Assignment 3a: Mini Drone Race

Abhinav Modi
Masters of Engineering in Robotics
University of Maryland, College Park
Email: abhi1625@umd.edu

Prateek Arora
Masters of Engineering in Robotics
University of Maryland, College Park
Email: pratique@terpmail.umd.edu

I. Introduction

The objective of the project is to navigate through a colored window of known size but unknown position and orientation. In our case the window is yellow in color. The challenge was to detect the window in different lighting condition. Several techniques were used for window detection to account for orientation and illumination change. Finally, after window detection the pose of the quadrotor with respect to the window is computed for the purpose of navigation.

II. Data

The data for this project was recorded manually from the Leopard Imaging camera on the quad. The data was collected in a rosbag file at a resolution of 800x460 at 60 fps from various angles and in different lighting conditions. In order to train the GMM we required prior knowledge of yellow color. For this, we used a function similar to roipoly in MATLAB created by Jörg Döpfert [1].

III. Our Approach

The approach is subdivided into two parts: (1) window detection, where the data from the camera is filtered and Gaussian mixture model is trained on the data and (2) pose estimation, where 6 DOF pose of quadrotor is computed with respect to the window. Various components to the pipeline are discussed in detail below:

A. Window Detection

Given a yellow colored window, color based segmentation was the obvious choice. We started off with a simple range based threshold for the yellow color. The segmentation output for this was not very good as some pixel values on that window didn't lie in the defined range. This problem is like selecting a small cuboidal subset in a cubical RGB color space with each dimension ranging from 0-255 which encloses all the values of yellow. This surely works in an ideal scenario but in real world, due to introduction of noise and different lighting conditions the desired range of values occupies a weird volume in this 3D color space.

B. Alternative Color Spaces

To make this estimation apart from RGB other color spaces were also tried like HSV, LA*B*. Since the camera didn't have auto exposure the quality of data was quite terrible, thus the performance was similar in all the color spaces. To get an estimate of the number of Gaussian, histogram of data samples recorded were plotted to check the peaks in the data for each channel.

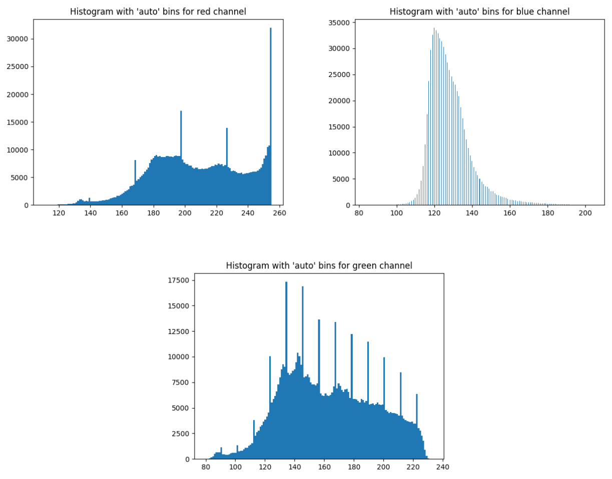

Figure 1: Histogram of intensity values in RGB channels respectively

C. Single Gaussian and Gaussian Mixture Models

In order to tackle illumination changes Single Gaussian and Gaussian Mixture Models are used to predict the color of the window and get a binary mask. Single Gaussian models the color space perfectly for a certain illumination, but doesn't work well for drastic change in illumination. For account for this, GMM are used, which was trained on the data we collected. By observing the peaks in the histogram of pixel intensity values we determined the number of Gaussian that was best for our training data set. Fitting six Gaussians was more accurate than four, but the former was much slower to compute than the later. Finally four Gaussian GMM was selected as it offered a good trade-off between accuracy and speed.

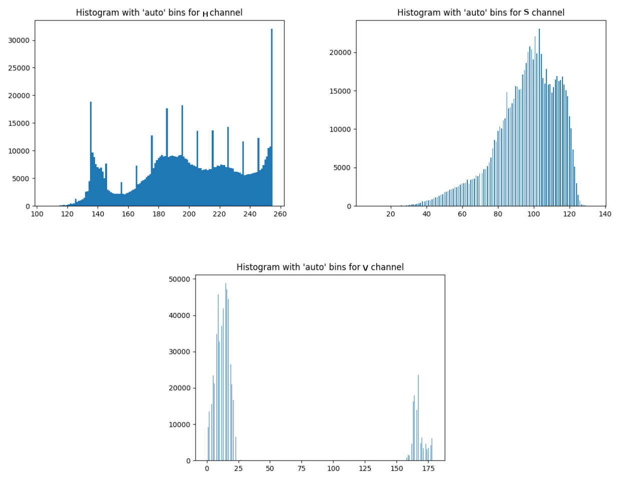

Figure 2: Histogram of intensity values in HSV channels respectively

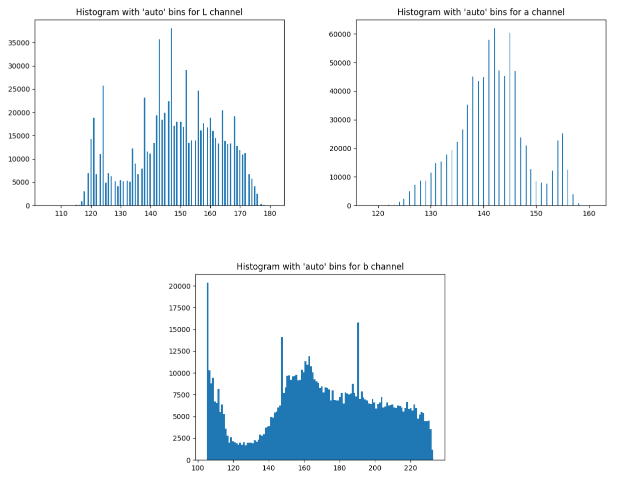

Figure 3: Histogram of intensity values in LA*B* channels respectively

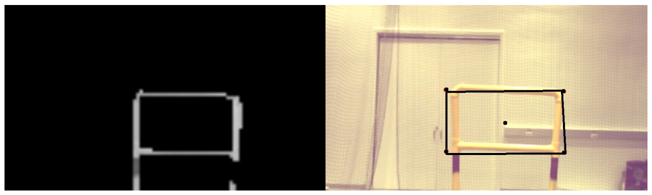

Figure 4: Histogram of intensity values in LA*B* channels respectively

D. Pose Estimation

After the four corners of the windows are robustly computed, perspective-n-points algorithm is used to estimate the pose of the quadrotor. Since the dimensions of the window are known aprior, it is a trivial task to compute 6 DOP pose. The world frame is arbitrarily chosen as the top left corner of the window. We used opencv function “cv2.solvePnP” to perform PnP.

Note: Camera coordinate system is as follows: positive x points into the image, positive y points to the left of image and positive z points upwards.

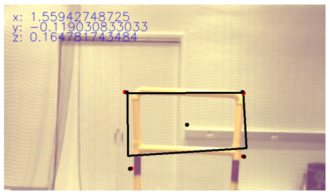

Figure 5: 3D Pose of window center w.r.t. camera

E. Control Policy

After taking off from the ground the quad survey the area by yawing slowly. As soon as the window is detected a simple PD control law is used to follow the trajectory of desired waypoints. This PD loop runs until the quadrotor aligns itself to window and is at a distance of 1 meter from the window. After this, the quadrotor pushes forward in open-loop state.

IV. Discussion and Conclusion

There were a lot of problems encountered while doing this project. First, the input images received from the camera were of very bad quality. There was yellowish-green tint in the whole image which made it harder to collect good data for the color yellow as the model would sometimes assume white backgrounds as yellow due to the tint. This was reduced to only an extent by tweaking the bias correction values in the camera configuration file. Further image quality was improved using ’gamma correction’. This improved the overall contrast and brightness of the image. We used a value of $\gamma = 1.5$ for our purpose.

After getting an estimate of the window mask we performed morphological operations, mainly closing to make the mask more connected and then used hough lines with a very high probability on this mask to get multiple lines only along the sides of the window. Finally we approximated the window by taking a convex hull of the predicted hough lines and perdicted the 4 corners.

The output of the GMM was very slow and the output of the hough lines was very noisy. The combination of these issues aggravated the noise in the estimate of corners even for small vibrations. The GMM inference was increased by downsampling the image and then resizing it again for running PnP, but still the maximum rate we could achieve was 10-12 Hz. To further reduce the noise in the estimates we created a moving average filter which would store the corner detections of the previous 5 frames and then used the mean of these corners as our corner estimates for the current frame. The output for this can be seen in the video available in the submission folder.

References

[1] Roipoly equivalent in python. https://github.com/jdoepfert/roipoly.py.