MicroCART Final Report

ECpE Senior Design Team sdmay19-20

Team Members: Anthony Bertucci, Sarah Koch, and James Talbert
Advisor: Dr. Philip Jones
Project Git Repository: git.ece.iastate.edu/danc/MicroCART

Table of Contents

Table of Figures

Terms

IMU	Inertial Measurement Unit - A device capable of determining the rotation of, and forces acting on, the sensor. Used to determine orientation of the quadcopter.
LIDAR	Light Detection and Ranging - A sensor that uses a laser to determine the range to the nearest object directly in front of it. Used to determine elevation from the ground.
Optical Flow	A sensor using a camera to determine the motion of the quadcopter in the horizontal plane, relative to the ground.
PWM	Pulse Width modulation - A signal coding that uses the timings between edges in the signal to convey a value

1 Project Summary

The Microprocessor Controlled Aerial Robotics Team (MicroCART) project is centered around the development of a quadcopter (see Figure 1 below) and camera tracking system. This project has been in development since 1998 and the current system has been passed down since 2006. MicroCART aims to create a stable and easy to use platform for controller design, testing, and research. The quadcopter flies primarily in the Distributed Sensing and Decision Making Laboratory (Coover 3050) within a twelve-camera infrared tracking system.

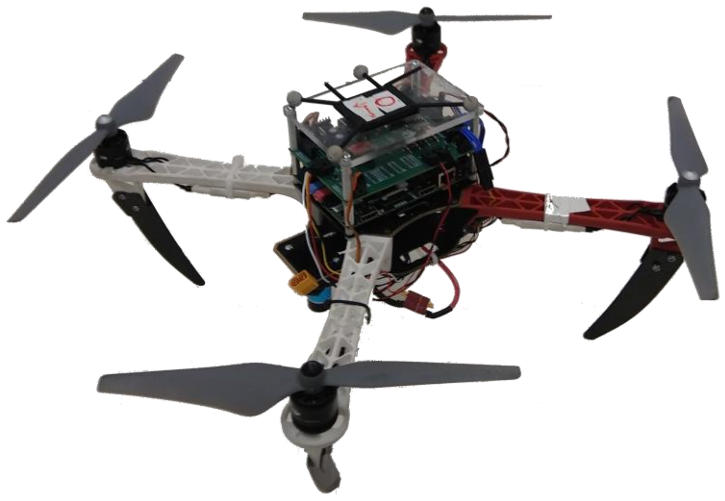

Figure 1: The MicroCART Quadcopter

The focus of the 2018-2019 MicroCART team was the development of multiple features intended to improve the usability of the system. To begin, the hardware platform on which the quad is built was upgraded to a newer version of Vivado in order to take advantage of modern advances in embedded system on a chip designs. A new PCB shield was fabricated to better organize the wiring and sensors located on the quadcopter, as well as provide power and signal debugging capabilities. In order to provide controls researchers with intuitive feedback on controller performance, the data-logging tool was upgraded to display graphs of real-time flight data. Two quadcopter mounts, capable of physically restraining the quad and prohibiting motion except along desired axes, were manufactured to aid in the characterization of controller behavior. In addition to improvements in physical controller testing, the Simulink tool was updated to include manual-mode flight simulation capabilities and documented to allow future users to navigate the simulator and utilize it for testing the behavior of new controllers. Finally, a new structure for the quad’s software was proposed which would create modular subsections for sensor and actuator data, communications, and the control algorithm. When implemented this will allow different controllers to be easily swapped into the quad software and make the platform more effective and accessible as a means of testing multiple controllers.

It is our hope that these additions serve to make MicroCART a more stable platform for research as well as allow future senior design teams to more easily implement their own changes to the project.

2 System Design & Development

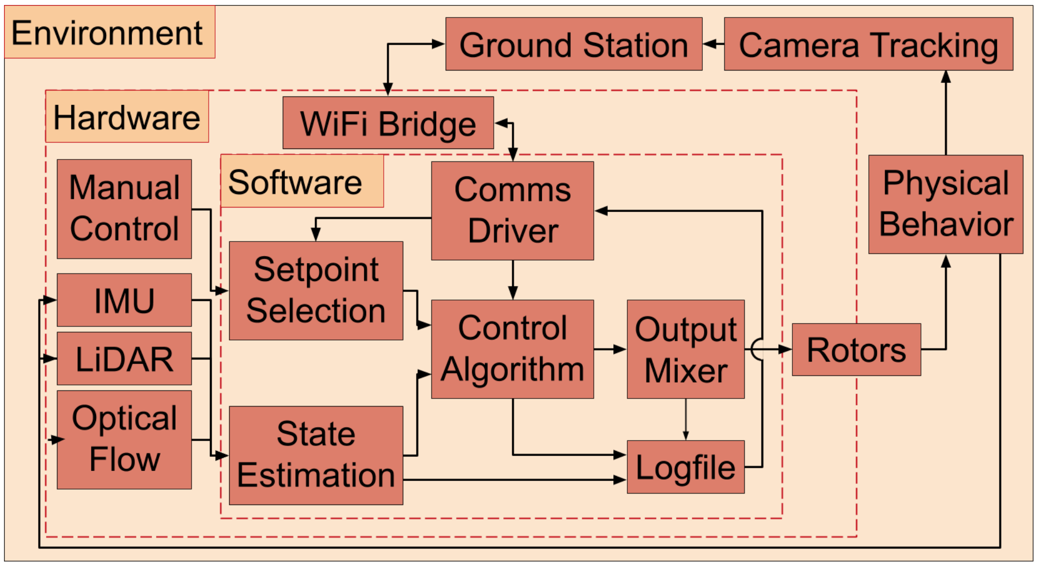

Figure 2: High-level System Diagram

The system is designed as nested feedback control loops, with the outer loop being formed by the camera system, ground station and quadcopter, and the inner loop being formed within the quad by internal sensors, the controller, and the rotors. The quadcopter can be operated without the outer loop in manual mode, allowing for limited control with just an RC controller to pass commands from the user.

The groundsation software is implemented in C++ with the GUI developed with QT5. It is split into the backend, which relays data between the cameras user(s) and quadcopter and the user interfaces which allow a user to execute commands and/or view state from a script, or a graphical environment. The connection between the ground station and the quadcopter is a WiFi link, allowing for low data latency.

The quadcopter uses a FPGA development board as the compute system, providing opportunities for research users to test accelerator designs on an embedded control system. The control software uses a bare-metal C application with the main control loop running at 200 Hz. Camera data is updated at 100 Hz, but the internal sensors are sampled at the full 200 Hz. The quadcopter software uses a hardware abstraction layer that allows us to run tests in a simulated environment through the use of mock hardware drivers on a Unix platform. The use of a system ‘database’ with independent structures for different system components including hardware drivers, control algorithm data, sensor and actuator information, and user input allows for functions to be explicitly given only the relevant information. The physical signal routing and power regulation are provided by a custom PCB that utilizes latching connector housings, preventing accidental disconnects or loose connections.

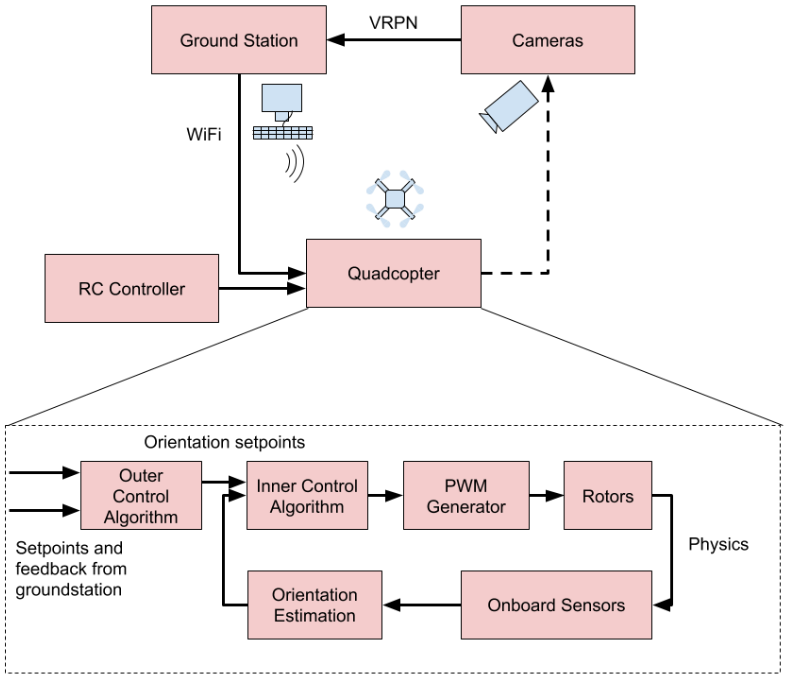

Figure 3: A view of the system as nested control loops

When operating in manual mode, the outer control algorithm is unused, and the orientation setpoints are taken from the RC controller. In addition, there is an override function that allows for characterization of the physical quad’s mathematical model through the use of test stands. Mounting the quad into the test stands allows the operator to constrain 5 of the 6 degrees of motion. Doing so, allows for tests that could not be performed safely in the air while flying, such as commanding a forward tilt of the quadcopter and checking how well it handles disturbances. Two such mounts have been built, one allowing for rotation about the vertical axis (yaw), and one for rotation about a horizontal axis (pitch or roll).

3 Implementation

Ground Station

The ground station consists of a backend, frontend, command-line-interface (referred to as CLI), VRPN network connected to the quad itself and user GUI that interact with one another in the following way:

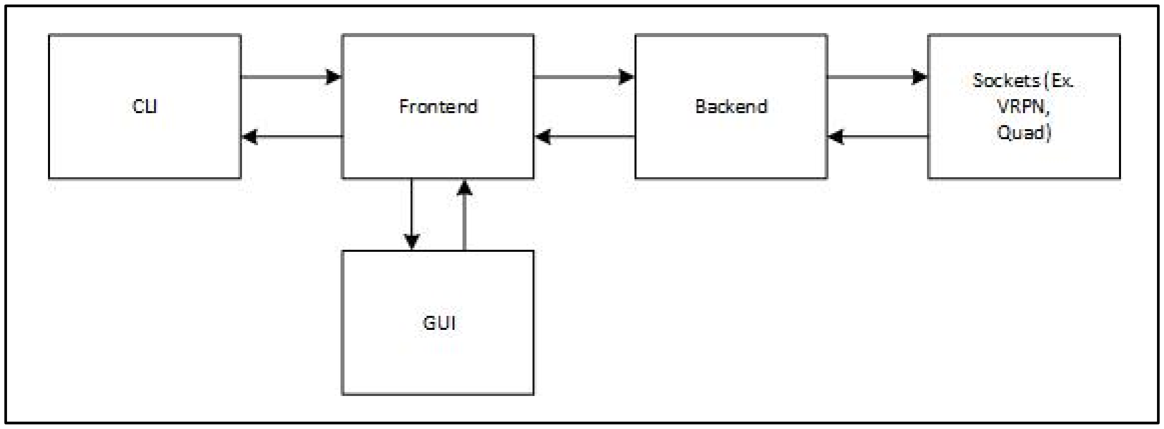

Figure 4: A representation of the ground station interfaces

After data is sent to the local VRPN from the quad, this data is packaged and delivered to the backend (implemented in C). This backend decodes messages, sends commands back to the quad and, when necessary, sends updates of information to the frontend. The frontend, also implemented in C), receives information from either the CLI or the GUI (depending on user preference) and sends this packaged information to the backend. The CLI consists of a shell that reads in simple single line commands from the user, allowing for the implementation of scripts of commands to be sent to the quad. The GUI was designed in C++ utilizing QtCreator, and involves the display of more complex data and data analysis to the user as well as allows for the user to send new commands to the quad.

A main area of development this year was to enable the transfer of flight data to the ground station in real time. When the user requests real-time data, the GUI starts a background MATLAB process that can run continuous analysis on files which are populated by data that is continually streamed from the quad and sent over the VRPN to the backend. This MATLAB process can then parse the file in order to create detailed graphs of flight data as it arrives. The addition of real-time data graphing makes the MicroCART platform more convenient and practical for researchers since they can now visualize their data and infer meaning from it immediately.

Quadcopter Hardware

On-board the quad, the sensors and actuators use a combination of custom and pre-built hardware blocks to interface to the software. The on-board sensors (IMU, LIDAR, Optical Flow) interface using the built-in I2C controllers, and the RC controller and actuators use custom PWM-handling hardware blocks. Software interfaces to these blocks using memory-mapped registers. The IMU is capable of some internal filtering of accelerometer and gyroscope data, but all sensors go through a filtering and sensor fusion process in software after being sampled. The control algorithm uses the setpoints and position data from the groundstation to determine the desired orientation, and then uses the on-board sensors and ground station feedback from the cameras to achieve that orientation. For example, if the quad is too far forward, then the quad will decide to tilt backwards, which will cause it to drift back. In order to tilt backwards, it will increase the power to the front two rotors until the quad is at the correct angle. As the quad approaches the desired position, it will begin to level out, and eventually will hold position at the setpoint. A visual representation of this can be seen on the next page.

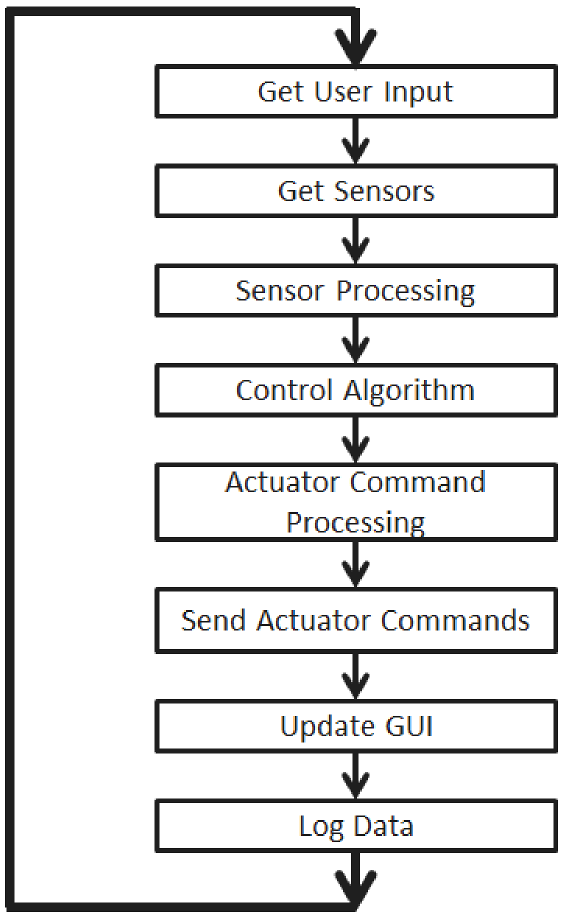

Figure 5: The quadcopter software flow.

Custom PCB Shield

The PCB that routes the signals between the sensors, actuators, and control board allows for easier maintenance and debugging. Every signal connection to the Zybo control board has a test point, so that the signals can be checked without interfering with the operation, and the IMU has a direct physical connection to the board, reducing vibration. Each sensor/actuator has a dedicated, keyed connector, prevent reverse-plugging. The board also has a voltage divider connecting the battery and an analog input channel, allowing for software monitoring of battery voltage. Schematics and design files are available in the project repository.

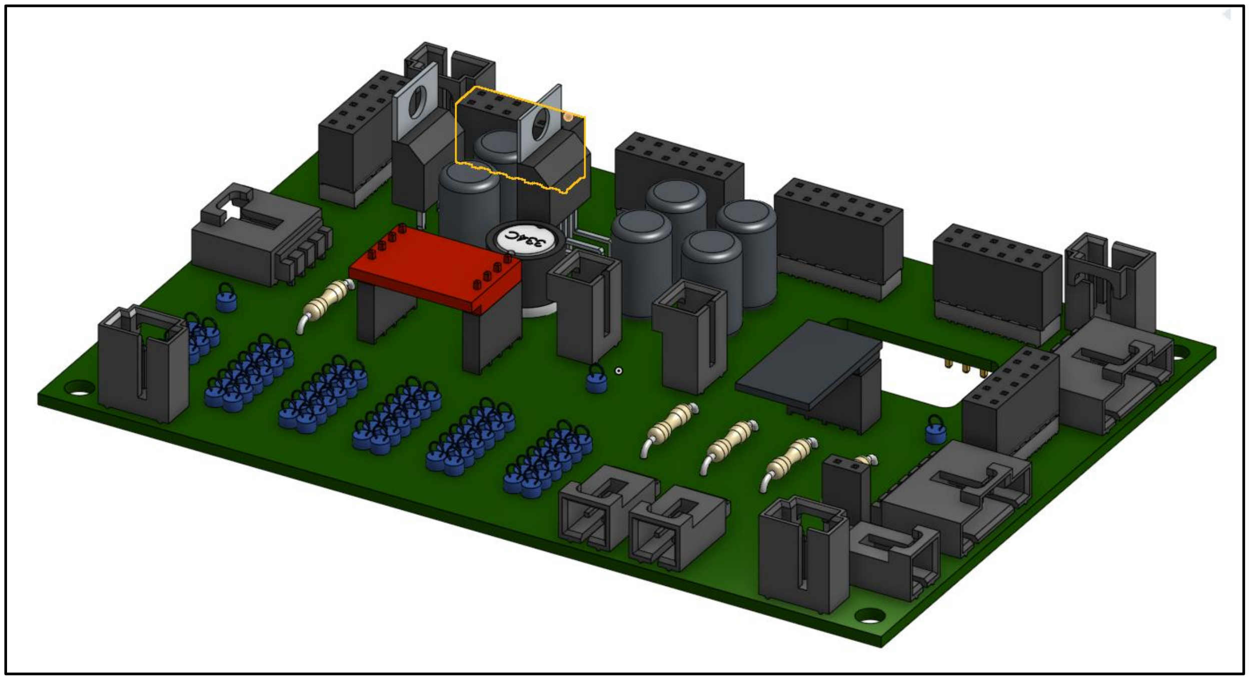

Figure 6: A CAD rendering of the assembled shield.

Controller Simulation

Test points on the PCB, such as those for PWM signals being sent to each of the four motors, can also be cross-referenced with simulated signals generated through a Simulink-based model of the quadcopter. The simulator is can be broken down into 5 different subsections: inputs, the control algorithm, actuators, sensors, and the camera system. Each subsection is represented by physics-based equations that describe the properties of the physical system,the details of which can be found in Matt Rich's thesis.

Inputs can be either set point locations (for autonomous flight mode simulations) or RC inputs (for manual flight mode simulations) and are specified by the user through a series of combined step inputs before running the simulation. In the case of autonomous mode, step inputs can correspond to positions along the x, y, and z axes, as well as to yaw angle. For manual mode, step inputs correspond to correctional movement along the x, y, and z axes, as well as for yaw correction. The inputs are sent to the control algorithm along with the most recent sensor data, which also includes location data from the camera system when running autonomous flight mode simulations. Currently, the simulated control algorithm can be either a PID or LQR controller depending on user selection. The control algorithm then sends output commands (PWM signals) to the actuators (quad motors), which results in a response from the simulated quadcopter. Sensors observe the quad’s response and send updated data to the control algorithm, creating a closed-loop system.

5 Testing, Validation, and Evaluation

Controller Implementation

Running the simulator creates a 3D visual rendering of the flight, which the user can watch to gain a more intuitive understanding of the simulated quadcopter behavior. This feature may be of particular interest while designing new controllers as it is a simple method for detecting flight instability and evaluating performance. After running a simulation, the user can also view a data log of the commands sent by the control algorithm to the actuators. The values contained in this log can be compared with voltages measured on the motor PWM test points on the PCB shield of the real-life quadcopter. As the motors can remain disconnected while probing the PCB test points, this cross-validation feature has the distinct advantage of allowing the user to validate that the controller on the quad has been correctly implemented through software without needing to risk damage to the quadcopter through flight tests.

Quadcopter Mounts

To facilitate the safe operation of the quad during testing and to collect characterization data about the quad, a set of mounting clamps and a test stand were developed in addition to the existing stand. The clamps allow for easy insertion/removal of the quad from the test stands, and the stands constrain 5 of the 6 degrees of freedom of the quad. The yaw-test stand allows for rotation about the vertical axis only, and the pitch/roll stand allows for rotation about either horizontal axis. To determine the response characteristics of a controller to a roll, pitch, or yaw command the quad is mounted in the stand and given the command, then the response is observed with the camera system and the quad’s positional data is logged for evaluation. If there is an error in the controller, and the quad does not behave correctly, it is observable through the flexing of the stand, but the quad does not pose a safety hazard to those in the lab. Additionally, the quad itself can be characterized by setting input to the motors directly and observing its movement. An output override command was also developed to facilitate this.

6 Project and Risk Management

Flight-based projects, as well as those with autonomous navigation features, carry an inherent risk of accruing damage each time they operate. Because both of these features are implemented by MicroCART, the project has good reason to invest in features that will allow users to perform tests with minimal to no risk of damaging the system. The following measures are currently in place to minimize risk to the system and its operators:

- Tethering the quadcopter to limit the flight range during flight tests
- Using testing mounts to hold the quad in place and only allow motion along a desired axis
- Using the Simulink simulator to view the performance of new controllers
- Using simulated actuator data to validate new controllers have been correctly implemented in the quadcopter software

The MicroCART project uses custom components, such as the quadcopter PCB shield, and expensive equipment, such as the camera tracking system. Reducing the risk of damage increases user safety and reduces time spent on fixing broken parts, but it also protects the financial investments of the ECpE department.

7 Conclusion

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. By improving the various systems that make up the MicroCART project as a whole and increase its usability by both researchers as well as future senior design teams, we have brought the platform closer to completing this initial goal. Maintaining focus on these key areas will be essential for creating an effective and useful research platform for many graduate classes to come.