in accordance with ISO/IEC 17050-1)

No. HH2010012801

Product(s):	Spektrum AR8000 Receiver
Item Number(s):	SPMAR8000
Equipment class:	1

The objects of declaration described above are in conformity with the requirements of the specifications listed below, following the provisions of the European R&TTE directive 1999/5/EC:

EN 301 489-1 v.1.6.1	**General EMC requirements for Radio equipment**
EN 301 489-17 v.1.2.1	

Signed for and on behalf of:
Horizon Hobby, Inc.
Champaign, IL USA
Jan 28, 2010

[illegible]

Steven A. Hall
Vice President
International Operations and Risk Management
Horizon Hobby, Inc.

Instructions for Disposal of WEEE by Users in the European Union

This product must not be disposed of with other waste. Instead, it is the user's responsibility to dispose of their waste equipment by handing it over to a designated collection point for the recycling of waste electrical and electronic equipment. The separate collection and recycling of your waste equipment at the time of disposal will help to conserve natural resources and ensure that it is recycled in a manner that protects human health and the environment. For more information about where you can drop off your waste equipment for recycling, please contact your local city office, your household waste disposal service or where you purchased the product.

FCC Information

This device complies with part 15 of the FCC rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

Caution: Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This product contains a radio transmitter with wireless technology which has been tested and found to be compliant with the applicable regulations governing a radio transmitter in the 2.400GHz to 2.4835GHz frequency range.

US patent number 7,391,320. Other patents pending.

Appendix F

Senior Design Conference Slides As Presented

CuriPilot

Greg Cusack, Karan Daryanani, Nathaniel Tucker
Advisors: Dr. Christopher Kitts, Dr. Sally Wood
May 12th, 2016

Agenda

- Project Background
- Objectives
- Initial Design
- Final Design
- Testing and Results
- Lessons Learned

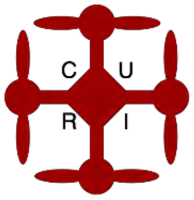


Figure 1: CuriPilot Logo

Objectives

- Show that the Intel Curie Processor is a viable choice for running a drone's stabilization control system.
- Help Intel bring up a new revision of the Curie board suitable for UAVs
- To develop a fully functional UAV autopilot system

Figure 2: Intel Curie Processor

Functional and Nonfunctional Requirements

CuriPilot ***will…***

- maintain stability while hovering
- allow for stable user controlled flight
- allow for altitude locked flight

CuriPilot ***will be…***

- modular
- maintainable
- robust
- reliable
- safe

SANTA CLARA UNIVERSITY

Design and Testing Constraints

- Stabilization system must run on the Intel Curie
- Stabilization system must be implemented in embedded C
- Limits of a Real Time Operating System
- Illegal for us to fly outdoors

Figure 3: Drone Development Cage

SANTA CLARA UNIVERSITY

About the Curie

- 32-bit, low-power, Intel Quark Processor
- 400MHz clock speed
- 384kB Flash Memory
- 6 axis IMU containing the onboard accelerometer and gyroscope
- Real-time Operating System
- Same platform used on snowboards in 2016 Winter X Games!

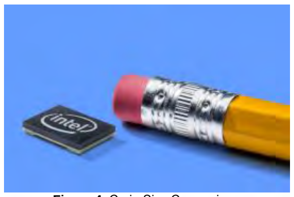


Figure 4: Curie Size Comparison

SANTA CLARA UNIVERSITY

Initial Work and Roadblocks

- What we had:
  - Rev.1 breakout board
  - Outdated Curie software
- What happened?
  - “Bricked” our Rev.1 board
- How did we react?
  - Restructured scope of project
  - Began developing on other parts of the system besides stabilization

Figure 5: Drone with 1st Board

SANTA CLARA UNIVERSITY

Curie Breakout Board v3

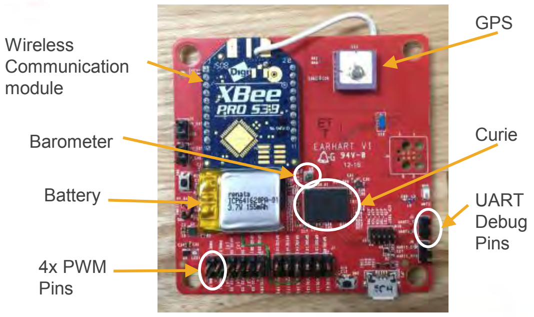


Figure 6: Earhart Board

System Architecture

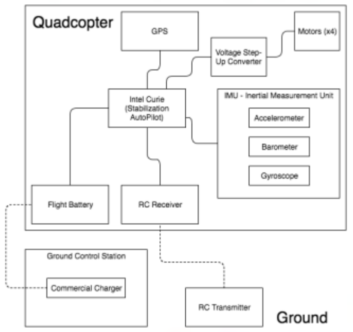


Figure 7: High-level System Architecture

Introduction to Drone Flight

- Roll, Pitch, and Yaw
- Rate Control vs Angle Control
- Altitude Lock

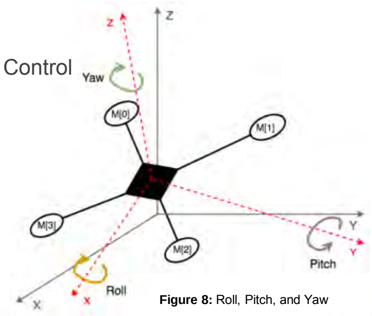


Figure 8: Roll, Pitch, and Yaw

PID Control Overview

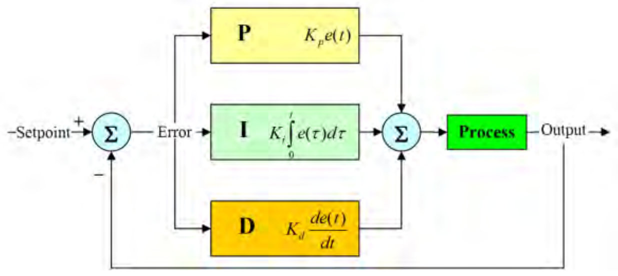


Figure 9: PID Control Diagram

Rate Stabilization

- KPX, KIX, KDX represent gain values used for the control system
- Proportional:

```
rate_err = (desired_rate - gyro_x) * KPX
```

- Integral:

```
int_err = (rate_err + int_err) * KIX
```

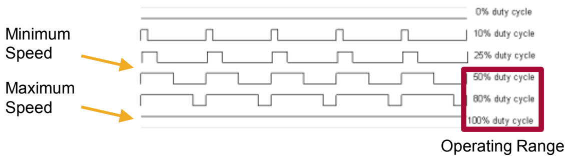
- Derivative:

```
deriv_err = (rate_err - rate_err_prev) * KDX
```

SANTA CLARA UNIVERSITY

Control Loop Output

- Use accelerometer & gyroscope data to calculate output PWM signals for each motor (0,1,2,3)



- Signal is sent through our 3.3V -> 5V converter and then to the 4 ESC's (Electronic Speed Controllers)

SANTA CLARA UNIVERSITY

Output to Motors

Motor[1]:
- roll adjust
- pitch adjust
+ yaw adjust

Motor[2]:
- roll adjust
+ pitch adjust
- yaw adjust

Motor[0]:
+ roll adjust
- pitch adjust
- yaw adjust

Motor[3]:
+ roll adjust
+ pitch adjust
+ yaw adjust

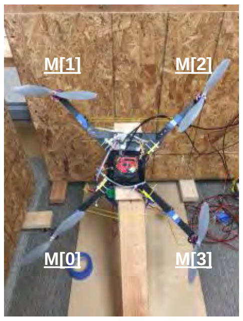


Figure 11: Drone on Test Stand

SANTA CLARA UNIVERSITY

Gain Testing for Stabilization

- Ran MATLAB script with simulated gyroscope data to quickly test algorithm and gain values
- On Rev. 1 of the board, we had ADC pins which allowed us to use Potentiometers to manually adjust the gains at run-time
- Began with adjusting KPX
  - Start with small value and work way up until relatively balanced
- Moved on to KIX and KDX with same strategy

SANTA CLARA UNIVERSITY

MATLAB Simulation

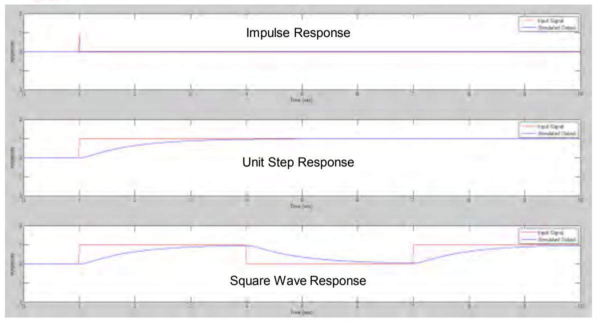


Figure 12: MATLAB Plots

SANTA CLARA UNIVERSITY

Progress to Date on GPS Integration

- MAVLink (Micro Air Vehicle Link) is a protocol that enables communication to unmanned vehicles
- MAVLink XML Generation
  - Supports: stabilization, altitude lock and waypoint navigation
  - Converts common XML using Python to MAVLink logic headers
  - Heartbeat stabilization loop of 1hz (1 second)

SANTA CLARA UNIVERSITY

Progress to Date on GPS Integration

- QGroundControl torn apart to support custom MAVLink
- Establish stable connection between Curie and QGroundControl

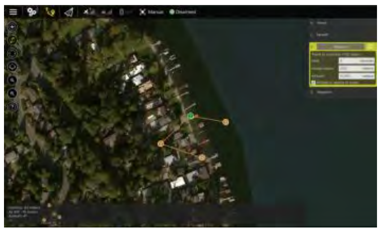


Figure 13: QGroundControl Graphical User Interface

SANTA CLARA UNIVERSITY

Choosing a Communication Method

- Onboard XBee vs. RC Controller (Spektrum AR8000)
  - XBee
    - Less reliable for flying since signal is easily dropped
    - Integrated on Curie breakout board
  - AR8000
    - Well documented, reliable, hobbyist device
    - Requires extra PWM input

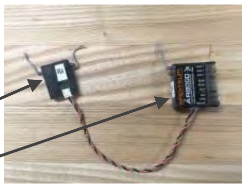


Figure 14: AR8000 Throttle Controller

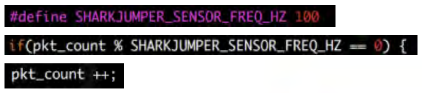
SANTA CLARA UNIVERSITY

Gyroscope Offset and Accelerometer Calibration

- Gyroscopes tend to drift over time
  - Implemented on offset in order to obtain accurate data for the control system
- Calibrated accelerometer for angle control algorithm
  - Found what accelerometer value corresponds to -1G and 1G
  - Used linear interpolation to calculate intermediate G values

Adjusting Packet Rate

- Severe delay in Curie's response time hindered stabilization capability
  - Only writing 1% of available data to rotors
- Adjusted code to utilize significantly more sensor data for rotor control
  - Averaged every 5 readings and wrote average to ESCs.



```
#define SHARKJUMPER_SENSOR_FREQ_HZ 100
if(pkt_count % SHARKJUMPER_SENSOR_FREQ_HZ == 0) {
pkt_count ++;
```

Figure 15: Packet Rate Control Code

"Live" Demo

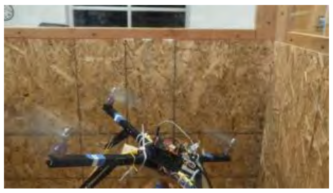
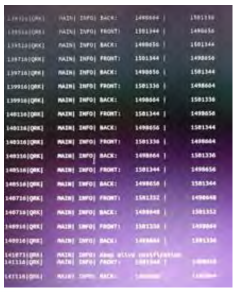


Figure 16: Input Data from Gyroscope



```
139316|QRK|   MAIN| INFO| BACK:     1498664 |        1501336
139516|QRK|   MAIN| INFO| FRONT:    1501344 |        1498656
139516|QRK|   MAIN| INFO| BACK:     1498656 |        1501344
139716|QRK|   MAIN| INFO| FRONT:    1501344 |        1498656
139716|QRK|   MAIN| INFO| BACK:     1498656 |        1501344
139916|QRK|   MAIN| INFO| FRONT:    1501336 |        1498664
139916|QRK|   MAIN| INFO| BACK:     1498664 |        1501336
140116|QRK|   MAIN| INFO| FRONT:    1501344 |        1498656
140116|QRK|   MAIN| INFO| BACK:     1498656 |        1501344
140316|QRK|   MAIN| INFO| FRONT:    1501336 |        1498664
140316|QRK|   MAIN| INFO| BACK:     1498664 |        1501336
140516|QRK|   MAIN| INFO| FRONT:    1501344 |        1498656
140516|QRK|   MAIN| INFO| BACK:     1498656 |        1501344
140716|QRK|   MAIN| INFO| FRONT:    1501352 |        1498648
140716|QRK|   MAIN| INFO| BACK:     1498648 |        1501352
140916|QRK|   MAIN| INFO| FRONT:    1501336 |        1498664
140916|QRK|   MAIN| INFO| BACK:     1498664 |        1501336
141073|QRK|   MAIN| INFO| [illegible]
141116|QRK|   MAIN| INFO| FRONT:    [illegible] |    [illegible]
[illegible]
```

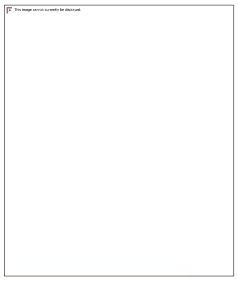
Figure 17: Output PWM Duty Cycles

Challenges

- 2 active cores conflicting

ARC
- Sensor Reads
- Subscribe to message data
- Hardware Configurations

QUARK
- Main Loop
- Queue management and scheduling
- Stabilization logic and motor output



This image cannot currently be displayed.

Figure 18: Kernel Panic

Recap

- Initial Goals
  - Fully stabilize a quadcopter running on the Curie
  - Develop user controlled flight
  - Begin altitude locked flight
  - Begin development on waypoint control
  - Build Curie into Yuneec drone

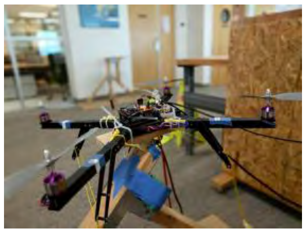


Figure 19: Drone with 3rd Board revision

SANTA CLARA UNIVERSITY

Recap

- Accomplishments
  - Integrated rate and angle control into system
  - Successfully developed a systematic way to calibrate the Curie's onboard sensors
  - Began waypoint control development
  - Brought up new board with Intel
  - RC control development
  - Identified and solved multiple firmware issues within the Curie

SANTA CLARA UNIVERSITY

Future Work

- Fine tune PID gains
- Fully integrate RC control
  - Spektrum AR8000
- Finish altitude lock flight control
- Integrate Curie into Yuneec drone

Figure 20: Yuneec Typhoon Q500 4K

SANTA CLARA UNIVERSITY

Advice to Future Development Teams

- Intel is around to help you!
  - Do not be afraid to ask the Intel development team for help
- Regularly calibrate sensors
- Use available ADC pins and potentiometers to adjust gains in real time

SANTA CLARA UNIVERSITY

Acknowledgements

- Intel
  - Jeff Ota
  - Steven Xing
  - Mike Rosen
- Robotics Systems Lab
  - Dr. Christopher Kitts
  - Addison Fattor
  - Robert McDonald

YUNEEC ELECTRIC AVIATION · Santa Clara University School of Engineering · intel

SANTA CLARA UNIVERSITY

Questions?

SANTA CLARA UNIVERSITY

Figure References

- Figure 2: http://www.electropages.com/2016/01/intel-curie-based-board-is-aimed-at-developing-wearable-technology/
- Figure 4: http://iq.intel.com/americas-greatest-makers/
- Figure 9: https://drstienecker.com/tech-332/7-pid-control/
- Figure 10: https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM
- Figure 13: http://www.qgroundcontrol.org/screenshots
- Figure 20: http://www.pcmag.com/article2/0,2817,2487122,00.asp

SANTA CLARA UNIVERSITY

Technologies Used

- Intel® Curie and Curie Breakout Board
  - Stabilization control, altitude lock computations, and RC control
  - Peripheral sensors, communications, and GPIO pins
- Programming Languages: Embedded C
  - Used for control system implementation (running on Curie)
- Vim
  - Development environment
- Python
  - MAVLink header generations

SANTA CLARA UNIVERSITY

Board Revision Comparison

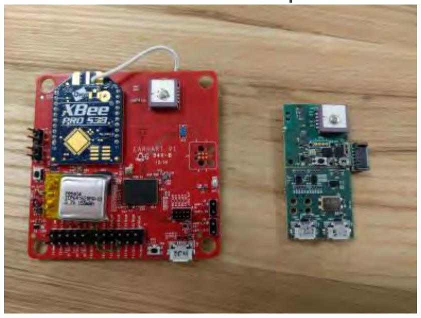
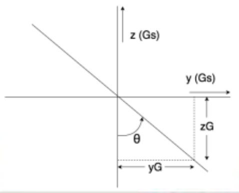


Figure 6: Updated Breakout Board and Updated Wearable Device

SANTA CLARA UNIVERSITY

Angle Control for Initial Stabilization

- yG = (accel_Y - min_Y)*2/(max_Y - min_Y) - 1
- zG = (accel_Z - min_Z)*2/(max_Z - min_Z) - 1
- $\theta = \tan^{-1}((1-yG)/(1-zG))*180/\pi$



SANTA CLARA UNIVERSITY

Pictures

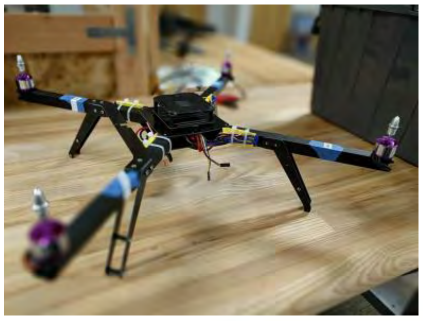


Figure 1: Frame of the drone

SANTA CLARA UNIVERSITY

Pictures

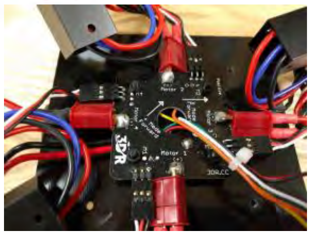
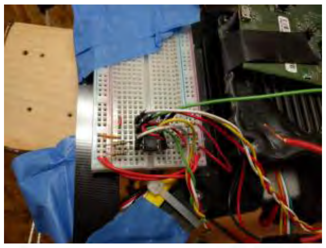


Figure 2: Power Distribution Board (Located inside drone frame)

Figure 3: Step up converter



SANTA CLARA UNIVERSITY

Getting a New Breakout Board

- What we needed:
  - The updated software!
  - Accelerometer/Gyroscope (on Curie)
  - GPS (Waypoint navigation)
  - Barometer (Altitude lock)
  - 4 PWM output pins
    - Pulse Width Modulation
  - Onboard, rechargeable battery
  - UART/USB debugging capability
    - Universal Asynchronous Receiver/Transmitter
    - Universal Serial Bus
  - Wireless communication capability

SANTA CLARA UNIVERSITY

Pictures

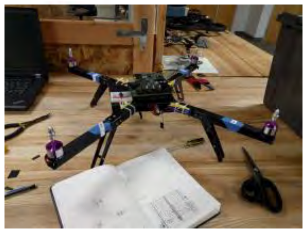


Figure 4:
Drone with 1st
Board

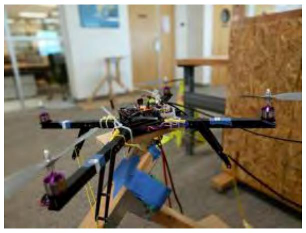


Figure 5:
Drone with 3rd
Board revision