Otto: The Personal Cameraman

Seth Kielbasa, CSE, Albion Lici, CSE, Noah Portnoy, CSE, and Andrew Sousa, EE

***Abstract*—Otto is the personal cameraman that introduces a new way to capture your life's most exciting moments. The system is an autonomous quadcopter that is designed to follow and record a user performing an individual action sport. By maintaining a visual lock on the user during his or her performance, Otto is able to capture the entire experience through an on-board high resolution video camera. Once finished, the user can gather video recordings from the drone and share them with loved ones.**

I. Introduction

HUMANS are social beings that yearn to share their experiences with family and friends. Some of the most exciting experiences to share are those that include extreme circumstances. Individual action sports, such as skiing, wake-boarding, mountain biking, and skateboarding, are activities that people truly enjoy capturing and sharing. Filming these moments is extremely hard since the sports are done at high velocities or in relatively dangerous conditions. We have set out to eliminate this challenge for amateur extreme sports performers.

Previously, this challenge has been addressed by various subpar solutions. Some performers hold a camera [1] while doing their action sport in order to capture the moment. This solution is dangerous because the performer is putting some of their focus on filming and not necessarily on their own actions. Another option for the performer to attain a recording of himself is to have someone ride or otherwise move alongside him and record him [1]. This solution is even more unsafe because it requires the cameraperson to move with the performer at a presumably high rate of speed and focus almost entirely on the recording. There are also several products that have the same goal of recording action sports performers. These solutions attempt to track the user with some combination of GPS localization, additional wearable hardware, hardware connected to the user's phone, and long-range Bluetooth communication; our solution is different in both its implementation and operation. Furthermore, these products, such as AirDog and HEXO+, are in the development stages and are not yet on the market. Thus, action sports performers, of which there are 4.88 million in the United States [2], have not yet found the appropriate solution for capturing all of the amazing things they do.

Otto is the personal cameraman for capturing and recording amazing third-person aerial images. This product will make the recording process safer for action sports performers and enable them to capture a unique aerial view of their performances. Additionally, it will deliver functionality that has never been feasible for everyday consumers. This technology could have applications in many fields, including medicine, military, and home security. Along with these applications that may be beneficial to society, there are opportunities for misuse as well. Criminals could use this technology to follow people, enabling them to stalk others in an obscure way. We attempt to curtail this by making the user's smartphone both the controller and the tracking device.

In order to deliver this ambitious prototype, we developed requirements and constraints to keep our project within the proper scope. In our development of these requirements, we wanted to ensure that the user can pay no attention to Otto during the recording and focus entirely on his or her performance. From this, we concluded that the system must initiate and maintain a visual lock on the user throughout the entire performance. We also require that the system must have safety features to minimize the possibility of injury to the user when operating Otto; the system thus has safety locks on both the smartphone application and the quadcopter itself to prevent undesired liftoff. A full list of our requirements can be found in Table IV of Appendix B. Along with these requirements are the quantitative system specifications that Otto must abide by; these can be found in Table I below.

TABLE I
System Specifications

Specification	Goal	Actual
Maximum drone/user separation distance	30 m	30 m
Minimum drone/user separation distance	5 m	10 m
Average flight time (fully charged)	10 min	10 min
Maximum speed of drone	30 mph	20 mph
Maximum angular velocity of drone in yaw	1.8 rad/s	1.8 rad/s
Minimum quality of video recording	720p30	720p60
Total drone mass	< 1500 g	1400 g
Throttle level required to lift the drone	50% throttle	64% throttle

II. Design

A. Overview

Otto introduces a new way for amateur extreme sports

S. K. Author from Westfield, MA (e-mail: skielbas@umass.edu).
A. L. Author, from Revere, MA (e-mail: alici@umass.edu).
N. P. Author from Newton, MA (e-mail: nportnoy@umass.edu).
A. S. Author from Dartmouth, MA (e-mail: acsousa@umass.edu).

performers to film themselves. Otto is a drone, more specifically an autonomous quadcopter, that can track and record a user. To do this, we needed to start with a robust foundation; details about the drone system's hardware are covered in Section B. The tracking software and hardware both reside on this drone platform. The drone hovers and flies using flight control software developed by the theam; more information on that can be found in Section C. The FollowMe feature, which is covered in section D, allows the drone to follow and track the user. Finally, the mobile communication system allows for reliable communication between the user's smartphone and the drone; technical details of this system are covered in section E. Within the mobile communication system, the mobile application running on the user's smartphone provides the user interface through which the user can operate Otto.

The tracking functionality relies on the user keeping an Android smartphone on their person throughout their performance. The user can open the Otto application on their smartphone and press the on-screen Takeoff button. The Android application will disable the Takeoff button if conditions are unsatisfactory for flight; see section E for more information about this feature. Once the Takeoff button is pressed, the drone will commence liftoff and rise to the appropriate pre-defined altitude. Then FollowMe will initiate, and Otto will immediately proceed to follow the user while maintaining a pre-defined drone-user separation distance. Otto will also turn to face the user while following her, so that the user may be in the video recording. When the user has finished her performance, the user may press the Land button within the app; at this point, Otto will slowly descend at its current GPS location and decelerate its motors upon impact with the ground. Once Otto is on the ground for a few seconds, it will shut off its motors entirely.

The FollowMe feature works by leveraging GPS modules on both the drone and the user's smartphone. With knowledge of the user's GPS location relative to the drone, FollowMe is able to command the following of the user. An additional component of the FollowMe feature, camera tracking, was planned and developed but ultimately never integrated into the final system prototype; see Appendix A for more information about the camera tracking system. As it stands, the FollowMe feature calculates actions for the drone to perform and outputs flight commands to the flight control software in order to achieve those actions. The FollowMe feature is visually represented in the Otto Block Diagram; Figure 1.

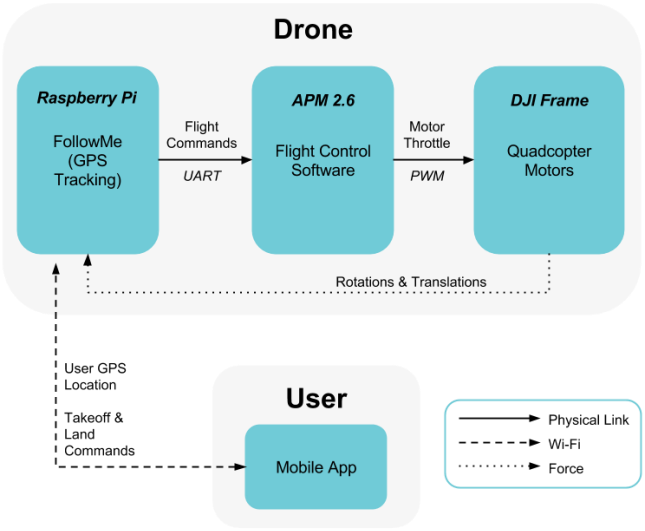

Fig. 1. A high-level view of Otto's main components.

B. System Hardware

First, we introduce the system's hardware. The airframe of the quadcopter is the DJI Flame Wheel F450, which is made of a hardened plastic material to ensure a rigid flight with enough strength to endure high impact landings [3]. To provide thrust for this frame, we have four SunnySky brushless DC motors which have a 980 kv rating; this means that for every volt applied, the motor will attempt to produce 980 revolutions per minute (RPM) [4]. The motors rotate 10x4.7 inch carbon fiber composite propellers. This motor-prop combination ensures that there is enough upward force to allow the drone to hover at 64% throttle, where the drone's total mass is measured to be approximately 1400 g. This has been thoughtfully designed so that at approximately 80% throttle, the motors can quickly maneuver the drone. Each motor is operated by a three-phase signal; the system generates these signals locally with electronic speed controllers, or ESCs.

We chose four ESCs with the SimonK firmware loaded on them, as they seemed to be the most promising consumer ESC offering today [5]. The ESCs receive a pulse-width modulated (PWM) signal from the flight control board, which tells the motors the angular velocity at which they should rotate. In turn, the ESCs output a polyphase signal to rotate the 14-pole DC motors. The flight control board is the APM 2.6 board manufactured by 3D Robotics. It contains all of the necessary hardware for flying: a 3-axis gyroscope, an accelerometer, and a barometer.

Both the FollowMe feature and the flight control software run on the APM 2.6 board, where the FollowMe feature sends pitch, roll, and yaw commands to the flight control software. In order for the FollowMe feature to provide these commands, it must be aware of the drone's location. Thus, there is a GPS module on the drone, specifically the u-blox LEA-6H GPS module with a Taoglas patch antenna [6], that is connected to the APM 2.6. See Section D for more information about the

FollowMe feature.

The core hardware component of the mobile communication system is the Raspberry Pi Model B+ on-board the drone [7]. The Raspberry Pi serves as a means of relaying messages between the user's smartphone and the flight control board. In order to establish a Wi-Fi communication link between the user's smartphone and the Raspberry Pi, a 2.4 GHz Wi-Fi module is attached via USB to the Raspberry Pi. More about the mobile communication system can be found in Section E.

The drone is powered by a 5200 milliampere-hour (mAh) lithium-ion polymer (LiPo) battery [9]. This battery was chosen such that the drone can meet the ten minute flight time requirement. We previously estimated that this battery would provide 11.4 minutes of flight time (see "Mixed Flight Time" in Figure 6 of Appendix B for calculated results), and we have found in practice that the drone is capable of 10 minutes of flight time or more on a fully-charged battery. Power from the battery is distributed to the four on-board ESCs. Each ESC then converts that power, as well as the input from the flight control board, into a three-phase power signal which is output to its respective DC motor. Each ESC also has a battery eliminator circuit (BEC) which outputs 5V 2A DC, although these go unused in our final prototype. An additional fifth ESC, also powered by the battery, has its own battery eliminator circuit, which serves as a dedicated power supply for both the flight control board and the Raspberry Pi. The Wi-Fi module attached to the Raspberry Pi is then powered by the Raspberry Pi itself.

Finally, there is a high-resolution video camera on-board the drone: the GoPro Hero camera. This device has an internal battery as an independent power source as well as its own data storage. The GoPro camera is mounted in line with the drone's nose and is used to capture the high-resolution video recording of the user.

C. Flight Control Software

The flight control software resides on the APM 2.6 flight control board and maintains constant knowledge of the orientation of the drone. Using readings from on-board gyroscopes and accelerometers, the system deploys a set of algorithms to determine appropriate values to feed the four individual ESCs, which then update the motors. The software has been built from scratch and utilizes libraries obtained from the open-source ArduCopter project; these libraries primarily serve as a means of reading from the multitude of sensors on our flight control board.

This crucial subsystem was developed for the 3D Robotics APM 2.6 flight control board [11]. This Arduino-based microcontroller contains an MPU-6000 sensor chip that features three pairs of gyroscopes and accelerometers, one pair per axis. An Integrated Development Environment (IDE) tailored specifically for the APM board was the site of all flight-related software design and testing.

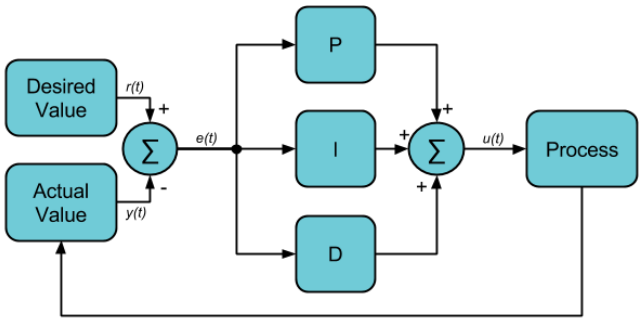

Fig. 2. A block diagram representation of the PID control algorithm [10].

The algorithm centered at the heart of the flight control software is the proportional-integral-derivative, or PID, controller algorithm. Figure 2 provides a visual representation of the PID controller algorithm; this is used a total of six times in the flight control software, where each instance can be abstracted as a "PID block". Figure 7, which can be found in Appendix B, provides a visual representation of the flight control software handling incoming data and converting it into values capable of driving the motors [10]. Each attitude PID block takes in a desired value that is compared to the actual value of the accelerometer. This first calculation is then input into an attitude rate PID block to be compared against the values captured by the gyroscopes. The outputs of these attitude rate PID blocks are then used to adjust the motors in the proper manner to compensate for the measured errors. Each of the three axes (pitch, roll, and yaw) possesses its own set of PID constants to control the rate at which the errors are magnified. These values were finely tuned to keep Otto from oscillating and becoming rapidly unstable in the air.

As Otto has been designed to be totally autonomous from the user's perspective, flight commands are being entirely computed by the FollowMe feature running on the flight control board. When Otto is airborne, the FollowMe feature constantly feeds pitch, roll, and yaw commands to the flight control software. The flight control software then feeds these through the previously mentioned PID control loops, along with sensor readings, in order to calculate the appropriate motor adjustments.

Completion of this subsystem required knowledge obtained through numerous Electrical and Computer Engineering courses that we have completed over the past three years. ECE 353 Computer Systems Lab 1 gave an initial exposure to the C programming language, which is syntactically and functionally very similar to the Arduino microcontroller language used in this project. Additionally, ECE 373 Software Intensive Engineering provided guidance on how to successfully plan the composition of a program. A junior-year ECE design project elective on firefighting robots provided foundational knowledge in robotics engineering and provided valuable experience utilizing sensors and motors with a microcontroller.

For obvious safety reasons, numerous tests were conducted on each new revision of the flight control software before

being used with live motors. The first round of flight software development involved simple scripts to print sensor readings and motor outputs to the console for analysis. Through this technique, it could be confirmed that the sensors were reading appropriate values and that the software was providing reasonable output to the motors. From there, a testing rig was built to allow the drone, now with the attached motors and propellers, to have free motion over one axis. This allowed for visual confirmation that the drone could respond to the manual RC control of either pitch or roll and then stabilize itself with minimal oscillation. With further use of the testing rig, the PID constants mentioned above were tweaked to reduce the oscillation effects. Once the PID control was sufficiently refined, outdoor tests were conducted at low throttle levels before manual takeoff and flight was achieved. While airborne, the spinning or oscillation of the drone frame could be observed, and the software was adjusted to counteract these undesired effects. More recent tests, conducted mostly during the winter months, required large open indoor spaces in which to test autonomous takeoff, landing, and altitude holding functionality. To satisfy this need, the second round of testing was conducted in the Boyden and Totman gyms on the UMass Amherst campus; these were the ideal spaces to develop those autonomous features previously mentioned. The last round of flight testing was the most difficult as it involved testing the tracking functionality, which required decent weather in order to obtain a GPS signal. Fighting the weather, primarily the wind, put a heavy burden on the PID control loops to compensate for the sudden changes of orientation recorded by the accelerometers and gyroscopes.

D. FollowMe

FollowMe is Otto's navigation and guidance system. The goal of this system is to act as the pilot of Otto; it shall control Otto's altitude, attitude, and heading such that Otto will follow the user from a defined separation distance during his or her performance. To accomplish this task, our system leverages three main sensors: the barometer, the magnetometer (compass), and the GPS receiver. FollowMe uses GPS location information about the user and the drone to navigate Otto to the appropriate location. We have broken down the FollowMe system into four main subsystems: altitude control, heading control, attitude control, and autonomous takeoff and land.

D-1. Altitude Control

The altitude control system relies on the barometer to provide accurate altitude information to the system. The barometer is a high resolution altimeter sensor which measures atmospheric temperature and pressure; from these measurements, the barometer calculates the drone's current altitude relative to its takeoff elevation. Our barometer provided an altitude resolution of 10 centimeters [12], although it was only accurate to the meter as altitude readings tended to drift over time.

To understand this subsystem, it must be clear that the altitude of the drone is a function of the throttle, if we assume horizontal flight and ideal environmental conditions. The throttle is the control signal that dictates the amount of power that the motors should consume, which is directly correlated with the amount of lift the motors provide. The altitude control system is a PID feedback control system with a dynamic feedforward component; see Figure 3 below. The input to the feedback system is altitude error and the output is a motor throttle command. The altitude error is the difference between the desired altitude set in software and the current altitude measurement from the barometer. Next is the feedforward component; we chose to use a feedforward component because we were able to determine what the "hover throttle" of our system was. The hover throttle is the amount of motor throttle that will overcome the force of gravity acting on the drone. The hover throttle is a function of the weight of the system and the current battery voltage; as the battery voltage decreases, a higher throttle command is required in order to make the drone hover. Since the weight of the system is known, we were able to hard-code that component of the hover throttle. To account for the battery's voltage during flight, we created a function that takes as input the known system-weight-compensation throttle and appropriately adjusts it based on the current battery voltage. The output of this function is an accurate estimation of the current hover throttle. In this architecture, the feedforward component overcomes the force of gravity on the drone and the feedback loop only has to make small changes in throttle to keep the drone at the desired altitude.

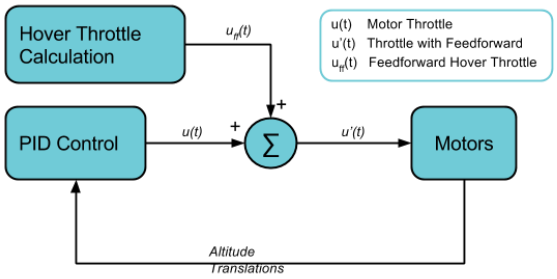

Fig. 3. Altitude PID feedback control with feedforward component.

D-2. Heading Control

The heading control system is the system that controls the heading of the drone; the heading can be thought of as the nose of the drone. This system is designed to keep the heading of the drone pointed to the user at all times. We made this design choice because the video camera is fixed to and aligned with the nose of the drone, so when the drone's heading is pointed directly at the user, the user will be in the center of the frame. This system takes as input the GPS locations of the user and the drone as well as the heading of the drone as measured by the magnetometer. To control the heading of the drone, we use a PD feedback control loop. The input to the feedback system is heading error and the output is a yaw command for the flight

control software to carry out. The heading error is the difference between the drone's current heading, which is read directly from the magnetometer, and the drone's desired heading, which is generated from the bearing. The heading is the direction that the drone is facing, measured in degrees away from North. The bearing is the heading at which the drone's nose would point to the user. The bearing is calculated by performing trigonometry on the GPS coordinates. More specifically, the tangent of the difference in latitude components over the difference in longitude components is used to calculate the bearing. The bearing is updated at a rate of 2 Hz; it is limited by the frequency of received GPS location data for the drone and user. Due to the inaccuracies of the GPS modules on the phone and the drone, we found that the bearing was fairly accurate only when the drone was approximately 10 meters (or more) away from the user.

D-3. Attitude Control

We preface this system with an explanation of the two coordinate systems at play in the attitude control system. In Figure 4, you will see that there are two coordinate systems: one colored blue (xb, yb, zb) and another colored red (xe, ye, ze). The blue-colored coordinate system is the drone's body axis coordinate system; this system changes as the drone rotates in space. The pitch axis is yb with pitch angle theta, the roll axis is xb with roll angle phi, and the yaw axis is zb with yaw angle psi. The red coordinate system is the inertial axis, or earth axis (hence the "_e" convention). It is a fixed coordinate system [14]. Another point to note about this system is that the x-axis (xe) is approximately aligned with the earth's longitude lines and y-axis (ye) approximately aligned with the earth's latitude lines. We say approximately because they are aligned on a small scale (10 to 100 meters) but on a large scale the longitudinal and latitudinal lines are not straight. With this knowledge, the attitude control system can be described.

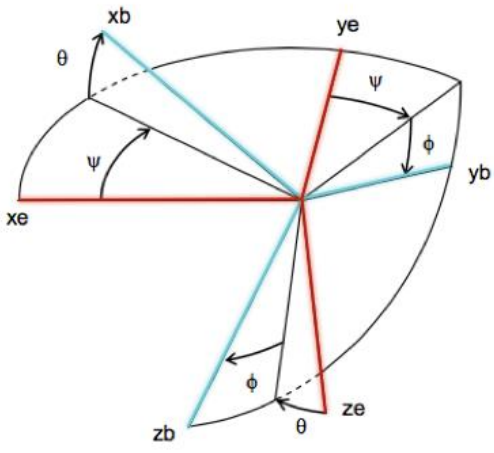

Fig. 4. Body axis to earth axis conversion diagram [14].

The attitude control system controls Otto's pitch and roll angles. Pitching the vehicle will make it move forward or backward and rolling will lead to a leftward or rightward movement, both relative to the drone's current position. Similar to the other FollowMe subsystems, the attitude control system is a PI feedback control system with inputs of desired separation distance and GPS error and output of pitch and roll commands. The first input, the desired separation distance, is the horizontal distance the drone shall be from the user; this is input to the attitude control system as an integer. The two additional inputs to the system are the GPS coordinates of both the drone and the user. These three inputs are used to synthesize Otto's target coordinate, which is the appropriate distance away from the user (the separation distance). To communicate the method of calculating the target coordinate, we can think of the right triangle formed between the two GPS points, where each coordinate is at the tip of the acute angles of the triangle. Each leg of the right triangle respectively has a length equal to either the latitudinal or longitudinal difference between the two points in space. These latitude and longitude errors are then scaled such that the hypotenuse of the triangle is equal to the separation distance, whilst the angles of the triangle stay fixed. These two latitude and longitude errors are then added to the user's actual GPS coordinates to determine the drone's target location. This system is updated at a frequency of 2 Hz as it is limited by the rate at which the GPS coordinates of the drone and user are updated.

Once the target coordinates have been calculated, the error between the drone's current GPS coordinates and the target coordinates can be derived. Before this error is input into the attitude control feedback loop, it must be converted into the body axis coordinate system (assuming horizontal flight). To do this conversion, we multiply the GPS coordinate error by the yaw rotation matrix [15], which can be seen in Figure 5. The input to the matrix is psi, which is the yaw of the drone relative to 0 degrees north; this represents the relationship between the body axis and the earth axis. The output of this multiplication can now be thought of as the x error, which will control the desired roll, and y error, which will control the desired pitch; these values, along with the drone's actual pitch and roll attitudes, are input into the PI feedback system. At the output, pitch and roll commands are sent to the flight control software.

$$\begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Fig. 5. Yaw rotation matrix, where psi is the yaw of the drone relative to north.

D-4. Autonomous Takeoff and Land

The final piece of the FollowMe system includes both the autonomous takeoff and land functions that enable the user to begin and end Otto's flight from the mobile device. Upon pressing their respective buttons on the phone, Otto will receive a signal to change flight states and initiate the appropriate action. If that action was takeoff, throttle will initially be set to zero and then immediately be set to a value slightly above the hover throttle, which as previously mentioned is determined by both the system-weight-compensation throttle and the current battery voltage. As the barometer reading approaches the desired altitude, the throttle is adjusted to be closer to the hover throttle in a linear fashion. Meanwhile, pitch, roll, and yaw commands are being controlled by the takeoff function such that Otto maintain its current location and heading. Upon reaching the desired altitude, the throttle output is set exactly to the hover throttle, allowing Otto to maintain altitude. From here, the FollowMe tracking system takes control and begins to command the following of the user.

The land function was more difficult to implement as it required much more precise maneuvers to successfully and safely perform the correct task. To start the initial descent, the throttle is set to a value slightly below the hover throttle to allow for some downward movement. The z-axis velocity and acceleration relative to the ground are monitored via the barometer and z-axis accelerometer, respectively. Should Otto begin to fall too fast, the throttle is set back to the hover throttle until no downward movement is being detected. Just as in the takeoff function, pitch, roll, and yaw commands are being controlled by the land function during descent to maintain the current location and heading. The z-axis accelerometer is used to detect an acceleration spike in the opposite direction of gravity (a positive value), signifying that Otto has hit the ground. The throttle value is then stepped down at set intervals if the sensors continue to indicate that Otto is no longer moving, implying a successful landing. After enough of these steps have occurred, we can be confident that Otto is stationary on the ground; the land function will then stop the motors entirely, at which point Otto is safe to approach.

This concludes the description of Otto's FollowMe navigation and guidance system. All of the feedback control loops work together to guide the drone through space, continuously pointing to and following the user while maintaining a constant altitude. The integrated FollowMe system can be seen in Figure 8 of Appendix B. To accomplish this task, we used knowledge from many courses across Electrical and Computer Systems Engineering. At the core of this system is feedback control; although none of the team members had studied the subject previously, we certainly learned feedback control "on the fly" and applied its theories to this system. We also used knowledge from Computer Systems Lab I and II (ECE 353 and 354) to develop and debug FollowMe's embedded software. The team also followed the general software development and testing practices that were taught in ECE 373 Software Intensive Engineering.

E. Mobile Communication System

The Mobile Communication System enables the communication between the user and Otto. To establish this communication link, the Mobile Communication System had to be broken down into two sub-components: a phone application and a messaging protocol.

The phone application runs on an Android-powered smartphone. It includes two interactive, "swipeable" screens: the Controls screen and the Diagnostics screen. The goal for this design approach was to make it easy and intuitive for the user to control Otto. The Controls screen presents the user with three buttons: Takeoff, Land and Power. Takeoff is used when the user would like to initiate takeoff of the drone. This button gets enabled and disabled as a function of the status of the system's diagnostics. Things such as the GPS signal accuracy, battery level, and Wi-Fi connection are checked before the button is enabled. The Land button enables the user to initiate the landing functionality of the drone. The Power button enables the user to initiate a shutdown of the Raspberry Pi in order to prepare Otto for a safe system shutdown. Another aspect of the app is the Diagnostics screen, which presents the user with the diagnostics of the system; these include the altitude of the drone, the user-drone separation distance, and the drone's battery voltage, among others. On this screen, the user is also capable of manually pinging Otto in order to ensure that the Wi-Fi link is still fully operational. The phone application by default pings Otto every second by sending a packet using the User Datagram Protocol, or UDP. For every ping packet that the phone sends, the app expects one back. If the app sends a ping packet and does not receive one in response within 3 seconds, we assume that there is something wrong with the Wi-Fi link.

The tasks of the Android application have been broken down into four threads executed in parallel: the user interface (UI) thread, the network send thread, the network receive thread, and the ping thread. Any rendering of visual objects and UI updates are performed by the UI thread. The UI thread is also responsible for spawning all other threads for this application; it can therefore be thought of as the main thread [13]. The network send thread handles all outgoing network data intended for the Raspberry Pi by transmitting messages that have been added to the transmit FIFO queue. The network receive thread handles all incoming traffic from the Raspberry Pi that is being sent to the Android app. Receiving from the network is a synchronous, or blocking, task. Finally, the ping thread is responsible for executing the pinging mechanism between the drone and the Android app.

The messaging protocol enables the system to have bidirectional communication between the phone application and the drone. The most important message communicated in this link is the GPS coordinates of the user; they are sent to the drone at rate of 2 Hz. Because we use UDP, the

communication on the Wi-Fi channel is inherently unreliable. Communication packets containing GPS coordinates might get lost due to collisions, dropped because of a full queue, or other factors; as such, the 2 Hz update rate is not guaranteed. Attempting retransmission will provide lagging coordinates as an input to FollowMe; therefore we ignore any dropped messages. The mobile communication system is designed to be robust, and so does not halt when packets are dropped. However, the system will see a decrease in performance in this scenario. The app communicates with the Raspberry Pi on-board the drone via a Wi-Fi Link operating on the 2.4 GHz band. The Raspberry Pi is connected to the APM flight control board via a UART communication link operating at a rate of 115200 baud. The protocol defined here is ASCII character based and is structured with a start character, a 3 byte message identifier, the message itself, and an end-of-packet character. Table II below shows an example of a message string that the phone app will interpret as the battery level of the drone. Tables V and VI, which can be found in Appendix B, list all message types supported by the system along with a short explanation for each.

TABLE II
EXAMPLE OF SENDING DRONE BATTERY LEVEL TO THE PHONE

Start Byte	3 Byte Message ID	Message	End Byte
$	BTS	11.812	!

Threaded programming and objected oriented design principles used in the development of this system were introduced in ECE 373 Software Engineering. The networking topics used here, such as the User Datagram Protocol, were presented in ECE 374 Computer Networks and the Internet. The Android application was developed using both the Java programming language and the Android API, the latter of which is provided and documented by Google. The Android app was written using an object-oriented design approach. An Android extension tool package was also used alongside the Eclipse IDE in order to make the development of the app more efficient.

III. PROJECT MANAGEMENT

Team Otto is comprised of three computer systems engineers (CSEs) and one electrical engineer (EE) who have diverse backgrounds. Seth Kielbasa has worked with robotics in the past as part of the UMass Amherst firefighting robot team; as a member of Team Otto, he was responsible for the flight control and stabilization algorithms in the first semester and for the takeoff and land functions in the second semester. Albion Lici has completed multiple internships at Teradyne, where he gained much insight into interfaces between hardware and software; this knowledge has proven to be very useful in his work on Team Otto. During the first semester, Albion was responsible for the GPS tracking component of the FollowMe feature which was developed and implemented on the Android device; during the second semester, Albion was responsible for the mobile communication system. Noah Portnoy also has robotics experience as he led the UMass Amherst firefighting robot team for two years. Noah was responsible for the camera tracking component of the FollowMe feature during the first semester. During the second semester, he was responsible for parts of the FollowMe system, namely the altitude control system and components of the attitude control system, as well as the system control logic for the drone. In addition, Noah managed and maintained the project's code base. Andrew Sousa is the EE of the team and he brings robotics experience from his work leading the IEEE Micromouse group to successful completion of an autonomous robot. Andrew is the team manager and was responsible for all of the drone hardware during the first semester; in the second semester, Andrew was responsible for the design and implementation of the FollowMe system (which includes much work from other members of the team).

This semester, we did not follow a standard Gantt chart as it did not seem to work well for many of the team members. Instead, we used an online tool hosted by GitHub that allowed team members to add issues or desired features/functions to the list, assign team members to work on them, and easily set deadlines and goals. This worked significantly better for our team's dynamics.

Over the course of the year, the team worked relatively well together. We all had a clear vision of the final product from beginning to end, though the smaller "demos" in between were less clear and more difficult to come up with. There was always an enlightening discussion at our weekly all-hands meetings with our advisor, Professor Christopher V. Hollot, who continued to ask us thought-provoking questions and help steer the team in the right direction. Our advisor was often so kind as to meet with us at other times to discuss various components of the many feedback control systems that lay within our system. The team communicated almost daily through either an online messaging service or in person; in addition, the team met weekly to discuss each individual's progress from the week. We also shared important information such as data sheets, calculations, and experiment results on cloud-based storage hosted by Google, while we kept all of our code under version control in a GitHub repository. Finally, we conducted large-scale or high-level communication via email. We were in nearly constant contact due to both our project's difficulty level and our commitment to deliver Otto.

IV. CONCLUSION

Following the completion of the Final Product Review, the team has a functional prototype of the original product design. Otto can successfully track a mobile device while keeping the on-board high resolution camera fixed on the user at all times. From the user's standpoint, there is no need for any manual control of flight, other than the takeoff and land functions that are integrated into the mobile app. While the camera tracking system that was presented in our Comprehensive Design Review did not make it onto the final product, Otto is successfully able to perform the actions that were originally set

out to be accomplished.

Overall, this project has been a tremendous learning experience for the entire team as it incorporated numerous engineering challenges that had to be solved throughout the course of the last two semesters. To start, this project was a significant coding challenge, written mostly from scratch with the help of many available libraries. A short range Wi-Fi network was established with a messaging protocol to send information from the mobile device to the phone, a common occurrence these days but not one previously accomplished by the team. An Android application was designed and built to provide user control of Otto, transmit the user's GPS coordinates to the drone, and display key diagnostics that the user may wish to monitor while flying Otto. Finally, the team learned of the many difficulties created by building a project that must cooperate with the weather, as was seen on the final SDP demo day when the wind was too severe to perform a live demo. The weather experienced during the final weeks of this project made it difficult to run the appropriate tests to ensure tracking was operational.

APPENDIX

A. Camera Tracking

The camera tracking system was developed with the intent of controlling the yaw of the drone such that the drone may face the user and thus the user may be in the video recording. If it were integrated into the drone system, the camera tracking system would be able to process video frames from a low-resolution camera and locate the user in the frame by looking for the specially colored jersey that the user would wear during his performance. As described in section D, the FollowMe feature uses GPS data to accomplish the yaw control functionality that camera tracking would have provided. Without camera tracking integrated, the only loss is that camera tracking likely would have provided more accurate data with which to command yaw when the drone is within a certain range of the user.

We chose not to integrate the camera tracking system into the final prototype in order to focus on the more critical components of the project. We did not choose to abandon camera tracking integration because it might be too difficult; rather, we believed that one or more components of the final prototype might end up non-functional if we were to assume the additional task of integrating camera tracking into the system. Following is a description of how the camera tracking works in isolation and how it may be integrated into the system in the future.

The camera tracking system can serve as an additional core component of Otto's FollowMe feature, allowing the drone to accurately keep the user in the video frame. In order for the camera tracking system to uniquely identify and track the user in the environment, the user must wear a distinctly colored jersey. The camera tracking system can attain a visual lock on the user upon drone takeoff by scanning for the color of the user's jersey. Once the visual lock has been acquired and the FollowMe feature has been initiated, the camera system can continuously track the user. If the user begins to veer out of the video frame boundaries, the camera tracking system can send yaw control output in vector form to the FollowMe feature. This output communicates how the drone should reorient itself along the yaw axis to maintain a visual of the user. The FollowMe feature can take this data from the camera tracking system and synthesize it with GPS data to form unified output in the form of flight commands to the flight control software. As of the team's Midway Design Review, camera tracking is a functional, closed-loop system operating at 7.5 Hz that can track an object of a certain color and keep the object in the video frame by commanding a servo motor to rotate the camera along the yaw axis.

The camera tracking software can reside on Otto's main computer, a Raspberry Pi Model B+ [7]. The software is written in the Python programming language and harnesses the OpenCV (Open Source Computer Vision) library to detect the colored object by heavily processing the video frames that are captured by a Logitech C310 USB webcam [8]. Specifically, the camera tracking software performs the following OpenCV transformations [16] on each video frame: cvtColor to convert the image from the RGB to the HSV color space, inRange to get a mask [17] of the image consisting only of those pixels that fall within the desired object's color range, dilate to dilate the shapes present in the mask so as to smooth out the shapes' edges, findContours to detect all shapes present in the image, and contourArea to measure the area of the detected shapes and to select the most prominent shape in the image. From here, the software determines the center of the detected object within the video frame, and continuously checks to see if the center of the object moves outside a programmed set of bounds centered about the middle of the frame. Upon detecting that the object has moved outside of these bounds, the software currently outputs servo commands for the yaw axis over a serial output. The serial output then goes to an Arduino Uno, which sends digital output to the servo. [Note that the Arduino IDE and programming language were used to develop the servo-controlling software that runs on the Arduino Uno.] Once the object moves back within the specified bounds, the software no longer sends servo commands. This simulates how the camera tracking software can send yaw commands to the FollowMe feature to keep the user in the video frame.

Several tests have been conducted to assess the performance of the camera tracking system. The environment for these tests used fluorescent lighting, thus having a color temperature of approximately 3000 K [18], and the camera was positioned to face a white background that filled the entire camera frame. Any object displayed to the camera was always kept approximately two feet away. For these tests, the system was configured to track a red object. The variables in the tests are as follows: (1) the color of the object, and (2) either moving the camera to simulate the effects of drone movement, or keeping the camera stationary. For each test, we monitored the system's performance for two minutes in the same

environmental conditions. Detection rates and false positive rates were calculated using frame counts provided by the camera tracking system. The results are summarized in Table III below.

TABLE III
CAMERA TRACKING SYSTEM TESTS

Test Scenario	Detection (True Positive) Rate	False Positive Rate
Moving red object always in frame, stationary camera	98.7%	0%
Moving red object always in frame, moving camera	100%	0%
No object in frame, stationary camera	N/A	0%
No object in frame, moving camera	N/A	0%
Blue object always in frame	N/A	0%
Green object always in frame	N/A	0%
Yellow object always in frame	N/A	0%

While there are many factors contributing to the performance of the camera tracking system in these tests, we can conclude that the system is capable of tracking a red object under certain conditions with a high degree of accuracy. We can attain similar performance in tracking an object of another color by simply changing the HSV range within which to look for an object.

The system's performance outdoors has been qualitatively tested with the user wearing a bright red jersey, the camera mounted on the drone and connected to the Raspberry Pi, and the drone a distance away from the user that meets the system specifications. We found the outdoor performance of the camera system to be suboptimal, only sometimes detecting the user's red jersey and occasionally detecting other red objects instead (such as vehicle taillights). However, that does not mean the camera tracking system cannot work outdoors; in fact, it may work very well with further adjustments.

One reason why the camera tracking system performed poorly outdoors is that while the color temperature of the indoor testing area was about 3000 K, color temperatures outdoors range from 5500 K to 6500 K [18]. By measuring HSV values for the red jersey both indoors and outdoors, we were able to confirm that the color temperature affects the camera's perception of color. To solve this issue, one can simply modify the HSV range specified in the program so that the range best represents the HSV values of the target; in this case, it would be the colored jersey of choice in both sunny and cloudy outdoor environments. A potential issue still to be addressed is the presence of objects in the environment that are similar both in color and size relative to the user's jersey. However, this issue can most likely be mitigated with a careful selection of both jersey color and HSV range.

The development of the camera tracking system relied heavily on the material covered in several Electrical and Computer Systems Engineering courses. ECE 353 Computer Systems Lab 1 and ECE 373 Software Intensive Engineering together provided a deep and fundamental understanding of software that allowed for the creation of the camera tracking system. ECE 354 Computer Systems Lab 2 introduced image processing and manipulation techniques that were formative in the design of the camera tracking system.

B. Figures and Tables

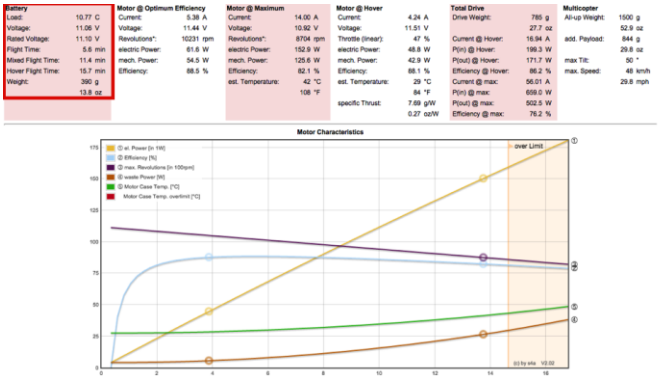

Fig. 6. Estimated performance of the drone based on its motors, propellers, battery, and drone weight. Battery performance estimates, including flight times, are highlighted with a box. "Mixed Flight Time" is a combination of hovering and maneuvering, the latter of which requires more thrust. Calculations provided by [19].

TABLE IV
SYSTEM REQUIREMENTS

	Requirement
1	Track user through a fusion of two sensors: GPS and camera
2	Collect GPS location of user through a Wi-Fi connection to user device
3	Collect finer location data of user through camera tracking
4	Carry out user-defined takeoff and land commands
5	Maintain a user-defined drone/user separation distance
6	Allow user to start and stop video recording
7	Video recording is high-definition (720p or better)
8	Must maintain visual lock on user for duration of recording
9	Drone will take preliminary measures upon reaching critical battery level
10	Safety lock in hardware and software

TABLE V
DRONE TO PHONE APPLICATION MESSAGE IDS

Message Identifier	Message Explanation
LAT	Drone Latitude (double)
LON	Drone Longitude (double)
BTS	Drone Battery Status (double)
ALT	Drone Altitude (double)
CRS	Drone Climb Rate (double)
GPS	Drone GPS Status (int)
GPA	Drone GPS Accuracy (double)
WPG	Wi-Fi Ping Response
SRD	Separation Distance (double)
BRG	Bearing to User (double)

TABLE VI
PHONE APPLICATION TO DRONE MESSAGE IDS

Message Identifier	Message Explanation
LAT	User Latitude (double)
LON	User Longitude (double)
BTS	Drone Battery Status (double)
TKF	User Takeoff (short)
STP	User Land (short)
WPG	Wi-Fi Ping Request
PSH	Raspberry Pi Shutdown (short)

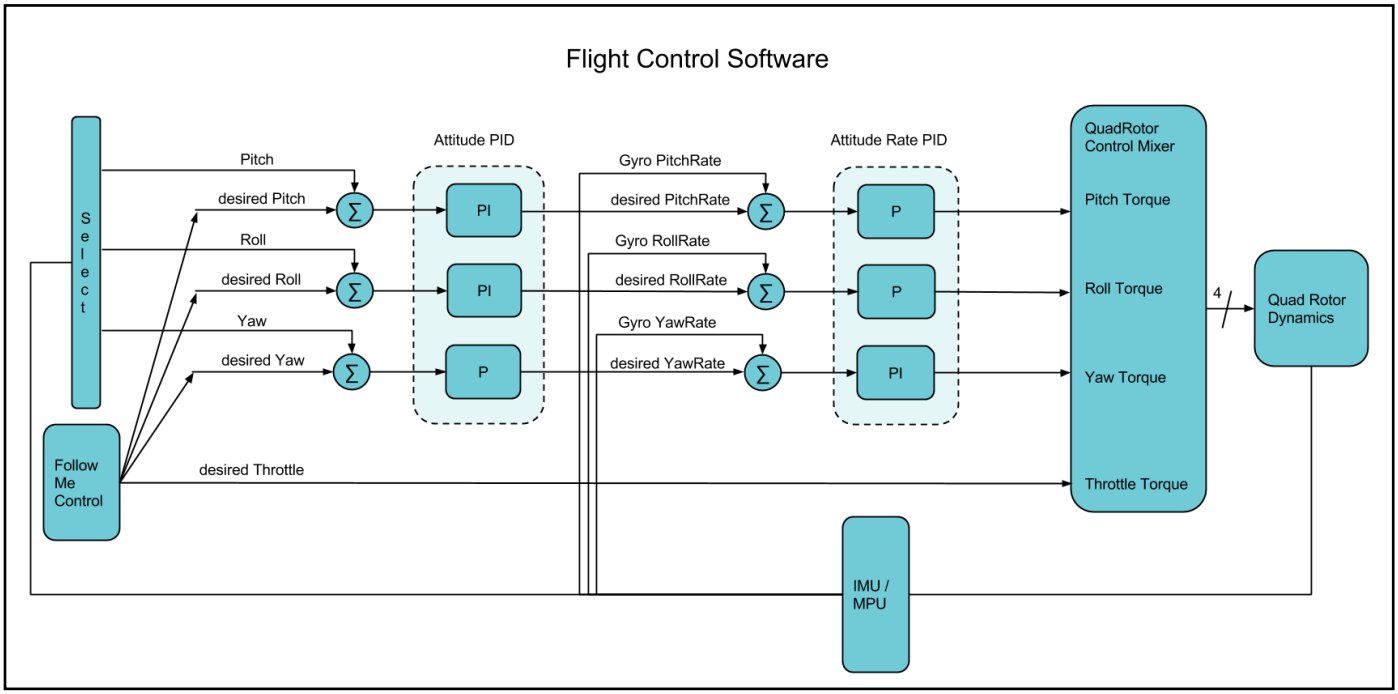

Fig. 7. A block diagram representation of the Flight Control Software.

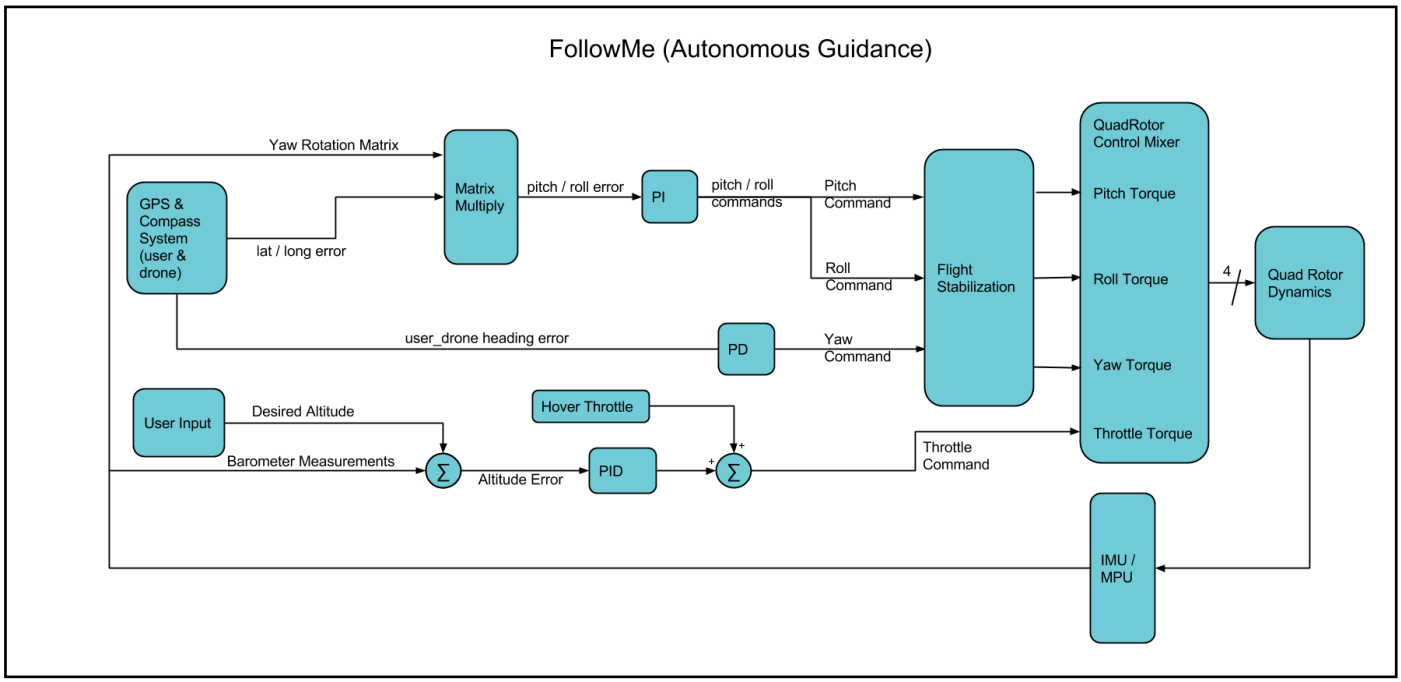

Fig. 8. The complete FollowMe guidance system.

ACKNOWLEDGMENTS

Our team would like to thank our advisor, Professor Christopher V. Hollot, for his excellent advising methodologies and desire to help us succeed. We would also like to thank Francis M. Caron for helping us with all requests related to the Senior Design Project lab and Paul Pounds for providing us with some MATLAB simulations that together served as a basis for our FollowMe guidance system.

REFERENCES

[1] B. Rose. (Accessed 2013, February 14). *How to Get Better Action Cam Footage* [Online]. Available: http://gizmodo.com/5983584/getting-better-action-camera-footage

[2] "Number of people who are very interested in extreme/ action sports in the United States (USA) from spring 2008 to spring 2014," Statista, New York, NY, Stat. Rep., Spring 2014 [Online]. Available: http://www.statista.com/statistics/229006/people-who-are-very-interested-in-action-sports-usa/

[3] DJI Innovations. (Accessed 2014, December 15). *Flamewheel ARF Kit* [Online]. Available: http://www.dji.com/product/flame-wheel-arf/feature

[4] P. Pine. (Accessed 2014, December 15). *What does KV mean?* [Online]. Available: http://www.flyelectric.com/ans.kv.html

[5] (Accessed 2014, December 15). *SimonK ESC User Guide* [Online]. Available: http://www.robotshop.com/media/files/pdf/lynxmotion-simonk-esc-guide.pdf

[6] (Accessed 2014, December 15). *3DR uBlox GPS with Compass Kit* [Online]. Available: http://store.3drobotics.com/products/3dr-gps-ublox-with-compass

[7] (Accessed 2014, December 15). *Model B+* [Online]. Available: http://www.raspberrypi.org/products/model-b-plus/

[8] (Accessed 2014, December 15). *Logitech HD Webcam C310* [Online]. Available: http://www.logitech.com/en-us/product/hd-webcam-c310

[9] (Accessed 2014, December 15). *Lumenier 5200mAh 3s 35c Lipo Battery* [Online]. Available: http://www.getfpv.com/lumenier-5200mah-3s-35c-lipo-battery.html

[10] G. Owen. (Accessed 2014, December 15). *How to Build Your Own Quadcopter Autopilot / Flight Controller* [Online]. Available: https://ghowen.me/build-your-own-quadcopter-autopilot/

[11] (Accessed 2014, December 15). *APM 2.6 Set* [Online]. Available: http://store.3drobotics.com/products/apm-2-6-kit-1

[12] (Accessed 2015, May 6). *MS5611-01BA03 Barometric Pressure Sensor* [Online]. Available: http://www.meas-spec.com/downloads/MS5611-01BA03.pdf

[13] (Accessed 2014, December 15). *Processes and Threads* [Online]. Available: http://developer.android.com/guide/components/processes-and-threads.html

[14] William Premerlani et al. *Direction Cosine Matrix IMU: Theory* [Online]. Available: https://gentlenav.googlecode.com/files/DCMDraft2.pdf

[15] Duane T. McRuer, et al. *Aircraft Dynamics and Automatic Control*. Princeton, NJ: Princeton University Press, 1973.

[16] (2014, April 21). *OpenCV 2.4.9 Documentation* [Online]. Available: http://docs.opencv.org/

[17] (2014, March 26). *Mask (computing): Image masks* [Online]. Available: http://en.wikipedia.org/wiki/Mask_%28computing%29#Image_masks

[18] (Accessed 2014, December 15). *Color Temperature* [Online]. Available: http://en.wikipedia.org/wiki/Color_temperature

[19] (Accessed 2014, December 15). *xcopterCalc - Multicopter Calculator* [Online]. Available: http://www.ecalc.ch/xcoptercalc.php