

Explorations in
Autonomous Flight

Santiago Saldaña

Senior Honors Thesis

May 11, 2007

Explorations in Autonomous Flight

Santiago Saldaña

May 11, 2007

Abstract

In order to verify the feasibility of a process, one may go about proving it in two different ways, by use of proofs on paper or by creating a working prototype. This paper discusses the use of the latter to create a developmental aerial platform capable of sustaining flight and orientation with respect to an object of interest. The project was created using an agile approach to software engineering and was developed from the ground up, focusing on the different components of the overall system, that being physical engineering as well as software development and integration.

1. Introduction

1.1 Goals

The purpose of this project spawned from several areas of interest, particularly from the capabilities of projects undertaken at the RoboRealm¹ website. A tutorial on their website described the process of tracking a color using a wheeled robot using only video and the Center of Gravity algorithm. This approach seemed very simple yet effective and it is the goal of this project to extend that principle into three dimensions by developing an indoor aerial platform capable of tracking an object based solely on color. In addition to keeping the object in its field of view the aerial platform must be capable of remaining in a stable hover using only visual feedback provided while tracking the object of interest.

1.2 Secondary Goals:

In addition to achieving its primary purpose of tracking an object based solely on color, the platform will be required to be inherently stable, physically robust and adaptable. This will ensure that the capabilities of the platform will be extensive enough to support future research.

1.3 Components

Throughout this paper, several terms will be used, in reference to the key components of this project. Here is a list of relevant terms.

PCTx –

Black box component available from www.endurancrc.com², this component connects between the computer and the Remote Control Transmitter and translate API calls into signals the transmitter can then relay to the helicopter

RoboRealm –

Machine Vision software that was used extensively throughout this project.

COG –

Center of Gravity is an area of an image having the highest overall average intensity.

Cyclic –

Cyclic refers to the control of forward and lateral attitude of a helicopter.

Delegate –

Accessor function used in C sharp to call functions located on a different thread.

2. Physical Engineering**2.1 Inherent Stability**

The demands of indoor flight often dictate aerial maneuvers within confined spaces, therefore requiring an aerial platform to be capable of handling such requirements. It is therefore critical that a vision based flight system which has no feedback as to the absolute orientation of the platform relative to the ground be inherently stable. As such, this was determined to be the most important obstacle to overcome.

2.1.1 Starting Platform

The platform was based on E-flite's Blade CX micro helicopter³. The Blade CX is an off the self, ready-to-run coaxial micro indoor helicopter. The coaxial design means that the torque cancellation problems encountered in single rotor helicopters is not present. The problem encountered when using single rotor helicopters is that they rely on a tail rotor to cancel out the torque produced by the main rotor. The use of the tail rotor then introduces lateral forces that typically cause sideways drift in a single rotor helicopter if cyclic control is not introduced. Thus a delicate balance is required among multiple forces to enable a single rotor helicopter to remain in a relatively stationary hover. For the purposes of stability, a coaxial helicopter is the way to go as it provides a simple method for countering torque while also eliminating the lateral drift introduced by the use of a tail rotor in conventional single rotor helicopters. Having overcome the torque cancellation problem, the next goal was to customize the Blade CX for the further stability required by a flight control system guided solely by visual cues.

2.1.2 Going Separates

The standard design of the Blade CX includes a 3-in-1 electronic gyroscope (gyro), radio receiver, and dual electronic speed control (ESC) achieved through the use of a single MOSFET (metal-oxide semiconductor field-effect transistor). Although this design was carefully designed for reduction of weight, its integrated nature caused some heat related issues due to the integrated nature of the onboard gyro. The 3-in-1 was designed to compensate for differences in room temperature on startup, however, there is no efficient way to eliminate the heat produced by the single MOSFET used to power the motors within the 3-in-1. The result of this was that the gyro would start to heat up

midway through flight and the readings from it would be skewed, causing the helicopter to rotate to one side.

The standard solution to this problem was found on the www.RCGroups.com forums and is described as going “separates”. What this technique implies is the replacement of the 3-in-1 with a separate gyro, a receiver, and a separate ESC for each motor. This design achieved the isolation of the gyro from the primary source of heat on the helicopters, the motors.

2.1.3 Cyclic Stability

The second group of stabilization modifications were focused on increasing the cyclic stability of the aircraft. Two methods were evaluated in order to increase the stability of the helicopter. The first was a gyro based setup while the second was a mechanically based setup.

The first attempt involved the use of 3 gyros, one for each axis of rotation, to cancel out unwanted changes in orientation. This method proved partially successful. The design did result in an overall increase in stability, but the balance was precarious as the gain on each gyro could only be turned up to 10% to 15% before oscillation problems would occur. These oscillation problems consisted of the gyro continuously overcompensating, causing the aircraft to swing back and forth in ever increasing swaths. The overall result was a minor gain in stability with a 9.2 gram increase in weight over a single gyro system.

The second and most successful attempt relied on the use of the stabilizer bar already present on the helicopter. Unlike the bottom rotor, which is controlled by servo input, the top rotors are freely hinged to the top hub and are only controlled by the current position of the stabilizer bar. The stabilizer bar changes the angle of the top rotor depending on how it is tilted. The stabilizer bar works as a large gyroscope. If the helicopter tilts in one direction the inertia of the stabilizer bar will cause it to resist any change in its rotational axis, and because it is attached to the top rotors, the top rotors will exert a force opposite to the direction of the movement of the helicopter. With this in mind, the second attempt simply involved using the weights from a second stabilizer bar and adding them to the first. This resulted in a large increase in stability with a small increase in weight.

2.2 Physical Robustness

The process of research and development is often a trial and error process and thus the possibilities of hardware failure are much higher than proven systems. This led to the need for a physically robust aerial platform that could withstand a minor crash without major structural damage and with minimal costs. In order to increase the physical robustness of the aerial platform, the structurally weak components were isolated and replaced. These components consisted of the upper rotor hub and the lower rotor hub. Both of these components were replaced with aluminum counterparts. After the upgrade, crashes typically result in damage only to the plastic blades.

3 Software Design

This project was designed with flexibility in mind, and as such makes use of two software applications, freely distributed RoboRealm Machine Vision software and a custom application written using Microsoft Visual C# 2005 Express Edition. The integration of both of these applications through the use of sockets results in a custom application that can be fed information from any future program via a very standardized form of exchanging information.

3.1 RoboRealm

RoboRealm is a freely distributed Machine Vision application designed for the use in a variety of experiments. It is widely used in vision based experiments because of its numerous modules, extendibility, frequent updates, and its extensive support network. RoboRealm is not open source, however, it is written with experimenters in mind and as such provides various ways to extend the capabilities of RoboRealm by interfacing with user-made programs. This project made use of the Center of Gravity module, the RGBFilter module, the Mean Filter module, and the RoboRealm Socket Server extension.

3.1.1 Center of Gravity

The Center of Gravity (COG) module provides a wrapper for the primary algorithm used within this project to track color. The COG algorithm compares the intensity values of an individual pixel with the total intensity value of all the pixels within a particular image. For the primary stage of this project, only the COG X and Y position variables were used.

Borrowed with permission from the RoboRealm website¹:

The COG is calculated by:

$$COG_X = COG_X + (I*x)$$

$$COG_Y = COG_Y + (I*y)$$

$$Total = Total + I$$

for each pixel where $I = (R+G+B)/3$ and x,y is the current pixel location. The resulting COG is then divided by the Total value:

$$COG_X = COG_X/Total$$

$$COG_Y = COG_Y/Total$$

to result in the final x,y location of the COG.

The problem with this algorithm is that it favors more intense colors over darker colors. To remedy this, a color filter is used so that only the color of choice is present and the intensity of all other colors is dropped to 0.

3.1.2 RGBFilter

The RGBFilter module works by amplifying the red value of an individual pixel based on the relative difference between that the R value of a pixel and the G and B values of a pixel. It works as following:

Borrowed with permission from the RoboRealm website¹:

$$R = ((R-B)+(R-G))$$

$$G = 0$$

$$B = 0$$

R is then normalized with respect to the maximum red value.

This process tends to raise the intensity of darker colors, thus in order to combat this problem a minimum hue and intensity filter is also built into this module. The minimum hue filter was used within this project and an experimentally determined value of 91 was used as a hue threshold for tracking the color red. Although the hue threshold value works well to increase the specificity of the center of gravity algorithm, the problem of the appearance of random background noise in the color of interest still remains. In order to resolve this problem, a Mean filter is used first on every image of the video stream.

3.1.3 The Mean Filter

The Mean Filter module was used to filter out unwanted noise present in the environment. The success of the color tracking algorithm depends on the isolation of the color of interest to within one location on the screen. For example, if a small red object were present in the upper right hand corner of an image, and the object of interest was present in the lower left hand corner, then the coordinates of the center of gravity would be skewed. This same situation is present if many single pixel size objects are present in the background. This is where the Mean Filter comes into play. What the Mean Filter module does is that it averages the R,G, & B values of a picture with those of surrounding pixels within a specified window. Below is an example from the RoboRealm website:

Borrowed with permission from the RoboRealm website¹:

For example, given the grayscale 3x3 pixel window;

22	77	48
150	77	158
0	77	219

Center pixel = $(22+77+48+150+77+158+0+77+219)/9$

The center pixel would be changed from 77 to 92 as that is the mean value of all surrounding pixels.

An experimentally determined window size of 17 was used for this module. This is a relatively high window size, but this value was used because as the values of pixels are averaged together, the intensity of small isolated pixels of the desired color are reduced resulting in an image that is more suitable to Center of Gravity detection based on a single desired color.

The combination of these modules provided the tracking portion of the project, however the need for custom processing to map a response to a change in Center of Gravity resulted in a need to extend RoboRealm's current capabilities. This resulted in the use of the VBScript module and the transition to using the Socket Server module to extend the capabilities of RoboRealm as needed for the aerial platform.

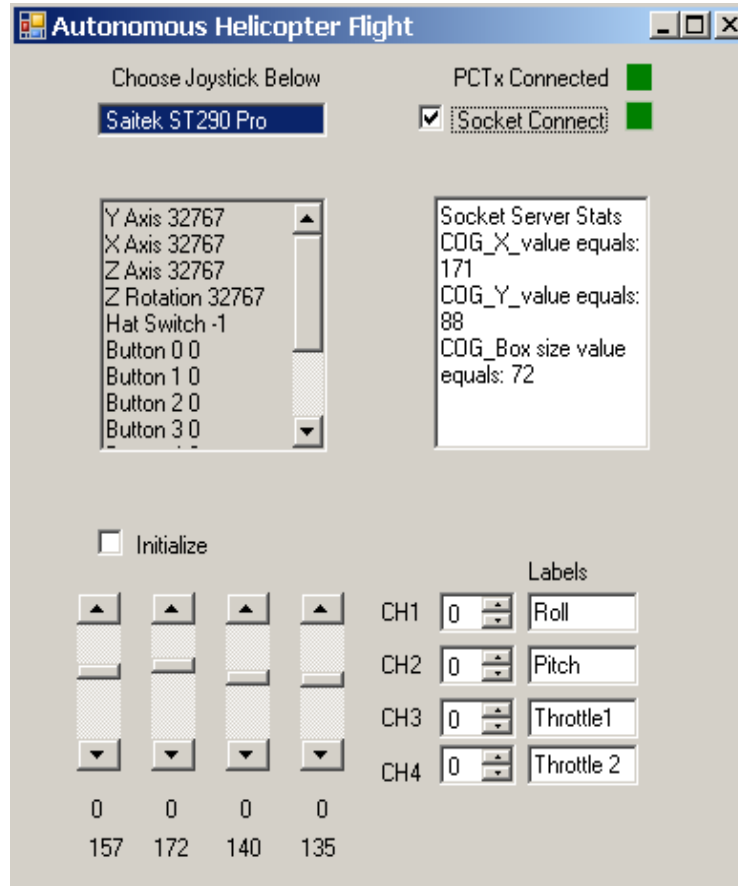
3.1.4 Socket Extension Module

The RoboRealm extension that was used in this project was the socket extension module. The socket extension module consisted of a Socket Server set to send out the various variables within the RoboRealm processing pipeline via port 5050. The Socket Server extension allowed for maximum flexibility of the program as it allows for the video to be processed on one machine and the control of the helicopter to be processed on another.

3.2 Microsoft Visual C#

The application was written using Microsoft Visual C# 2005 Express Edition. This platform was chosen for its easy operability with the various hardware components included in the project. The application is multithreaded, allowing for greater response time from the Graphical User Interface (GUI) as well as allowing the application to run with user input only, or by additionally incorporating the data retrieved from the RoboRealm socket server application. The result is an application consisting of three threads, the GUI, the joystick worker thread, and the socket server/client handler thread.

3.2.1 The GUI



The GUI is meant to provide an easy method of access to all the key variables seen involved in maintaining orientation during flight. The GUI was designed specifically to work with Saitek’s ST90 pro joystick because of its low cost (~\$20) and USB compatibility. The joystick is required to be plugged in before the program is initiated so that the program has an opportunity to find the joystick. If it is connected, the joystick will appear under the “Choose Joystick Below” label. In addition to having the joystick connected at this point in time, the user must also have the PCTx connected to the computer. After the user selects the joystick, a worker-thread will be spawned that will check to see if the PCTx is in fact connected. If the PCTx is connected the worker-thread will raise an event in the main form that will cause the GUI to change the display value of the box next to the PCTx from red to green. The GUI also supports user input in the form of trim values for any of the 4 channels currently supported by the program. The labels next to the trim values may also be modified, although they are reset to their default values each time the program is run.

3.2.2 The Joystick Thread

If the joystick and the PCTx are connected when the user selects the joystick name in the GUI then the main form will spawn off the joystick worker-thread to handle retrieving data from the joystick. If it is not connected, the program will display a

message saying that the PCTx is not connected, and the program must be restarted. Meanwhile, the worker-thread will continuously poll the joystick for data until the program exits. Data from the worker-thread will be written back to the main form via a delegate which will display the raw joystick axis and button values in the grayed out box beneath the joystick name. In addition to displaying the raw joystick values in the grayed out box, values for the X, Y, Z, and rZ joystick values corresponding to Roll, Pitch, Throttle and Yaw will be displayed in the scroll bars for CH1 – CH4.

3.2.3 The Socket Server

The Socket Server thread was developed as a modification to a simple socket server extension example provided by RoboRealm. The example contained a section to add your own code to retrieve additional variables from the RoboRealm socket server. Additional variables were added to the computation by comparing the name field of each arriving packet to determine whether it was a user defined variable or an image variable. An example of the parsing used is shown below.

```
if (name.Equals("cog_x"))
{
    tempCogX = System.Convert.ToInt32(Encoding.ASCII.GetString(data, 0,
len));
}
else if (name.Equals("cog_y"))
{
    tempCogY = System.Convert.ToInt32(Encoding.ASCII.GetString(data, 0,
len));
}
```

As you can see, the data is transmitted as ASCII strings and must then be converted to an integer value for storage and manipulation. The socket server thread also writes to shared memory on the main form and uses a common mutex located on the GUI thread to perform this in a thread safe manner. In the example below, m_form refers to the GUI / main form of the application.

Example:

```
//WRITE TO SHARED MEMORY//
m_form.mut.WaitOne();
m_form.imageData.height = imageData.height;
m_form.imageData.width = imageData.width;
m_form.imageData.COG_X = tempCogX;
m_form.imageData.COG_Y = tempCogY;
m_form.imageData.COG_BOX_SIZE = tempBoxSize;
m_form.mut.ReleaseMutex();
//DONE writing to shared memory//
```

4 Flight Algorithm

The joystick worker-thread does all the real work in the application as it integrates the values retrieved from the socket-server connected to RoboRealm, trims values from the main form, as well as raw joystick values, and calculates an appropriate output to send to the PCTx. The PCTx is connected directly to the transmitter and relays a desired response to the helicopter. The joystick thread has two modes it works on, automode and user controlled mode. If the user presses the fire button on the joystick then automode is

set to false and the user has direct control over the joystick. This was developed in response to a need of having an emergency recovery system in case there was any undesired response from the color tracking system. If button2 on the joystick was pressed then automode was set to true and the worker thread attempted to incorporate the values available from RoboRealm. This was performed only if a variable on the main form had indicated that the socket server had connected successfully.

The actual algorithm used to control the orientation of the helicopter was intended to incorporate a simple PID loop using the distance from the center of gravity as a measurement of displacement while storing previous samples to have a measurement of velocity and acceleration. Unfortunately hardware difficulties delayed this stage of programming and a simpler approach had to be taken. In order to keep the process safe, the algorithm takes into account the physical position of the throttle as a set point for throttle calculations. This is used so that in case of an emergency, the user can throttle the helicopter all the way down or switch to manual mode without having a tremendous switch in the amount of throttle present at that time. In order to control yaw the algorithm takes a measurement of the difference between the center of the screen and the center of gravity and multiplying it by a range factor. The range factor was used to limit how quickly the program could respond to a change in the position of the center of gravity or the helicopter. Additionally corrections were made to both propellers, so that one would be slowed down and the opposite one sped up to increase the response rate of the helicopter. To decrease the possibilities of oscillations, a differential control was used and the area near the center of the screen had half the effective potential for correction as did the far sides of the screen. Additionally, in order to provide a counter balance to over compensation, the algorithm maintains a Boolean variable named *Extremes* that keeps track of whether or not the helicopter was recently in either the far right or far left of the screen. This Boolean is then checked if the center of gravity is higher than 25% of the screen width or lower than 75% of the screen width. If this is the case then it is assumed that the helicopter is compensating for a difference between the two and that it is accelerating towards the center. If this is the case, then a counter *ODF* is used to send a certain number of pulses in the opposite direction of the helicopters current velocity in order to attempt to avoid overcompensation from overshooting the center of the screen. The very center of the screen only uses one half of the possible correction in order to maintain oscillation to a minimum, but to still allow for active tracking when the helicopter is directly facing the object of interest. Additionally, in order to ensure that the helicopter's throttle did not continue to accelerate in the case that the helicopter managed to lose track of the object of interest, a LockThrottle Boolean is used to maintain a constant throttle set point if the helicopter is tracking an object near the edge of its field of vision. This means that the helicopter will not track up and down if an object of interest is moving too fast out of its field of vision.

Example Code:

```
double amount2Add = ((double)((m_form.imageData.COG_X - imageData_width / 2.0) / (imageData_width / 2.0)) * (Rz_range / 2));  
  
double comparison = imageData_width - m_form.imageData.COG_X;
```

```

    if (comparison > imageData_width * 0.75 || comparison <
imageData_width * 0.25)//left hand side of screen or right side
    {
        Extremes = true;
        LockThrottle = true;
        if (amount2Add < 0)
            new_Ch4 += amount2Add;
        else
            new_Ch3 -= amount2Add;
            pulseCounter++;
            OFFcounter = 0;
    }

else
    {
        LockThrottle =false;
        if (Extremes)
            {
                if (pulseCounter < ODF/2)
                    {
                        amount2Add *= -0.5;
                        pulseCounter++;
                        if (amount2Add < 0)
                            new_Ch4 += amount2Add/2;
                        else
                            new_Ch3 -= amount2Add/2;
                    }
                else
                    {
                        Extremes = false;// done compensating
                        pulseCounter = 0;
                    }
            }
        else//inside part of screen, close to center
            if(amount2Add <0)
                new_Ch4 += amount2Add/2;
            else
                new_Ch3 -= amount2Add/2;
    }
}

```

In addition to making adjustments to the yaw of the helicopter in order to keep the object of interest in view, the helicopter also had to adjust its altitude to keep the vertical component of the object of interest in view. This was done by using an accumulator approach incorporating a set point. As mentioned in the previous section, the user would continue to have some throttle control in the case of an accident. The user would initially raise the throttle to a point where the helicopter was becoming light on the ground, then automode could be engaged. A Flight_Height function was called that first filled a queue with previous comparison's between the center of the screen and the height of the center of gravity of the object of interest. The queue was filled to the numberInBuffer_Height items before it was used. After being filled, when a new item was added, the first item of the queue was removed and the average of the queue was calculated and the average value was returned. This value was then used as the correction factor to the current throttle value of the helicopter. The end effect of this accumulator/ set point combo was

a helicopter that reacted slower to fast movements because of the averaging effect of the accumulator. This allowed for more graceful rises and descents.

```
//Height auto adjust
Flight_Height(ref Height_accumulator, ref Height_accumulator_average,
ref Height_range, ref numberInBuffer_Height, ref bufferFilled_Height);

    if (!LockThrottle)
    {
        new_Ch3 += Height_accumulator_average;
        new_Ch4 += Height_accumulator_average;
        Old_Height_accumulator_average = Height_accumulator_average;
    }
    else
    {
        new_Ch3 += Old_Height_accumulator_average;
        new_Ch4 += Old_Height_accumulator_average;
    }
if(PCTxConnected)
pctx.Send((int)new_Ch1, (int)old_Ch2, (int)new_Ch3, (int)new_Ch4, 0, 0,
0, 0, 0); //moved channel 2 to channel 6 old channel 2 because of
channel2 inversion
```

5. Future Goals

The original goal of this project was to create an aerial platform capable of tracking color, but within that goal was the purpose of creating a platform that was inherently stable, physically robust and adaptable. The reason for these subsidiary goals was a long term goal of creating a semi-autonomous aerial platform capable of sending back sensor information from afar. The robustness and stability was needed so that in the case of a loss of signal, the helicopter would be able to descend safely. Therefore, future plans include exchanging the helicopter's receiver with a breadboard, while incorporating range finders with Bluetooth and video to create a truly robust aerial reconnaissance platform.

6. Conclusions

The goals for the original project were met as the helicopter was able to lift off the ground and track an object of interest based solely on color. The platform was very stable and was capable of flying "hands-off" for up to 30 seconds with zero drift. Additionally, the design of the helicopter was improved, strengthened where necessary and lightened considerably. The helicopter can now sustain minor crashes with zero down time. With these improvements in place, the aerial platform is better equipped to handle any future research demands.

7. References

1. *Vision for Machines*. (n.d.). Retrieved May 1, 2007, from <http://www.roborealm.com/>
2. *PCTx*. (2007). Retrieved May 1, 2007 from <http://www.endurance-rc.com/pctx.html>
3. *Blade CX RTF Electric Coaxial Micro Heli*. (n. d.) Retrieved March 20, 2007, from <http://www.horizonhobby.com/Products/TechnicalSpecs.aspx?ProdID=EFLH1200>