The AR Drone LabVIEW Toolkit:
A Software Framework for the Control of Low-Cost Quadrotor Aerial Robots

A Thesis

submitted by

Michael Mogenson

In partial fulfillment of the requirements
for the degree of

Master of Science

in

Mechanical Engineering

TUFTS UNIVERSITY

May 2012

Advisor: Chris Rogers
Committee Members: Jason Rife, Matthias Scheutz

Abstract

For this research the AR Drone quadrotor by Parrot was reverse engineered to work with the graphical programming environment LabVIEW in an attempt to create a low cost, easy to use, entry level aerial robotics platform for students, educators, and researchers. The result of this work is the AR Drone LabVIEW Toolkit. The Parrot AR Drone is a low-cost consumer oriented quadrotor equipped with video cameras and WiFi communication. The AR Drone LabVIEW Toolkit can maneuver the AR Drone, read data from its on-board sensors, decode the video stream from the two cameras, and analyze the video frames with a variety of included image processing algorithms. This thesis consist of a discussion of how the AR Drone LabVIEW Toolkit functions followed by demonstrations of some sophisticated autonomous behaviors programmed for the AR Drone and an evaluation of the AR Drone LabVIEW Toolkit based on classroom testing and educator testimonial.

Acknowledgements

I would like to thank the people at the CEEO and elsewhere who helped me learn the skills required to complete this project, especially Rob Linsalata, Evan Krause, and Michalis Kalaitzakis. Thank you to my advisor, Chris Rogers, for the freedom I've enjoyed in my research and for constantly pushing for new innovation in educational technology, as well as Ethan Danahy, Jason Rife, and Matthias Scheutz. Thank you to my family for providing me the means to further my education and Elspeth for her support. Finally, thank you to all the faculty, staff, and grad students in the ME department. You've taught me how to be an engineer and make me wish I could stay another year.

Table of Contents

1 Introduction

1.1 Motivation

Over the past decade the quadrotor has become the standard platform for micro aerial vehicle (MAV) research and aerial robotic projects. This is a result of the simplicity and effectiveness of the quadrotor design, the decrease in component costs, and the increase in performance and capabilities of the components.

A quadrotor is a small helicopter, typically the scale of 1 to 3 feet across with four propellers arranged in counter rotating pairs on a "+" or "x" frame. Each propeller is driven by an electric motor, which is powered by an on-board battery pack. These craft are autonomously or remotely operated. An on-board computer or microcontroller and a variety of sensors including accelerometers, gyroscopes, and magnetometers maintain the quadrotor's stability.

The quadrotor is one of the easier aerial platforms to control. All maneuvering is accomplished by adjusting the rotational speed of each propeller relative to the rest. Propellers located directly across the center frame from one another spin the same direction. Adjacent propellers spin in opposite directions. To perform a maneuver such as rotating the front of a quadrotor down, the front propeller's speed is decreased and the back

propeller's speed is increased by the same amount. This causes a force moment that pitches the pitches of the quadrotor down while maintaining the same upwards thrust for small rotation angles and equilibrium in every other dimension. Other modes of movement can be seen in Figure 1.1.

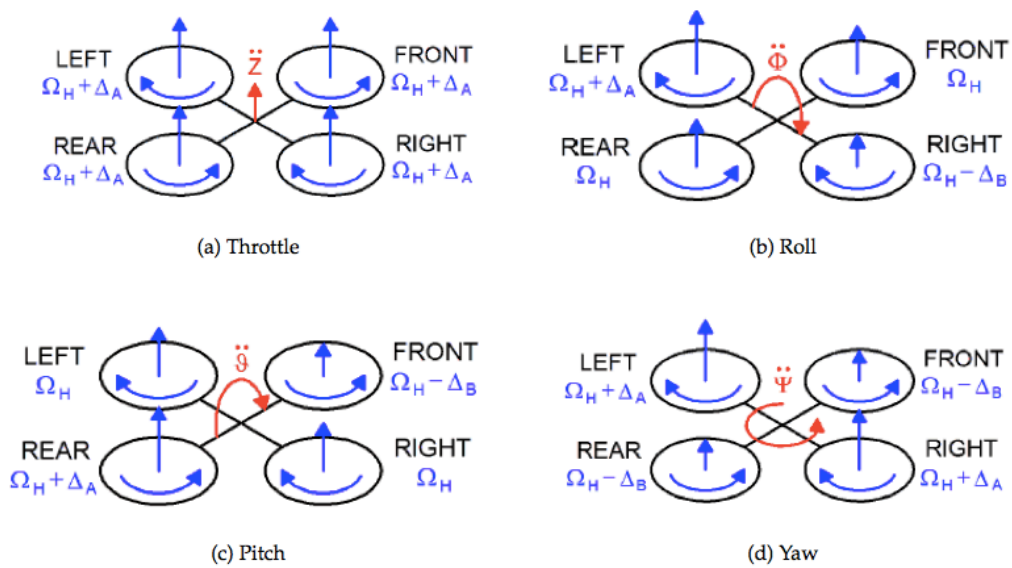


Figure 1.1: Methods for quadrotor movements[1]

Over the years, the electronics that make up a quadrotor have dramatically improved within the categories of cost, performance, and weight. Miniature computers and microcontrollers fit on a single board and are now powerful enough to run sophisticated control algorithms. Micro electronic mechanical systems (MEMS) sensors provide accurate information and take up a tiny amount of space. Finally, brushless DC motors and electronic speed controllers are powerful and more energy efficient. When

[1] Image courtesy of AR Drone Developer's Guide

used in conjunction with lithium polymer lightweight batteries, flying times of up to 20 minutes can be achieved [1].

Many researchers, students, and educators in high school, college, and graduate institutions are attracted to the quadrotor platform for the reasons specified above [2]. Quadrotors are useful for robotics research such as testing of control algorithms for fast and aggressive aerial maneuvering, developing robot-to-robot networking for collaboration, and other swarm theory applications [3]. Quadrotors may also be used as a tool to teach non-robotics subject matter such as physics and critical thinking. A flying quadrotor craft is an exciting reward for a student who must first learn how to calculate some parameter or create some other part necessary to complete an autonomous program. A literature review shows that some educators are already using quadrotors in a classroom setting [4] [5] while others are attempting to build quadrotors specifically for education [6] [7].

There are two main options to obtain a quadrotor suited for education or research: purchase a model from manufacturers such as Ascending Technologies [1] or DraganFly [8], or build one from scratch. Neither of these options is ideal. A professional quadrotor such as the Draganfly X4 costs over $12,000 [8]. Remote control toy quadrotors can be had for a few hundred dollars but these are not able to interface with a computer and do not have sophisticated sensors. Alternatively, the process of building a quadrotor from scratch is long and arduous. Successfully constructing a quadrotor requires knowledge of circuit design, embedded programming,

machine shop fabrication, and experience with remote control hobby equipment. Once the parts have been sourced and the craft has been constructed, the control algorithms must be developed and implemented on the quadrotor's computer or microcontroller. These control algorithms must be correctly tuned to ensure the craft's stability. Even with extensive simulation, in-flight testing is a must and crashes and broken parts are a certainty. Many home-built quadrotors evolve though a trial and error process.

The high cost of professional quadrotors and complexity of home-built quadrotors limits their use by students and hinders their adoption by academic institutions with smaller budgets. One additional commercial quadrotor was released in 2010. The French cell phone component company, Parrot, created a consumer-oriented quadrotor called the AR Drone. The AR Drone, shown in Figure 1.2, retails for $300, can connect to a computer with WiFi, and has a number of useful sensors such as accelerometers, gyroscopes, an ultrasonic altimeter, and two video cameras: one facing forwards and one looking down. An onboard computer is capable of automating takeoffs, landings, and maintaining a stable hover while streaming video from the cameras and sensor data over WiFi. I chose this quadrotor for this work due to its price, available sensors, and WiFi communication. A comparison of the AR Drone to other commercially available quadrotors can be seen in Table 1.1.

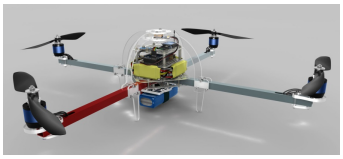
Figure 1.2: The Parrot AR Drone quadrotor in flight[2]

Because the AR Drone was designed as a toy, the only officially supported way to fly it is with a free iPhone or Android phone app. The AR Drone fits the needs of most students, researchers, and educators, but there is no way to control it from a desktop computer. A link to a computer is necessary so that users can write code to control the AR Drone and monitor it while in flight. If there were a way to link the AR Drone to a computer, the user could read sensors and live video streams, command and maneuver the AR Drone, and implement control algorithms to close the loop between sensor inputs and control outputs to create sophisticated autonomous behaviors. This work describes a software package running on a computer

[2] Image courtesy of http://www.flickr.com/photos /minhocos/5830590347/

as a means to establish a link to the AR Drone and automate communication and control. Students, educators, and researchers are able to write their own programs but are not forced to create low-level features such as the communication management between the computer and AR Drone and the video stream decoding.

Table 1.1: Commercially available quadrotors

Quadrotor	**Sensors**	**Price**
Parrot AR Drone 	3-axis accelerometer, 3-axis gyroscope, sonar altimeter, front & bottom cameras	$300
Ascending Technologies Hummingbird 	3-axis accelerometer, 3-axis gyroscope, 3-axis magnometer, barometric pressure sensor, GPS	$5,000
DraganFly X4 	3-axis accelerometer, 3-axis gyroscope, barometric pressure sensor	$12,000
DIY Drones ArduCopter 	3-axis accelerometer, 3-axis gyroscope	$860

1.2 Existing Technologies

There are a number of projects to either supply a low-cost capable quadrotor or control the AR Drone from a computer. Two organizations, Mikrocopter [9] and DIYDrones [10], supply kits to build a quadrotor. The Mikrocopter kits cost about $1200 but include everything necessary to build a fully functioning quadrotor. The DIYDrones kit was developed by volunteers and costs $860. Both of these kits require electronics knowledge and soldering skills. The included microcontrollers are pre-loaded with code, but the control algorithms still need to be tuned for each individual craft. Out of the box, these quadrotors cannot be controlled by a computer. However, a wireless serial module can be added to each craft with additional soldering and modification of the embedded code.

On the AR Drone side, Parrot has released an official SDK [11]. The SDK was created to aid developers building iPhone and Android apps to fly the AR Drone. Because the SDK was not intended to be used by general hobbyists, it lacks detailed documentation and can only be compiled on certain versions of Linux and Windows. Additionally, the SDK is written in plain C with a mix of French and English function names and variables, making it difficult to understand.

Individual AR Drone owners have reverse-engineered the communication and control schemes of the quadrotor. They have written

software toolkits for a variety of computer languages. The majority of these toolkits are easier to use than the official SDK, although different projects are in various states of completion.

There is a Java application to control the AR Drone called ARDroneME [12]. This application was created by MAPGPS, a member of the AR Drone developers forum and one of the first people to reverse engineer the AR Drone. His application shows, in full detail, how to communicate with and control the AR Drone. It is also written entirely in Java, meaning it is platform independent and not proprietary. Although this project has tremendous potential, recently, development has stopped and MAPGPS has moved on to other things. Some key features, such as the ability to view video steams from the on-board cameras, have not been implemented.

The ARDroneME application was written as a stand-alone program, not a framework. This makes it difficult to extract reusable code that can be used for new programs. A project called AutoPylot [13] has taken a different approach to easily controlling an AR Drone by creating Python and MATLAB wrappers for the official SDK. Python is an interpreted language, like Java, that uses easy to read syntax and MATLAB uses a loosely typed version of C and is a standard in the technical computing industry. These two versions of AutoPylot provide the user with the full capability of the official SDK such as video streaming and data logging in an easier to use fashion. The MATLAB version is centered around one call back function that provides frames of video and sensor data from the AR Drone. Once this information is within

MATLAB, it can be analyzed with any of the included mathematical and image processing algorithms. This is a very powerful control suite for the AR Drone. However, since the MATLAB and Python code is linked to the official SDK, AutoPylot is only available for Ubuntu Linux computers. There is also a complicated setup procedure to have MATLAB compile the official SDK.

1.3 Conclusion

The quadrotor platform provides a means for aerial robotics research and for the use of aerial robotics as a teaching aid for other areas of study such as controls and mechatronics. The technologies behind quadrotors have evolved to the point where these machines are capable of sophisticated and intelligent autonomous behaviors. The two factors preventing the wide spread adoption of quadrotors throughout academic institutions are cost and ease of use. The release of the AR Drone quadrotor by Parrot solved the issue of cost. A framework of computer code to control the AR Drone from a PC, with functions to access all of the AR Drone's important features, would solve the ease of use problem.

2 Implementation

2.1 Introductory Statement

In this thesis I will present a software toolkit for controlling the AR Drone. I will discuss its functionality and its effectiveness at enabling students and educators to create autonomous programs for the AR Drone. I will then evaluate the performance of individual parts of this toolkit and present feedback from real world usage of this toolkit from classroom testing and student projects. This software toolkit has been implemented in the graphical programming environment LabVIEW by National Instruments and is therefore called the AR Drone LabVIEW Toolkit.

The LabVIEW programming environment is composed of functions called VI's (for virtual instruments). As shown in Figure 2.1, a VI appears graphically as a block with inputs and outputs. A VI can be connected to another VI by virtually wiring them together. Entire programs can be built up this way and sub-VI's can be created from multiple VI's to include in larger projects. Each program has a Block Diagram where the VI's are located and a Front Panel where graphical user interface (GUI) items are presented to the user and manipulated.

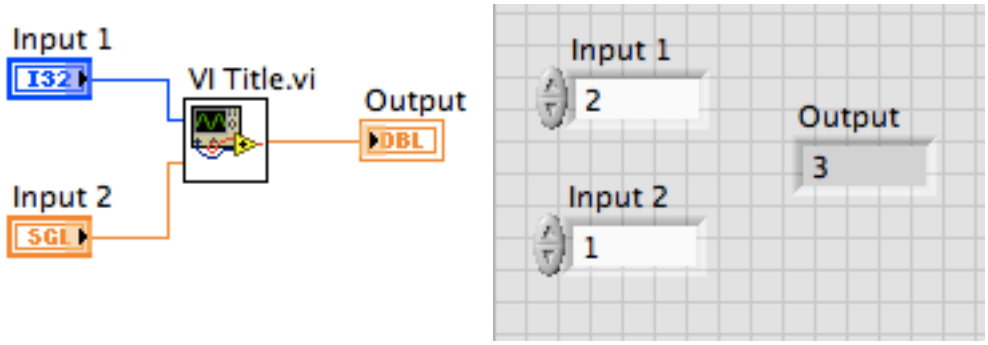


Figure 2.1: The Block Diagram (left) with a VI that has inputs and an output, also shown on the Front Panel (right)

The LabVIEW dataflow structure is well suited to robotics applications [14]. Just like in controls block diagrams, information and signals flow through wires from one VI to the next. It is easy to trace wires to understand the functionality of a LabVIEW program. LabVIEW can be easier for novice programmers to learn than text based languages because there are no function names or critical syntax to remember [14].

Unlike text-based computer languages, LabVIEW does not execute in line order. Each VI executes as soon as its inputs are available. This means multiple VI's can execute at the same time. This inherent parallelism is beneficial for programs that require asynchronous communication.

The AR Drone LabVIEW Toolkit is composed of high-level VI's that perform important tasks such as maneuvering the AR Drone, reading and decoding the video stream, and collecting sensor data. Since each task requires only one VI, the total number of VI's the user has to interact with is kept to a minimum. This organization attempts to make the AR Drone LabVIEW Toolkit more accessible to users. The AR Drone LabVIEW Toolkit

VI's are shown in Figure 2.2. These VI's can be used in conjunction with LabVIEW's many included VI's to analyze the data coming from the AR Drone and calculate output controls to be sent back.

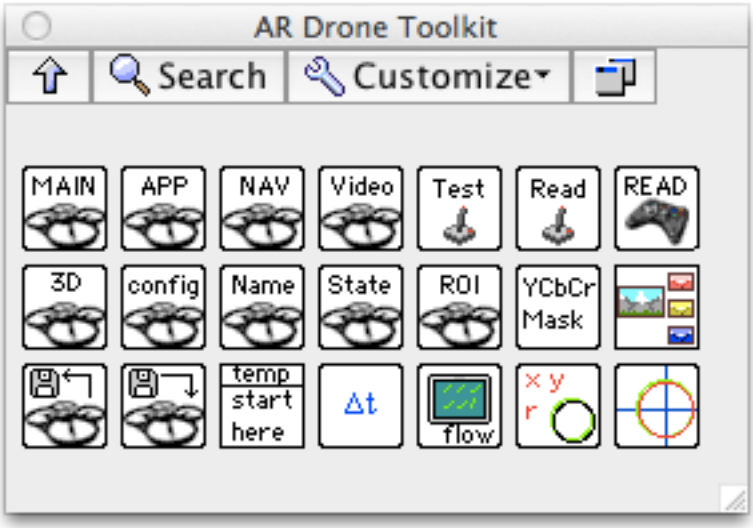


Figure 2.2: Palette of VI's comprising the AR Drone LabVIEW Toolkit

Figure 2.3 shows a simple teleoperation program made with the AR Drone LabVIEW Toolkit. It is visually simple considering the networking that occurs between the computer and the AR Drone. Only four VI's from the AR Drone LabVIEW Toolkit are required. The joystick and Main VI convey the movements the user makes with a connected USB joystick to the AR Drone. The Video and Nav VI's read and display the AR Drone's video stream and on-board sensor data. These VI's are surrounded by a While Loop which calls these VI's continuously until the user stops the program.

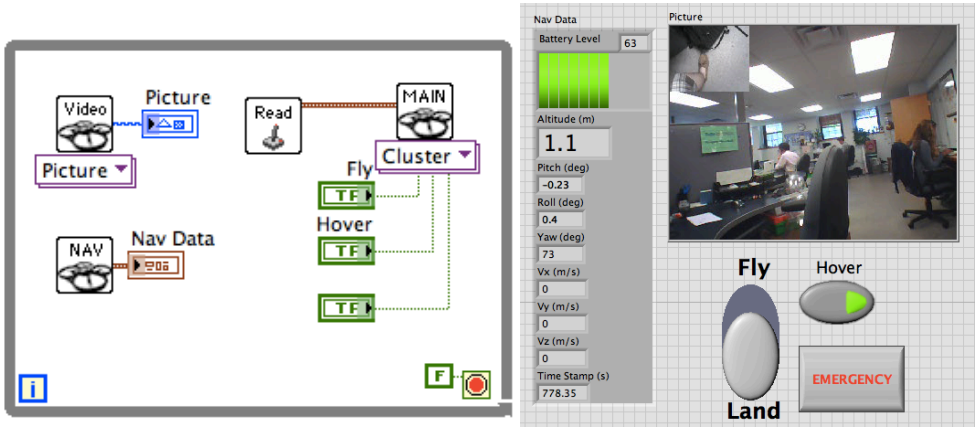


Figure 2.3: Block Diagram and Front Panel of a teleoperation program

The following sections will detail how specific VI's function and how they can be used.

2.2 Communication and Control VI's

The computer must first be connected to the AR Drone via WiFi in order to control the AR Drone. When the AR Drone is powered on, it creates a local ad-hoc WiFi access point that may be joined from the computer. On this network the AR Drone assigns itself a static IP address and opens up some predefined ports for communication. Figure 2.4 shows the main channels for communication. The video stream and sensor data is sent from the AR Drone to the computer and state commands to control the movement of the AR Drone are sent from the computer to the AR Drone. All communication happens asynchronously.

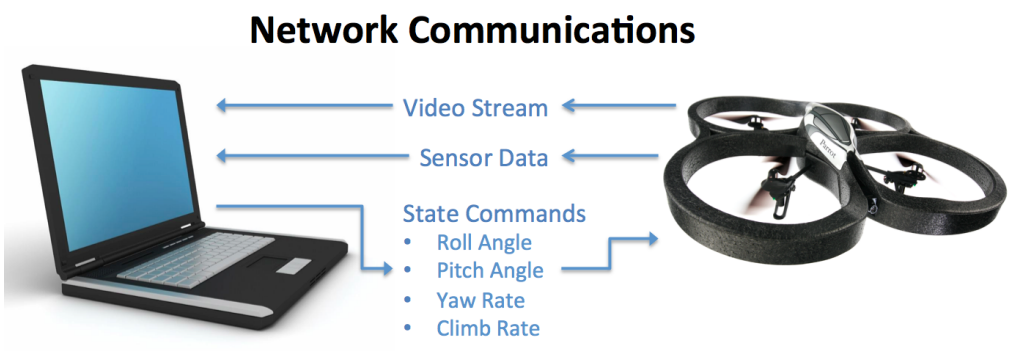


Figure 2.4: Flow of information between the AR Drone and computer

2.2.1 Main VI

The most critical VI in the AR Drone LabVIEW Toolkit is Main VI. Main VI sends control commands to the AR Drone and keeps communication channels up and running. Main VI communicates with the AR Drone via the user datagram protocol (UDP). UDP allows packets to be sent from the computer to the AR Drone and vice versa without prior setup of transmission channels. It is simple and easy to implement but it does not guarantee that each packet will reach its recipient, or that packet that go though arrive in the order they were sent. To combat this problem the AR Drone uses a sequence number. The sequence number is an integer, which starts at one and increases by one for each UDP packet sent. The sequence number is embedded in each UDP packet sent is synchronized between the computer and AR Drone. The AR Drone only acts on UDP packets with a sequence number greater than the AR Drone's current sequence number. This prevents the AR Drone from executing old commands.

Contained within each UDP packet is one or multiple commands

recognized by the AR Drone. Each command begins with the characters *AT**, contains one or multiple pieces of information, and terminates with a line feed character, represented as *<LF>.* For example, in order to initiate a takeoff the Main VI sends the AR Drone a UDP packet containing the command *AT*REF=seq, state<LF>* where *seq* and *state* are arguments. In this command *seq* is the sequence number discussed above and *state* is a 32-bit integer where each bit is a boolean toggle for some option on the AR Drone. For instance, bit number 9 is the takeoff or landing state of the AR Drone. By toggling bit 9 from 0 to 1, the AR Drone is instructed to takeoff. Bit number 8 of *state* is the emergency state of the AR Drone. If this bit is flipped from 0 to 1 the AR Drone will immediately stop all motors (and plummet to the ground if flying). When bit number 8 is flipped back to 0 the AR Drone will recalibrate its internal sensors and prepare for flight again.

Main VI also utilizes the command *AT*PCMD=seq, hover, roll, pitch, climb, yaw<LF>* where *roll, pitch, climb,* and *yaw* are arguments that control the motion of the AR Drone and *hover* tells the AR Drone to maintain its current position. *Roll, pitch, climb,* and *yaw* are all single precision floating-point numbers between -1 and 1. These values are type cast into 32-bit unsigned integers before being inserted into the *AT*PCMD* command. For *yaw* and *climb,* the sign of the number controls the direction with which to move the AR Drone and the magnitude controls the rate at which to move. In Figure 2.5, a value of 1 for *yaw* would rotate the AR Drone about the Z_w axis in the direction of the arrow at the fastest allowed rate. A value of -0.5 for

yaw would pivot the AR Drone in the opposite direction around the Z_w axis half as fast. *Climb* causes the AR Drone to ascend and descend in height. *Roll* and *pitch* set the desired roll and pitch angles with respect to the X_w and Y_w axes. A value of 1 for *roll* brings the AR Drone to the maximum allowed roll angle and a value of 0.5 would be half of that max angle. Finally the *hover* parameter can be equal to 0 or 1. A value of 1 instructs the AR Drone to ignore the *roll* and *pitch* commands and attempt to keep itself positioned over its current location.

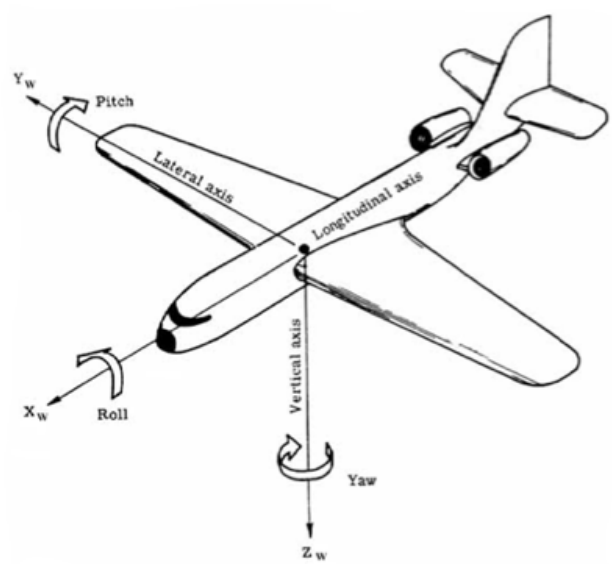


Figure 2.5: Coordinate system of the AR Drone[3]

Main VI also employs the *AT*COMWDG=seq<LF>* command. This command must be sent at least every 250 milliseconds to keep communication between the computer and AR Drone active. If the AR Drone doesn't receive this command it will assume that communication has been lost and perform a controlled landing.

[3] Image courtesy of AR Drone Developer's Guide

The first time Main VI is executed at the beginning of each new flight, it performs a setup procedure to initialize the AR Drone for flight and set any program specific preferences. The setup procedure relies on many additional commands, most notably *AT*CONFIG, AT*CTRL, AT*REF, and AT*FTRIM*. The setup procedure sets limits for the movements of the AR Drone including max roll and pitch angle, max yaw rate, a max climb rate, and max allowed altitude. The communication channels are setup using the AR Drone and computers IP addresses and port numbers for sending and receiving video and sensor data. Flight parameters such as if the foam propeller protector is being used and if the AR Drone is flying indoors or outdoors are set. The active video camera for video streaming is also specified. Video camera options include the front camera, bottom camera, and picture in picture for front in bottom, and bottom in front. These setup preferences are available to be set by the user in a Setup Parameters cluster assigned as an input to Main VI. Finally, the setup procedure conducts a manual calibration of the AR Drone's on-board sensors and toggles the AR Drone's LEDs from red to green to signify that it's ready for flight.

During flight, Main VI performs a couple of actions every time it is executed. It takes in new values for *roll, pitch, yaw,* and *climb* and checks if they are a number. If not, the values are coerced. *Roll, pitch, yaw* and *climb* are typecast into 32-bit integers and packaged into an *AT*PCMD* command and placed into a newly formed UDP packet with a new sequence number. Main VI looks up the amount of time that has passed since it was last called

and, if needed, inserts an *AT*COMWDG* command into the UDP packet. An *AT*REF* command containing the desired takeoff and emergency states is also inserted into the packet. Finally, a port on the computer opened and the UDP packet is sent to the AR Drone.

Main VI selectively delays the sending of commands to ensure that the UDP packets being sent are dispatched at a constant 30 Hz. This constant rate of communication prevents the AR Drone from being overloaded by commands. The performance of the AR Drone's on-board controller can suffer if too much processing time is spent dealing with incoming UDP packets causing erratic movements, delayed response to commands, and the potential for crashes.

2.2.2 Video VI

Another crucial VI in the AR Drone LabVIEW Toolkit is Video VI. Video VI reads UDP packets containing video frames sent from the AR Drone, decodes them, and presents them as either images or clusters containing raw pixel data.

The first step to obtaining video from the AR Drone is to initialize the video stream. On first call, Video VI sends a UDP packet containing a decimal 1 to the AR Drone's video stream out port. This instructs the AR Drone to begin broadcasting video to the IP address from which that packet was sent. The other option is to have the AR Drone publicly broadcast the video stream for any computer on the network to read. UDP packets containing

compressed video frames will now be delivered to the computer's video receiving port. The most current packet is kept in a buffer for Video VI to read. If Video VI does not retrieve that packet within a certain amount of time, or a new packet arrives, it is trashed.

If the front camera is selected, video packets will be sent about every 66.7 milliseconds with a resolution of 320 x 240. If the bottom camera is selected, video packets will be sent about every 16.7 milliseconds with a resolution of 176 x 144. After it is received, each video frame sent to the video decoder to be uncompressed.

The AR Drone encodes its video in a modified H.263 universal variable length codec (15). On the AR Drone, each video frame is first split into rows 16 pixels high called Groups of Blocks (GOB). Each GOB is then split into Macroblocks 16 pixels wide. Every 16 x 16 pixel Macroblock is stored in the YCbCr type 4:2:0 color space corresponding to 6 blocks of 8 x 8 8-bit integer values. These blocks are shown in Figure 2.6. The first 8 x 8 blocks are the Y or luminous pixel values for the Macroblock in top left, top right, bottom left, bottom right order. The last two 8 x 8 blocks are the chroma red and chroma blue values. Because the human eye is less sensitive to color than to intensity, space is saved by only storing one chroma red or chroma blue value for every four Macroblock pixels, hence the 8 x 8 size for the 16 x 16 Macroblock.

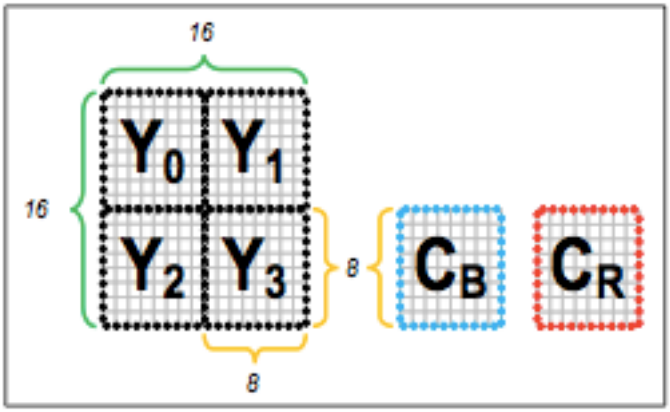


Figure 2.6: YCbCr components of a Macroblock[4]

Each block is then compressed using the JPEG encoding process shown in Figure 2.7. First, a two-dimensional direct cosine transform converts each block into a frequency domain representation. The bit-depth of each value in the block is temporally increased to 16 bits in order to store the frequency domain data. The block is then quantized by dividing the entire block by a DC offset and rounding to the nearest integer. This process causes many of the smaller values to go to zero. The last step is to use entropy encoding to reduce the number of bits needed to store the binary data for each block. This allows the video stream to be sent in smaller sized packets, reducing the chances for lost or corrupted packets. Entropy encoding consists of first rearranging each block's values in a zigzag shape starting at the top left and working down to the bottom right. Then, a run-length encoder (RLE) algorithm takes binary representation of the rearranged block and replaces long strings of zeros by a run-length code marking down how many zeros have been removed. A Huffman encoder

[4] Image courtesy of AR Drone Developer's Guide

performs a similar process to remove strings of ones.

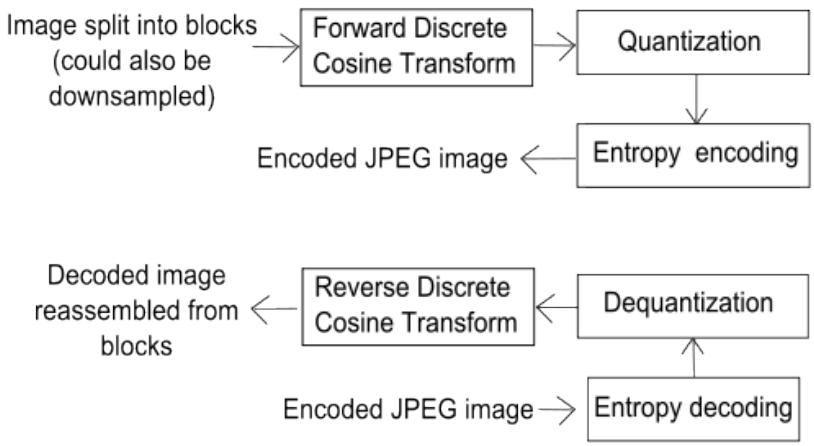


Figure 2.7: The JPEG encoding process[5]

After compression, each block is combined into Macroblocks and GOB's, padded with status bits detailing start & end of frame, start & end of GOB, and start & end of Macroblock, and packaged into a UDP packet to be sent off. The compression ratio between original image to final packet varies from 12.5 to 32.5 depending on the contents of the original image.

To decode each video frame, this process is run in reverse. Each block is uncompressed and recombined into Macroblocks and GOB's, and eventually the entire YCbCr image. To make future processing easier, these YCbCr images are converted into the standard 24-bit RGB color space. This decoding process is implemented in LabVIEW within Video VI.

The execution time of Video VI is lengthy compared to the rest of the AR Drone LabVIEW Toolkit due to the complex nature of decoding video. It

[5] Image courtesy of AR Drone Developer's Guide

takes 3.0 milliseconds to decode a 320 x 240 frame from the front camera and 0.6 milliseconds to decode a 176 x 144 frame from the bottom camera. The structure of Video VI has been organized so that while one part of the VI is decoding the current video frame, another part is already fetching the next frame so it will be ready to go. It takes 0.1 millisecond to retrieve the next video frame from a received packet. A drawback to this approach is that it necessitates a dummy frame to begin the process. Therefore, any image processing algorithms that get images from Video VI should not use the first frame as any sort of reference.

2.2.3 Nav Data VI

Nav Data VI is the final VI contained within the AR Drone LabVIEW Toolkit that has routine direct contact with the AR Drone. Nav Data VI receives and parses the AR Drone's internal sensor information, called navigation data. Each time Nav Data VI is executed it returns a cluster consisting of the battery level in percentage, altitude in meters, pitch, roll, and yaw in degrees, X_w - Y_w - Z_w ground velocity in meters per second, and a time stamp accurate to the microsecond.

Similar to Video VI, the Nav Data VI sends a UDP packet to the AR Drone's nav data out port, instructing the AR Drone on whether to send nav data packets to the computer's IP address or to multicast them. In the AR Drone LabVIEW Toolkit, Nav Data VI deals with an extended nav data packet that arrives every 5 milliseconds and is 292 bytes long. This nav data packet

contains extra sensor data and debug information. These packets are long strings of bytes from which Nav Data VI extracts specific bytes and combines them to form integer and floating-point values to add to the output cluster. For example, byte number 226 though 230 are combined to form the time stamp. The first 11 bits of this 32-bit number are seconds and the remaining 21 bits are microseconds. It takes 0.1 millisecond to retrieve and parse a packet of navigation data.

2.3 Supporting VI's

The AR Drone LabVIEW Toolkit also includes a number of other VI's to manage, debug, and customize the AR Drone. Read Joystick & Test Joystick VI use the *lvinput* framework from the TETRIX Toolkit [16] to poll a USB joystick or gamepad about its state. Joystick Tester VI displays all axes and buttons on LabVIEW's Front Panel so that the user can determine which axis corresponds to which direction of movement on the physical joystick. Read Joystick VI lets the user select which axes corresponds to which AR Drone movement command: *roll, pitch, yaw,* or *climb* and invert them if necessary.

There are two VI's for managing the firmware of the AR Drone Test Firmware VI and Upgrade Firmware VI. The AR Drone LabVIEW Toolkit was designed for a specific AR Drone firmware and requires this firmware to be installed. Test Firmware VI puts the AR Drone into a FTP server mode and downloads a file containing the version of the current firmware. It checks

this number to determine if the firmware needs to be updated. Firmware Updater VI also flips the AR Drone to FTP mode then uploads the required firmware. A hack is used to force the AR Drone to update its firmware: the current firmware version is set to 0.0.0. The actual process of uploading and downloading files from a FTP server within LabVIEW is done with system level calls specific to Windows, Mac OS X, or Linux.

Lastly, Name VI changes the name of the WiFI network that the AR Drone creates with an *AT*CONFIG* command and 3D Picture VI displays a 3D representation of the AR Drone that uses the nav data to match the AR Drone's orientation. Read Config VI telnets into the AR Drone and retrieves a file where all of the user's custom configurations are stored. The contents of this file can be checked against the preferences set in the Setup Parameters cluster to make sure they are being correctly set.

2.4 Additional VI's and 3rd Party Integration

Most robots follow the *Sense – Think – Act* pattern [17]. Video VI and Nav Data VI are the *Sense* functionality for the AR Drone LabVIEW Toolkit and Main VI is the *Act*. The VI's discussed in this next section provide some means to *Think*: that is, take the outputs from Video VI and/or Nav Data VI and extract some meaningful information that can be sent to Main VI. Because speed is always a factor in aerial robots, the performance of these

VI's was analyzed using the Performance and Memory Toolbox from within LabVIEW and will be discussed.

The first such VI is State VI. State VI takes the nav data and estimates the AR Drone's position in the ground X-Y-Z Cartesian coordinate system plus yaw orientation. It does this by first populating a rotation matrix between the AR Drone's coordinate system and the ground with the Euler orientation angles (*roll, pitch, yaw*) from the nav data then applying that rotation matrix to the velocities measured from the bottom camera. These rotated velocities are integrated into a position with the high precision nav data time stamp and the trapezoidal method. The trapezoidal method is defined as:

$$x = \frac{1}{2}\sum_{i=1}^{N}(t_{i+1} - t_i)(v_{i+1} + v_i)$$

Equation 2.1

Where x is the position for a direction, $t_{i = 1..N}$ are the time stamps, and $v_{i = 1..N}$ are the velocities in that direction.

The yaw orientation on the AR Drone is sensed using only a single rate gyro, and drifts over time. This gyro drift has been measured over time to be roughly constant 0.076 degrees per second. A drift corrected yaw combined with the X-Y-Z location of the AR Drone.

The output of State VI is only as accurate as the nav data fed into it. It is particularly sensitive to variations in the velocities. Because the AR Drone's velocities are measured from an optical flow algorithm, they can

report incorrect values if the image of the ground lacks features or if shadows move across the camera's view. Additionally, the position from integration will drift over time due to round-off error. Therefore, it is best to use the output of State VI for only short amounts of time. Another solution is to use the boolean reset input of State VI to reset the estimated location when the AR Drone gets to a known location.

Each frame of video sent back from the AR Drone contains a wealth of information. Using image processing techniques, one can extract useful information from these video frames. The AR Drone LabVIEW Toolkit has been designed to work with the LabVIEW image processing library developed by Chris Rogers at Tufts University and Christophe Salzmann at EPFL in Switzerland [18]. This library is capable of many things, but one common application suited to the AR Drone is locating the position of some object in the image with a color very different from the background. One example is locating a red ball in a white room. For this scenario, the first step is to mask the image with a red mask. This removes every pixel that does not have a value for red above a certain threshold. The image is now a binary image containing pixels with a value of 1 for the pixels that passed the masking, and a 0 for those that did not. This image is fed into a blob detector that identifies the size and location of a group of 1's. The largest blob is most likely the red ball. Its location can be used as the set point in a controller to have the AR Drone track and follow the red ball.

Blob detection is data intensive process requiring each pixel of an image be analyzed. Yet, it is critical that the blob detector execute quickly or else the entire program will slow down. Main VI cannot update the AR Drone with a new set of commands until the blob detector supplies it with new coordinates for the tracked object. This delay induced in the communication can make the AR Drone unresponsive and unstable. Therefore, I selected a faster blob detection algorithm for the AR Drone LabVIEW Toolkit. This fast blob detector uses a contour-tracing algorithm developed by Fu Chang et al [19]. This algorithm scans though a binary image until a nonzero pixel is located then searches clockwise around that pixel until another nonzero pixel is located and updates its location. It repeats the clockwise search for each new pixel location until the entire contour of the blob has been traced and the algorithm has gotten back to the first nonzero pixel. In this manner, the contour-tracing blob detector can locate blobs in the image without having to scan though every pixel. This approach proves fast enough for aerial robotic applications. The contour-tracing blob detector can analyze a 320 x 240 pixel image from the AR Drone's front camera in 0.9 milliseconds.

Another image processing VI that has been added to the AR Drone LabVIEW Toolkit is an optical flow algorithm called Dense Optical Flow VI. Dense Optical Flow VI calculates the velocity of pixel translation from one image to the next for twelve different regions within the image. It is based upon the image-interpolation technique of M.V. Srinivasan [20]. This algorithm takes each portion of the original image and translates it by a

reference Δx_{ref} and Δy_{ref} in the positive and negative directions. The number of pixels translated, Δx and Δy, in each direction from one frame to the next can be found by simultaneously solving the following system of equations:

$$\left(\frac{\Delta x}{\Delta x_{ref}}\right)\iint (f_2 - f_1)^2 dxdy + \left(\frac{\Delta y}{\Delta y_{ref}}\right)\iint (f_4 - f_3)(f_2 - f_1) dxdy = 2\iint (f - f_0)(f_2 - f_1) dxdy$$

$$\left(\frac{\Delta x}{\Delta x_{ref}}\right)\iint (f_2 - f_1)(f_4 - f_3) dxdy + \left(\frac{\Delta y}{\Delta y_{ref}}\right)\iint (f_4 - f_3)^2 dxdy = 2\iint (f - f_0)(f_4 - f_3) dxdy$$

Equation 2.2

Where f_0 is the first frame, f is the next frame, $f_1 = f_0(x + \Delta x_{ref})$, $f_2 = f_0(x - \Delta x_{ref})$, $f_3 = f_0(y + \Delta y_{ref})$, and $f_4 = f_0(y - \Delta y_{ref})$.

Other optical flow algorithms pick out features that can be tracked from one image to the next. This VI is considered a dense optical flow algorithm because it does not rely on features within an image. This makes the VI less computationally intensive and more applicable to a wide range of textures within the camera. The downside is that Dense Optical Flow VI is only accurate for small movements between frames, on the same order of magnitude as Δx_{ref} and Δx_{ref}. Testing shows that Dense Optical Flow takes 1.1 milliseconds to process each new image and calculate new flows for the twelve segments.

Another feature added to the AR Drone LabVIEW Toolkit is the ability to process images in different color spaces. There is a YCbCr VI which converts the RGB images from Video VI back to YCbCr type 4:4:0 color space,

meaning there is an intensity, chroma red, and chroma blue value for every pixel. There is also a YCbCr Mask VI which performs a mask on pixels for a threshold range of chroma red values and chroma blue. This has the useful application of separating pixels belonging to a persons face from the background since the color for most Caucasian people's skin tone occupies a very narrow range of chroma red and chroma blue [21]. These VI's can be used to create face or hand tracking programs.

Circle Detector VI fits a circle to points in a binary image using a least squares approach. This VI uses a closed form solution proposed by Dale Umbach [22] to minimize the following expression:

$$\sum_{i=1}^{n}(r-\sqrt{(x_i-a)^2+(y_i-b)^2})^2$$

Equation 2.3

For a radius r and a center point (a,b) where (x_i,y_i) is the location of each non zero pixel.

Circle Detector VI tries to best fit a circle to every nonzero pixel in a binary image, whether it is part of a circular object or not. This makes Circle Detector VI very susceptible to background noise and requires prior image processing to provide Circle Detector VI a clean binary image. Figure 2.8 shows an image of a round hoop and Circle Detector VI's best fit circle. Circle Detector VI took 0.8 milliseconds to fit a circle to this image containing over 3000 nonzero pixels after being masked for green.

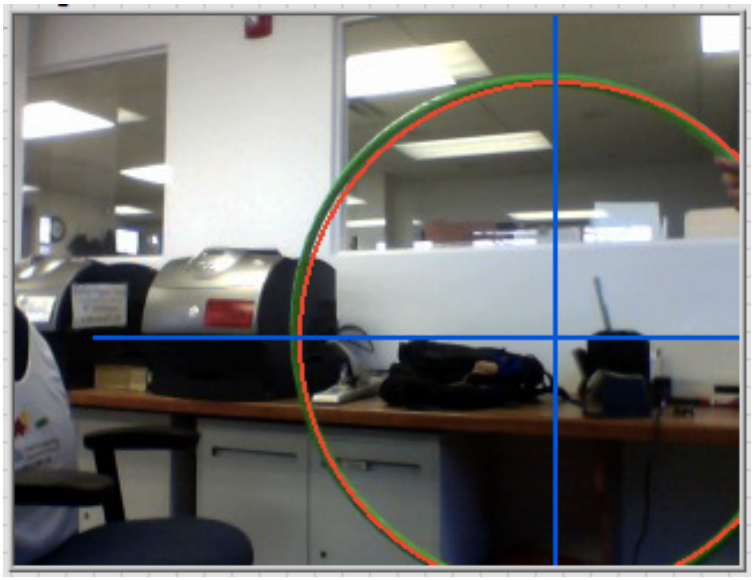


Figure 2.8: Image of green hoop with best fit circle plotted in red and center point in blue

A problem that arises when tracking objects in the video frames from the AR Drone is that the movement of the object in the image is coupled to the movement of the camera on the AR Drone. Adaptive ROI VI is a solution to this problem. Adaptive ROI VI uses the orientation information from Nav Data VI to select a region of interest (ROI) within the video frames and translate the ROI around the image to keep the contents of the ROI fixed and level. From experimentation, it was found that the contents in a video frame translate down 5.5 pixels for every degree *pitch* increases. Adaptive ROI VI translates the ROI in each video frame 5.5 pixels in the opposite direction for every degree change in *pitch*. It also performs a rotation in the opposite direction of equal magnitude for every degree change in *roll*.

Since the ROI will always be smaller than the original image, pixel information is lost. However, using Adaptive ROI VI means additional image

processing can be performed independent of the orientation of the AR Drone. This trade off was deemed acceptable since simplicity is a priority for the AR Drone LabVIEW Toolkit.

Finally, many things can be accomplished with the LabVIEW image processing library created by Chris Rogers and Christophe Salzmann and the image processing VI's created for the AR Drone LabVIEW Toolkit. However, some users may require more sophisticated image processing algorithms for advanced projects. These users may use the OpenCV library with the LabVIEW to OpenCV wrapper functions. OpenCV is an open source C and C++ image processing library with many higher level functions such as face recognition and pattern matching (REFERENCE). The LabVIEW to OpenCV wrapper functions take the form of a C framework, which contains the entire OpenCV library. This C framework can be loaded into LabVIEW and its functions called by the Call Library Function Node VI. When a function from the C library is passed an image from the AR Drone it converts it to the IPLImage structure used by OpenCV. From there, OpenCV can do its processing on the image and return a modified image or other data in the OpenCV specific formats to the C library. The C library then packages that data into arrays of bytes and returns it to LabVIEW.

2.5 Conclusion

Only one full program called Joystick Flight Example VI is included

with the AR Drone LabVIEW Toolkit. It is a direct teleoperation program that uses a USB joystick or gamepad to control the AR Drone. It also has aviation themed Front Panel indicators to display video and nav data. It was included to provide immediate gratification to new users and demonstrate how some of the important VI's are used. The rest of the VI's in the AR Drone LabVIEW Toolkit have full context help documentation and labeled input and output terminals. These VI's are designed to be used in new programs, to be explored, and to be recombined in new ways for new behaviors. Since Main VI handles all outgoing communication with the AR Drone and error checks the commands before sending out packets it is hard for the user to accidently crash the AR Drone with a poorly written program. Even if the LabVIEW program crashes or freezes while the AR Drone is flying, the AR Drone will remain aloft and Main VI will reestablish control over the AR Drone the next time the LabVIEW program is launched.

All of these previously discussed VI's form the complete AR Drone LabVIEW Toolkit. The AR Drone LabVIEW Toolkit is modular and decentralized by design. Only Main VI is required for every AR Drone program, since without Main VI commands will not be sent from the computer to the AR Drone. The remainder of the VI's can be added or removed as seen fit. One of the benefits of LabVIEW is that modifications to a program do not require changing header files or recompiling. Additionally, since the AR Drone is blind to the processing being done by the computer, the program can be modified without restarting the AR Drone.

3 Demonstrations

3.1 Introduction

The following three programs were constructed from the AR Drone LabVIEW Toolkit to test the functionality of the toolkit and the performance of the AR Drone: a face tracking program, an indoor navigation program, and a fly though hoop program. Only VI's from the AR Drone LabVIEW Toolkit, the LabVIEW image processing library, and the base installation of LabVIEW were used with the exception of the first two programs, which used the OpenCV image processing library. Documentation for the OpenCV functions used and the theory behind their operation can be found in the OpenCV documentation [23]. I chose these three programs to demonstrate the wide variety of autonomous behaviors that can be created with the AR Drone LabVIEW Toolkit. In the next chapter, we will look at programs created by users of the AR Drone LabVIEW Toolkit.

3.2 Face Tracker

The first demonstration program is a face tracking application called Face Tracker VI. The Face Tracker VI passes video frames from Video VI to OpenCV via the wrapper functions. OpenCV identifies a face, tracks it, and

returns the location and size to LabVIEW. The AR Drone is then maneuvered to keep the camera pointing towards the face and level.

Face Tracker VI is structured like a state machine. When the program is launched, the default state calls OpenCV's cvHaarDetectObjects( ) function. This function segments a series of ROI's from the supplied image within a range of sizes and orientations, as specified by the function's arguments, and uses a Haar Classifier to compare each ROI to a template of hundreds of sample faces. The Haar Classifier looks for changes in contrast such as a face's darker recessed eye sockets and brighter nose and cheeks. If cvHaarDetectObjects( ) returns a match for an image Face Tracker VI flips the state machine into a new state which initializes OpenCV's Camshift tracker. The Camshift tracker is passed an image with the ROI containing the face object detected by cvHaarDetectObjects( ). The Camshift tracker then calculates a color histogram for that ROI. After this color histogram is created, Face Tracker VI switches the state machine to the final, face tracking state and supplies new video frames to the Camshift tracker. The Camshift tracker compares every pixel of each new video frame to the color histogram and assigns it a probability of belonging to a face. After a video frame is analyzed, Camshift identifies the region of highest probabilities and returns the location of the center of area.

This location is used as the input to a two dimensional proportional controller. This controller attempts to zero the difference between the face location and the center of the video frame by rotating the AR Drone and

increasing or decreasing its altitude. A screen shot from a video of the face tracking program in action [24] can be seen in Figure 3.1. The proportional controller is tuned to be slightly over damped and does not require a derivative term for smooth operation. Even with just a proportional controller the AR Drone tracks accurately and quickly.

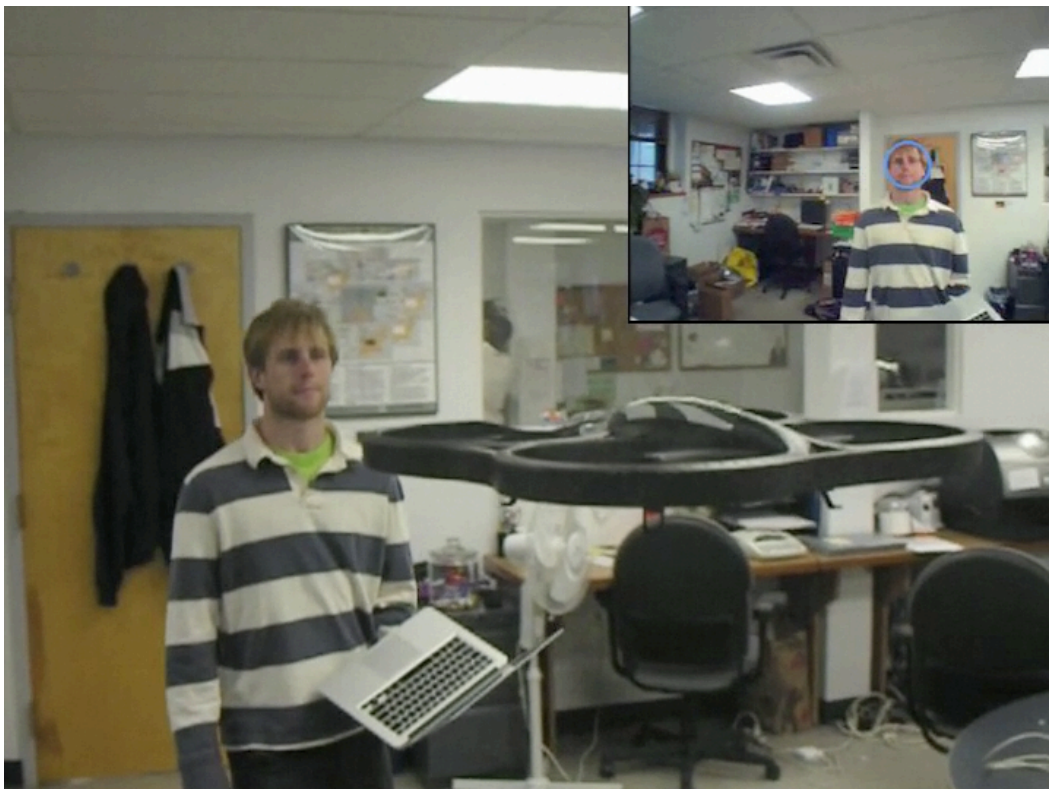


Figure 3.1: Face Tracker VI in action. AR Drone's view in upper right corner with identified face marked with blue circle.

3.3 Vanishing Point Navigation

The second demonstration to be discussed is an indoor navigation program, which uses vanishing point detection to direct the AR Drone towards the end of a hallway while avoiding walls. This program is called

Vanishing Point Navigation VI. The vanishing point navigation algorithm superimposes vectors on top of edges within the incoming video frames and calculates the approximate intersection point between each vector. For a hallway, the most prominent edges are (1) the intersections between ceiling and walls, (2) walls and floor, and (3) the outline of doors and windows. If the vertical edges are ignored, vectors superimposed on edges will intersect at the center of the end of the hallway.

To detect strong edges, the video frames are passed to OpenCV via the wrapper functions and the OpenCV function cvCanny( ) is called. This function uses a Canny Edge Detector to process the incoming image with a variety of steps to isolate edges. An edge is defined as a sharp intensity gradient. The function cvCanny( ) returns a binary image with every pixel associated with an edge set equal to one and the rest are zero. This binary image is passed to cvHoughLines2( ), which employs a Linear Hough Transform to calculate the location and orientation of vectors corresponding to straight edges. The Linear Hough Transform plots several lines though each nonzero pixel and determines which line best fits the nearby neighborhood of nonzero pixels. It adds the slope and intercept of each line to a two dimensional accumulator space. After this procedure is completed for every nonzero pixel the local maximums of the accumulator space are found via thresholding or numerical differentiation. The slopes and intercepts of the most common vectors are returned to Vanishing Point Navigation VI.

Vanishing Point Navigation VI organizes these vectors into a system of equations in the form $Ax = b$:

$$\begin{bmatrix} -m_1 & 1 \\ -m_2 & 1 \\ \vdots & \vdots \\ -m_n & 1 \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} y_1 - m_1 x_1 \\ y_2 - m_2 x_2 \\ \vdots \\ y_n - m_n x_n \end{bmatrix}$$

Equation 3.1

Where m is the slope of each vector and *(x, y)* is a point that lies along the vector. The vector $\begin{bmatrix} c \\ d \end{bmatrix}$ is the intersection point. Equation 3 is over specified requires a pseudo inverse of matrix A in order to find the solution.

Figure 3.2 shows the binary image obtained from cvCanny( ) with red vectors obtained from cvHoughLines2( ) and a yellow square marking the approximate intersection point between the vectors. Similar to Face Tracker VI, the intersection point is used in a proportional controller to keep the intersection point in the middle of the front camera's field of view. When a valid intersection point exists the AR Drone is pitched forwards at three degrees to progress down the hallway.

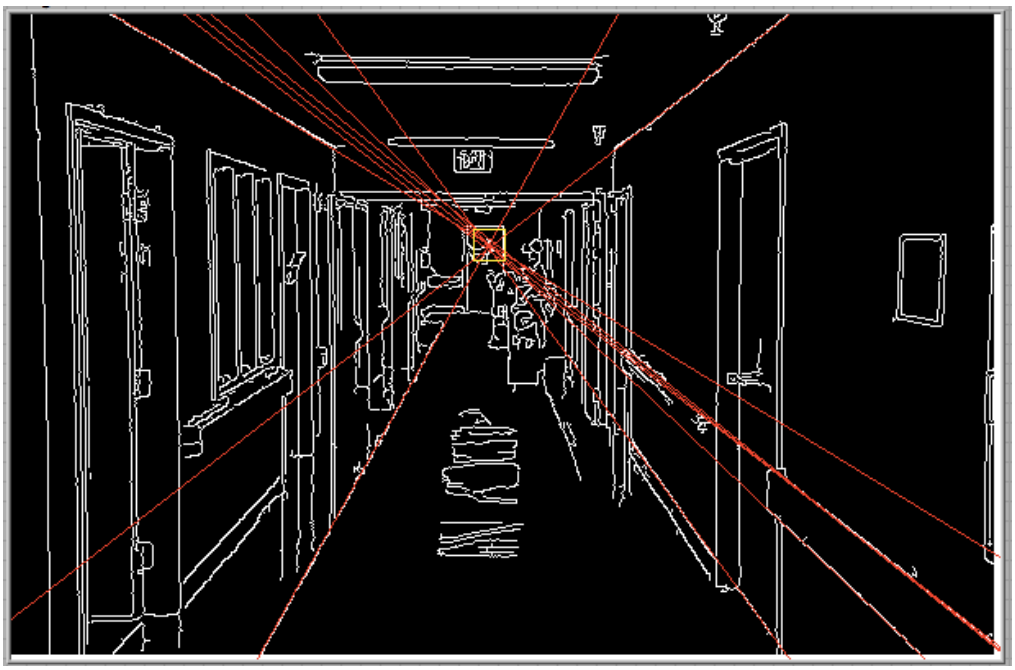


Figure 3.2: Vanishing point from intersection of vectors superimposed on detected edges

A video of Vanishing Point Navigation VI in action [25] shows the AR Drone successfully navigating its way down a hallway despite occasional disturbances such as people crossing in front of the AR Drone and trash bins and other items partially obstructing the AR Drone's view. Vanishing Point Navigation VI fails under two conditions: (1) extreme changes in hallway geometry such as 90 degree turns and (2) when no edges are in view such as when the AR Drone is facing a featureless wall.

3.4 Fly Through Hoop

The final demonstration is a program to pilot the AR Drone though a hula-hoop entitled Fly Through Hoop VI. The AR Drone detects a hoop

within the view of the front camera, then orients itself towards the center of the hoop, flies forwards until it has passed through the center, and lands. The hoop is wrapped in bright green tape to stand out from the background.

Fly Through Hoop VI takes the video frame from Video VI and first applies a green mask. The threshold for the mask is set high enough so that only pixels part of the green hoop pass though the mask. The resulting binary image is then sent to Circle Detector VI. Circle Detector VI fits a circle to the binary image and provides the center point and radius of the circle. The center point and radius are used in a controller structured like a state machine. When there is no circle detected, the controller outputs zeros for all movement commands sent to the AR Drone. When a circle is detected, the *yaw* and *climb* axes of movement are manipulated to bring the center point of the circle to the center of the front camera's field of view.

Adaptive ROI VI is used to keep the center of the circle aligned with the AR Drone even when the AR Drone is pitched or rolled off level. The AR Drone only proceeds forward towards the hoop when the error between the center of the hoop and the center of the camera's field of view is below 24 pixels. This prevents the AR Drone from closing the gap between itself and the hoop before it can orient itself towards the hoop. If the radius of the hoop exceeds a threshold while proceeding towards the hoop, the controller switches to a movement script. The AR Drone moves forward for three seconds and then lands. When the circle has a very large radius, the controller knows the AR Drone is nearly though the hoop and can safely

proceed with its current course.

A video of Fly Through Hoop VI in action [26] shows the AR Drone successfully navigating through the confined space of a hoop. The AR Drone identifies a hoop, orients itself towards it, and begins flying towards the hoop. When the position of the hoop is changed, the AR Drone stops, reorients itself, and proceeds again, this time traveling through the hoop. A screenshot from this video can be seen in Figure 3.3.

Figure 3.3: The AR Drone traveling through a hoop.

While this method of flying through a hoop is neither optimized nor fast, it is robust to mild disturbances of hoop position (until the AR Drone has entered the hoop) and independent of environment (as long as the environment is not the same green color as the hoop). Circle Detector VI has the benefit of fitting a circle to a binary image of the hoop when only a

portion of the hoop is visible, possibly identifying the center of the hoop outside the bounds of the image. This has the benefit of virtually increasing the front camera's field of view.

4 Evaluation

4.1 Student Testing

The AR Drone LabVIEW Toolkit was tested for two days at the Boston University Academy high school in Boston, MA. Six 9th and 10th grade male students volunteered to beta test a version of the AR Drone LabVIEW Toolkit for two 45-minute periods. They were supplied with an AR Drone and two laptop computers with LabVIEW and the AR Drone LabVIEW Toolkit installed. After a 10-minute introduction to LabVIEW (only two students had used LabVIEW before) and a crash course in AR Drone control and image processing the students were instructed to create a program to autonomously control the AR Drone. There were two requirements: (1) the AR Drone must move in response to some external stimulus, but manipulating controls on the LabVIEW Front Panel was not considered external stimulus and (2) the AR Drone's video cameras must be utilized and some analysis of the video stream must be performed.

The students decided to create a red ball tracking program. Their final program can be seen in Figure 4.1. This program takes each video frame from Video VI, masks it for red, then sends it to the Find Column VI from the image processing library. Find Column VI splits the image into a number of columns and returns the column number containing the most

nonzero pixels. For this program the image is split into three columns, meaning Find Column VI will return 0, 1, or 2. Since Main VI requires control inputs to range from 1 to -1, the students subtracted 1 from the output of Find Column VI and wired the result to the *yaw* input of Main VI. The result of this program was that when the red ball was held in the left third of the AR Drone's front camera's view the AR Drone will rotate left, if the ball is held in the right third the AR Drone will rotate right, and if the ball is held in the middle third the AR Drone will remain stationary.

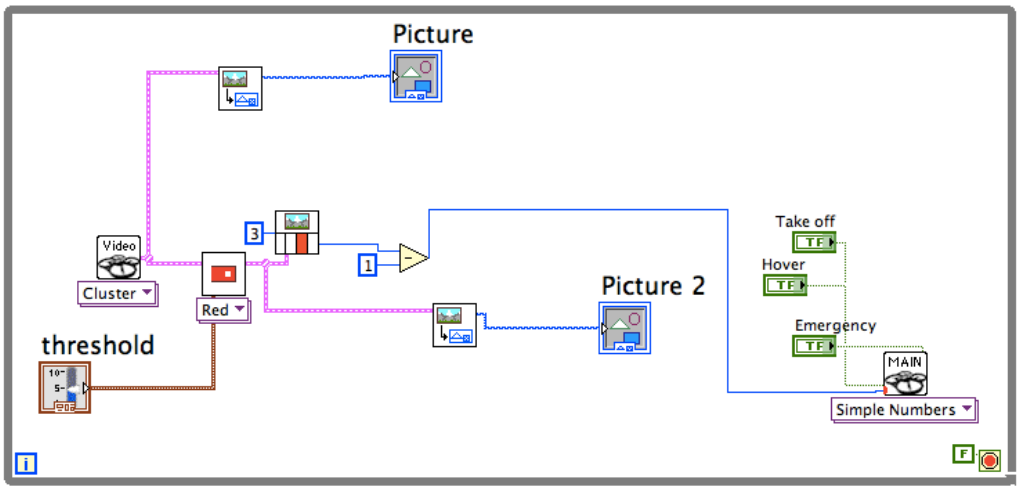


Figure 4.1: Red ball tracking program created by BU Academy students

The high school students created the previous program through trial and error over the span of a half an hour. I provided technical assistance as needed. During this time, the two computers proved beneficial. The students split themselves up into two groups. The first group worked with one computer to discover which input commands had what effect on the AR Drone by peeking into the wire running from Read Joystick VI to Main VI.

The second group used the other computer to explore the VI's in the image processing library. They were able to use the live video feed from the computer's web cam to develop the red ball identification part of the final program and copy and paste it into the first group's program.

On the second day of testing, the students were tasked with expanding their program. The students decided to create a person tracker that could identify a person within the view of the front camera and orient the AR Drone to follow him/her. The first modification was to replace the red mask with YCbCr VI and YCbCr Mask VI. With the right settings, these VI's segment out skin colored pixels from incoming images, thus detecting a person. The next modification was to add an instance of Find Row VI with three rows. This VI functions the same as Find Column VI except it splits the image into three rows instead of columns. The row number with the most nonzero pixels was used to direct the AR Drone to climb, descend, or remain level.

Once again, while the first group worked on the control of the AR Drone on one computer, the second group experimented with image processing on the other computer. The image processing group manipulated the settings of YCbCr Mask VI until they had isolated a person from the background based on skin color. The settings for accurate detection were then added to YCbCr Mask VI in the main program. During testing the AR Drone proved to be too "jittery" (as described by a student) so the max allowed yaw rate and climb rate were decreased in the Setup Parameters cluster.

By the end of the second 45-minute period the group of students had successfully created a person tracking AR Drone. The students' program wasn't foolproof since the AR Drone would not necessary track the same person from frame to frame. It would orient itself towards the column and row with the most skin colored pixels, which could be an arm, a leg, or a bystander's face. However, the person tracking program created by the BU Academy students is an achievement considering it performs roughly the same action as Face Tracker VI without relying on OpenCV and was developed in a short amount of time.

Student testing highlighted improvements that needed to be made to the AR Drone LabVIEW Toolkit. The version of the AR Drone LabVIEW Toolkit they tested had a dedicated preflight setup procedure VI. It proved easy to forget to run this VI first before the rest of the program. The final version of the AR Drone LabVIEW Toolkit moved the setup procedure within Main VI to be called automatically. Additionally, an option to wire in individual numbers for the *roll, pitch, yaw,* and *climb* inputs of Main VI instead of a cluster of values was added. Finally, the blob detector used at the time proved too slow for the students' programs and degraded the performance of the entire system. This observation led to the fast blob detector previously discussed.

Over the course of both sessions, these volunteer students had positive responses towards the AR Drone LabVIEW Toolkit. They were eager to have a working AR Drone and were willing to learn about new subjects

such as image processing in order to complete their goal. The students were proud of their work with the AR Drone. Their final program was demonstrated out in the hallway of the school to a small audience of friends that had come to see the AR Drone quadrotor. In general, these students thought that the AR Drone was "cool" and the fact that they volunteered their free period during the day is evidence of their excitement to use the AR Drone.

4.2 Educator Testimony

Educator William Church from Littleton High School in Littleton, NH has used the AR Drone and AR Drone LabVIEW Toolkit in a variety of educational settings. He has expressed excitement about the possibility of students using aerial robotics in the classroom, "I have watched this technology go from university labs to the State. It is now where a school can afford to purchase one of these units. It is amazing. To enable students to tinker with these tools can open their eyes to what is possible for them right now!"

Mr. Church has used the AR Drone with the AR Drone LabVIEW Toolkit to teach more than just robotics. In a college science course about space, Mr. Church used the AR Drone as a model for a spacecraft. He says, "While it is certainly not a space craft (propellers don't work in space!), it represents all of the systems one has on a space craft -- propulsion, position

control, data, and communication… While we can not go out and tinker with real space craft, we can explore how the quadcopter system works."

Finally, Mr. Church and other educators flew the AR Drone during a teacher workshop about using data collection in the classroom. Mr. Church explains, "We used the quadcopter system as an example of remote data collection. Teachers learned to fly the system by joystick, using the camera view alone to control its path. While it flew, they collected visual information. The visual information they collected was a list of objects they could see in the camera views. The experience of flying something by camera served as a discussion topic around collecting data in a remote place."

Mr. Church has demonstrated how a low cost quadrotor may be used in an educational setting to teach subjects other than robotics. Previously, cost prohibited quadrotors from being used in the classroom. However, the AR Drone and the AR Drone LabVIEW Toolkit have provided a classroom ready aerial robotics platform.

4.3 Conclusion

The AR Drone is an exciting piece of hardware that interests students and educators. A group of students at Boston University Academy were inspired to learn about programming, controls, and image processing in order to make the AR Drone perform the task they desired. One educator, William Church, has used the AR Drone to start discussions on topics broader

than just robotics such as forces, communication, and data collection.

The AR Drone LabVIEW Toolkit allowed these users to utilize the AR Drone to its full potential. It allowed them to start to understand how this quadrotor functions and how the AR Drone, computer, cameras, and software work together to create an autonomous flying craft. It has been shown that the AR Drone and AR Drone LabVIEW Toolkit have a place in the classroom and that the AR Drone LabVIEW Toolkit can successfully be used by students and educators.

5 Conclusion

The quadrotor is a capable aerial robotics platform, which is of interest to students, educators, and researchers. In 2011 the FIRST robotics competition league attempted to tap into the enthusiasm about quadrotors by creating the Collegiate Aerial Robotics Demonstration (C.A.R.D.) competition [27]. The inaugural competition consisted of ten schools participating in two different challenges where quadrotors had to assist autonomous ground robots and perform cooperative tasks. The quadrotors could be either teleoperated or autonomous. Videos of the competition show that teams had difficulty constructing quadrotors that could hover and be maneuvered in a stable manner [28]. A proposed third challenge where quadrotors would be manually or autonomously piloted through a series of hoops was scrapped due to the difficulty.

The AR Drone combined with the AR Drone LabVIEW Toolkit would be a good base robotics package for the C.A.R.D. competition, similar to the starter robotics kits available for other FIRST competitions [29]. With the AR Drone LabVIEW Toolkit, these college teams could focus on their aerial robotics application instead of the details of maintaining stable flight.

The AR Drone and AR Drone LabVIEW Toolkit is capable of performing the challenges put forth by the C.A.R.D. competition. For example, it was previously demonstrated that a program can be written to

autonomously fly the AR Drone though a hoop. This program needs only to be expanded in scope to pilot the AR Drone through a series of hoops & accomplish the third C.A.R.D. challenged, deemed too difficult. Furthermore, Fly Through Hoop VI was developed in less than a week due to the fact that the basics of communicating, maneuvering, and reading data from the AR Drone had already been accomplished. Only the image processing and control algorithms had to be created.

The AR Drone LabVIEW Toolkit has been used for student projects at a number of universities around the world. A group of students at the King Fahd University of Petroleum and Minerals in Saudi Arabia have created a program to pilot the AR Drone through a series of way points with the goal of autonomously inspecting oil pipelines from the air [30]. In email correspondence about their experience working with the AR Drone LabVIEW Toolkit, one of the project members Sohaib Darwish said, "We had only four weeks to deliver the presentation but I want you to know that we took two weeks to build the interface from zero."

Work with the Boston University Academy students and universities abroad demonstrates that the AR Drone LabVIEW Toolkit is effective an enabling students to create sophisticated aerial robotic behaviors in a short amount of time. The Boston University Academy students were able to prototype their programs and test and iterate to a fully functional control algorithm using a stable and inexpensive aerial robotic platform. Previously, to implement this level of intelligent control with a quadrotor would require

a $5000 – $12,000 research level quadrotor with a dedicated indoor space equipped with nets and motion capture cameras. The classroom setup required to use the AR Drone LabVIEW Toolkit consists of a Mac, Windows, or Linux PC with WiFi capabilities, a $50 student edition of National Instruments LabVIEW, and the $300 Parrot AR Drone.

It should also be noted that the AR Drone is light weight, has soft & flexible propellers which automatically shut off in the event of a collision, and a protective foam bumper. It is a much safer aerial robot to fly indoors compared to other research grade quadrotors.

The Parrot AR Drone and AR Drone LabVIEW Toolkit is a capable aerial robotics platform targeted towards students, educators, and researchers. It has proved valuable to student groups who have used it. I am making this software package available to download free of charge [31], in the hope that adoption of this platform will increase across academia and that it will continue to offer a low barrier of entry and high ceiling of capability for people interested in aerial robotics.

6 Appendix

6.1 AR Drone Commands

Each command is given in text form with C style format specifiers in place of arguemnts. The arguments are listed in order and <LF> is the line feed character with an ASCII value of 10.

Command	Arguments	Description
AT*REF=%d,%d<LF>	1. Sequence number 2. 32 bit integer representing the bit-field that controls the drone	Controls the basic behavior of the drone such as take-off, landing, and emergency stop
AT*PCMD=%d,%d,%d,%d,%d,%d<LF>	1. Sequence number 2. Flag enabling hover mode (0 or 1) 3. Roll 4. Pitch 5. Climb Rate 6. Yaw Rate	Sends progressive commands to make the drone translate or rotate. Roll, Pitch, Climb Rate, and Yaw Rate are floating point values in range of [-1..1]
AT*FTRIM=%d<LF>	1. Sequence number	Sets a reference of the horizontal plane for the drone's IMU
AT*MTRIM=%d,%d,%d,%d<LF>	1. Sequence number 2. Trim for roll 3. Trim for pitch 4. Trim for yaw	Sets an offset for the progressive commands
AT*ZAP=%d,%d<LF>	1. Sequence Number 2. Video stream with value [0..3]	Selects which video stream to broadcast midflight. Options, in order, are front camera, front within bottom camera, bottom, bottom within front

AT*CONFIG=%d,%d,%d<LF>	1. Sequence number 2. Name of option to set surrounded by double quotes 3. Option value between double quotes	Sets a configurable option on the drone. See Appendix 6.2 for details
AT*GAIN=%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d<LF>	1. Sequence number 2. K_D (roll & pitch) 3. K_P (yaw) 4. K_I (yaw) 5. K_P (roll & pitch) 6. K_I (roll & pitch) 7. K_P (altitude) 8. K_I (altitude) 9. K_P (climb rate) 10. K_I (climb rate) 11. K_P (hover) 12. K_I (hover)	Sets gains for drone's control loops
AT*COMWDG=%d<LF>	1. Sequence number	Resets the drone's communication watchdog
AT*PWM=%d,%d,%d,%d,%d<LF>	1. Sequence number 2. Front left 3. Front right 4. Back right 5. Back left	Sends PWM values directly to the motors bypassing the drone's control loops
AT*LED=%d,%d,%d,%d<LF>	1. Sequence number 2. Animation 3. Frequency 4. Duration	Makes the motor LEDs blink in a predetermined sequence
AT*ANIM=%d,%d,%d<LF>	1. Sequence number 2. Animation 3. Duration	Makes the drone execute a predefined movement
AT*CTRL=%d,%d,%d<LF>	1. Sequence number 2. Control mode 3. File size	Specifies whether the drone should acknowledge each command received

6.2 AT*CONFIG Options

This is the full list of options for both arguments of the AT*CONFIG command. Option names and values should be sent as text and surrounded by double quotation marks.

Option Name	Option Value	Description
GENERAL:navdata_demo	TRUE/FALSE	If TRUE is selected the drone sends a reduced packet of nav data every 30 ms. FALSE sends the full packet every 5 ms.
CONTROL:euler_angle_max	Floating point number in range [0..0.52]	Sets maximum allowed roll and pitch angles in radians
CONTROL:control_vs_max	Floating point number in range [0..20000]	Sets maximum allowed climb rate in mm/s
CONTROL:control_yaw	Floating point number in range [0..8.72];	Sets maximum yaw rate in radians/s
CONTROL:altitude_max	Integer in range [500..5000]	Sets maximum altitude in mm
CONTROL:outdoor	TRUE/FALSE	Adjusts sensitivity of the drone for outdoor vs. indoor flying
CONTROL:flight_without_shell	TRUE/FALSE	Lets drone know whether bumper shell is being used
VIDEO:video_channel	Integer in range [0..3]	Selects which video stream to broadcast at beginning of flight. Options, in order, are front camera, front within bottom camera, bottom, bottom within front

7 References

[1] "AscTec Hummingbird." Ascending Technologies GmbH. http://www.asctec.de/asctec-hummingbird-autopilot-5/ (accessed March 25, 2012).

[2] Y.M. Zhang. “Using Cutting-Edge Unmanned Aerial Vehicles (UAV) Technology for Courses Teaching.” *EDULEARN11 Proceedings. (2011)*: 3563-3573.

[3] "The Quadrotor's Coming of Age." Illumin. illumin.usc.edu/162/the-quadrotors-coming-of-age/ (accessed March 25, 2012).

[4] Krajnik, Thomas, Vojtech Vonasek, Daniel Fiser, and Jan Faigl. "AR-Drone as a Platform for Robotic Research and Education." *Research and Education in Robotics – EUROBOT.* (2011): 172-186.

[5] Sahin F. and Walter W., “Multidisciplinary Microrobotics Teaching Activities in Engineering Education”, *Proceedings of the 2003 ASEE Annual Conference & Exposition*, June 22-25, 2003, Nashville, Tennessee.

[6] Ulbrich, Peter, Rudiger Kaptiza, and Christian Harkort. "I4Copter: An Adaptable and Modular Quadrotor Platform." *Proceedings of the 2011 ACM Symposium on Applied Computing* (2011): 380-396.

[7] "Flight Dynamics and Control Lab - Dr. Taeyoung Lee" School of Engineering & Applied Science at The George Washington University. http://www.seas.gwu.edu/~tylee/people.html (accessed March 25, 2012).

[8] "Draganflyer X4 Four Rotor UAV Helicopter." Draganfly.com Industrial Aerial Video Systems & UAVs. http://www.draganfly.com/uav-helicopter/draganflyer-x4/ (accessed March 25, 2012).

[9] "MiKroKopter." MiKroKopter. http://www.mikrokopter.de/ (accessed March 25, 2012).

[10] “DIY Drones." DIY Drones. http://www.diydrones.com/ (accessed March 25, 2012).

[11] "ARDrone API." ARDRONE Open API Platform. https://projects.ardrone.org/ (accessed March 25, 2012).

[12] "ARDroneME - Java (J2ME) based AR.Drone Controller - by MAPGPS." Google Code Project Hosting. http://code.google.com/p/ardroneme/ (accessed March 25, 2012).

[13] "AR.Drone AutoPylot." Homepage of Simon D. Levy. http://home.wlu.edu/~levys/software/ardrone_autopylot/ (accessed March 25, 2012).

[14] Wang. E., LaCombe, J., and Rogers, C., "Using LEGO Bricks to Conduct Engineering Experiments," *Proceedings of the 2004 ASEE Annual Conference & Exposition*, June, 2004, Salt Lake City, UT.

[15] Piskorski, Stephane. "ARDrone Developer Guide SDK 1.5." https://projects.ardrone.org/projects/list_files/ardrone-api (accessed October 5, 2010).

[16] "LabVIEW Add-ons for LEGO MINDSTORMS NXT." NI Developer Zone. http://zone.ni.com/devzone/cda/tut/p/id/4435 (accessed March 25, 2012).

[17] Linsalata, Robert S. " Development of a Universal Robotics API for Increased Classroom Collaboration within Robotics Education." Tufts University, 2012. United States -- Massachusetts: *Dissertations & Theses @ Tufts University; ProQuest Dissertations & Theses (PQDT).* Web. 25 Mar. 2012.

[18] "LabVIEW Image Processing Library." DropBox.com. http://dl.dropbox.com/u/14022170/Code/ImageProcessing.zip (accessed March 25th, 2012).

[19] Chang, Fu, Chun-Jen Chen, and Chi-Jen Lu. "A Linear-Time Component-Labeling Algorithm using Contour Tracing Technique." *Computer Vision and Image Understanding* 93, no. 2 (2004): 206–220.

[20] Srinivasan, M. V.. "An Image-Interpolation Technique for the Computation of Optic Flow and Egomotion." *Biological Cybernetics* 71, no. 5 (1994): 401-415.

[21] Chang, Henry, and Ulises Robles. "Skin Color Model." EE368 Final Project Report - Spring 2000. www-cs-students.stanford.edu/~robles/ee368/skincolor.html (accessed March 25, 2012).

[22] Umbach, Dale, and Kerry Jones. "A Few Methods for Fitting Circles to Data." *IEEE Transactions on Instrumentation and Measurements* 52, no. 6 (2003): 1881 - 1885.

[23] "OpenCV 2.1 C Reference." OpenCV Wiki. http://opencv.willowgarage.com/documentation/c/index.html (accessed March 25, 2012).

[24] "AR Drone Face Tracking." [July 29, 2011]. Video clip. Accessed March 25, 2012. YouTube. http://youtu.be/VghVtljvWew.

[25] "AR Drone Vanishing Point Navigation." [July 29, 2011]. Video clip. Accessed March 25, 2012. YouTube. http://youtu.be/Lbkdkf5UQuM.

[26] "LabVIEW Controlled AR Drone Autonomously Flies Through Hoop." [March 9, 2012]. Video clip. Accessed March 25, 2012. YouTube. http://youtu.be/BQ9dFGxQpvY.

[27] "Home." Collegiate Aerial Robotics Demonstration. http://collegiateaerialrobotics.org/home (accessed March 25, 2012).

[28] "Collegiate Aerial Robotics Demo (CARD) Semi-Final." [May 2, 2011]. Video clip. Accessed March 25, 2012. YouTube. http://youtu.be/hum2TfdJyYo.

[29] "2012 FRC Kit of Parts Technical Resources." USFIRST.org. http://www.usfirst.org/roboticsprograms/frc/2012-kit-of-parts-technical-resources (accessed March 25, 2012).

[30] "Controlling Ar.Drone Using LabVIEW to Follow a Path (First Test)." [January 15, 2012]. Video clip. Accessed March 25, 2012. YouTube. http://youtu.be/_W50BKr-Br8.

[31] "Ardronelabviewtoolkit." The AR Drone LabVIEW Toolkit. http://ardronelabviewtoolkit.wordpress.com (accessed April 4, 2012).