MicroCART 2018-2019

DESIGN DOCUMENT

Team #20

Client/Advisor

Dr. Phillip Jones

Team Members

Tony Bertucci - Ground Station Lead
Sarah Koch - Controls Lead
Tina Li - Quad Software Lead
Nina Moriguchi - Flight Simulation Lead
James Talbert - Hardware Lead

Team Email

sdmay19-20@iastate.edu

Team Website

http://sdmay19-20.sd.ece.iastate.edu

Revised: Nov. 29 / Version: 2.0

Table of Contents

List of Figures

List of Tables

Definitions

Term	**Definition**
CLI	Command line interface
Continuous Integration	automated process of running tests on every commit to the repository
Demo	Short for demonstration; this is one of the deliverables of the project: a demonstration of the quad's capabilities, for example, doing a backflip with the quad, finding an object and following it, communicating with a second quad to perform flight patterns
GPS	Global Positioning System; space-based radio navigation system using satellites to determine position; proposed to find position (x, y) when not in the lab using the VRPN system
Ground station	The application that runs on a host computer that communicates with the quad via a Wi-Fi connection and sends it coordinates to the quad
GUI	Graphical user interface
IR	Infrared wavelengths of light longer than visible light; used in the VRPN system to determine the position of the quad
LIDAR	Light Detection and Ranging; this is a system for determining the altitude (z) of the quad using the onboard sensor
Optical Flow	system using pattern of motion of objects, surfaces, and edges caused by the relative motion between the and the scene to determine position; used by the quad to calculate position (x, y) when not in the lab using the VRPN system
PID	Proportional-integral-derivative control system; standard control algorithm used on the quad
Quad	Short for quadcopter; this is the hardware platform we use in this project
Setpoint	in a control system, the target value for an essential variable
VRPN [1]	Virtual-Reality Peripheral Network; this is the system used to determine the position (x, y, z) and orientation (ϕ,θ,ψ) of the quad in the lab using a set of 12 stationary cameras and an IR transmitter on the quad

1 Introduction

Microprocessor Controlled Aerial Robotics Team or MicroCART project is centered around the development of a quadcopter (see **Error! Reference source not found.** below) and tracking s ystem. This project has been in development since 1998 and the current system has been passed down since 2006. The project aims to create a stable and easy to use platform for researching control theory. The quadcopter flies primarily in the Distributed Sensing and Decision Making Laboratory within a twelve camera infrared tracking system.

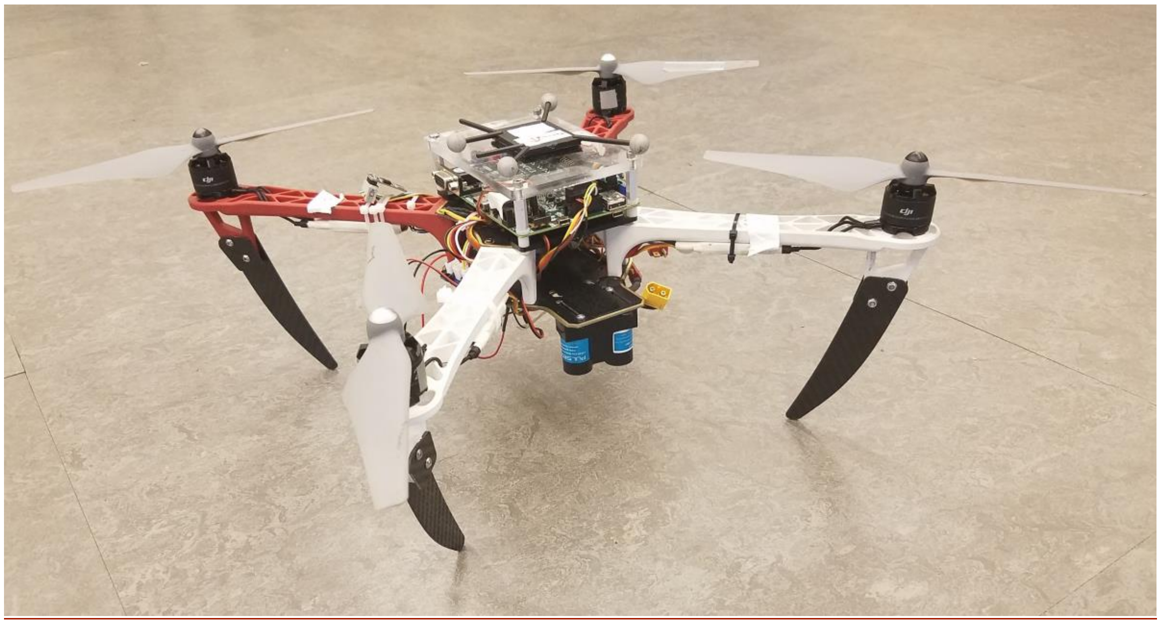


Figure 1-1 MicroCART quadcopter

1.1 Acknowledgement

MicroCART is a project that is assisted by graduate students working in controls under Dr. Phillip Jones. As this is an ongoing project, previous team members will also be providing help in understanding the current system. As such, we would like to acknowledge the assistance that has been and will be provided by:

- Matthew Cauwels
- Robert Buckley
- Matt Rich
- Dr. Phillip Jones
- Dane Larson
- Matthew Kelly
- Austin Rohlfing
- Eric Middelton

1.2 Problem Statement

We will improve upon an existing platform that is used for research in controls and embedded systems and for departmental demos. The platform will be improved by adding increased, reusable testing of all systems, adding documentation to increase the speed for new users to get started, and by adding new system features.

1.2.1 Problem

The MicroCART platform designed in previous years had many flaws that hindered its use for research and demo purposes. The previous platform failed to familiarize the user(s) of the system in a time horizon that would make it viable for research. This is because the system did not have ample documentation available for the users of the system to learn about the platform and its uses. From the viewpoint of a user running demos the area within the VRPN system the platform as it stood is most stable when confined to flying within the VRPN system as the optical flow navigation cannot hold position during flight. This means that the areas available to the quadcopter for demos can only utilize the small amount of space in the lab for a demo. Lastly, the quadcopter is controlled using a PID controller that requires logical guessing and checking to tune, we now have a new linear controller that can be computed faster and be tuned around multiple points on the nonlinear model.

1.3 Operational Environment

In order to fly using the VRPN software for position and orientation data, the quad must be inside of a small area (less than 10 m^2) inside of Coover 3050. This lab is designed to cause very few environmental impacts on the quadcopter. Through the use of ventilation, window shades, and Coover's heating and air conditioning, the lab has a nearly constant light and temperature with little to no accumulated dust to affect air quality.

Additionally, the project relies on two Linux computers. One is used to control the ground station software and the other contains build tools for the FPGA on the quadcopter itself. This provides a platform for development that is consistent across team members and easier to demo as there are not issues with building or ensuring correctness of the various communication aspects used for the project.

1.4 Intended Users and Uses

The primary set of end users is composed of future MicroCART members and controls graduate students at Iowa State. We also have demonstrations for prospective students, someone from the two categories (the user) will be running the demo for them (the audience). This means that the users can be assumed to have competence in using multiple forms of programs (for example, either GUI or CLI) and in reading general technical documentation.

The other goal listed above regards the modular implementation of new control algorithms as a research opportunity for graduate students. These users have three primary needs from our product. The first is a robust and reliable system to decrease variation in test results. This includes having sturdy quad hardware, low communication latency, and a bug-free user interface. The second need is to have modular software with complete documentation to allow for them to alter the implementation themselves, without the need for significant system rework or intervention of the MicroCART design team. Finally, these students will need the data from the system characterization in order to form their models. This includes information about mass, moments of inertia, motor resistances, rotor areas, and many other properties that determine the true behavior of the quadcopter. There is also a need to make sure that any new features are documented clearly so that the next group of students that add features will be able to ramp up quickly.

1.5 Assumptions and Limitations

1.5.1 Assumptions

- Our VRPN camera system as it exists provides sufficiently accurate position data
- Our VRPN camera system as it exists provides sufficiently frequent updates
- The quadcopter will be used within the camera system

1.5.2 Limitations

- The quadcopter must use a wireless link to the ground station
- Accuracy of onboard sensors (e.g. optical flow, LIDAR, IMU, GPS)
- Latency and range of the wireless link between the quadcopter and the ground station
- The quadcopter must be physically tethered to the lab floor

1.6 Expected End Product and Deliverables

The quadcopter system consists of three major subsections: the quadcopter hardware/software, the ground station, and the control systems. Each of the subsections is essential to meet the desired objectives and fulfill requirements. Documentation and demos are also a major deliverable for our project and will be discussed.

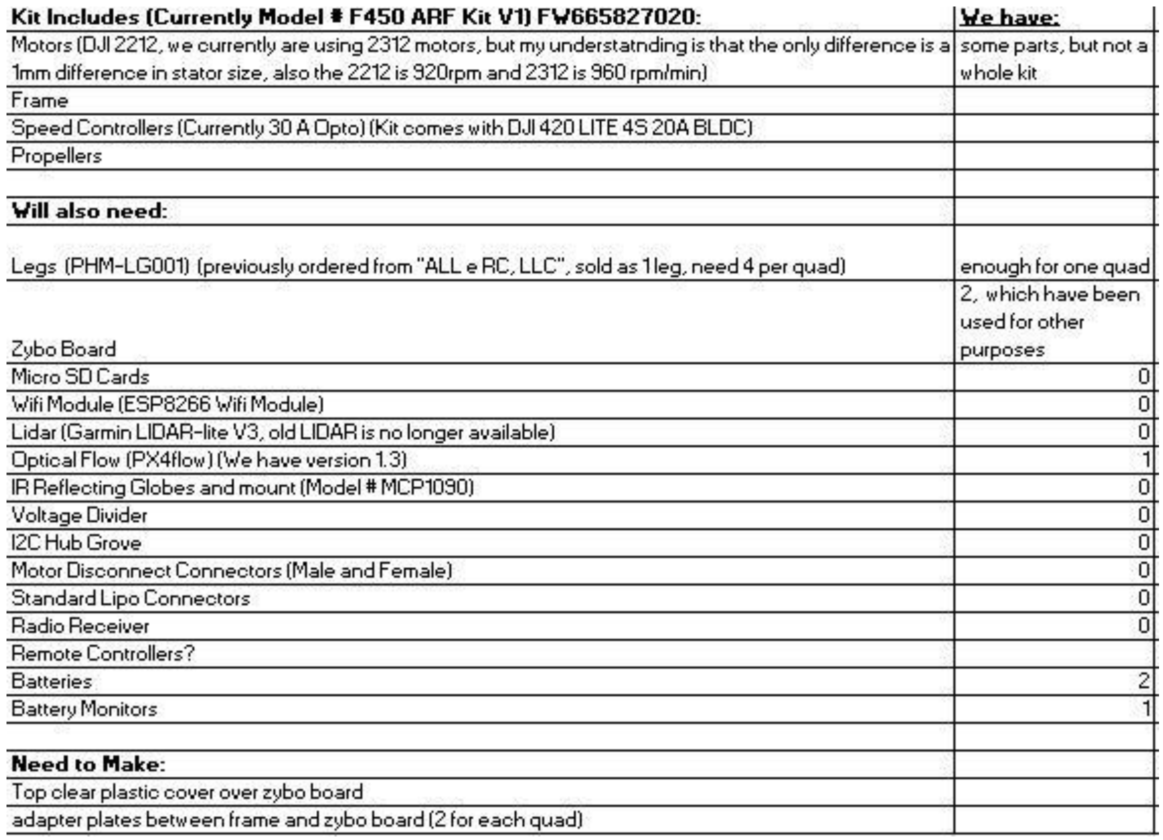
1.6.1 Quad Hardware and Software

1.6.1.1 Vivado Upgrade Testing

The entire quad software platform relies on the Xilinx toolchain. The software we were using to develop the Quad software is the Xilinx toolchain. Specifically, XPS, which is known to be a very non-user friendly and dated piece of software. One of the major goals of this project was to transition to the new and improved Vivado to program the FPGA hardware. In addition, we added unit tests for the hardware modules in simulation as part of our continuous integration pipeline.

1.6.1.2 Second Quad

Our client advisor requested that we develop a second quad for available flight. The difficulties in building a second quad revolve around the lack of documentation of parts used on the quad, along with the availability of the previous generation of parts. The new quad needed to be able to run the same software on a different set of hardware.



Kit Includes (Currently Model # F450 ARF Kit V1) FW665827020:	We have:
Motors (DJI 2212, we currently are using 2312 motors, but my understatnding is that the only difference is a 1mm difference in stator size, also the 2212 is 920rpm and 2312 is 960 rpm/min)	some parts, but not a whole kit
Frame	
Speed Controllers (Currently 30 A Opto) (Kit comes with DJI 420 LITE 4S 20A BLDC)	
Propellers	
Will also need:	
Legs (PHM-LG001) (previously ordered from "ALL e RC, LLC", sold as 1 leg, need 4 per quad)	enough for one quad
Zybo Board	2, which have been used for other purposes
Micro SD Cards	0
Wifi Module (ESP8266 Wifi Module)	0
Lidar (Garmin LIDAR-lite V3, old LIDAR is no longer available)	0
Optical Flow (PX4flow) (We have version 1.3)	1
IR Reflecting Globes and mount (Model # MCP1090)	0
Voltage Divider	0
I2C Hub Grove	0
Motor Disconnect Connectors (Male and Female)	0
Standard Lipo Connectors	0
Radio Receiver	0
Remote Controllers?	
Batteries	2
Battery Monitors	1
Need to Make:	
Top clear plastic cover over zybo board	
adapter plates between frame and zybo board (2 for each quad)	

Table 1-1 Parts Order for second Quad

1.6.1.3 Quad Hardware Upgrade

The client has requested we upgrade the quads to the latest Zybo board that includes a connection for a pi camera. The current quads have many long wires and loose connections that are potential points of failure. Our team will create a custom pcb board to replace those connections, as well as install the newer zybo boards.

1.6.1.4 Quad Software Upgrade

Currently the quad can only fly reliably within the VRPN Camera system. By integrating the GPS and improving controls, a quad that will reliably fly outside will be aimed for. The improved communication and multi-client capability developed last year opens up the possibility of networking and quad coordination. To take advantage of the newer zybo boards and cameras the client requested, a version of Linux with OpenCV, to be run on the second processor on the board, will be explored. Software surrounding these new functionalities will be developed.

1.6.1.5 Power Regulator PCB Board

The PCB will monitor battery usage and make sure that the user is alerted when battery power is low, and that the quad lands whenever battery power is too low. It will also make sure the battery usage is efficient as possible, and no more power than necessary is drawn when the quad is idle.

1.6.1.6 Sensor Data Sent in Real Time

Because the quad now uses Wi-Fi instead of Bluetooth, it is possible to explore sending sensor data in real time rather than logging sensor data and sending it after the quad lands. The user should explain what type of data they want even during mid-flight, and the quad should be able to send that data back to the ground station immediately.

1.6.2 Ground Station

1.6.2.1 Transmission of Flight Data in Real Time - Prototype by January 1, 2019. Final by May 1, 2019

The communication standard currently setup between the Ground Station and the quad supports the transmission of flight data and performance information after a flight has completed. Due to the amount of information being transferred, this process usually takes a considerable amount of time to complete. We would like to improve the communication occurring between the quad and the Ground Station to support the transmission of flight data in real time. This will improve the quad's status as a research platform by allowing for easy and timely analysis of flight and controls data.

1.6.2.2 Updated GUI for Flight Data Information - Prototype by January 1, 2019. Final by May 1, 2019

Currently, the GUI does not support displaying flight data to the user in real time as it arrives from the quad. The GUI will be updated to include a display for flight data that the user can see, as well as interact with to choose the types of data they wish to be seeing and recording. This will allow users to have a high level of control over the type and amount of data they are seeing during flight time.

1.6.2.3 Multiple Object Interaction Capabilities - Prototype by March 1, 2019. Final by May 1, 2019

Currently the ground station supports having multiple quads connected at the same time. In order to maintain a safe environment, both for observers and for the quads, the ground station will be improved to include position and orientation analysis for all connected quads in order to ensure collisions between objects connected to the ground station do not occur. This allows for a higher level of safety when conducting controls experiments involving multiple quads, as well as improve the expected lifetime of components and quads.

1.6.3 Controls Systems

1.6.3.1 Model Linearization and LQR Controllers

The primary deliverables of this year's team were a modular linearization of the system model and a pair of LQR controllers. The linearization is a script that uses symbolic MATLAB derivation of the nonlinear model provided by Matt Rich in [1]. This allows a future user to change the nonlinear model and immediately recompute the system linearization. Similarly, the linearization is also dynamic on the measured system parameters, so no further work needs to be done to account for potential future changes of physical properties (e.g. using bigger rotors or a frame with a greater mass).

1.6.3.2 System Parameterization Instructions

Because there are now two quadrotors, it is more important than ever to be able to measure and track the physical properties of each quadcopter. As such, the controls team aggregated parameter measurement procedures from both Rich's [1] and McInerney's [4] research, as well as from un-versioned documentation from the previous year's team. These were formed into a series of four parameter identification instruction documents, written in Markdown and stored on git, that contained straightforward instruction, consistent variable usage, and (where necessary) example MATLAB scripts. Additionally, a Markdown document was created to track all relevant parameter values and instruction sets.

1.6.4 Continuous Integration

1.6.4.1 Quad Simulator

The quad simulator models a virtual flight dynamics environment for various flight tests. The current established model in the simulator does not model rotor dynamics; however, it still offers a reliable platform for performing sanity checks of the changes in controls and quad software. The current simulator uses a slightly modified version of the actual quadcopter controls. The simulator also offers input and output through sockets which enables control to be running outside of the simulator. Our team will focus on improving the simulator model and integrating the simulator with the automated environment of GitLab.

1.6.4.2 Upgrade Testing Framework

Continuous Integration is the system that tests changes to code using the virtual quadcopter software. To make the tests more standardized and provide more flexibility in writing the tests, the tests were ported from a custom barebones testing framework to a standard testing framework, Unity [5]. This provides a fully developed set of testing functions that can be used by future teams. We plan to also increase test coverage and write tests for new features.

1.6.4.3 Automated Testing for Controls Output

We plan on creating a test that would gather information about the controls output in real time, then get the actuator data from the simulator and make sure that the two points match.

1.6.5 Documentation

The year before, many areas of the code, especially those relating to ground station and quad software, were lacking documentation. The ground station contains four main components that are separated well but adding functionality was not explained nor is it mentioned that this is custom communication between the ground station and quad. The quad software is designed in a way that makes it so external directories must be used in build tools and there is also no explanation of the hardware running on the quad. Last year's team made it their goal to have documentation for all existing demos, documentation consistent in all code, and documentation for the research done during their time on the team. To follow up on that goal, our team will continue adding documentation. This year, the areas of controls model and simulation, ground station, the CI Testing Framework, along with pure hardware plans need improvement and organization.

1.6.6 Demos

As one purpose of this project is to showcase the talents within this department, new demos needed to be developed to showcase yearly changes. These demos are performed to controls classes as well as to undergraduate students. We plan to implement the following major demos:

1. Have a quad that tracks an object on the ground, or in air, and maintains a set distance away from it.
2. Have multiple quads perform synchronous movements
3. Have multiple types of quads running at the same time flying together.

2 Specifications and Analysis

2.1 Proposed Design

2.1.1 Overview

At the highest level, the proposed system consists of three parts: the quadcopter, the ground-station, and the camera-based object tracking system. The way these systems communicate is shown below.

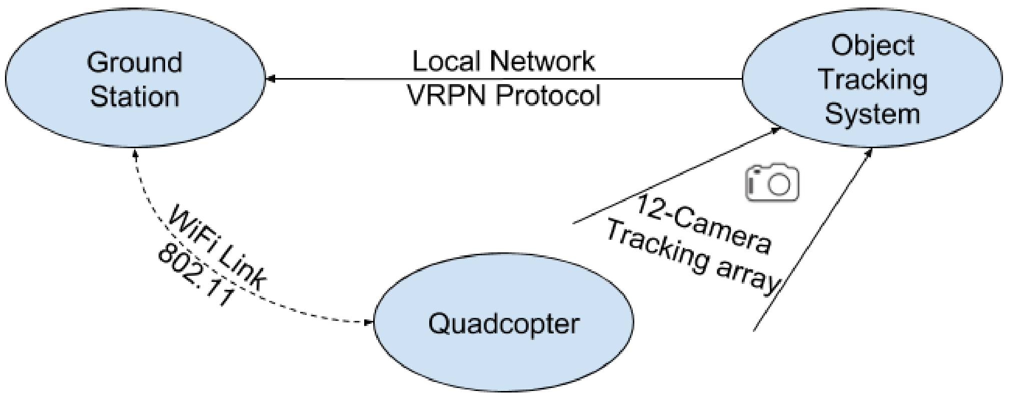


Figure 2-1 Top-level System Diagram

Each of these systems is of course made of up various internal subsystems. The object tracking system is a black box, but the ground station and quadcopter are developed by the team. The object tracking system uses an array of infrared cameras to track markers on the quad's frame, it then streams the position and orientation data to the ground station, which in turn distributes the useful elements to the quadcopter over a Wi-Fi link [2]. The quadcopter uses this data in combination with internal sensors and setpoints from the ground station to control its position, which the cameras can observe.

2.1.2 Quad Hardware and Software

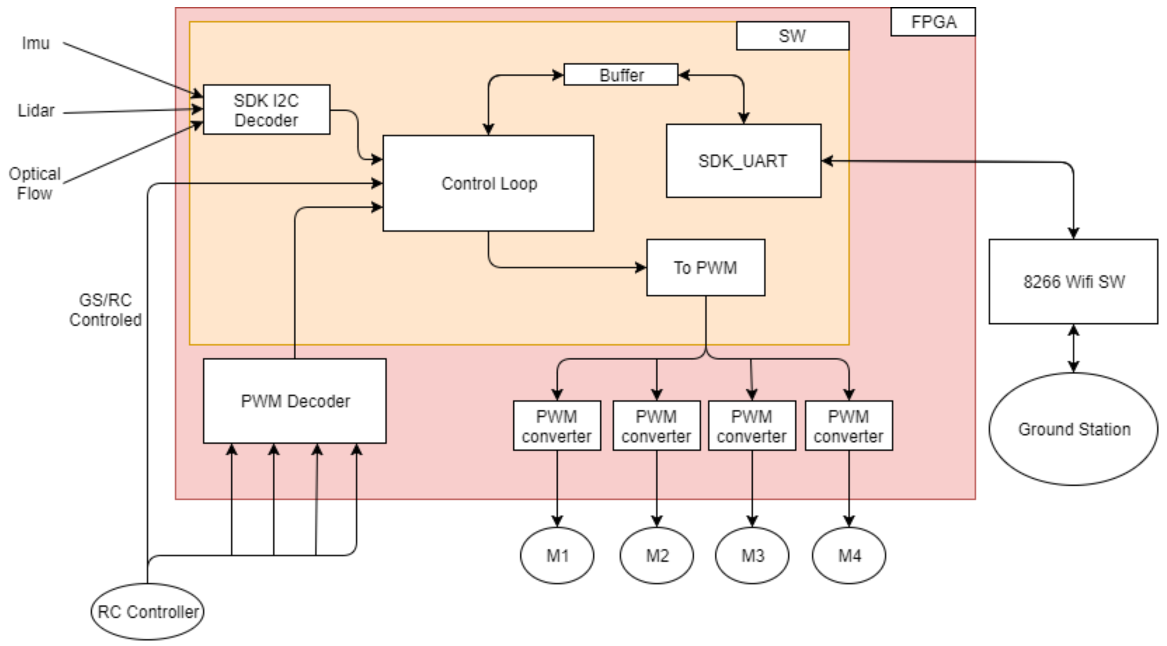


Figure 2-2 Quadcopter System Diagram

The quadcopter is built from a Flamewheel 450 airframe, with a Zybo Z7020 control board. This board includes a Xilinx Zynq Z7020 FPGA. On board the quad are also a Wi-Fi bridge, an RC receiver, an optical flow sensor, a LIDAR sensor, and an Inertial Measurement Unit (IMU). The FPGA includes the hardware needed to interface to all of these devices, and to generate PWM to control the Electronic Speed Controllers (ESCs) for the motors. The hardware design for the FPGA is implemented in Xilinx Vivado.

Within the FPGA, a processor core runs a baremetal (no operating system) program that runs a continuous loop:

- Read Sensor Data
- Process/Filter Sensor Data
- Run Control Algorithm
- Output Actuation
- Log data

Here, the input from the RC controller and the ground station are considered sensor values. The RC receiver has 6 channels, two of these channels are used for configuration (active/killed and manual/autonomous) and 4 are used in manual mode for piloting the craft. When in manual mode, the quad does not use any data from the ground station and uses only internal sensors to attempt to follow user commands (roll, pitch, yaw, throttle). In this mode, the quad will drift around the room at speeds dependent on the pilot, the calibrations of the sensors, and any air drafts in the room.

2.1.3 Controls

The controls for the quad are currently implemented using nested proportional-integral-derivative (PID) controllers. There is a set of PIDs for each of the three Cartesian components of position (x, y, z) and one for yaw (rotation around the z-axis). These were chosen because they achieve a very configurable approach to quadcopter controls, as modifications to the quad can be accounted for by simply adjusting the various PID constants.

The problem with PID controllers is that they contain almost no information about the system physics, and once tuned to reasonable values control cannot be reliably improved except through modifying the coefficients by hand to meet qualitative judgements. The primary change we wanted was to create a controller based on a physical model of quadrotor actuation, which can serve as non-trivial starting point for future controls research on this platform. Specifically, the plan was to implement an LQR controller capable of flying the quad to prove the correctness of our model and its computed linearization.

2.1.4 Ground Station

The overall architecture of the various components for the ground station will stay consistent but the network architecture, as well as backend functionality, will be improved. The ground station is currently well-designed allowing for a backend server, a frontend for clients to use for communication with the backend, and various clients such as the GUI or CLI (more details on each interface can be found later in the report). The benefits of the system as it stands is that the communication and server are kept with the backend so that clients do not need added complexity to deal with the different objects that are connected. The frontend provides a simple interface that clients can pass data to and get a response as needed. This again hides the backend implementation from the clients and this interface is simplified and provides all functionality that the quad and backend have to offer.

The current communication standard for the MicroCART system allows for formatted packets to be sent between the ground station and the quad both during flight time for regular flight instructions as well as after flights for transmitting flight data logs. The ground station (and therefore, the quad) will be updated so that it will request and receive flight data during operation over the UART connection currently used to send instruction data. This information will then be formatted and displayed to the user on an updated GUI. The user will also be able to choose the type(s) of information they want the ground station to display. This change allows for a higher ease of use of the quad as a research platform as the types of data displayed will be determined by the user.

The next major change involves the integration of safety features regarding the control of multiple objects into the backend. In addition to maintaining the individual position, velocity and orientation data of each of the trackables connected to the ground station, the ground station will also conduct additional checks to ensure dangerous scenarios such as collisions between quads do not occur. . This tracker will loop through all objects and provide them each with position information assuming they are using the VRPN system. This will also bring about changes in the GUI which will consist of a means of warning the user that dangerous conditions will/have occur/occurred.

2.1.5 Continuous Integration

The original Continuous Integration (CI) system ran a suite of tests that performed checks on parts of the quad software, using a set of sockets to simulate the drivers used on the quad. It relied on a basic testing framework, created by a previous team member, consisting of a single assert function. To address these limitations, we plan to add an additional part to the testing procedure to test the controls themselves. This would involve interfacing with a flight simulator and connecting the controls used on the quadcopter to the simulator, with the output of the simulator connected as inputs to the control model and the outputs of the control model connected to the inputs of the simulator to provide throttle levels to the motors of the quad in simulation. Automated tests that integrate with this simulator will also be made to test the ground station. In addition, we plan to replace the testing framework currently in use with a more powerful C testing library, Unity [5]. To do this we will work to convert the existing tests to use Unity.

2.1.6 Quad Software and Tests

To improve debugging, we want to send sensor data and actuator results in real time. We are also giving the user the option to select which sensor data they want information from. The sensor data task has two parts : program the quad to send data in real time, and validate that the data has been sent quickly enough. The more sensor data that is sent, the more it will slow down the quad. To test how much sensor data can be sent at once, our approach is to log the latency and calculate it from there. As for sending the sensor data, we've tried two approaches: using the existing logging framework, and sending data using that framework, or just sending the sensor data directly right after data comes in, using the uart driver. Another possible approach is to create a thread that continuously polls the output from various sensors.

2.2 Design Analysis

2.2.1 Quad Software

In terms of Quad software, we have currently not made many modifications to the system from a functional perspective. We have looked into modifying the way our system boots to allow for multiple different types of sensors as feedback, but to no success yet. one thing I think we really need to implement is a better system of testing. When we attempt to test any changes to the system it can sake several minutes and in turn slow development time significantly. One idea of making a wall plug to power the board and sensors but not the motors as a testing platform instead of the batteries. This would enable faster testing iterations and improve development speed significantly. Our solutions as of now seem to give us strengths in functionality but at the sacrifice of future development time increasing. this is due to hardware acceleration being costly (in terms of time) to modify and test as opposed to a software solution.

2.2.2 Controls

As described in the Proposed Design section, the plan is to implement a nonlinear control in a finite number of linearized segments. This solution will have more precision than the existing PID controllers by computing control signals directly from the theoretical dynamics of the quad. This model will use a very precise representation of the quad obtained from planned work in system identification. To emphasize the point from above, this approach allows for higher precision - and thus speed - than a PID implementation at the cost of being more difficult to configure when the quad changes and having a smaller range of operation if not enough linear segments are included.

2.2.3 Ground Station

We currently have a robust framework and backend with a bare-bones GUI implemented for controlling a single quadcopter. Moving forward we plan on using the backend only modifying what is needed to implement safety features for multiple quads and fix any bugs we find. However, we will focus heavily on GUI development and making our platform one that is extremely easy to work with for demos and research. As defined in 1.6.2 we plan on adding real time flight data transmission, redesigned GUI, and improved multiple object tracking capabilities. Each of these parts will either make research easier to use, take less time to collect data, better review the data gathered, and allow for more complicated and impressive demos.

2.2.4 Continuous Integration

Integration of new features into the system is done through a series of tests ran automatically after every commit in the online Git repository. Tests are written in scripting programming languages such as Perl or Python. The merge request merge is unlocked upon successful run of the test scripts. MicroCART Simulator (MCS) will be a virtual environment for the current virtual quadcopter. Currently, the MCS is in the early stage of development and it is dependent on the successful completion of the quadcopter flight model description. Once completed, we will be able to simulate virtual flight and thus test the controls software along with our current simple software test.

3 Testing and Implementation

3.1 Interface Specifications

The top-level system includes the camera system, the ground station, and the quad. Both the camera system and quad interface to the ground station. The ground station relays messages between the camera system, the quad, and the user. It is necessary for the ground station to relay input to the quad ~100 times per second, and the latency must be less than 10 ms. Onboard the quad, the software interfaces to the sensors and motors through the FPGA hardware design. This hardware design uses memory-mapped peripherals to link the processor and external devices. These interfaces need to be low-latency, with the motor output and sensor data being updated about 200 times per second (< 5 ms of latency).

3.2 Hardware and software

Testing the system involves hardware, software, and integration testing. The hardware tests run in simulation automatically by the continuous integration system, though there are additional tests that must be run manually, on the physical quad system. The simulated tests require Xilinx Vivado, the FPGA design/simulation environment. The software also has tests that are part of the continuous integration system. These tests require a C/C++ compiler, QT to compile the ground station UI, and a machine to run them on. Integration tests consist of running the quad and seeing if it flies correctly. In theory, if any piece is broken, the quad should not be able to fly. Integration tests require the camera system, the ground station, a Wi-Fi-bridge, and the quad.

3.3 Functional Testing

3.3.1 Hardware

The hardware design includes the wiring and components, as well as the FPGA design. The FPGA design consists of many IP blocks attached to the fixed portion of the FPGA, some of which are custom built for our project. The vendor supplied components are assumed to be well tested. The custom blocks are tested using a combination of simulation tests, and software projects.

The electrical components and their wiring are a potential failure point for the quad and are not easily tested automatically. Each component and the requisite wiring to make it function is tested by a flight, operating these items requires a person to be present. There are unit-functional-tests in the project that a user can use to test a single sensor or other device without making intentional use of other devices. To mitigate the risks associated with wiring in a vehicular platform, we have opted to use locking connectors in our design. These reduce the risk of in-flight disconnection, and reduce the maintenance required.

The simulation tests for the custom IP cores test the core functions of the module, without the logic that provides the software interface. These tests can catch and identify behavioral bugs in the module's logic. The software application tests use hardware designs that have as few blocks as possible to test the full functionality of the custom IP block. The general format is that the application asks the user to attach any needed wires, then generates some input for the input blocks, or triggers the output blocks to generate, then evaluates the result. These tests can catch a broader range of errors, but are less capable of identifying the source, and cannot be as easily automated.

3.4 Non-Functional Testing

Due to the nature of this project, functional and non-functional testing often overlaps in nature and scope. While a 5ms control loop time seems like a non-functional requirement, this level of speed for control loops is necessary to keep the quad in the air. Our intent is to allow others to use our solution as a base to modify to their needs, which also makes internal design quality and documentation a functional requirement.

3.5 Process

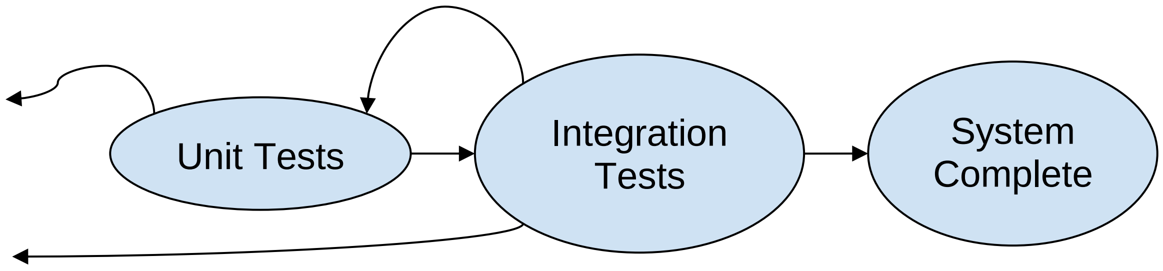


Figure 3-1 Iterative Testing Process

3.5.1 Quad Hardware and Software

The quad hardware and software are tested both together and separately. Each has unit tests that test specific portions of the system (communications packet format, PWM output timing) and can be tested together by testing the flight-capability of the quad. The hardware unit tests are done with VHDL testbenches under simulation in Xilinx Vivado. These can be run automatically and have scripts that can generate a failure if the simulation reports a problem. The quad software has built in test cases that can be used if the quad software is compiled as a test build. Doing this removes the dependency on the hardware of the FPGA design and allows the tests to be run on any machine. In addition to these tests, there are single-application, minimally integrated tests that allow a user to run a test that includes software and hardware, but only as needed for a specific subsystem (such as checking that the software can read IMU data).

3.5.2 Controls

To test a control system, there are basically 2 methods: simulate the design, and run it on the real system. Simulating the design is safer but requires an accurate model of the quadcopter which is built on multiple physics-based equations. Running the controller on the real system does not require a model of the quad, but if the controller does not act as expected or if the actual system differs greatly from the model the controller is based on then the system may fail, sometimes catastrophically. We strive to do as much testing of the controller in simulation as we can, using tools such as Matlab, and take the appropriate safety measures (tether and maintaining a safe distance) when we do need to fly the quad.

3.5.3 Ground Station

Due to the stability of the backend and VRPN setup already present in the ground station code, no large changes to the overall functionality of the backend are intended to be made. Instead, pre-existing and pre-tested basic commands are to be used in any and all extended functionality involving communication with the quad. This allows testing for the Ground Station backend code to be verified via simple system performance tests with a predetermined stable build on the quad.

Frontend changes to the UI and their effects are readily and easily determined due to the visibility of the UI and changes to the UI code in the QTCreator framework and can thus be verified as such.

3.6 Results

3.6.1 Quad Hardware

The quadcopter's hardware has encountered several failures and minor problems. During our initial demonstration, we had a power failure to the IMU due to faulty/loose wiring. None of the current team considered that the wiring might be loose, and the debugging process took a long time. In the end, we ran the demy by unplugging and re-plugging the power cable to the IMU. As we developed hardware tests, we encountered some minor problems in the PWM capture and generation timings. These resulted in small errors (ranging from 1 to 18 clock cycles) that would not have been noticeable in the integrated system but were nonetheless an error. All discovered errors in the tests have been resolved. The biggest problem with the hardware testing has been integrating it into the CI framework. This required ETG to update/reconfigure the machine used for automated testing.

3.6.2 Controls

Our tests of the quad's PID controller have been successful, though we have not had many practical flight tests. Our demonstration at the ECpE scholars fair was generally a success, barring a delay due to electrical failure.

The previous MicroCART team worked to create an operational LQR controller that would be usable as an alternative controller for the quadcopter. While they were largely successful in the completion of its design, hardly any actual testing has been completed on the new controller. We are taking steps to reduce risk to the quadcopter in the event that the LQR controller does not function by first attempting to test it using a simulator and by observing if its behavior in Matlab matches the behavior we expect from it when given specific inputs. Ultimately, the usefulness of this approach is somewhat limited due to uncertainty surrounding the accuracy of the model quadcopter in the simulator and the equations used to model the quadcopter's behavior. If there is enough error in our model, then we may be unable to catch errors in our controller until we are able to perform tests on the physical quadcopter.

3.6.3 Ground Station

Currently, the Ground Station has issues sending large packets while still maintaining performance standards. This is due to insufficient levels of bandwidth available to send data during a standard control loop. This will eventually cause problems as transferring additional flight data during runtime (thereby satisfying the real time transfer of flight data requirement) will require larger or more packets to be sent. In order to mitigate this, it will become necessary to run tests on the quad's and ground stations UART connection to determine the maximum amount of data that can realistically be transferred while still maintaining performance and runtime standards, and then utilize this data to send an optimal size/number of packets per control loop.

4 Closing Material

4.1 Conclusion

Overall, the purpose of this project is to provide a stable and accessible research platform for graduate students to test their controls and embedded systems algorithms, as well as be a demonstration piece to show in departmental demos. In order to create as useful a platform as possible, the best course of action is to continue increasing the stability and dependability of the features that already exist on the quad, as well as introduce key new features that are necessary for researchers and demonstrators to complete their work. Maintaining focus on these key areas will be essential for creating an effective and useful research platform for many graduate classes to come.

4.2 Appendices

Project Repository: https://git.ece.iastate.edu/danc/MicroCART

4.3 References

[1] "Virtual Reality Peripheral Network," [Online]. Available: http://vrpn.org/. [Accessed 29 November 2018].

[2] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,* IEEE Standard 802.11, 2012.

[3] M. Rich, "Model Development, system identification, and control of a quadrotor helicopter," Iowa State University Digital Repository, 2012.

[4] D. Wehr, "ESP8266 WiFi Latency Testing," 17 September 2016. [Online]. Available: https://docs.google.com/document/d/1VU99wMgkqK2EgbNLdqrdhvj9iikfqk2gtUYQ367K5-Q/edit#heading=h.soog8emj18jx.

[5] I. McInerney, "Development of a multi-agent quadrotor research platform with distributed computational capabilities," Iowa State University Digital Repository, 2017.