Window Detection and Navigation with the PRG Husky - USING 1 LATE DAY

Team `sudo rm -rf *`

Abhishek Shastry
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: shastry@umd.edu

Animesh Shastry
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: animeshs@umd.edu

Nicholas Rehm
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: nrehm@umd.edu

***Abstract*—In this report, a window detection algorithm and trajectory planner is implemented on the PRG Husky for ENAE788M: Hands on Autonomous Aerial Robotics. The window detection scheme utilizes a Gaussian mixture model for color segmentation and Hough line transform for window corner detection. Vehicle location is calculated with respect to the window so that a trajectory through it may be generated and carried out by an odemetry feedback based outer loop control scheme. An EKF is used to refine the vehicle location with respect to the window and to ensure accurate position throughout the flight. Results show that this method is effective in successfully navigating through colored windows.**

I. Introduction

Traversing through a window in three-dimensional space with a quadrotor is a difficult task that requires the use of computer vision and robust trajectory planning. A window of constant color can be loosely detected through the use of color segmentation with additional processing of the binary mask to further filter out noise. To detect window corners from this mask, a combination of Hough line transform and Harris corner detection may be used. If the parameters of the camera and coordinates of the real window corners are known, a camera pose can be extracted from the image given the detected corners. This estimated pose can be used to generate a trajectory through the window which can be refined as additional pose estimates are generated.
Link to the result videos: Click Here

II. Camera Calibration

The Leopard Imaging global shutter USB camera must be calibrated for color due to the output image being raw sensor data combined with possible color change from the new wide-angle lens. It was found that the provided driver for the camera simply scales the output values for the red, green, and blue channels by a constant intensity value. This ultimately has no effect on color segmentation ability because every pixel is modified in the same way. These parameters were loosely tuned for the sake of consistency and to prevent an over or under-exposed image, but this had little effect on the achieved color segmentation.

The camera matrix K and distortion coefficients D were computed with the camera calibration implemented in ROS. These are required in extracting the position and orientation of a known object in the camera frame and are specific to the camera sensor and lens used. The camera matrix and distortion coefficients found through calibration are given as:

$$K = \begin{bmatrix} 743.60 & 0 & 409.0 \\ 0 & 750.18 & 213.9 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$D = [-0.3378, 0.1423, -0.0015, -0.0036, 0.0] \tag{2}$$

III. Color Segmentation

A basic Hue-Saturation-Value (HSV) color space was selected to be the basis for the color segmentation implementation for the following three methods. Note that a slight Gaussian blur is applied to the original test image before the binary mask is generated to reduce noise in the final output.

A. Simple Color Thresholding

A simple method for generating a binary color mask is achieved by iterating through every pixel in an image and assigning binary values for pixels that either fit a certain color criteria or do not. The HSV color space allows for the color's hue to be the main parameter that is checked for the correct desired color. Low values of saturation and upper and lower values of the value parameter can also be thresholded out, as these colors represent shadows and highlights that are not desired. The results of this method performed on a test image (Figure 1) are shown in Figure 2 for a yellow gate after manual fine tuning of the threshold parameters.

Fig. 1. Raw test image

Fig. 2. Binary mask generated using simple color thresholding

B. Single Gaussian

Single Gaussian color segmentation was implemented in python where we wrote our own Gaussian functions to evaluate the probability of each pixel being a part of the window. It was not used in final implementation because we found it to be very slow for any real-time implementation.

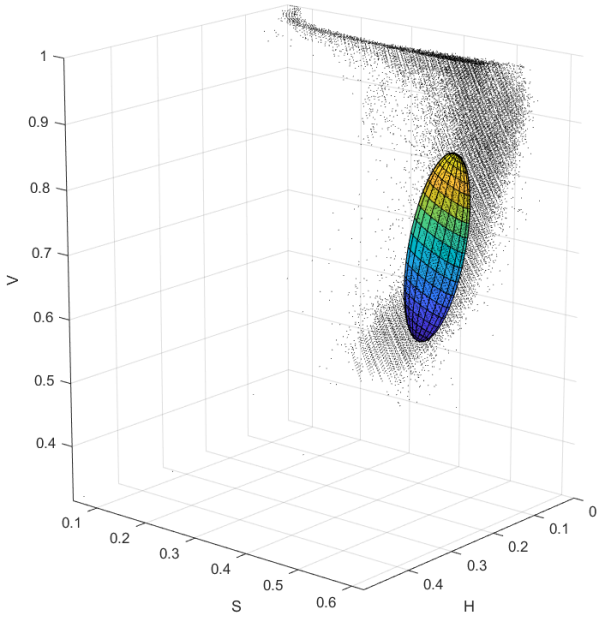

Fig. 3. Single Gaussian fit to test data in HSV color space

Fig. 4. Binary mask generated using single Gaussian

C. Gaussian Mixture Model

GMM color segmentation was implemented in python which was trained on the dataset collected under different lighting conditions. It was not used in final implementation because we found it to be very slow for any real-time implementation.

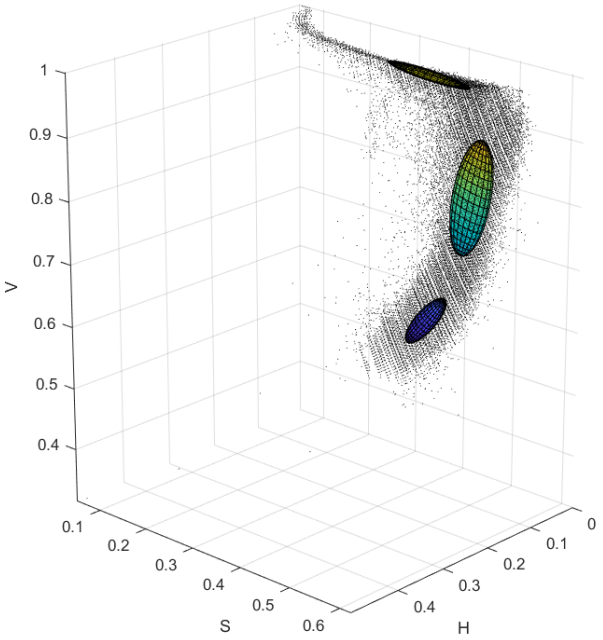

Fig. 5. GMM fit to test data in HSV color space

Fig. 6. Binary mask generated using GMM

IV. Window Gate Detection

The window detection algorithm implemented for this project is a multi-step process that has been refined to provide a reliable and consistent result. The final color segmentation method implemented for the live demo was the basic color thresholding because the other methods were very slow when implemented in python.

A. Pre-processing and Binary Mask Generation

The raw image from the onboard camera is first converted to the HSV color space and scaled down to 138x240 pixels to reduce computation power required for processing each video frame. Noise in the binary color mask became an issue for images any smaller than this size. A slight Gaussian blur is applied before color segmentation is performed.

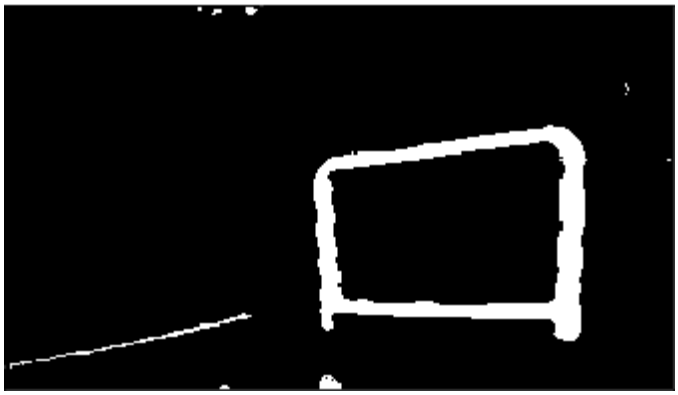

Fig. 7. Binary mask after color segmentation

B. Erosion and Dilation

The binary color mask is eroded and then dilated to eliminate any noise that made it through the color segmentation step. Dilation assists in reconstructing any discontinuities in the binary window mask.

Fig. 8. Binary mask after Erosion/Dilation

C. Selection of Largest Contour

Every contour in the binary image is identified and only the largest is kept, which is assumed to be the window by this point.

Fig. 9. Binary mask after largest contour isolated

D. Line Fitting

Lines are fit to the edges of the largest contour using a Hough line transform and are slightly dilated to eliminate any gaps between similarly positioned lines.

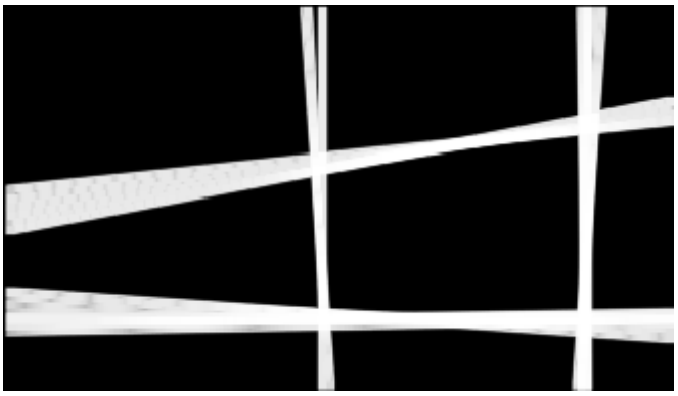

Fig. 10. Lines fitted to largest contour

E. Corner Detection

Intersections of the lines are detected by Harris corner detection implemented in OpenCV. This locates four corners at each line intersection which are then clustered together using HBDSCAN to generate the final window corners.

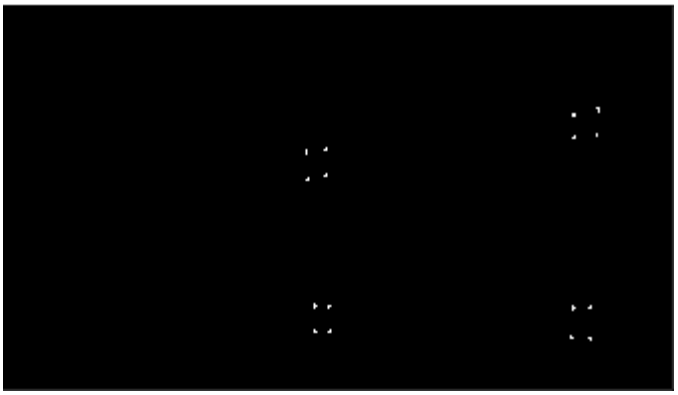

Fig. 11. Corners of line intersections detected using Harris corner detection

Fig. 12. Final window corners detected

V. Camera Pose Recovery

If four corners of a window are successfully detected, a camera pose relative to the fixed window is computed. The solvePnP function implemented in OpenCV is provided with known window corner coordinates in the fixed world frame, pixel locations of the corresponding four window corners, the camera matrix K, and camera distortion coefficients D. The recovered pose for test image shown in Figure 1 is re-projected in the form of a .5 meter line at each corner on the original image shown in Figure 13.

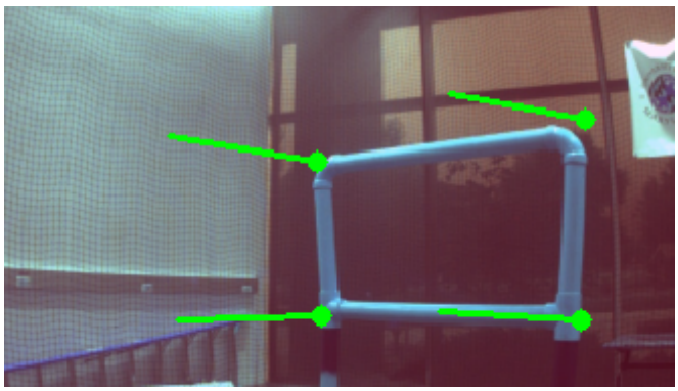

Fig. 13. Recovered window pose with .5 meter lines extending out from detected corners

VI. Trajectory Planner and Controller

A. EKF

An EKF was implemented on top of the estimate of gate pose from vision after converting it to the inertial frame. The inertial frame was chosen because both the state and observation model reduces to identity matrices in this frame and hence EKF reduces to simple linear KF and is therefor an optimal observer which is easier to implement. The pose estimate from EKF was more stable

B. Search

A search algorithm was implemented to search for the window in the arena after take-off. Search was done using a spiral pattern in Yaw-Z plane and bounded by max and min

values in Z and yaw. Search will terminate as soon as EKF publishes a window pose with low enough covariance.

C. Waypoint

Initial waypoint after a lock on the window is confirmed is 2.1 m in front of the window where the quad goes as soon as the search terminates. The distance to the window is then slowly decreased to about 1.5 m while maintaining the yaw aligned to the window. At 1.5 m , the window covers the entire camera frame. Hence, after that the quad is just commanded to go forward by 3m to go though and clear the gate.

VII. Results and Conclusion

The window detection and navigation method implemented in this paper was successful in completing 3 out of 4 of the tests within the allotted time. Issues with lighting conditions led to reduced color segmentation performance and thus difficulty getting the EKF to converge on the window pose. While the color segmentation was tuned to work well with nearly all lighting conditions, direct sunlight through the blinds in the room led to sever overexposure of the raw camera image and hindered the ability to detect the window accurately. In the future, auto white balance and exposure correction can assist in mitigating these effects. Under controlled lighting conditions, the window detection and navigation method implemented in this paper had a very high success rate. Time performance could be improved by further tuning the EKF and possibly loosening the tolerances of the intermediate waypoints. GMM model could be made faster by either writing it in a different C++ ROS node or by using inbuilt numpy or scikit functions in python. This will allow it to be used for online segmentation purposes.