DESIGN OF A VISION-BASED CONTROL SYSTEM FOR QUADROTOR SWARM AUTONOMY

by

Christopher Wesley

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Electrical Engineering

Charlotte

2015

Approved by:

_____

Dr. James M. Conrad

_____

Dr. Bharat Joshi

_____

Dr. Ronald Sass

ABSTRACT

CHRISTOPHER WESLEY. Design of a vision-based control system for quadrotor swarm autonomy. (Under the direction of DR. JAMES M. CONRAD)

Recently, small radio-controllable aircraft known as quadrotors, or quadcopters, have become very popular. These aircraft have the ability to vertically takeoff and move in any direction with great stability. They are also capable of carrying small loads, depending on the size of the quadrotor and strength of its motors. The most common applications of quadrotors for the average consumers are recreational activities such as recording video from high altitudes and other angles not accessible by humans. However, the applications of quadrotors and their usefulness in data acquisition extend far beyond leisure and simple delivery. The precision with which a quadrotor can move makes this aircraft a perfect candidate for reconnaissance of dangerous environments. When several quadrotors are networked together, this forms what is called a swarm. A quadrotor swarm can be a very effective way of performing tasks. The research that will be presented shall convey how this type of technology can be achieved.

ACKNOWLEDGEMENTS

I would like to thank everybody directly and indirectly involved with the production of this thesis. I would like to thank my family and my friends for giving me support throughout my college career. I would like to thank my mentor Dr. James Conrad for providing me with the means to pursue this thesis. I would also like to thank my friend and colleague, Terrill Massey, who worked on his thesis alongside mine and made the task of completing much more bearable.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Ah	ampere-hour
AIOP	All-in-One-Pro
DC	direct current
ESC	electronic speed controller
HDMI	high-definition multimedia interface
I/O	input/output pins
IR	infrared
LED	light-emitting diode
Li-Po	lithium-polymer
nm	nanometers
PID	proportional-integral-derivative
PWM	pulse-width modulation
RPM	revolutions per minute
USB	universal serial bus
V	volt

CHAPTER 1: Introduction

Quadrotor applications broadly fall under two categories: manipulating a payload and surveillance [15]. Many situations occur in which there is a need of some type of search, rescue, or exploratory effort required. In some cases, the environment in which the search takes place may be inaccessible to humans or may include hazardous material or unsafe structures, such as a nuclear disaster site where there may be radioactive material or falling objects. In cases such as these, humans need a way to precisely maneuver through an area while recording information and possibly retrieving objects. While manually remote-controlled quadrotors may be able to accomplish these tasks, it may take longer than necessary and there is the possibility of inefficiencies because of lack of coordination between pilots. A quadrotor swarm proposes to share the knowledge of any singular unit with the entire swarm. In terms of surveillance or surveying. this means that quadrotors in a swarm will not search places that others have already searched. In terms of moving payloads, this means that a quadrotor that is about to drop a payload off will know where other quadrotors have drop their packages. Where it may take a pilot several minutes to fully search a foreign environment it would take an autonomous swarm a fraction of the time, and in situations in which time is of the essence this makes a substantial difference. Current research into quadrotor swarms does not take into account close-quarters or inaccessible environments. Those networks require the use of an off-board coordination system such as a camera device that sees all of the quadrotors and issues commands from afar. The research presented here aims to place the coordinator within the swarm itself, granting more flexibility to the swarm. In order for a quadrotor swarm to be effective, the participating quadrotors will need to establish situational awareness with respect to

each other. In order to realize a swarm with situational awareness, each quad rotor is outfitted with a vision system and a communication system. The vision-system allows the quadrotor to establish its position with respect to other quadrotors within the swarm, whereas the communication algorithm establishes a method to coordinate the movements within the swarm to prevent collision and promote situational awareness throughout the entire swarm. The coordinator quadrotor will be using the vision system to adjust its position relative to a stationary LED beacon. This beacon ensures that the system has baseline vision data in order to mitigate swarm position oscillations. The vision-system is established through using a camera and a set of beacons on each quadrotor. Through careful management of movement commands, the vision system will be able to maintain the position of the swarm. The communication algorithm is realized through a wireless transceiver and a decision making process performed by the coordinator. The swarm algorithm allows the coordinator to make decisions on the movement of the surrounding quadrotors and the formation of the swarm overall. There is the possibility of a variety of formations, each of which the coordinator will reside in the middle of the swarm. The communication algorithm also establishes a decision routine for variability within the swarm. For example, if certain quadrotors of the swarm encounter an obstacle, the information will be relayed to the coordinator. This will be used by the coordinator in order to issue commands for the followers to avoid obstacles. With research conducted with the vision system and communication algorithm, a swarm formation that has on board movement processing can be realized. This allows for a more autonomous system. In the future, quadrotor swarm functionality may become even more useful for other applications. The complex coordination that swarms can provide may be used for construction projects or autonomous surveillance. The purpose of this thesis is to thoroughly examine the methods that have been constructed in order to successfully create a quadrotor swarm system.

CHAPTER 2: BACKGROUND

2.1 What is a Quadrotor?

A quadrotor is multi-rotor helicopter capable of producing lift using four vertically oriented rotors. Quadrotors utilize two sets of propellers, in this case fixed-pitch: two rotate clockwise, and the other two counter-clockwise. Each propeller is rotated via a brushless DC motor. By varying the RPM of each rotor individually, the quadrotor's thrust and lift characteristics can be changed in order to adjust its roll, pitch, and yaw; thus giving the ability to perform a full range of aerial movement, including roll/pitch motion, altitude adjustment, and yaw rotation as shown in Figure 2.1 below. For a quadrotor, a change in roll causes movement to the left or right with respect to the forward direction of the quadrotor, while pitch causes movement forwards or backwards. By varying the speed each propeller rotates, quadrotors can fly agilely [10]. By using a radio frequency transmitter, PWM commands are sent to the flight controller, which as explained above will vary the RPM of each motor. A transmitter is required to have at least 4 channels, each supporting one degree of freedom for the quadrotor: throttle, yaw, pitch, and roll. Radio channels allow for multiple types of commands to be sent to the controller for simultaneous command issuing. Channels and firmware can be configured to other movement types for the quadrotor. For this application it is necessary to have a transmitter and receiver that support at least 5 channels. The fifth channel will be used to activate the quadrotor's altitude hover capability, one of the vital functions of a quadrotor swarm.

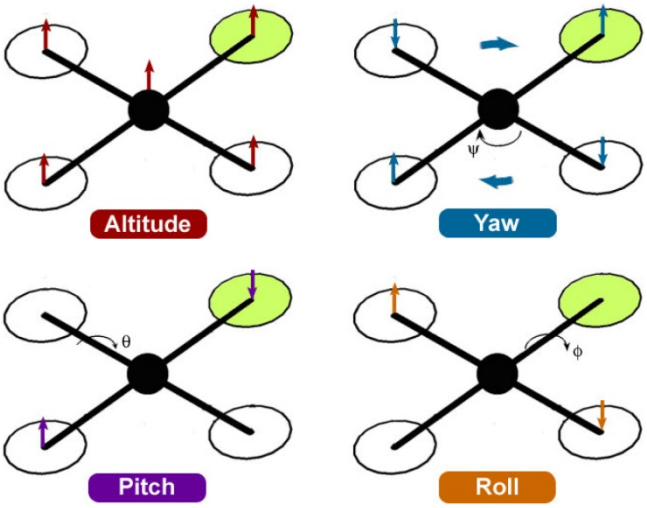


Figure 2.1: Quadrotor degrees of freedom [16]

2.2 Why Quadrotors?

Quadrotors have become a popular platform for hobbyists and professionals alike for good reason. Because of their design quadrotors are a very stable aircraft, making them capable of performing the same functions of helicopters such as vertical takeoff, lateral movement, precision control, the capacity to carry loads, and the ability of maintain a single position. In addition to this, quadrotors have the advantage of being fully symmetrical along two axes, which means it will easily maintain stability as long as its load is equally distributed across itself. As opposed to helicopters that have two rotors, one large main rotor and a smaller rotor, with a large torque on the main rotor, quadrotors have four rotors to distribute the torque. This means that each propeller is spinning with less kinetic energy and reduces the amount of damage the rotors can do should there be a crash. Quadrotors also have a more compact and modular form factor than helicopters, making assembly and maintenance simple, as well as providing a platform on which electronics, sensors, and other equipment can be easily mounted. Depending on the size of the quadrotor and the torque of its motors, it is capable of carrying a variety of loads. Recently, quadrotors have been constructed that carry upwards of 50 kg [6]. The variance of the size and strength

of quadrotors and their unique applications makes swarm control even more useful. Current quadrotor technology encompasses many aspects of quadrotors, including hardware, sensors, autonomy, software tools, customizability. A multitude of quadrotor flight controls exists, which when used in conjunction with on-board sensors, provide great stability and foundation for autonomy. The sensor types included on a quadrotor flight controller can consist of gyroscopes, accelerometers, barometers, ultrasonic, and global positioning systems (GPS). Flight controller configuration tools, such as Mission Planner, allow quadrotor-configuration and settings to easily implemented into the flight controller's memory. The controller can have these settings configured: PID controller constants for flight control, ranges for the radio-controller PWM commands, way-point navigation settings, advanced flight modes, and various sensor calibration.

2.3 Current and Potential Quadrotor Applications

The current applications for autonomous quadrotors include reconnaissance, search and rescue, surveillance, inspection, goods transportation, construction, security, and even personal entertainment. As opposed to robotics ground vehicles or ambulatory robots, quadrotors have the obvious advantage of changing position on a vertical axis. By having the capability of moving in three dimensions and the ability to carry loads, quadrotors have a myriad of possible uses. Professional photographers use them to take aerial images of landscapes and high-altitude areas of interest. The military uses UAVs for aerial reconnaissance, search and rescue missions in urban environments. Potential applications include: surveillance of privately property and war zones [11] [7], creating a deployable wireless communication network [17] [5], and in disaster relief [10]. Quadrotors have found surveillance application, including: riotous political movements, noninvasive inspection of buildings and public structures [8], and unsafe natural events such as mudslides and volcanoes [19]. Other applications currently being explored are delivery services, i.e. Amazon drones or Matternet [18] [9].

Figure 2.2: University of Pennsylvania swarm [12]

2.4 Swarm Functionality

It is said that there is power in unity and there is power in numbers. For robotics applications, this is no exception. Using many quadrotors simultaneously for single large task is where quadrotor swarm potential lies. Where it may take one a person a few hours to survey an area of land, a swarm of quadrotors would take a fraction of the time; where a single quadrotor may not be able to lift a certain load, multiple quadrotors could lift it. A quadrotor swarm proposes to share the knowledge between all of the quadrotors. This means that quadrotors in a swarm will not search places that other quadrotors have already searched. Where it may take a pilot several minutes to comprehensively search a foreign environment it would take an autonomous swarm a fraction of the time, and in situations in which time is of the essence this makes a substantial difference. Current research into quadrotor swarms does not take into account close-quarters or inaccessible environments. Those networks require the use of and off-board coordination system such as a camera device that sees all of the quadrotors and issues commands from afar. The research presented here aims to place the coordinator within the swarm itself, granting more flexibility to the swarm.

2.5 Current Swarm Research and Limitations

Current research into quadrotor swarm formations consist of very agile and well coordinated groups of quadrotors. They are controlled by an external system with a series of stationary cameras looking down at the swarm, such as the system shown in Figure 2.2. Limitations of this type of system stem from the fact that the cameras are a stationary system that are required to be placed above the quadrotors in order to issue commands to them. The most current work done on a vision system before the start of this thesis was the use of a Wiimote camera to detect IR LEDs [14].

2.6 Thesis Contribution

This research placed the control of the swarm into itself. By removing the limitations of an externally-operated system, the swarm has the capability to move without restriction. It will have more flexibility in ability to navigate through closed-off regions or areas that are otherwise incapable of having an off-board control scheme setup. Through use of a vision system implemented onto each quadrotor, the members of the swarm are given the ability to determine each other's location, even in dark, remote, and otherwise visually inaccessible locations. The vision system will be robust enough to work in environments that may not be suitable for use of GPS, or color-based object detection. This vision data will be held by a coordinator. The coordinator quadrotor will issue movement commands to follower quadrotors through a swarm communication algorithm.

CHAPTER 3: RESEARCH OVERVIEW

3.1 Objective

The main objective of this research is to examine the best method of creating an embedded processing platform that can be used by a set of quadrotors greater than two in order to maintain and establish a quadrotor swarm formation. The formation will consist of a coordinator quadrotor that will be situated in the middle of the node, or follower, quadrotors, as shown in Figure 3.1. The coordinator will have a set of beacons equipped to each of its four sides as well as a camera and will have initial visibility of a stationary beacon. Each follower quadrotor will be outfitted with a camera and is planned to initially have vision of the coordinator.

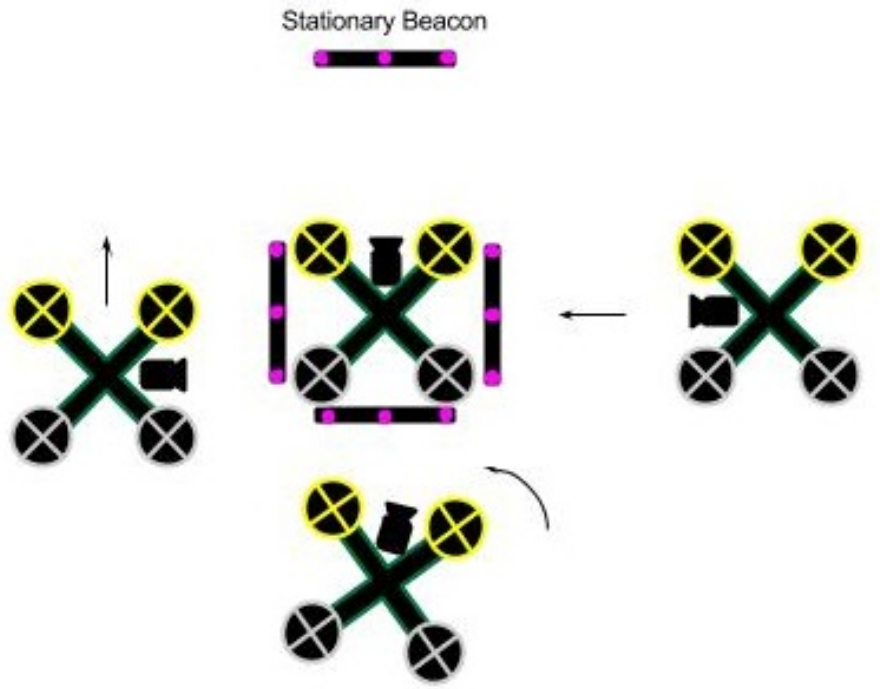


Figure 3.1: Example of a quadrotor swarm and basic functionality

A vision system will allow for the localization of the quadrotors and this will adjust for small variations in the positioning of the end devices with respect to the coordinator. The coordinator will continuously process position data of the stationary beacon, and adjust its own position in order to stay aligned with the stationary beacon. The follower will transmit its position with respect to the coordinator and it will record their position in space. After recording all of the position information of the followers, the coordinator will issue movement commands. To make communication simple, avoid collisions, and reduce other complexities, each quadrotor with be sent movement commands and execute them one at a time in round robin format. Each movement scenario proposed by a follower will be handled separately according to a hierarchy. This method prevents the system from becoming out of sync if there should be an error with a movement command. Another dimension of simplicity will be in the form of constant altitude positioning of all the quadrotors. The high-level flow diagram for the system is shown in Figure 3.2 below. All quadrotors in a formation will need to maintain the same altitude for the purpose of this design.

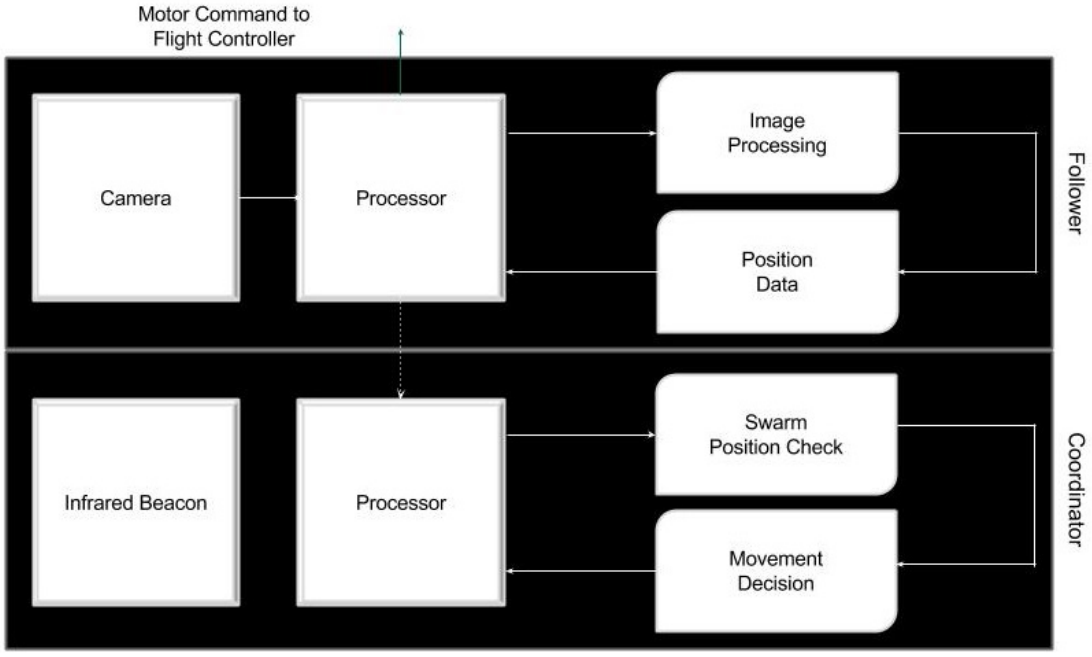


Figure 3.2: High-level flow diagram for interaction between coordinator and follower

This research focuses on developing swarm control on quadrotors, not designing a flight system for them. For this reason, a pre-existing flight controller is used in this design. The quadrotors platform used utilizes the Crius AIOP flight controller and the MegaPirateNG flight control firmware. This flight controller normally receives movement commands from a radio transmitter in the form of pulse-width modulated values whenever the controller joysticks are moved. These commands consists of the following: throttle for vertical movement, and yaw, roll, and pitch adjustment as well as trim adjustment controls for each of the movement joysticks. The flight controller receives the commands sent from the transmitter and converts this information into usable data. These data are run through the flight control algorithm and sent to the electronic speed control for the propeller motors. The receiver on the flight controller can be bypassed in order to send commands directly into the flight controller, in this case through the Raspberry Pi's general purpose input/output pins.

3.2 Equipment and Software Selection

3.2.1 Flight Controller

One of the major factors in the decision to use the equipment that has been chosen for this research is cost-effectiveness. The flight controller was chosen out of four possible candidates: the Crius AIOP v2, the Pixhawk, the APM2, and the PX4. These choices were dictated by the pre-existing components recommend by Ardupilot, an open source UAV platform for controlling quadrotors. As it can be seen in the Table **??** below, the Crius AIOP, shown in Figure 3.3 was the choice controller. The majority of the reason was because the swarm system need to be of low cost; the cost component of each controller held 50 percent of the weight. The Crius, priced at 40 dollars, was nearly 7 times cheaper than the most expensive flight controller.

Table 3.1: Flight controller decision matrix

	Crius AIOP	Pixhawk	APM 2.6	PX4
Cost	10	1	1	2
Performance	8	10	8	10
Support	7	10	10	10
Interfacing	10	10	10	10
Weighted Sum	93	55	51	50
Rank	1	2	3	4

3.2.2 Battery and Motors

Choosing a proper set of motors for the quadrotor was imperative for complete testing. The amount of battery power necessary to produce the torque to lift the quadrotor and maintain flight for a reasonable amount of time were the major concerns. In order to choose the most efficient motor given the criteria, a few battery calculations must be made. First, a running time can be determined for the quadrotor. In order to obtain useful flight information, e.g. drift magnitude, maximum altitude, swarm efficacy, a baseline of 10 minutes of flight was used. Also, a lithium-polymer battery with a 2.2 Ah capacity, 3 cells, and a nominal voltage of 11.1V be used, since Li-Po batteries are the standard for most medium sized quadrotors and most ESCs will work only on 2 or 3 cell batteries. From this, a safe operating current draw can be determined using this equation,

$$I_{op} = \frac{B_{cap}}{t_r} \tag{3.1}$$

$$I_{op} = \frac{2200mAh}{\frac{1h}{6}} = 13200mA \tag{3.2}$$

where I_{op} is the battery's operating current, B_{cap} is the battery capacity in mAh, and t_r is the running time that has been chosen. With the values predetermined above, the operating current will be 13200 mA, or 13.2 A. The operation current will allow us to choose motors that are suitable for the quadrotor. The static thrust of the

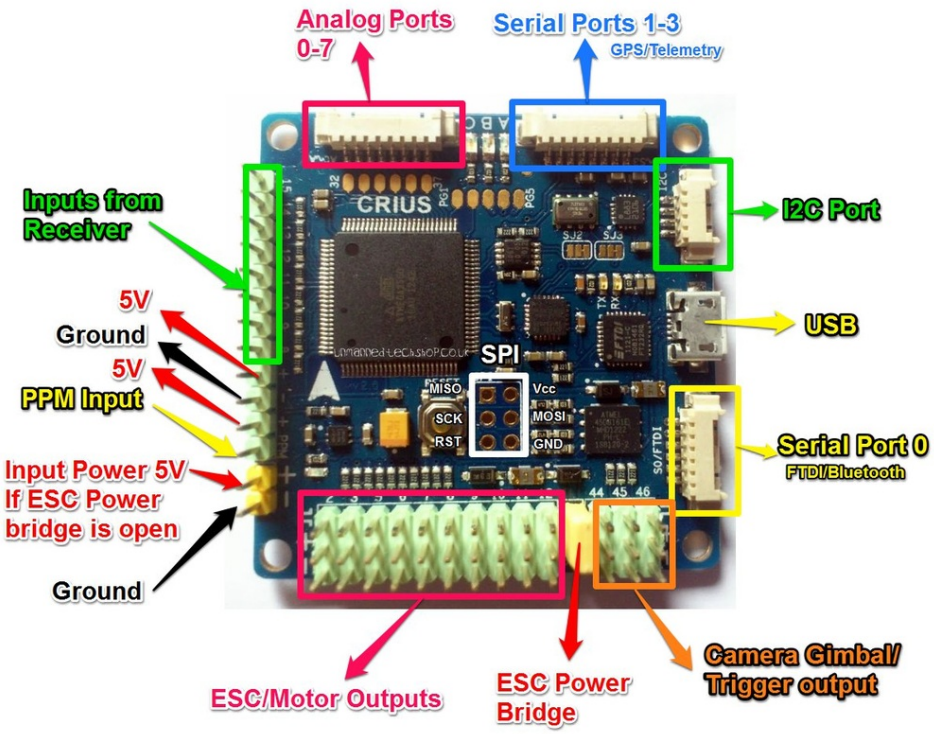


Figure 3.3: Diagram of Crius AIOP v2 [3]

quadrotor's propellers must also be determined. Because the quadrotors' purpose in this research is to move very precisely, but not necessarily quickly, static thrust calculation will apply to most situations at low ground speed. For this quadrotor, the propellers used will be 8045 slow fly propellers. In order to determine the static thrust, the power transmitted to the motor must be determined. Power can be found using the equation

$$P = K_p RPM^{power factor} \tag{3.3}$$

where P is the power delivered to the motors, K_p is the propeller constant, and power factor is characteristic of the propeller. Now the thrust of the motor and propeller will be determined. Based on momentum theory, the equation below can be used

$$T = \frac{\pi}{4} D^2 \rho v \Delta v \tag{3.4}$$

where T is thrust, D is the diameter of the propeller, ρ is the density of air, v is the velocity of the air at the propeller and Δv is the velocity of the air accelerated by the propeller. Commonly, the velocity of the air at the propeller will be half the value of the velocity of the air being accelerated by the propeller. Thus, the equation can be simplified to

$$T = \frac{\pi}{8} D^2 \rho \Delta v \tag{3.5}$$

The next equation will allow the power absorbed by the propeller from the motor to be calculated. Using this, the velocity of the air being accelerated by the propeller can be determined.

$$\Delta v = \frac{2P}{T} \tag{3.6}$$

Inserting into the thrust equation gives

$$T = [\frac{\pi}{2} D^2 \rho P^2]^{\frac{1}{3}} \tag{3.7}$$

Finally, using Newton's law, the static thrust in terms of mass can be calculated by the determined static thrust and the force of gravity, g.

$$m = \frac{T}{g} \tag{3.8}$$

$$T = \frac{[\frac{\pi}{2} 8^2 * 1.225 * 24^2]^{\frac{1}{3}}}{9.81} \tag{3.9}$$

$$m = 364.53 grams \tag{3.10}$$

It is known that the static thrust of each propeller is enough to lift 364.53 grams at a propeller speed of 10000 RPM. With this knowledge an appropriate motor can be chosen. The motor chosen will be the ST2210 brushless motor. It will produce 1050 RPM/V, 600 grams of thrust and will be able to operate on currents up to 10.2

A. Used with the batter chosen, the rotors will be expected to have the capability of operating at 115500 RPM. Theoretically, the quadrotor will have the ability to provide 1458.12 grams of static thrust, which will be the approximate weight limit of the quadrotor's frame, equipment, and any payloads.

3.2.3 Processor

Because of its high level of support and documentation, as well as its size, form factor, and low price , the Raspberry Pi was the optimal computer platform to have on the quadrotor. The functions of the Raspberry Pi would differ whether or not the quadrotor it was on was a follower or a coordinator. The Raspberry Pi Model B provides two USB ports, an Ethernet adapter, several general purpose I/O, HDMI, a maximum current draw of 500 mA at 5V, and the ability to interface with a camera, the Pi Camera. Shown in Figure 3.4, the Raspberry Pi's myriad of I/O pins will allow for even more versatility.

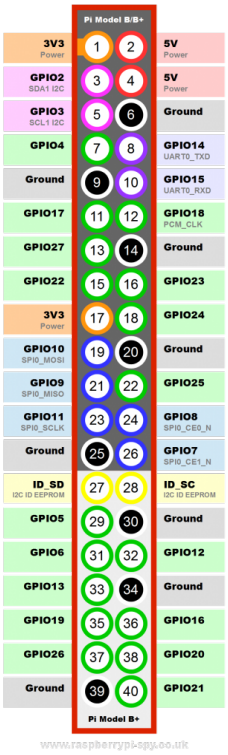


Figure 3.4: Raspberry Pi general purpose I/O [1]

3.2.4 Infrared Beacon

The initial design involved a small breadboard and three equally spaced infrared LEDs shown in Figure 3.5. The LEDs have a 940 nm wavelength and 20° beam-width. Each LED was spaced 3 inches apart, making the total length of the beacon approximately 6 inches, not including the unused breadboard space. In order to power the LEDs, three 1.5 V AA batteries were used in series with a small current limiting resistor. Creating a beacon with a consistent distance between each LED was paramount for the accuracy of measurements. It was also important to make sure that each LED was powered with the same amount of voltage in order to have consistent brightness across all LED. A bright LED allows the vision system to more easily calculate the contours of the image and determine the positions of the LEDs.

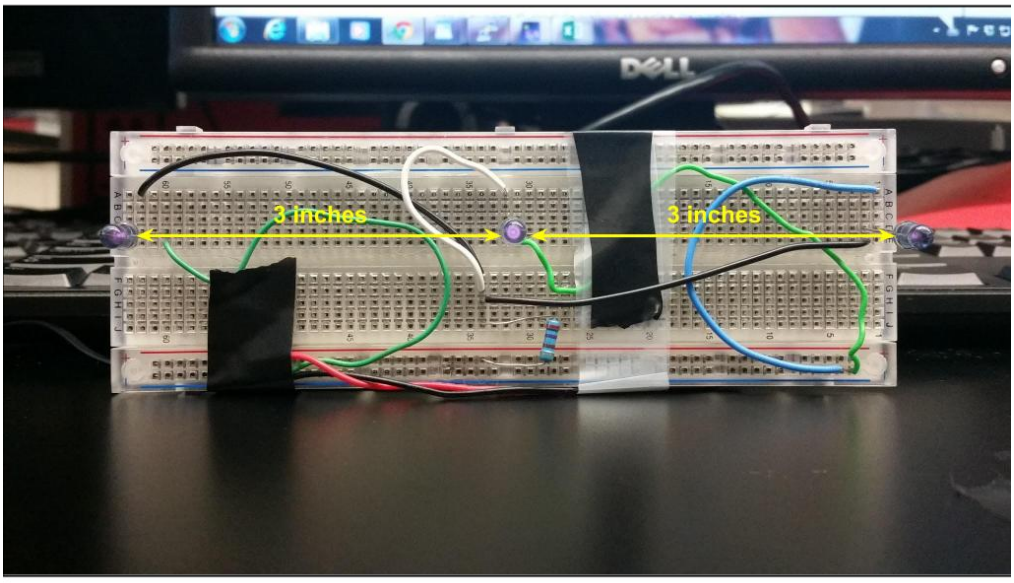


Figure 3.5: Prototype infrared beacon

After initial testing with the prototype beacon, focus was placed on finding an alternative beacon. Although the prototype beacon was sufficient for testing the

camera and processor's image processing capabilities, the prototype beacon could not be easily replicated. To solve this, it was decided that a Wii Sensor Bar would be used as the infrared beacon, shown in Figure 3.6. Since this item is mass manufactured, their dimensions would not differ very much. However, there are slight differences between the prototype beacon and the Wii Sensor Bar. Instead of having three LEDs arranged with equal spacing, this Wii Sensor Bar has six LEDs that are not all equally spaced. Instead, the LEDs are arranged into two groups of three, with a large gap in between the two groups. Changes would need to be made to the image processing code in order to accommodate for this difference. To easily deliver power to the infrared LEDs on the sensor bar from the Raspberry Pi, the default power connector can be replaced with a USB connector.

Figure 3.6: Wii Sensor Bar [2]

3.2.5 Camera

The camera used for the vision system was the Raspberry Pi's camera standard - the Pi Camera. The camera will easily be interfaced with the Raspberry Pi, with its own specific library. The camera is situated on a board that is 1 inch x 1 inch meaning that it has a low weight and compact form factor that can easily be mounted onto the quadrotor. It is a 5 mega-pixel camera with the possibility to take 2592 x 1944 pixel static images and can support up to 1080p video recording. Attached to the camera will be an infrared (IR) pass filter. This allows for the camera to see only the infrared beacon equipped to the coordinator. The idea behind using infrared lights for a vision system stem from the idea of all purpose environment utility. Having an infrared beacon means that there will be no interference from visible light spectrum, as well as being able to have detection in completely dark environments. Limitations of the camera come with the introduction of obstructions blocking the path of the camera, including solid opaque objects and possibly water in the form of a mist or rain.

3.2.6 Operating System

The operating system used with the Raspberry Pi was Raspbian, a Debian variant specifically made for the Raspberry Pi. This was chosen because it is the most typically used operating system for the Raspberry Pi and has the most documentation associated with it.

3.2.7 Programming Language

The programming language chosen was heavily dictated by the types of image processing libraries that are available for it. OpenCV is an open source image-processing library made to be used with Python and C/C++. Because of the simplicity with which code can be prototyped, and the pre-existing support for OpenCV, Python was the best candidate.

3.2.8 Other Software Tools

The firmware used for the AIOP was the MegaPirateNG open source firmware. In order to apply this firmware to the hardware Arduino software was used. After this, MissionPlanner was used to adjust the flight settings of the Crius AIOP. Finally MATLAB was used to analyze some of the data recorded from the vision system.

CHAPTER 4: PRELIMINARY QUADROTOR SETUP

This chapter will cover the basic steps required to properly interface with the quadrotor flight controller, transmitter and receiver, and the swarm communication algorithm.

4.1 Extracting Values from Transmitter

It may be that not all radio transmitters do not have the same joystick position to PWM correspondence. A method of determining the exact values that the receiver is output is necessary in order to completely and accurately recreate the PWM values that the transmitter is sending. In order to accomplish this, we must simply measure the output pins on the receiver while varying the controller's joystick positioning.

4.2 Constructing a Flight Test Environment

Testing the quadrotors for autonomous flight can be a challenging prospect. By having the inability to directly control what the aircraft is doing, the quadrotor or anything around it is susceptible to being damaged. This makes creating a safe testing environment for the quadrotor a vital component of solving autonomous flight. Although the quadrotor's destructive capability is not as great as a single rotor of equivalent lift, should the quadrotor hit a surface, the propellers may easily unscrew themselves from their motors and send the propeller cap as well as the propellers themselves flying. In order to avoid the potential self-destruction of a quadrotor in the event of crash, a means of providing a soft crash zone is required. This is accomplished by laying down a series of foam cushions along the area of the test environment.

A major component of flight testing is making efficient use of the batteries. This

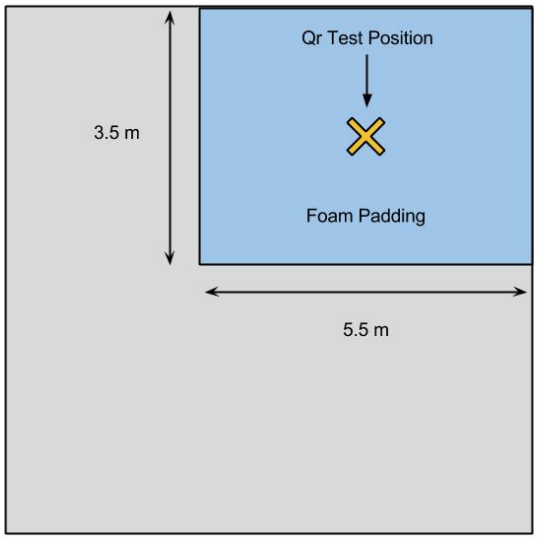


Figure 4.1: Illustration of safe quadrotor flight testing environment

means maximizing the lifespan of the battery through proper maintenance. Since the batteries used were standard for quadrotor flight, finding a battery recharger was trivial. In order to properly charge the batteries the battery cannot be charged at a current higher than the current capacity of the battery. This means for the battery used, the charging current must be at most 2.2Ah. However, charging at a rate less than the capacity, such as 70 - 80 percent or 1.54A - 1.76A and 4.2V per cell will extend the battery's lifespan and maximize capacity at full charge. Also, properly storing the batteries in a cool environment and at a 40 percent capacity will increase the battery's lifespan. Finally, partially discharging the battery will reduce stress on the battery and prolong the battery's life.

4.3 Quadrotor Drift

Initial tests with the quadrotor yielded problematic results. After loading the firmware onto the quadrotor we observed the barometer's output readings. For the

most part all altitude maintained by the quadrotor seemed to be stable, with a reasonable margin of error. However, when it came to the test flight we found that while the quadrotor would keep its altitude constant for the most part, it would begin to drift from its initial position even without input from the operator. Several tests were performed at varying altitudes in order to ascertain the extent of quadrotor drift. It was determined that the constants for the flight controller's PID control system would have to be adjusted. Fortunately, the PID values are easily changeable from the flight controller's interfacing software.

4.4 Swarm Algorithm

4.4.1 To Lead or to Follow

As mentioned before the processor for the quadrotor would behave differently whether or not it was a follower or a coordinator. As a follower, the responsibilities of the Raspberry Pi would as follows:

- to draw in images of the coordinator's beacon using the Pi Camera
- perform image processing on the images to determine the locations of the LEDs on the beacon
- calculate position information from the LED location data
- handling interrupt service routines for ultrasonic sensor data in the case of potential collision
- send position and potential collision and receive movement command information to and from the coordinator wirelessly
- issue pulse-width modulation commands to the flight controller...

As a coordinator, the responsibilities become the management of the followers and coordination of movement commands. These responsibilities include:

- measure position relative to the stationary beacon and perform self-adjustment
- receiving position information from followers and sending movement commands to followers
- routinely checking the position of all follower nodes in the swarm
- handling interrupt service routines for ultrasonic sensor data in the case of potential collision
- issuing movement commands if a follower leaves the marginal drift zone
- calculating and sending pathing commands to followers if the coordinator wishes to perform formation changes
- issue pulse-width modulation commands to the flight controller...

4.4.2 Optimal Quadrotor Positioning

A very important aspect of the quadrotor formation is the selection of the distances the quadrotors must keep from each other. Each quadrotor must have a sufficient amount of space around itself that will provide a margin of error for hovering.

CHAPTER 5: VISION SYSTEM

OpenCV contains a myriad of functions that are useful for this type of application. However, there are really only two that are vital to the system's ability to determine beacon information. The first major tool is the contour finding function. The second is the moment calculator.

5.1 Coordinator Baseline Margin of Error

Although the coordinator will be adjusting itself with respect to the stationary beacon, there will most likely be a margin of error associated with its position. Since drift characteristics of quadrotors tend to remain the constant, the coordinator must calculate its margin of error characteristics. In order to do this, the coordinator's vision system must sample images of the beacon over the course of a period and find in what vectors it has moved. Then, patterns must be found in the sample position data over the course of the period. The position data collected will then be able to be classified into one of three drift characteristics: linear drift, periodic drift, and erratic drift.

5.1.1 Linear Adjusted Drift

The case of linear drift in the position of the coordinator is the most simple to characterize. In this circumstance, the coordinator will have samples of the stationary beacon that have a linear change in position over the course of time. Once the system determines that the quadrotor has a constant linear drift, meaning that the period of sampling produces evidence of linear drift, the coordinator can characterize the information and provide the followers with adjusted movement commands.

5.1.2 Periodic Unadjusted Drift

In the case of periodic drift, the quadrotor will have an constant oscillation between positions, and will keep within the baseline angle of operation without the need of the vision system's compensation. This could occur in the form of linear drift from one position to the next, circular drift, or any specific periodic pattern of movement. This type of drift is simple to characterize as well, since the coordinator's position in time will be able to be determined based on previous data, as long as the system remains linear and time-invariant.

5.1.3 Erratic Drift

Erratic drift describes coordinator movement as being neither linear or periodic. In this case the vision system does not recognize the movement type as either.

5.2 Methods of Detecting Quadrotor Position

Two different methods for the detection of the presence of a quadrotor by camera were considered. The first was using a normal camera connected to a Raspberry Pi to record real time images of two different colored beacons while the quadrotor was in flight.

5.2.1 Color-based Detection

The images taken by the camera were be processed by a color detection algorithm. The algorithm consisted of masking the images two times on the beacon comprised of two colors. The position between the colors on the image would translate to identification information of a follower and the number of pixels of each beacon provided distance and angular information. This information would be used to suggest movement commands for the quadrotor to perform.

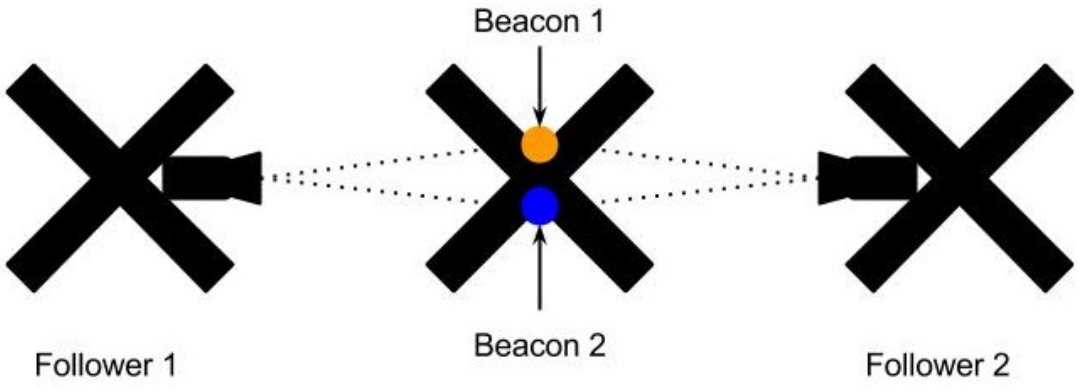


Figure 5.1: Example formation for color detection based vision system

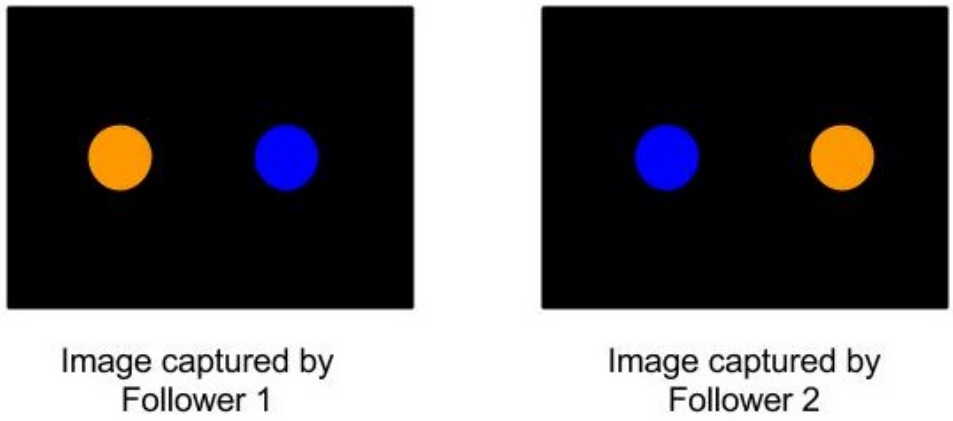


Figure 5.2: Example of images captured by differently positioned quadrotors

As it can be seen in Figures 5.1 and 5.2, the follower quadrotor would be able to know its own unique position in the virtual coordinate system after seeing the arrangement of the differently colored beacons on the coordinator. Unfortunately, this avenue was not a very viable scheme for the current system, because of processing time. Intervals between picture taken and data outputs would take 5 seconds. Even after reducing pixel operation time by decreasing the image resolution drastically to 200x150, increasing the trade-off between distance accuracy and runtime processing,

execution time would still take several seconds before providing the position information. Another drawback of this method was the inability to completely adjust for specific color ranges that are produced by the environmental ambient light. A dynamic color shifting algorithm would require to run periodically in order to compensate for different lighting situations. This algorithm was created, but the runtime of the program was even greater than that of the color detection algorithm. The solution was to reduce the amount of time processing pixels. This would be accomplished by two things. The first was to reduce time differentiating the beacons from the background. This was accomplished by using three infrared light-emitting diodes as a beacon and introducing an infrared filter to the camera. This method produces images that solely consist of a black background and three "blobs" on the image. In order to recognize the pixels of the LEDs on the image. Through use of the open source image processing library, OpenCV, determining the location of the LEDs becomes very simple. With the locations of the LEDs on the image, the actual distance the camera is from the beacon. By arranging the three LEDs in a row, all of the quadrotor's position information can be determined. Firstly, the lateral position of the quadrotor is found by simply detecting whether or not the middle LED of the beacon is centered on the image. If this LED is not in the center of the image, movement commands are sent from the camera quadrotor to the coordinator and will move such that the middle LED will be in the middle of the image. Next, The angle of the camera quadrotor to the beacon is checked. This is accomplished by calculating the difference between the distance between the right and middle LED to the distance between the left and middle LED. If the angle the camera is facing is not the same as the beacon, then these two distances will not be equal. Again, the quadrotor will send movement commands and adjust its position. Finally, the quadrotor will validate the distance between the camera and the beacon. Calculating the distance between either the left and middle or right and middle LEDs will yield a number

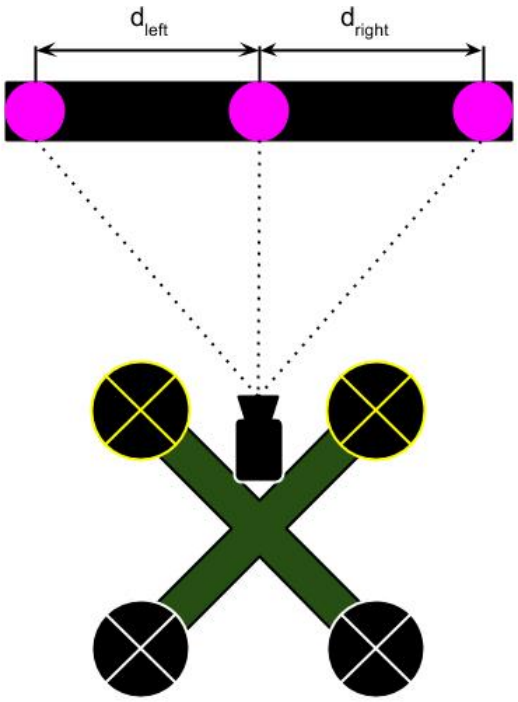


Figure 5.3: Distance measurement between quadrotor and beacon

that can be translated to distance information. Since this data was solved for in a previous step, it can be used again for this one. If the distance is above or below a certain threshold, the quadrotor will send movement commands and compensated for the discrepancies.

5.3 Camera Characteristics

With any camera, it is important to determine what types of settings to use for the application at hand. The Pi camera has a myriad of settings that can be used for a variety of lighting situations. In order to maximize the appearance of the IR LEDs on the image, a few of the default settings on the camera needed to be changed. Immediately, several of these settings can be eliminated as candidates for exposure options.

5.3.1 Camera Exposure Testing

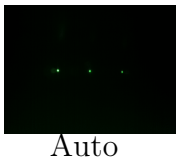
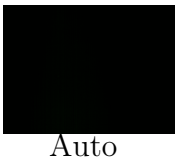
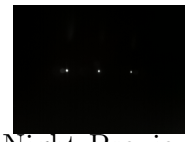
Out of the many settings that the Pi camera can be configured, it was decided that 12 test pictures would be taken using the most appropriate camera settings. These settings were taken with automatic white-balancing off or on and consisted of 6 different exposure modes: off, auto, night, night preview, back-light, and fireworks. Each entry in the Table 5.1 below is a picture taken with a different exposure types.

As can be seen from the sample images, auto-white balancing wa a necessary setting to have enabled in order to properly see the LEDs. The best candidates for camera settings consisted only of pictures that were taken using automatic white balancing. Out of the 6 pictures, a series of tests were constructed to determine the exposure type's ability to produce images that could best be processed by the contour finding algorithm. Each test consisted of an exposure type for the camera. Also, the camera took a 50 pictures of the beacon situated at a constant distance away. Each one of the pictures had the contour algorithm performed on it. The percent of times the algorithm was able to successfully detect the full beacon was be the measure of effectiveness. Next, the distance between the camera and the beacon was be changed and the same tests previously mentioned were performed. Ultimately, the camera setting that achieved the highest percentage of full beacon detection was the most effective option. As can be seen in the Table **??** above, the camera set to automatic white-balancing and an 'off' exposure type if by far the most effective. After determining the optimal exposure settings, the next task is to determine the other camera settings in order to produce the most reliable possible image processing system.

5.3.2 Optimizing Other Camera Settings

Other camera settings that the Raspberry Pi and OpenCV allow for that are pertinent to this specific arrangement include: resolution, ISO, and our image processor's thresholding limit. In order to thoroughly determine the optimal setting for each of

Table 5.1: Sample images of beacon at varying exposure types

AWB On	AWB Off
 Off	 Off
 Auto	 Auto
 Night	 Night
 Night Preview	 Night Preview
 Back-light	 Back-light
 Fireworks	 Fireworks

the parameters, trials were done on each of the setting's ranges.

5.3.3 Camera Resolution Properties

The first setting that was be tested was the resolution of the image. For this application, resolution will affect the distance ranges of the image processing algorithm. The hypothesis is that the higher the resolution of the image taken, the more pix-

Table 5.2: Percentages of full beacon recognition with varying exposure types at varying distances

Expoure Type	10 in.	12 in.	14 in.	16 in.	18 in.	20 in.	22 in.	24 in.
Off	100%	100%	100%	100%	100%	100%	100%	100%
Auto	60%	28%	36%	42%	32%	28%	32%	28%
Night	36%	44%	54%	40%	48%	44%	30%	30%
Night Preview	59%	40%	36%	70%	36%	46%	30%	40%
Backlight	51%	50%	78%	63%	42%	35%	44%	44%
Fireworks	49%	70%	76%	37%	30%	46%	28%	39%

els the camera will theoretically capture and be able to work with to determine the contours of the LEDs in the beacon. However, the trade-off would be the amount of processing time of each image would be increased as the resolution increases. Several test images of the beacon were taken with varying resolutions at a distance of 12 inches. These test shots are shown in Figures 5.4, 5.5, 5.6, and 5.7.

Figure 5.4: Low resolution beacon image

Figure 5.5: Medium resolution beacon image

Figure 5.6: High resolution beacon image

Figure 5.7: Ultra-high resolution beacon image

Testing on the maximum range using the optimal camera settings yields a maximum detection range of 48 inches. In order to test the efficacy of each resolution, 10 trials were conducted on each resolution with the beacon at 4 incrementally greater distances starting at 48 inches, for a total of 40 trials per resolution. The percentage of the time that the LED was detected and the average execution time of the program were the measures of the efficacy of the resolution. The processes performed during runtime were: taking the picture, determining the existence of the LED, finding the positions of each led, and printing test values. Values included were: LED detection rate, total number of trials run, and average runtime of the program.

5.3.4 Camera ISO Speed

Camera ISO speed describes the camera's sensitivity to light. Increasing a camera's ISO increases its sensitivity to light. The Pi Camera's ISO ranges from 100 to 800. The utility of this camera setting lies in its ability to increase the light that can be captured from the LEDs on the beacon. Using the camera's middle range resolution,

Table 5.3: Varying resolution LED detection test

Low Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	1.457s	1.458s	1.459s	1.461s
Mid Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	1.772s	1.779s	1.79s	1.788s
High Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	2.312s	2.33s	2.302s	2.311s
Ultra Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	3.707s	3.742s	3.664s	

a test was constructed that would compare the beacon's apparent brightness on the image as the beacon's distance increases.

5.3.5 Beacon Region of Recognition and Angles Limits

Using the optimal camera setting determined, the LEDs used for the beacon have a limited angle of visibility, thus the beacon itself will have a limited angle of visibility. Test were constructed that would determine the maximum angle the beacon could face from the camera. As can bee seen in Figure 5.8, the individual angle characteristic of the infrared LED used specifies a viewing angle of 120°.

A rough trial of 50 images of the beacon was taken at values ranging from 60° to -60° on the yaw axis relative to the camera. Also, The distance was varied from 36 inches to 72 inches at each angle. The idea of this test was to examine the functional ranges purported by the maximum angles found in the LED datasheet. Results confirm the accuracy of the datasheet, finding that full beacon detection occurs 100 percent of the time when the LED is not angled. Figure 5.8 also shows that the LED's intensity is reduced to 75 percent when the viewing angle is increased above 30°. We can see in Table 5.4 below that the maximum angle at which the LED beacon is visible is at

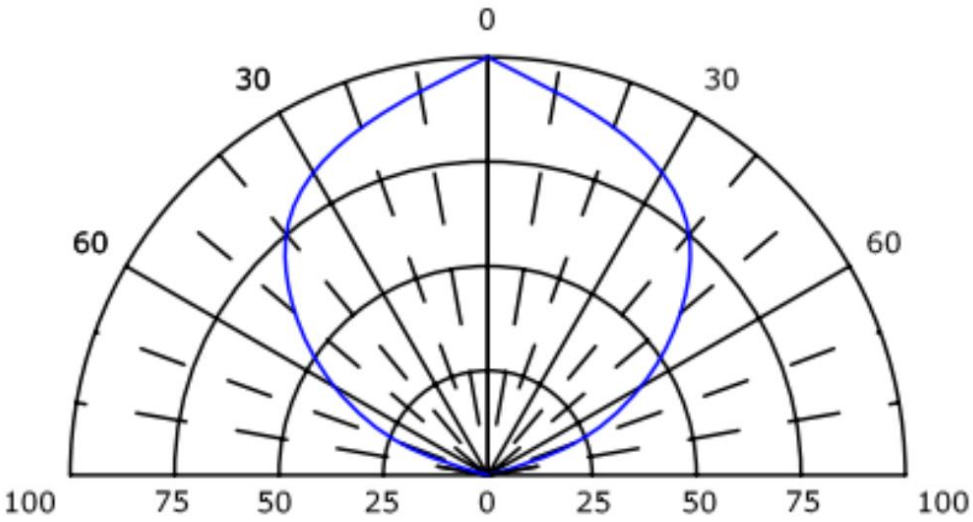


Figure 5.8: Characteristics of individual infrared LED angle vs. intensity [13]

30° at a distance of 18 inches.

Table 5.4: Rough trial of angle characteristics of LED beacon in 120° range

Low Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	1.457s	1.458s	1.459s	1.461s
Mid Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	1.772s	1.779s	1.79s	1.788s
High Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	2.312s	2.33s	2.302s	2.311s
Ultra Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	3.707s	3.742s	3.664s	

With the rough trials providing a range boundary, a second trial with an adjusted was required in order to accurately determine the angle characteristics of the beacon. The range was adjusted to 30° to -30° with step increases of 10° in between and the same distances being measured. As can be seen in the table below, the majority of angular displacement that the beacon can make while still being able to be recognized was at a 20° range. This result means that the follower quadrotor will have a maximum yaw displacement of up to 10° from the coordinator in order to provide position data to the coordinator.

Table 5.5: Angle characteristics of LED beacon in a 60° range

Low Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	1.457s	1.458s	1.459s	1.461s
Mid Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	1.772s	1.779s	1.79s	1.788s
High Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	2.312s	2.33s	2.302s	2.311s
Ultra Res	36 in	48 in	60 in	72 in
Detect Rate	100%	100%	100%	100%
Runtime	3.707s	3.742s	3.664s	

5.3.6 Experimentation with Beacon Distance to Pixel Spacing Relationship

Due to the nature of determining distance information from a 2-dimensional image, the relationship between true distance and the pixel spacing between each beacon requires sample data at various distances. In order to do this, pixel data calculated from a medium resolution image (1280x720) was recorded at distances measuring from 12 inches to 24 inches. Pixel data consisted of the average pixel difference between the left and middle LEDs, d_{left}, and the right and middle LEDs, d_{right}. The idea behind taking the average is that when the beacon is angled relative to the camera, d_{left} and d_{right} will be different, but the sum of the two will remain the same as if the beacon was not angled. The data points were then able to be plotted and examined. In Figure 5.9 below, the initial plot for the data was generated using MATLAB.

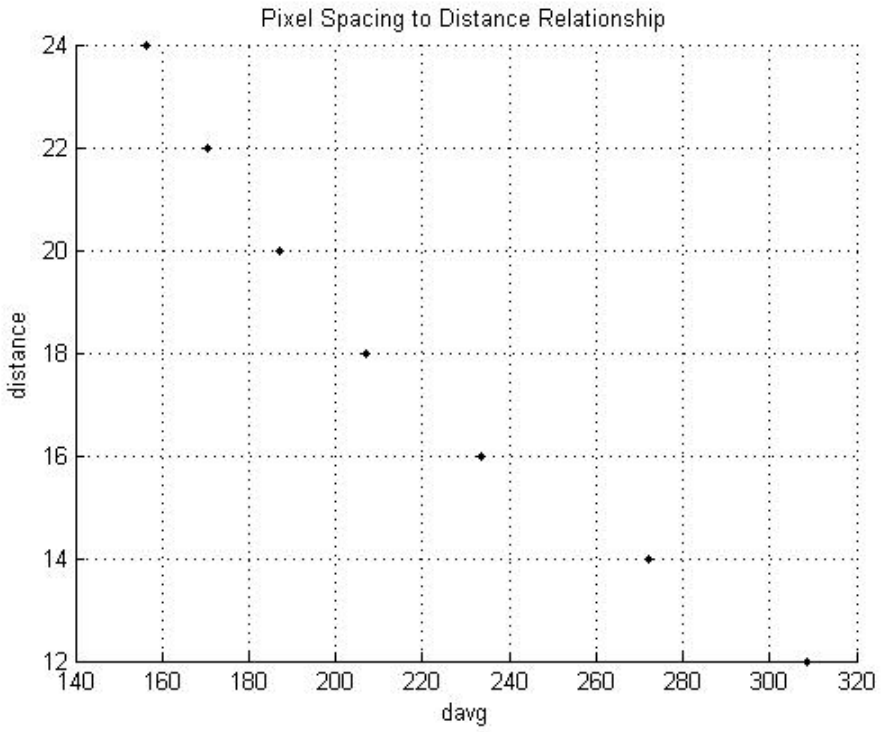


Figure 5.9: Pixel spacing to distance relationship

Figure 5.9 show the distance in inches versus the average number of pixels in between all of the LEDs in the image, d_{avg}. Through use of the MATLAB Curve Fitting Tool, finding a continuous relationship between 12 inches and 24 inches is made very simple. By performing a 2nd degree polynomial curve fitting, relationship can be approximated to a function:

$$f(x) = p_1 x^2 + p_2 x + p_3 \tag{5.1}$$

where f(x) is the distance between the beacon and the camera, x is the average pixel distance between the two halves of the beacon. Using the curve fitting tool, the constants can be determined, producing the approximation with 95 percent confidence bounds,

$$f(x) = 0.0003357x^2 - 0.2321x + 51.86 \tag{5.2}$$

The plot of the curve can be seen Figure 5.10 below.

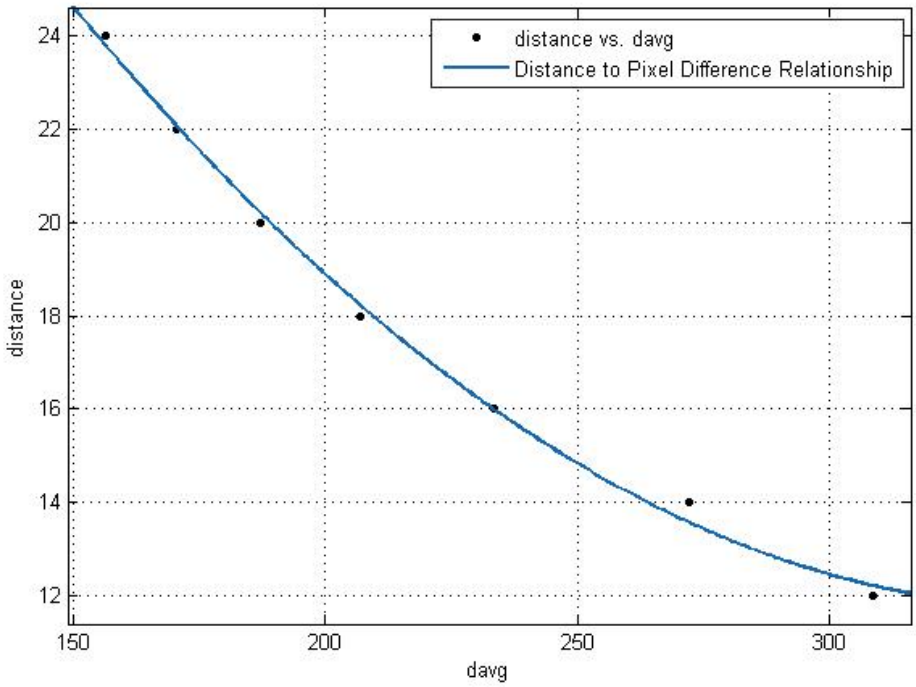


Figure 5.10: Pixel spacing to distance curve fitting

5.4 Trigonometric Calculation of Beacon Position Relative to the Camera For a Binary Vision System

The true distance of the beacon from the camera can be calculated through simple trigonometry. Being given the physical length of the beacon and the Pi Camera's field of vision characteristics, distance from the beacon to the camera can be easily determined. There several constants that need to be taken into consideration: the resolution of the image will be 400x300 pixels, making the middle of the image at 200 pixels, the ratio of pixels (pixratio) to the camera's view angle is 7.407 pixels per degree, and the physical length of the beacon is 7.5 inches. These two constants will differ depending on the type of camera used and the type of beacon used.

5.4.1 Distance Calculation for Beacon

When measuring the distance the camera is from a beacon, three sets of cases must be considered. In the first case set, the average pixel location of the beacon is near the center of the image. In the second case set, both sides of the beacon are either both on the left half or both on the right half of the image. The last case involves the lack of a full beacon image, meaning the image captured either has only one visible side of the beacon or no visibility of the beacon whatsoever. Relationships between the physical distances and the distances represented by the pixels of the image can be formed in order to accurately measure the location of the follower to the coordinator. Unlike the quadratic approximation determined through the use of test samples, these distance calculations are purely mathematical, and thus more accurate than the approximations. Each case will have its own dedicated set of equations.

Case 1a: the average of the pixel locations of the two sides of the beacon is the center of the image.

$$b_{avg} = mid \tag{5.3}$$

$$\theta = \frac{b_{right} - b_{avg}}{pixratio} \tag{5.4}$$

$$\theta_r = 90 - \theta \tag{5.5}$$

$$d = \frac{sin(\theta_r}{sin(\theta)} b_{len} \tag{5.6}$$

Where d is the distance between the beacon and the camera,

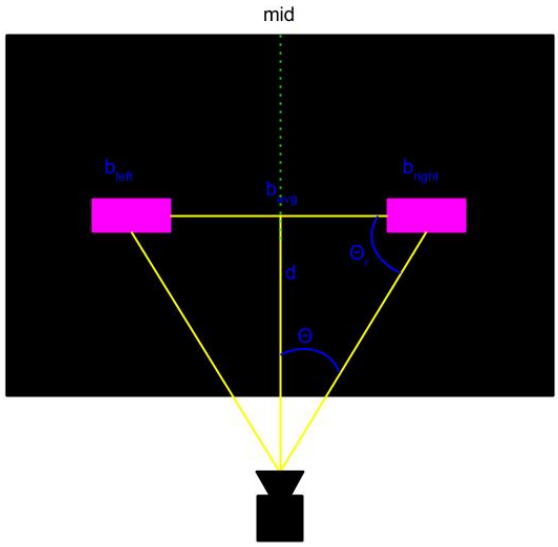


Figure 5.11: Case 1a distance scenario

Case 1b: the average of the pixel locations of the two sides of the beacon is to the right of the center of the image.

$$b_{average} > mid \tag{5.7}$$

$$\theta = \frac{b_{avg} - b_{left}}{pixratio} \tag{5.8}$$

$$\theta = \frac{b_{avg} - mid}{pixratio} \tag{5.9}$$

$$\bar{\bar{\theta}} = 90 - \bar{\theta} \tag{5.10}$$

$$\theta_r = \bar{\bar{\theta}} - \theta \tag{5.11}$$

$$d = \frac{sin(\theta_r}{sin(\theta)} \frac{b_{len}}{2} \tag{5.12}$$

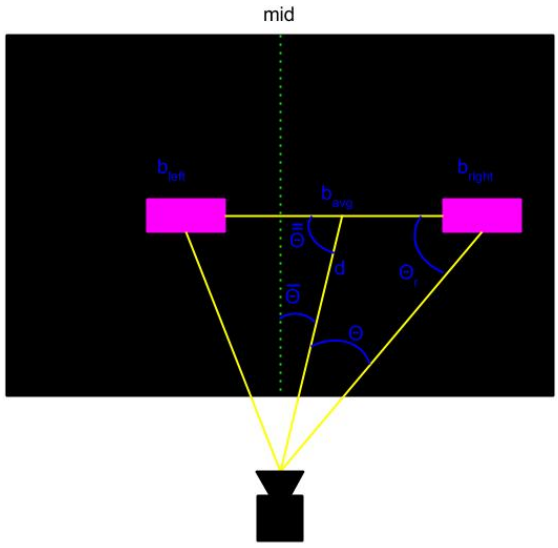


Figure 5.12: Case 1b distance scenario

Case 1c: the average of the pixel locations of the two sides of the beacon is to the left of the center of the image.

$$b_{average} < mid \tag{5.13}$$

$$\theta = \frac{b_{avg} - b_{left}}{pixratio} \tag{5.14}$$

$$\theta = \frac{mid - b_{avg}}{pixratio} \tag{5.15}$$

$$\bar{\bar{\theta}} = 90 - \bar{\theta} \tag{5.16}$$

$$\theta_l = \bar{\bar{\theta}} - \theta \tag{5.17}$$

$$d = \frac{sin(\theta_l}{sin(\theta)} \frac{b_{len}}{2} \tag{5.18}$$

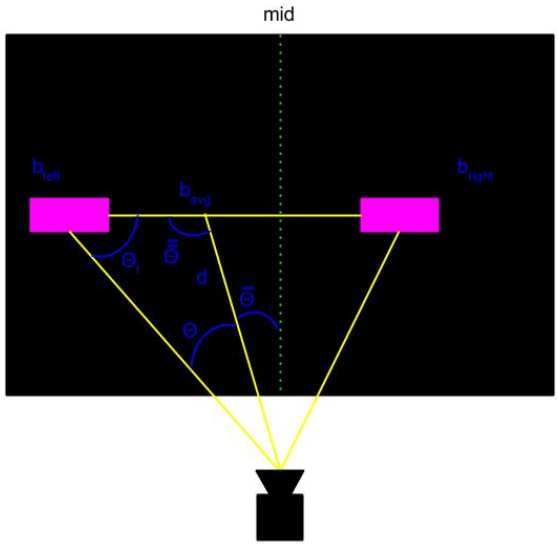


Figure 5.13: Case 1c distance scenario

Case 2a: both sides of the beacon are determined to be on the left half of the image.

$$b_{left} < mid, b_{right} < mid \tag{5.19}$$

$$\theta = \frac{b_{avg} - b_{left}}{pixratio} \tag{5.20}$$

$$\bar{\theta} = \frac{mid - b_{avg}}{pixratio} \tag{5.21}$$

$$\bar{\bar{\theta}} = 90 - \theta \tag{5.22}$$

$$\theta_l = 90 - \bar{\theta} \tag{5.23}$$

$$d = \frac{sin(\theta_l}{sin(\theta)} \frac{b_{len}}{2} \tag{5.24}$$

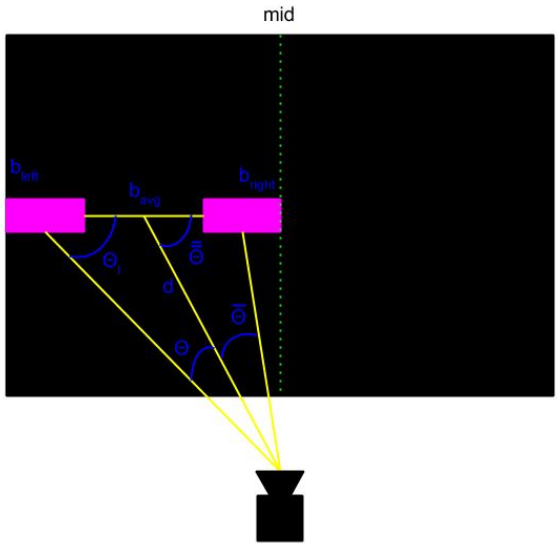


Figure 5.14: Case 2a distance scenario

Case 2b: both sides of the beacon are determined to be on the right half of the image.

$$b_{left} > mid, b_{right} > mid \tag{5.25}$$

$$\theta = \frac{b_{right} - b_{avg}}{pixratio} \tag{5.26}$$

$$\bar{\theta} = \frac{b_{avg} - mid}{pixratio} \tag{5.27}$$

$$\bar{\bar{\theta}} = 90 - \bar{\theta} \tag{5.28}$$

$$\theta_r = 90 - \theta \tag{5.29}$$

$$d = \frac{sin(\theta_r}{sin(\theta)} \frac{b_{len}}{2} \tag{5.30}$$

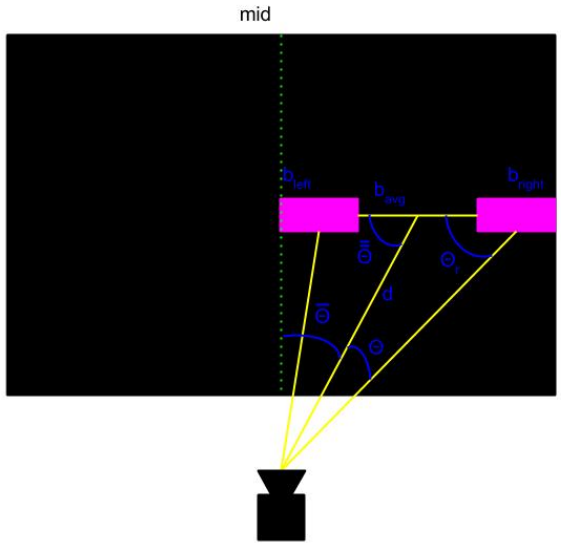


Figure 5.15: Case 2b distance scenario

Case 3: either only one side of the beacon is visible or the beacon is not visible at all.

5.4.2 Lateral Offset Calculation for Beacon

Lateral Offset can be described as the distance a follower quadrotor must move in order to be completely aligned with the coordinator. Much like the distance calculations, the lateral offset can be examined in terms of several cases. Also, values used in the distance calculations can be used also for the lateral offset calculations. The are other measurements that need to be made as well. Namely, the number of pixels that offset the middle of the beacon and the middle of the image.

Case 1: the average of the pixel locations of the two sides of the beacon is the center of the image. In this case, no calculation needs to be performed, since the beacon is already in the optimal lateral position.

$$b_{avg} = mid \tag{5.31}$$

$$x_{off} = 0 \tag{5.32}$$

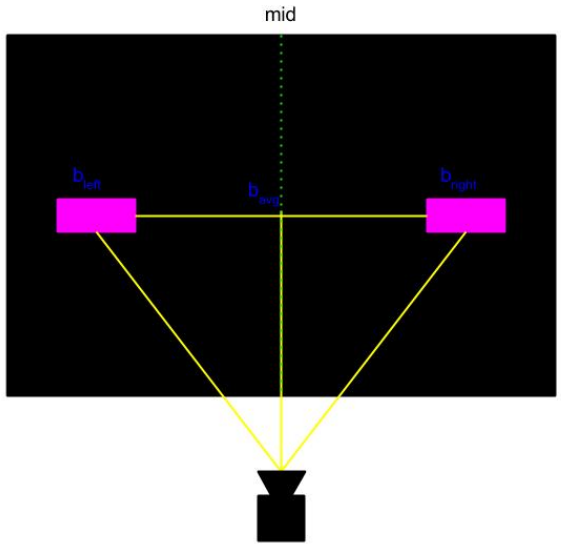


Figure 5.16: Case 1 lateral offset scenario

Case 2a: the average of the pixel location of the two sides of the beacon is on the left half of the image.

$$b_{avg} < mid \tag{5.33}$$

$$b_{pix} = b_{avg} - bleft \tag{5.34}$$

$$x_{pix} = mid - b_{avg} \tag{5.35}$$

$$x_{off} = \frac{x_{pix}}{b_{pix}} b_{len} \tag{5.36}$$

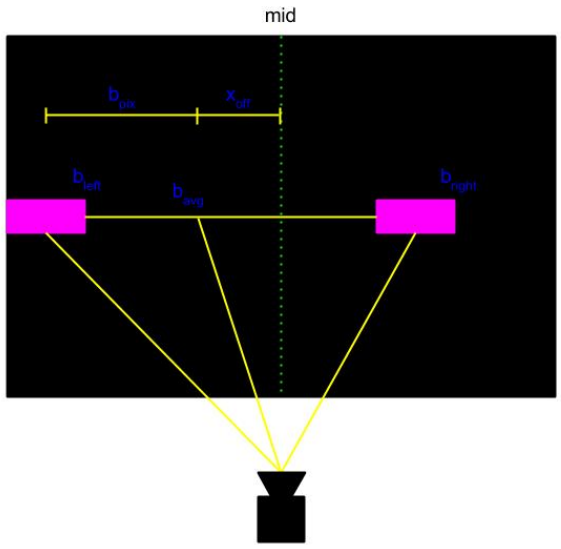


Figure 5.17: Case 2a lateral offset scenario

Case 2b: the average of the pixel location of the two sides of the beacon is on the right half of the image.

$$b_{avg} > mid \tag{5.37}$$

$$b_{pix} = b_{right} - b_{avg} \tag{5.38}$$

$$x_{pix} = b_{avg} - mid \tag{5.39}$$

$$x_{off} = \frac{x_{pix}}{b_{pix}} b_{len} \tag{5.40}$$

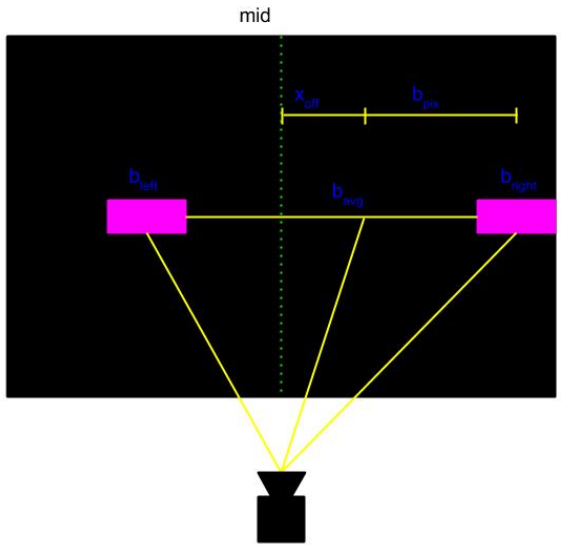


Figure 5.18: Case 2b lateral offset scenario

5.4.3 Angular Offset Calculation

Determining the angular offset of the beacon to the quadrotor can be delicate and is difficult to do through the means described previously. It was determined that using a rough method of angular offset calculation was the optimal route. The objective of this system is to determine if the quadrotor is at any offset at all, and in what direction. If the quadrotor were to be facing to the right of the coordinator, the coordinator would simply pass a command to the follower to turn left. The amount the follower would turn to the left would be a constant, meaning that it would take multiple steps to become optimally positioned if the follower is turned too far in either direction.

CHAPTER 6: CONCLUSION

6.1 Performance Limitations

The performance of the system can be defined as the rate at which the camera and processor can capture images, analyze the images for information about quadrotor positioning, use the data to generate movement commands for quadrotors and subsequently successfully transmitting the aforementioned movement command to a following quadrotor. Therefore, the performance of the system can be limited by multiple sectors of the quadrotor system. First by the camera's resolution and the speed at which it can capture image. Next, by the swiftness with which image data can be analyzed by the processor's quadrotor detection algorithm. Higher resolution images, which are necessary for formations required larger distances between quadrotors, will take longer times to process. The communication algorithm's efficacy determines the last factor of the system's overall performance. There is also a bottleneck that can occur when transmitting and receiving data. The longer it takes for a movement command to be successfully transmitted from a coordinator, the less fluidly the whole swarm will change state. As the size of the swarm increases and more clusters are added, the accuracy of the vision system information will decrease. With all of the slight variations of quadrotor drift, the position error will start to accumulate across the entirety of the swarm, increasing with every cluster added. Also, the coordinator must be able to move the stationary beacon in order to provide extensive mobility to the swarm. Should the coordinator lose sight or control over the stationary beacon the swarm will be more prone to unstable movement oscillations.

6.2 Stationary Beacon Displacement for Enhanced Mobility

One of the major functions of this swarm research is the ability of the swarm to minimize its position error due to unobserved drift. In order to accomplish this, the coordinator must maintain sight of a beacon that has constant physical position for reference. This, however, would limit the mobility if the swarm. Granting the ability for the coordinator to pick up the stationary beacon and place it into a new position in order to set a reference point for the swarm would be vital for the versatility of the swarm. As the coordinator has the capable of seeing the stationary beacon, it can simply fly to the beacon and lift it, keeping track of the direction it was facing. At this point, the quadrotor should issue coarse grain flight commands to the followers. Once the swarm meets at a location, the coordinator will drop the beacon and will resume the fine grain swarm maintenance algorithm.

6.3 Task Shift Capability

In order to have a truly robust system, it is imperative that the quadrotors have the ability to change roles in order to be a truly effective swarm unit. In order to realize this, each quadrotor must be equipped as both a coordinator and follower. Consequentially, each quadrotor will be equipped with a camera and a beacon. The utility in roll shifting lies in redundancy of a system. For example, that a quadrotor swarm with a hierarchy of follower nodes. In the case that the coordinator recognizes that all followers have lost sight of the it, the coordinator should broadcast a message telling the followers that it is in a non-operational state, and the follower at the top of the node hierarchy will take the position of the swarm.

6.4 Averaging Over a Margin of Error

The method of obtaining position data from calculations from successive images can be precise for a beacon that is stationary. However, using the same method on a beacon that is moving will not produce results that are truly representative

of the quadrotor's real-time position. In a practical situation, the quadrotors will never be completely stationary as long as they are in flight. This is because both the follower and coordinator may have drift, or the inability to maintain an exact position throughout time. Though the quadrotor's drift has been minimized and the quadrotor will stay in the same general position, slight variations between error margins of quadrotor drift in the current position will produce movement commands that are unnecessary and inaccurate. In order to correct for these slight variances, an averaging system was used on sets of successive images taken by the follower's camera in between communication events with the coordinator. For the entire duration of time between communication with the coordinator, the quadrotor will continuously record images of the coordinator's beacon and average the position one on top of another, producing a running average of the positions of the quadrotor. Assuming that the drift is occurring equally on each side of the follower's margin of error, the result should prove sufficient. The algorithm would resemble the following: record an image, determine the position data of the LEDs and sort positions, take a second image, determine the position data of that image, average the beacon locations of the two images, save the result as an average variable take another image, repeat process.

6.5 Long Range Coarse Grain Control

By integrating GPS into the swarm system, the swarm may be able to perform long range coordination. The coordinator would issue general commands and end position coordinate information for each follower, and the follower would operate accordingly. After performing whatever routines assigned, each follower would make its way to the rendezvous position, and the fine grain tuning algorithm performed by the coordinator would bring the followers back into maintenance position.

6.6 Node Identification through Beacon Pulsing

A drawback of using a binary vision system such as this infrared beacon is the inability to easily distinguish unique quadrotors. With a multicolor-based system, each quadrotor can have its identity linked with a color. In order to give uniqueness to a quadrotor in a binary system, one must be more creative. One method is to periodically pulse the LEDs on the beacon of a quadrotor. This way, images taken from one side of the coordinator will produce different results than images taken from other sides of the coordinator. However, it is imperative to have an image sampling system that can sample twice as fast as the frequency of the LED pulsing to avoid aliasing, as per Nyquist's Theorem. This however, requires an image processing system that can run fast enough to provide the most recent position data to the coordinator as well as capture images quickly enough to detect changes in the LED lighting pattern of the beacon.

6.7 Future Work

Though work in this field is still at its infancy, advancements in aerial swarm robotics are bound to happen at a fast pace. Already, multiple research groups are working on different forms of this research. It will not be long before this type of robotics will be publicly acknowledged and the social and ethical issues of using such technologies must be pondered. For productivity and efficiency of task completion, this type of technology would do wonders. As advancements in flight control systems and quadrotor hardware occur, the ability to create stronger and larger swarms will be more easily achieved.

REFERENCES

[1] Using the raspberry pi gpio with python. maxembedded .com/2014/07/using-raspberry-pi-gpio-using-python/.

[2] Wii sensor bar. http://www.amazon.com/Wireless-Infrared-Motion-Nintendo-Wii-Consoles/dp/B00BI43APY.

[3] All in one pro board setup with megapirates code. http://www.unmannedtech.co.uk/manualsguides/all-in-one-pro-board-setup-with-megapirates-code, 2013.

[4] Testing multiple pi camera options with python. http://www.raspberrypi-spy.co.uk/2013/06/testing-multiple-pi-camera-options-with-python, 2013.

[5] M.S. Alvissalim, B. Zaman, Z.A. Hafizh, M.A. Ma'sum, G. Jati, W. Jatmiko, and P. Mursanto. Swarm quadrotor robots for telecommunication network coverage area expansion in disaster area. In *SICE Annual Conference (SICE), 2012 Proceedings of*, pages 2256–2261, Aug 2012.

[6] B. Benchoff. Heavy lifting copters can apparently lift people. http://hackaday.com/2013/09/20/heavy-lifting-copters-can-apparently-lift-people, 2013.

[7] K. B. Culver. From battlefield to newsroom: Ethical implications of drone technology in journalism. *Journal of Mass Media Ethics*, pages 52–64, 2014.

[8] C. B. Eschmann, C. Kuo. Unmanned aircraft systems for remote building inspection and monitoring. pages 1–8, 2014.

[9] A. George. Forget roads, drones are the future of goods transport. *New Scientist, 219(2933)*, page 27, 2013.

[10] S. Gupte, P.I.T. Mohandas, and J.M. Conrad. A survey of quadrotor unmanned aerial vehicles. In *Southeastcon, 2012 Proceedings of IEEE*, pages 1–6, March 2012.

[11] A. Jaimes, S. Kota, and J. Gomez. An approach to surveillance an area using swarm of fixed wing and quad-rotor unmanned aerial vehicles uav (s). *In System of Systems Engineering, IEEE International Conference*, pages 1–6, 2008.

[12] R. Jonsson. Upenn's grasp lab unleashes a swarm of nano quadrotors. http://www .gizmag .com/grasp- nano-quadrotor-robots-swarm/21302/, 2012.

[13] CHINA YOUNG SUN LED TECHNOLOGY SO. LTD. Infrared light-emitting diode. YSL-R531FR2C-F1, 2010.

[14] A.J. Nash, C.M. Engel, and J.M. Conrad. Establishing and maintaining formations of mini quadrotors. In *SOUTHEASTCON 2014, IEEE*, pages 1–7, March 2014.

[15] A.J. Nash, T.E. Massey, C.J. Wesley, S.S Kosanam, and J.M. Conrad. Towards establishing and maintaining autonomous quadrotor formations. In *Informatics in Control, Automation and Robotics (ICINCO), 2014 11th International Conference on*, volume 02, pages 635–639, Sept 2014.

[16] S. A. Raza and W. Gueaieb. Intelligent flight control of an autonomous quadrotor. In Federico Casolo, editor, *Motion Control*. 2010.

[17] L. Reynaud and T. Rasheed. Deployable aerial communication networks: challenges for futuristic applications. *ACM MSWIM (International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems)*, pages 9–16, 2008.

[18] T. Smedley. Drones new mission: saving lives in developing countries. http://www. theguardian .com/sustainable-business/2015/, 2015.

[19] M. Waite. Journalism with flying robots. *XRDS: Crossroads, The ACM Magazine for Students*, 20(3):28–31, 2014.

APPENDIX A: PROGRAMS

A.1 Camera Testing Program

The code shown below is the code used to cycle through the Pi Camera's settings. This was used to determine the settings at which the camera should use in order to have the clearest image of the beacon as possible. Parts of the code are based on an article for testing the Pi Camera [4].

```
#!/usr/bin/env python
import os
import time
import subprocess

list_ex = ['off','auto','night','nightpreview','backlight','fireworks']
list_awb = ['auto']
photo_ev = 0
photo_width = 640
photo_height = 480
photo_rotate = 0
photo_interval = 0.25
photo_counter = 0
total_photos = len(list_ex)*len(list_awb)
camera_distance = 12
try:
    os.remove("photo_*.jpg")
except OSError:
    pass

try:
    for ex in list_ex:
        for awb in list_awb:
            photo_counter = photo_counter + 1
            filename = 'photo_' + ex + '_' + str(camera_distance) + 'inches'+'.jpg'
            cmd = 'raspistill␣-o␣' + filename + '␣-t␣1000␣-ex␣' + ex + '␣-awb␣' + awb
                 + '␣-ev␣' + str(photo_ev)+ '␣-w␣' + str(photo_width) + '␣-h␣' + str(
                photo_height) + '␣-rot␣' + str(photo_rotate)
            pid = subprocess.call(cmd, shell=True)
            print '␣[' + str(photo_counter) + '␣of␣'+ str(total_photos) + ']␣' +
                filename
```

```
            time.sleep(photo_interval)
    print "Finished␣photo␣sequence!"
except KeyboardInterrupt:
    print "\nGoodbye!"
```

A.2 Color-based Vision System

```
import cv2 as cv
import numpy as np
import picamera
import math
import io


#Constants
lower_blue = np.array([85,40,40])
upper_blue = np.array([100,255,255])
#lower_yellow = np.array([20,110,110])
#upper_yellow = np.array([30,255,255])
lower_pink = np.array([0,110,110])
upper_pink = np.array([20,255,255])
res = [640,480]

#NEED TO ADD :
# bg subraction pic needs to update whenever qr changes position
with picamera.PiCamera() as camera:
    camera.resolution = (res)
    camera.capture('bg.jpg')

bg = cv.imread('/home/pi/bg.jpg')
while(1):
    stream = io.BytesIO()
    with picamera.PiCamera() as camera:
        camera.resolution = (res)
        camera.capture(stream, format = 'jpeg')

    data = np.fromstring(stream.getvalue(), dtype=np.uint8)
    image = cv.imdecode(data,1)
    image = image - bg
    hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)
```

```
    kernel = np.ones((5,5),np.float32)/25
    #mask for possible blue values
    blue_mask = cv.inRange(hsv, lower_blue, upper_blue)
    blue_mask = cv.filter2D(blue_mask,-1,kernel)
    ret, blue_mask = cv.threshold(blue_mask,200,255,cv.THRESH_BINARY)
    blue_res = cv.bitwise_and(image,image,mask= blue_mask)

    pink_mask = cv.inRange(hsv, lower_pink, upper_pink)
    pink_mask = cv.filter2D(pink_mask,-1,kernel)
    ret, pink_mask = cv.threshold(pink_mask,200,255,cv.THRESH_BINARY)
    pink_res = cv.bitwise_and(image,image,mask= pink_mask)
    #yellow_mask = cv.inRange(hsv, lower_yellow, upper_yellow)
    #yellow_res = cv.bitwise_and(image,image,mask= yellow_mask)




#find center of blue object
    blueM = cv.moments(blue_mask)
    try:
        bx = int(blueM['m10']/blueM['m00'])
        by = int(blueM['m01']/blueM['m00'])
    except ZeroDivisionError:
        bx = 0
        by = 0
#find center of pink object
    pinkM = cv.moments(pink_mask)
    try:
        px = int(pinkM['m10']/pinkM['m00'])
        py = int(pinkM['m01']/pinkM['m00'])
    except ZeroDivisionError:
        py = 0
        px = 0

#count the number of pixels for each beacon to determine size
    bpixcount = cv.countNonZero(blue_mask)
    ppixcount = cv.countNonZero(pink_mask)


#Calculate angle camera is facing away from beacon
#for instances where the cam only sees one beacon
```

```
    if bpixcount > 2*ppixcount:
        print "The camera sees only the blue beacon. QR needs to turn 90 degrees."
    if ppixcount > 2*bpixcount:
        print "The camera sees only the yellow beacon. QR needs to turn 90 degrees."
#for slightly more precise turning
    if bpixcount > ppixcount+100:
        print "The QR needs to rotate left."
        angle_set = 0
    elif ppixcount > bpixcount+100:
        print "The QR needs to rotate right."
        angle_set = 0
    else:
        print "QR is at the correct angle."
        angle_set = 1

#Calculate distance each object is from camera
#radius = -distance + 75

    bradius = math.sqrt(bpixcount/math.pi)
    pradius = math.sqrt(ppixcount/math.pi)
    D1 = 75 - bradius
    D2 = 75 - pradius

    if angle_set:
        DN = (D1+D2)/2
        if DN > 50:
            print "QR needs to move forward."
        elif DN < 30:
            print "QR needs to move backward."
        else:
            print "QR is at the correst distance."


#These are only for debugging/user interface
#blend the images
    dst = cv.add(blue_res,pink_res)


#draw dot on middle of objects
    bradius = int(bradius)
    pradius = int(pradius)
```

```
    cv.circle(dst, (bx,by), bradius, (0,0,255), 3, 8)
    cv.circle(dst, (px,py), pradius, (255,0,0), 3, 8)
    font = cv.FONT_HERSHEY_SIMPLEX
    cv.putText(dst,'BACON_1',(bx+10,by+10), font, 1,(255,255,255),2,cv.LINE_AA)
    cv.putText(dst,'BACON_2',(px+10,py+10), font, 1,(255,255,255),2,cv.LINE_AA)

#Outputs
#print "The beacons are %s" % Db + " cm apart.
    print "The number of blue pixels is %s" % bpixcount + "."
    print "The number of pink pixels is %s" % ppixcount + "."
    print "The blue beacon is %s" % D1 + " cm away."
    print "The pink beacon is %s" % D2 + " cm away."
    cv.imshow('Captured', dst)
    cv.waitKey(30)
```

A.3 Binary-based Vision System

```
import numpy as np
import cv2 as cv
import operator
import io
import picamera
import picamera.array
import time
import math

tiny_res = [200,150]
low_res = [400,300]
mid_res = [1280,780]
high_res = [1920,1080]
ultra_res = [2592,1944]
font = cv.FONT_HERSHEY_SIMPLEX
res = low_res
beacInfo = [0]*4
white = [255,255,255]
green = [0,255,0]

class LED:
    ledCount = 0
    def __init__(self,name,loc,vis,group):
        self.name = name
        self.loc = loc
```

```
        self.vis = vis
        self.group = group
        LED.ledCount += 1

    def displayCount(self):
        print "Total␣LEDs:␣%d" % LED.ledCount

    def displayLED(self):
        print "Name:␣", self.name, ",␣Location:␣", self.loc, "Visibility:␣", self.vis

def contourSearch(im):
    imgray = cv.cvtColor(im,cv.COLOR_BGR2GRAY)
    ret,thresh = cv.threshold(imgray,5,255,0)
    dilation = np.ones((5,5), "uint8")
    imgray = cv.dilate(imgray,dilation)
    _,contours,_ = cv.findContours(thresh,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_SIMPLE)
    return(contours,imgray)

def findLeds(contours):
    cx = [0]*len(contours)
    cy = [0]*len(contours)
    M = [0]*len(contours)
    ledDict={'0': 0,}
    contourlist = [0]*len(contours)
    for c in range (0,len(contours)):
        M[c] = cv.moments(contours[c])
        try:
            cx[c] = int(M[c]['m10']/M[c]['m00'])
            cy[c] = int(M[c]['m01']/M[c]['m00'])
        except ZeroDivisionError:
            contourlist[c] = contours[c].tolist()
            cx[c] = contourlist[c][0][0][0]
            cy[c] = contourlist[c][0][0][1]
    for j in range (0,len(contours)):
        while cx[j] in ledDict:
            cx[j] += 1
        ledDict[cx[j]] = cy[j]
    del ledDict['0']

    return(ledDict)
```

```
def sortLeds(ledDict,contours):
    sortedLeds = sorted(ledDict.items(),key=operator.itemgetter(0))
    ledList = [0]*len(contours)
    LED.ledCount = 0
    for b in range (0,len(ledList)):
        ledList[b] = LED(str(b),sortedLeds[b],len(contours[b]),'none')
    return(ledList)


def ledGroup(ledList):
    leftCount = 0
    rightCount = 0
    leftList = []
    rightList = []
    ledSpacing = [0]*(len(ledList)-1)
    leftY,rightY,leftMean,rightMean,innerLeft,innerRight = 0,0,0,0,0,0
    oldDif = 0
    dif = 0
    for x in range (1,len(ledList)):
        dif = ledList[x].loc[0] - ledList[x-1].loc[0]
        ledSpacing[x-1] = dif
        if dif > oldDif:
            innerRight = ledList[x].loc[0]
            innerLeft = ledList[x-1].loc[0]
            oldDif = dif
    for x in range (0,len(ledList)-1):
        if ledList[x].loc[0] < (innerRight + innerLeft)/2:
            leftList.append(ledList[x].loc[0])
            leftY = ledList[x].loc[1]
            ledList[x].group = 'left'
            leftCount += 1
        else:
            rightList.append(ledList[x].loc[0])
            rightY = ledList[x].loc[1]
            ledList[x].group = 'right'
            rightCount+=1
    try:
        leftMean = np.convolve(np.array(leftList),np.ones((leftCount,))/leftCount)[(
            leftCount-1):][0]
    except ValueError:
        leftMean = 0
    try:
```

```
            rightMean = np.convolve(np.array(rightList),np.ones((rightCount,))/rightCount
                )[(rightCount-1):][0]
        except ValueError:
            rightMean = 0
        beacInfo = [leftMean,leftY,rightMean,rightY,leftCount,rightCount,ledSpacing]

        return(beacInfo)

def calcPosition(beacInfo):
        b_len = 7.5
        mid = res[0]/2
        b_left = beacInfo[0]
        b_right = beacInfo[2]
        b_avg = (b_left+b_right)/2
        pixratio = res[0]/54
        ledSpacing = beacInfo[6]
        angle = 0

        #distance
        if b_left < mid and b_right > mid:
            if b_avg == mid:
                b_pix = 1
                x_pix = 0
            elif b_avg > mid:
                b_pix = b_right - b_avg
                x_pix = b_avg - mid
            elif b_avg < mid:
                b_pix = b_avg - b_left
                x_pix = -(mid - b_avg)
        elif b_left < mid and b_right < mid:
            b_pix = b_avg - b_left
            x_pix = -(mid - b_avg)
        elif b_left > mid and b_right > mid:
            b_pix = b_avg - b_left
            x_pix = mid - b_avg
        else:
            x_pix = 0
            b_pix = 0

        theta = b_pix/pixratio
        theta_bar = x_pix/pixratio
```

```
    theta_bar2 = 90 - theta_bar
    theta_dot = theta_bar2 - theta

    #offset
    try:
        x_off = b_len*(x_pix/b_pix)
    except ZeroDivisionError:
        x_off = 0

    try:
        distance = ((b_len)/2)*(math.degrees(math.sin(math.radians(theta_dot))))/(
            math.degrees(math.sin(math.radians(theta))))
    except ZeroDivisionError:
        distance = 0

    #angle
    if len(ledSpacing)>3:
        leftSpace = ledSpacing[1]
        rightSpace = ledSpacing[3]
        spaceDiff = rightSpace-leftSpace

        if distance >= 12:
            angle = 7.14*spaceDiff-20.7
        elif distance >= 14:
            angle = 4.35*spaceDiff-4.78

    return(distance,x_off,angle)

with picamera.PiCamera() as camera:
    camera.resolution = (res)
    camera.contrast = 100
    camera.exposure_mode = 'off'
    camera.awb_mode = 'auto'
    camera.vflip = True
    camera.iso = 800
    with picamera.array.PiRGBArray(camera) as stream:

        while(1):
            startTime = time.time()
            camera.capture(stream, format = 'bgr')
            im = stream.array
```

```
            # print "capture time: " + str(time.time()-startTime)
            contours,imgray = contourSearch(im)
            ledDict = findLeds(contours)
            ledList = sortLeds(ledDict,contours)
           # print "detect time: " + str(time.time()-startTime)
    #beacInfo:
    #[0]-> mean x-coordinate of left-side LEDs (float)
    #[1]-> mean y-coordinate of left-side LEDs (float)
    #[2]-> mean x-coordinate of right-side LEDs (float)
    #[3]-> mean y-coordinate of right-side LEDs (float)
    #[4]-> number of leds in left group (int)
    #[5]-> number of leds in right group (int)
    #[6]-> spacing between consecutive leds
            beacInfo = ledGroup(ledList)

            distance,offset,angle= calcPosition(beacInfo)
            print "Distance (cm): " + str(distance)
            print "Offset:" + str(offset)
            print "Angle:" + str(angle)


    #displayThings(contours,beacInfo,im)
#             cv.putText(im,'Runtime: '+str(time.time()-startTime)+ ' s.',(10,40),font
    ,0.25,white,1,cv.LINE_AA)
 #            cv.line(im,(int(beacInfo[0]),int(beacInfo[1])),(int(beacInfo[2]),int(
     beacInfo[3])),green,1)
#             cv.putText(im,'Distance: '+str(distance)+' in.',(10,10),font,0.25,white
    ,1,cv.LINE_AA)
 #            cv.line(im,(res[0]/2,0),(res[0]/2,res[1]-1),white,1)
  #           cv.line(im,(res[0]/2,res[1]/2-10),(int((beacInfo[0]+beacInfo[2])/2),res
      [1]/2-10),white,1)
   #          cv.putText(im,'Offset: '+str(offset)+' in.',(10,20),font,0.25,white,1,cv
       .LINE_AA)
    #         cv.putText(im,'Angle: '+str(angle)+' deg.',(10,30),font,0.25,white,1,cv.
        LINE_AA)
     #        for x in range (0,len(ledList)):
      #           cv.putText(im,str(x+1),(ledList[x].loc), font, 0.25,white,1,cv.
          LINE_AA)
       #      cv.imshow('Display',im)
        #     if res == low_res:
         #        cv.imshow('Visibility',imgray)
```

```
#   cv.waitKey(30)
 # print "display time: " + str(time.time()-startTime)

  contours[:]=[]
  ledList[:]=[]
  stream.seek(0)
  stream.truncate()
```