Team 5 - QDMC - Avoid the Wall and Cross the Bridge

Vishnu Sashank Dorbala
University of Maryland
vdorbala@umd.edu

Tim Kurtiak
University of Maryland
tkurtiak@umd.edu

Ilya Semenov
University of Maryland
isemenov@umd.edu

Surabhi Verma
University of Maryland
sverma96@umd.edu

***Abstract*—This report presents the implementation of computer vision algorithms for detecting and estimating depth of a wall, detecting a river and bridge based on textures, and their corresponding controls to cross the bridge and avoid the wall.**

I. Problem Statement

The aim of this project is to detect and avoid two different obstacles using computer vision. The first obstacle is a wall with unknown dimensions and unknown height. The wall must be identified using the front facing monocular camera of the ENAE788M Bebop drone and the drone must fly above or below the wall while staying between it's posts. See Figure 1 below for a diagram of the wall and possible height configurations. The second obstacle is a river, which may not

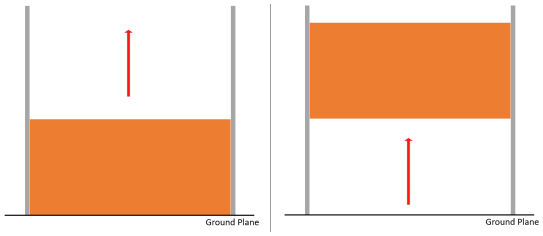

Figure 1: Diagram of Wall

be flown over. The river can only be crossed by navigating over a bridge, shown in Figure 2. The river must be detected using the down facing camera, which is a stereo greyscale camera. As such, the blue and brown colors can not be effectively used to distinguish the river and bridge from the background. Instead, a texture based recognition algorithm is implemented to identify and overcome this obstacle.

Figure 2: River Obstacle

II. Bridge Problem

The river and bridge must be flown over, prompting the use of the down facing Duo camera. Since this camera is gray-scale the identification of the river and bridge cannot take advantage of color. Additionally, this is a flat feature so stereo identification is also not helpful. Alas, we turn to texture based methods.

A. Law's Texture Energy Masks

The first attempted implementation made use of Law's texture energy masks. These masks are a a series of 2D convolutions that create, after some filtering, 9 variables describing a texture aspect for each pixel in the image.

Shown in figure 3 are the fundamental vectors that can create 16 possible combinations of matricies. These matricies are convolved with the image to create a 16 element vector at each pixel. The vectors corresponding to inverted convolutions are averaged together for rotation invariance (i.e. L5R5 and R5L5).

```
L5  =  [ +1  +4  6  +4  +1 ]  (Level)
E5  =  [ -1  -2  0  +2  +1 ]  (Edge)
S5  =  [ -1   0  2   0  -1 ]  (Spot)
R5  =  [ +1  -4  6  -4  +1 ]  (Ripple)
```

Figure 3: Law's fundamental vectors

These remaining 9-d vectors can be clustered with the k-means method. For robustness, the number of clusters is found by using the elbow method with a random subsample of the entire image for speed.

Looking at figure 4, we can see a grayscale image with various pbjects that would be difficult to discern with thresholding alone. However, after applying Law's energy masks we can see the image has neatly broken down into 4 base textures, shown in figure 5.

This method was discarded eventually due to the large computation time involved in taking 33 convolutions of the entire image.

B. Min-Max Differential Filter

This idea was inspired by Animesh from team sudo. A custom 2D convolution takes the difference between the maximum and minimum values of a window, and uses that as the value of the pixel at the center of the window. This creates a map of featured areas. Due to the intensely varying carpeting, and relatively uniform surface of the bridge and river a mask

Figure 4: Original image pre-textures

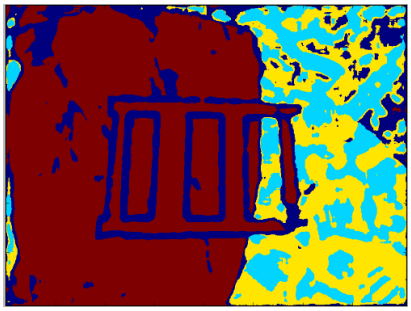

Figure 5: Laws textures result

of only the minimum values from this differential filter returns regions of interest as apparent in figure 6. Unfortunately there are other areas with few features in the netted area that are included in the mask, however, they tend to be darker areas as well. Because the bridge has a light color in the grayscale image, a bitwise AND operation of a simple threshold for higher values along with this differential filter low value mask returns the bridge and river reliably. This AND operation is shown in figure 7

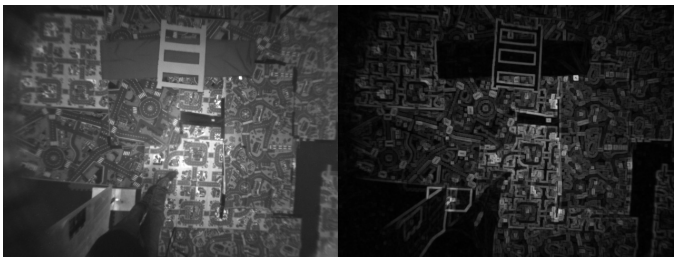

Figure 6: Differential filter of the image on the left is on the right

Figure 7: Regular thresholding on the left, Differential threshold in the middle, and resulting mask on the right

C. Poor Man's Differential Filter

This idea was inspired by Nick from team sudo. Due to the long processing time of a true differential filter, a similar result is obtained by using a liberal Canny filter, and utilizing only the areas with few edges. The resulting mask is noticeably worse but, much faster as you can imagine from the base output shown in figure 8.

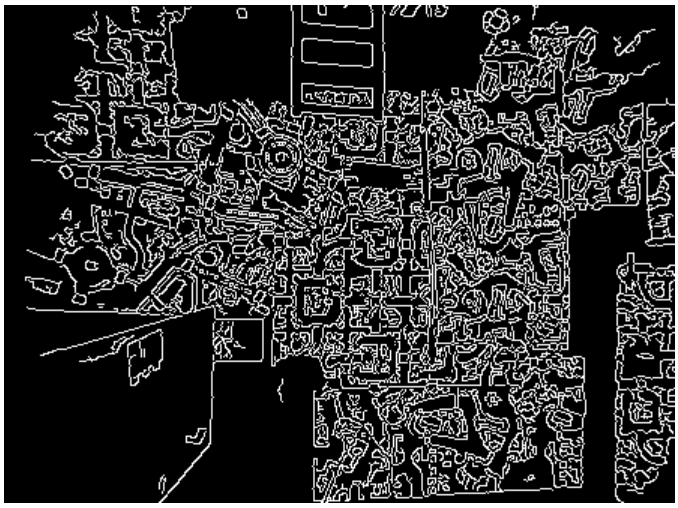

Figure 8: Example of a liberal canny result, morphological operations here can approximate the differential filter mask

This approach is combined with a custom double-threshold filter to produce an adequate mask. One issue is that even with the IR LED that the duo3d camera uses off, there is still a noticeable bright spot in the center of it's vision. The double mask uses two sets of thresholds for the inner and outer areas of the image to account for a drop in bridge brightness out of frame, without creating masks that are too inclusive.

The AND operation as before is used here as well.

This mask can be used with morphological operations to obtain the largest contour. The largest contour, assumed to be the outer perimeter of the bridge, is used to additionally find the holes in the bridge by filtering on: relative perimeter, distance to maximum contour, solidity, aspect ratio, and parallelism. These inner holes tend to lend well to accurate rotated rectangle fits, which allows the calculation of the line that crosses the bridge in the correct direction. The center of the bridge is assumed to be the center of the largest contour, A.

D. Bridge Controller

With the line and a point found, one can assume that the vehicle lies at the center of it's own image C. Therefore the line segment representing the minimum distance to the vehicle from the line AD can be found and used to inform the controller.

If this line segment distance is long, then the vehicle is at a skew to the bridge and should navigate to this line by attempting to find point D. If this distance is short, then the vehicle is lined up to cross the bridge and can proceed to

attempt to cross it. An example of how this is working is shown in figure 9, not that C is in a dummy location here for illustration.

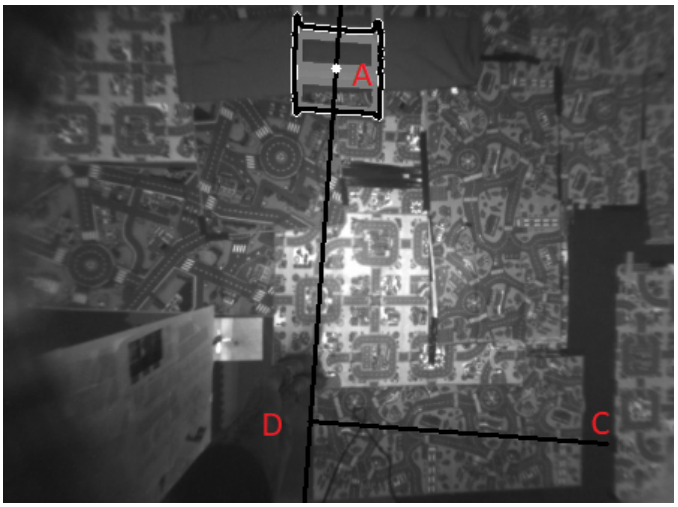

Figure 9: Example of calculations to approach bridge

III. Wall Problem

Our approach basically involves extracting the wall from the image. For this, we find SURF features located only on the wall and use a FLANN matcher to find correspondences in the incoming image stream from front facing camera. However, this led to a lot of incorrect data matching or outliers as seen in Fig. 10. This could be do to a lot of feature points in the environment, especially from the textured carpet. We therefore sought to reduce the search space of the matcher by obtaining a mask as described below. Fig 11 illustrates the process.

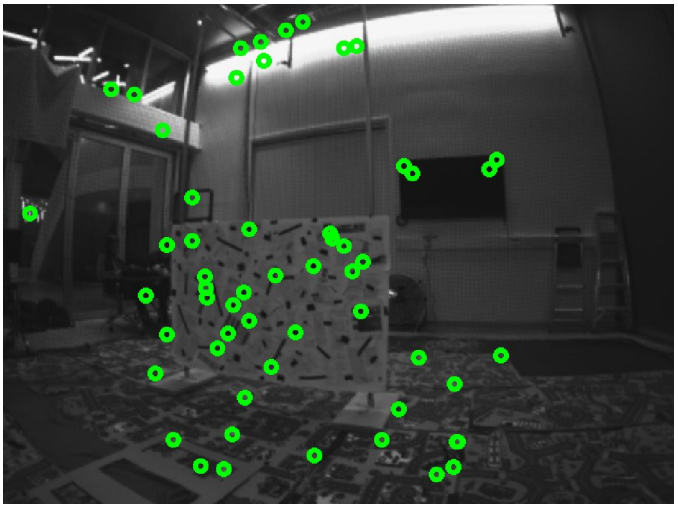

Figure 10: Several Outliers due to Feature Matching Errors

- Adaptive Thresholding on a grayscale median filtered image.
- Erosion for noise removal.
- Finding external contours on the resultant image.
- Applying a series of dilations to fill in the gaps in the contoured image.
- Inverting the image to find intermediate masks.
- Choosing the largest masks by area.

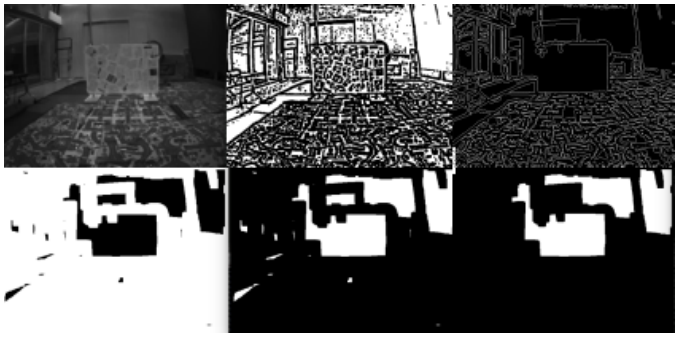

Figure 11: Obtaining a mask for feature matching

The mask eliminates the textured carpet in the image and allows only the wall and some background regions. Applying feature detection on the resultant image results in maximum points on the wall region. The matching is more robust with very few outliers as can be seen in Fig. 12.

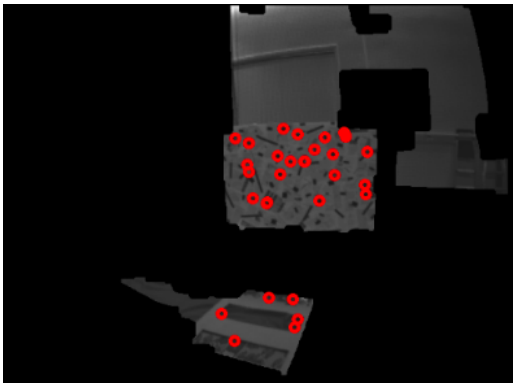

Figure 12: Feature matching in masked regions

Next, we apply K-means to cluster the matched points. The cluster with highest number of points would correspond to the wall. The mean of these points is the approximate center of the board. Fig. 13 shows the cluster points in red and blue and the green dot represents the centroid of the blue cluster. After post processing, the average centroid of a sample image looks like Figure 14.

It can be noted that the feature centroid is not always at the center of the wall. However, it is always on the wall. To aid in the robustness of this method, the matches are grouped by clusters and only the strongest cluster is kept. Additionally, the remaining points are passed through a RANSAC function and finally a moving average (low pass filter) of the previous 5 results was taken.

IV. Wall Control Strategy

The wall control strategy is simply to track towards the centroid of the wall features. A simple control strategy was

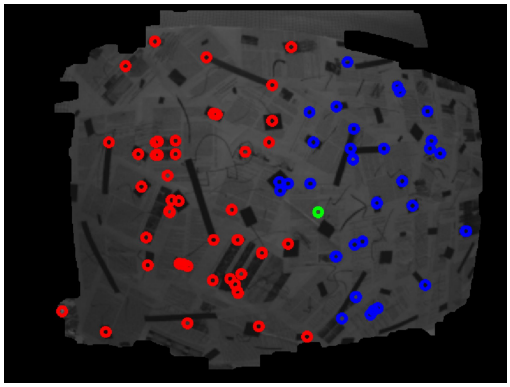

Figure 13: Wall Feature Detection

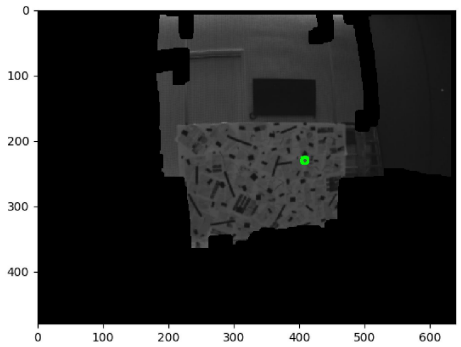

Figure 14: Average Feature Detection Centroid

implemented to track the centroid of the wall to the center of the camera in order to line up cross track. Once centered, the aircraft is instructed to slowly move forward toward the wall while maintaining a centered cross track.

We use the centroid found to determine if the wall is set up in a short or tall configuration, and whether to set vehicle altitude above or below the obstacle. The vehicle is then trained to maintain a heading towards the wall and pass through the plane before landing.

V. Results

The detection methods and control strategies worked very well in practice. The methods are sensitive to different levels of illumination, but overall work well without any major issues.

VI. Lessons Learned

- Detection algorithms must work quickly in order to base feedback control off of them. Slow detection causes sloppy and laggy control.
- Robust detection is the cornerstone to good control. Errant detection causes bad controller responses which can lead to a crash.
- The drone is pretty fragile, and can get damaged even during minor crashes. As these crashes can take up a lot of time to repair, we should tape the fuselage well to speed up our efficiency during demos.
- Another important test time trick we learnt was to make efficient use of roslaunch files along with bash scripts to execute routine tasks faster.
- The "*most important*" run-time lesson learnt perhaps was to land the drone if it looks like it's about to crash. Drone crashes take time to repair, and in our case, prevented us from successfully completing one of the tasks.

VII. Conclusion

- The bridge crossing worked well, and and used only the down facing camera output to find and go across the bridge.
- Although the wall avoidance task worked several times during our trials, it failed to successfully avoid the wall during the demo. This can be attributed mainly to the time loss due to the unexpected drone crash, and bad landing control.

VIII. References

- Lecture 16 Sanket, Nitin; ENAE 788M Lecture 16 https://drive.google.com/ file/d/1jcbLtuw7ptxDDn0Djl9SUXXX8ENVdTu2/view
- FlannMatcher https://opencv-python-tutroals.readthedocs.io/en/latest/pytutorials/pyfeature2d/pymatcher/pymatcher.html.