Project Proposal

Project Quad

Author:
Andy Fang

Supervisor:
Dr. Mark Brown

February 13, 2014

Contents

1 Overview

1.1 Quadcopter

1.1.1 Defination

A quadcopter, also called a quadrotor helicopter, quadrocopter, quadrotor, is a multicopter that is lifted and propelled by four rotors.[1]

Figure 1: A Maker Faire quadcopter in Garden City, Idaho[1]

1.1.2 Flight Control

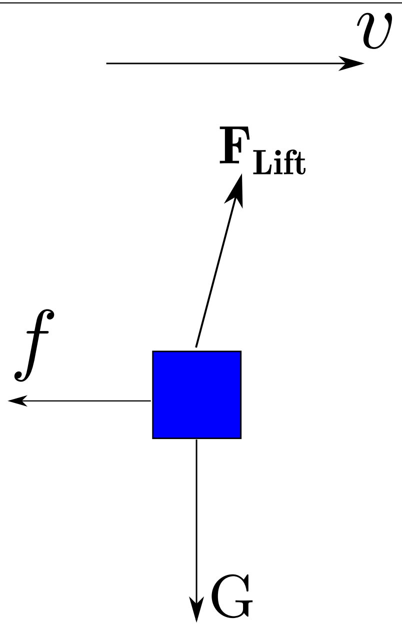


Figure 2: The forces effected on the quadcopter while making stable movement.

[**Hover**]

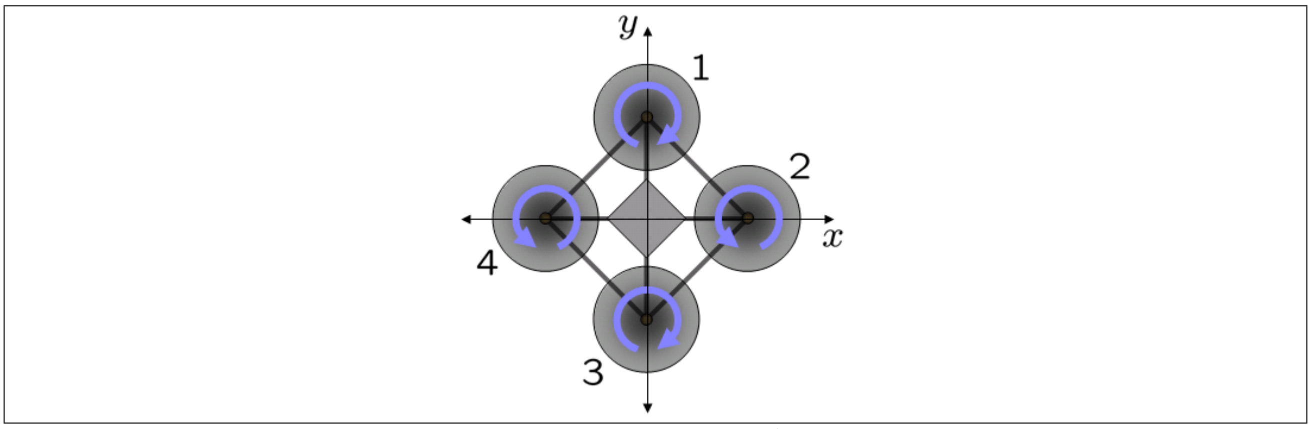


Figure 3: The example model of quadcopter.

Each rotor produces both thrust and torque. If all the rotors are spinning at the same angular velocity, and as the example shows, with rotor 1,3 spinning clockwise and rotor 2,4 spinning counterclockwise, the angular acceleration on the yaw-axis will be zero. This is the method to hovering.

[**Yaw**]

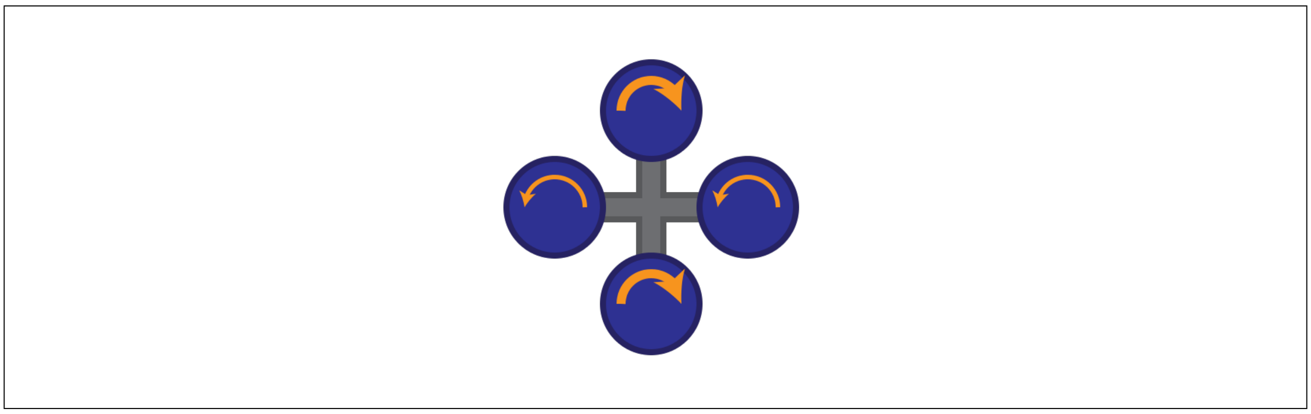


Figure 4: Yaw control

The method of making yaw control can be done by adjusting the angular velocity of one pair of rotors.

[**Pitch**]

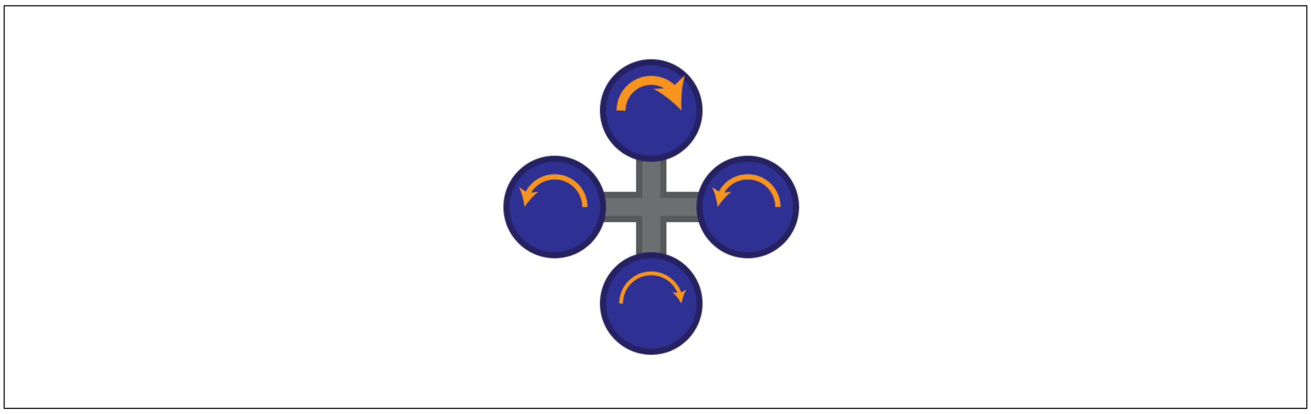


Figure 5: Pitch control

The method of making pitch control can be done by increasing one rotor's spinning velocity and decreasing the opposite rotor's spinning velocity.

[**Roll**]

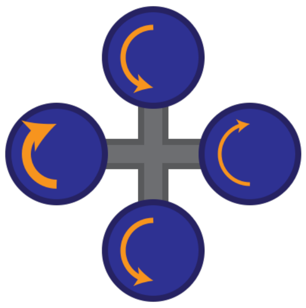


Figure 6: Roll control

The method of making roll control is similar to making pitch control.
It can be done by increasing one rotor's spinning velocity and decreasing the opposite rotor's spinning velocity.

1.1.3 Design Principles

This will probably remain an amateur project. Which means the time spent on this project will be relatively short. Therefore, we really want to keep things simple.

If there is an existing library that implemented a function we need, we can simply include the library in our project. Adoption of *de facto* or *de jure* standards make things not only *simple*, but also *easy to manage*.

After all, *combination counts*.

1.2 Computer Vision

The basic computer vision system in this project will be *OpenCV*[1].

There are several reasons for using *OpenCV*:

- Open source. *OpenCV* is in BSD license. Therefore, our work can be perfectly legal, without any violation of IP laws.
- A large, active online community.
- The Intel background makes it reliable.
- Implemented in C++, the library is really fast.

Now, it is high time for discussing CV related issues.

1.2.1 Quadcopter Identification

The optimal choice, in this case, is a Quadcopter positioning system involves stereo visions and four colored table tennis balls on each Quadcopter. The reason for that decision is fairly simple: *simplicity.* As seen in a youtube video, a research group did successfully implemented a tracking system which requires only mono-colored balls to locate quadcopters. [2] While this is possible, it is also too complex for a small amateur team like ours to make it. However, multi-colored ball system can simplify the mechanism greatly. With limited time and computing power, this is the only logical choice.

2 Optical Tracking

2.1 Overview

The Optical Tracking System is implemented with three subsystems:

1. Video Extraction
2. Pinhole Camera Model
3. Epipolar Geometry

[1] OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez. `http://opencv.org`

[2]Something titled "Machine", "athletic", "Quadcopter". They used reflection markers as the tennis balls in this case.

2.2 Video Extraction

The Video Extraction system's function is to extract the location of a patricular object within the video footage. The objects, in this case, are differently colored small balls. The small balls represent sever key points of the Quadcopter. By determining the location of the balls, we will be able to calculate the position and attitude of the Quadcopter. The procedure of video extraction is described as following:

1. Shotting: Get images from camera,
2. Transforming: transform the image to HSV colorspace,
3. Extracting: use tresholding to extract pixels in patricular color range, then use histogram and backprojection to determin the probablity that the pixels belong to the model,
4. Denoising: use opening and closing to denoise,
5. Determining: use *Lucas-Kanade* and *CamShift* algorithm to determin the centroid of the extracted pixels.

2.2.1 Shooting

In this project, v4l[3] is the middleware of OpenCV and camera. With *v4l* and *OpenCV*, reading an image from camera is as easy as following:

```
cap = cv2.VideoCapture(source)
img = cap.read()
```

2.2.2 Transforming

Different from the RGB colorspace, HSV colorspace has a unique character: its V channel represents the brightness. If we remove V channel from tresholding, the same color profile will be able to work in different lighting conditions. The transforming process is described below[4]:

$$M = \max(R, G, B) \tag{1}$$

$$m = \min(R, G, B) \tag{2}$$

$$C = M - m \tag{3}$$

$$H' = \begin{cases} \text{undefined}, & \text{if } C = 0 \\ \frac{G-B}{C} \mod 6, & \text{if } M = R \\ \frac{B-R}{C} + 2, & \text{if } M = G \\ \frac{R-G}{C} + 4, & \text{if } M = B \end{cases} \tag{4}$$

$$H = 60^\circ \times H' \tag{5}$$

$$V = M \tag{6}$$

$$S = \begin{cases} 0, & \text{if } C = 0 \\ \frac{C}{V}, & \text{otherwise} \end{cases} \tag{7}$$

The code used to transform the image is:

```
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

Note that by default, OpenCV uses BGR color space.

2.2.3 Extracting

OpenCV's built in treshold function is used to utilize multithreading. The process of tresholding is printed below:

```
mask = cv2.inRange( hsv,
                    np.array((0., 60., 32.)),
                    np.array((180., 255., 255.))
                    )
```

[3]a.k.a. Video For Linux

However, simply tresholding is often not enough. The object tracked can have a complex color feature. In order to track the entire object, it is necessary to consider all of the features. Histogram is used to model the color distribution of an object, and me can then map the probability of pixels using backprojection.

Backprojection uses the modeled histogram to find the probability that certain pixel belongs to the model.

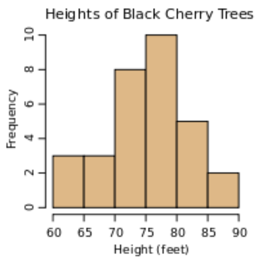


Figure 7: A sample histogram[5]

Consider an image matrix M, each pixel can be described as pair $(H_{i,j}, S_{i,j})$. We can use the two values to create a 2D histogram that represents the distribution of certain color range in H and S. Below is a sample 2D histogram generated by *OpenCV Python Sample*.

After generating the histogram, the probability that a pixel belongs to the model can be expressed by the

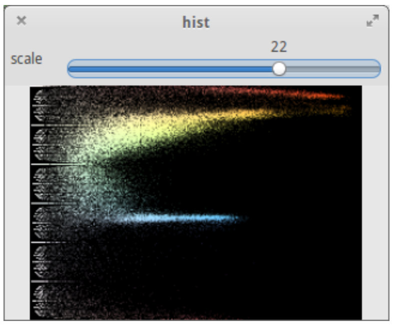


Figure 8: A sample 2D histogram

following steps:

1. Find the pair $(H_{i,j}, S_{i,j})$ of a pixel,
2. find the correspondent bin in the histogram,
3. get the relative frequency of the bin.

Then we can use the frequency to map a binary image. In this image, each pixel represents the probability that the correspondent pixel in the original image belongs to the model. Below is an example backproject image:

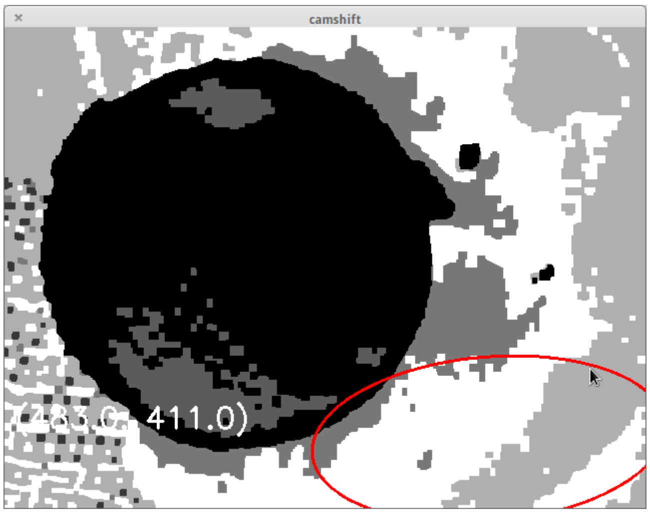


Figure 9: A sample backprojrction

2.2.4 Denoising

After calculating the backproject image, we will be able to use morphology transformation to remove noises. In this projcet, we used Opening and Closing to remove small noise pixels and fill black pixel holes in the backprojection.

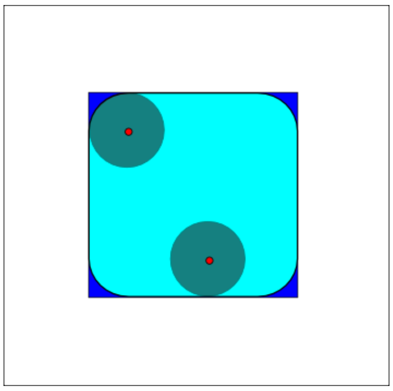


Figure 10: Sample Opening operation[7]

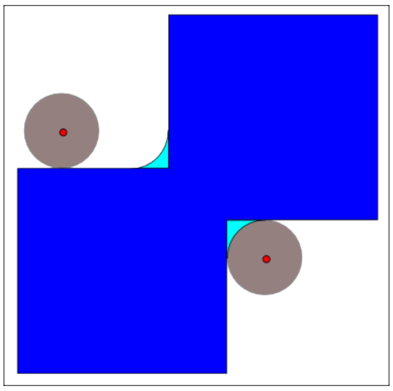


Figure 11: Sample Closing operation[6]

The result is significant. A great deal of noises is removed in this process:

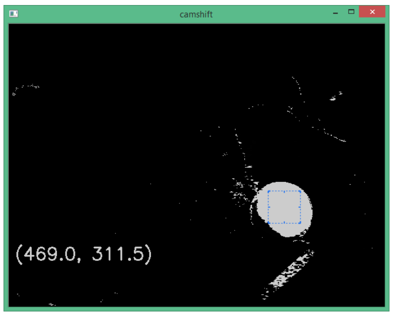
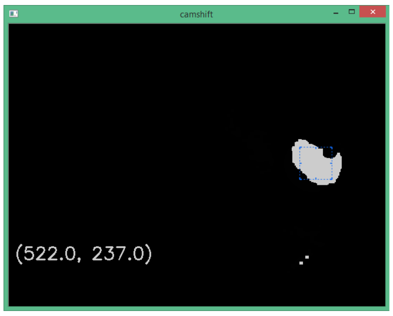


Figure 12: Befor morphology transformation



(522.0, 237.0)

Figure 13: After morphology transformation

2.2.5 Extracting

There are two stages of extracting. They are:

1. Detecting
2. Tracking

The difference between them is that *Tracking* requires the original location of the object to be known. That information should be fed with *Detecting* in the *first* frame of video source.

By *Detecting*, we find locations where the image of *source* is similar to *template*. The algotighm we used in *Detecting* is rather simple. It "shifts" the *template* on the *source*, and find the similarities bewteen the *template* and the image below.

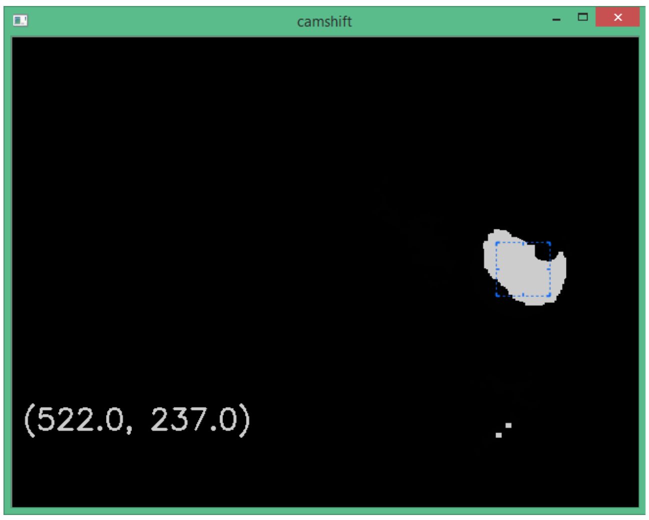
The method of template matching is printed below:

```python
import numpy as np
import cv2
import math

def match(template, image):
  '''
  Match template on image and return the centroid's coord.
  '''

  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

  result = cv2.matchTemplate(gray, patch, cv2.TM_CCOEFF_NORMED)
  result = np.abs(result) ** 3
  val, result = cv2.threshold(result, 0.01, 0, cv2.THRESH_TOZERO)

  minVal, maxVal, minLoc, maxLoc = cv2.minMaxLoc(result)

  height, width, depth = template.shape
  x, y = maxLoc
  x += width/2
  y += height/2

  return (x,y)
```

In order to make things clearer, we used a marker from the TV series *Person of Interest* to mark the point of interest. Here's the code we used to mark the object:

```python
import numpy as np
import cv2
import math

def draw_machine_mark(size, location, image):
    '''
    This function draws a machine mark on the image
    '''
    original = cv2.imread("assets/machine.png", -1)

    osize = math.sqrt(original.size)/2
    ratio = size / osize
    height, width, depth = image.shape
    timg = cv2.resize(original, (0,0), fx=ratio, fy=ratio)
    x, y = location

    image[:] *= 0.8

    x_start = max(0, x - size/2)
    x_end = min(x_start + size, width)
    x_start = min(x_end - size, x_start)
    y_start = max(0, y - size/2)
    y_end = min(y_start + size, height)
    y_start = min(y_end - size, y_start)

    roi = image[y_start : y_end, x_start : x_end]
    for c in range(0,3):
        roi[:,:,c] = timg[:,:,c] * (timg[:,:,3] / 255.0) + roi[:,:,c] * (1 - timg[:,:,3] / 255.0)
    image[y_start : y_end, x_start : x_end] = roi
```

Below is an image of the effect:



```
(522.0, 237.0)
```

Figure 14: Example of tracking effect

References

[1] Wikipedia , *Quadcopter.* `https://en.wikipedia.org/wiki/Quadcopter`

[2] GitHub , *ardupilot.* `https://github.com/diydrones/ardupilot`,`http://ardupilot.com/`

[3] OpenCV , *Meanshift and Camshift.* `https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_video/py_meanshift/py_meanshift.html?highlight=camshift`

[4] Wikipedia , *HSL and HSV.* `http://en.wikipedia.org/wiki/HSL_and_HSV`

[5] Wikipedia , *File:Black cherry tree histogram.svg.* `http://en.wikipedia.org/wiki/File:Black_cherry_tree_histogram.svg`

[6] Wikipedia , *File:Closing.png.* `http://en.wikipedia.org/wiki/File:Closing.png`

[7] Wikipedia , *File:Opening.png.* `http://en.wikipedia.org/wiki/File:Opening.png`