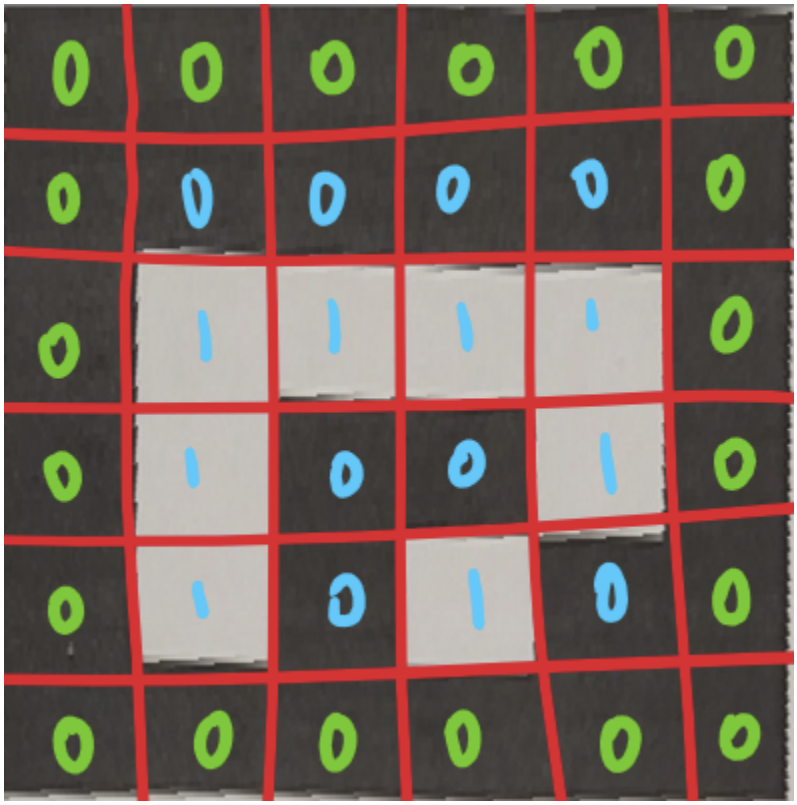
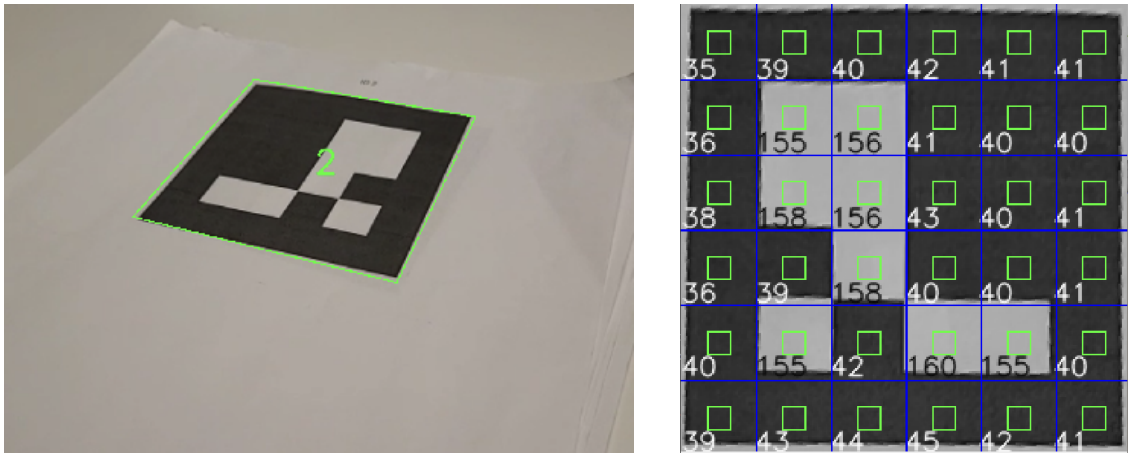
The classification component of my AruCo marker detection is a fairly straightforward process, and doesn't require the more complicated probabilistic classifiers we discussed in class. After the preprocessing pipeline is completed, potential detected markers are normalized via a perspective transformation to a square of known dimensions. These normalized markers are then ready for classification. The binary code they represent is designed to be easily readable and unambiguous, thus the straightforward approach. First, the image is sliced into a grid of N x N "pixels" based on the size of the marker. For example, I used markers with a 4 x 4 code so after adding the 1 pixel padding surrounding the code the marker size is 6 x 6.

Pictured below is an example of the grid pattern for marker ID 1. Padding is shown in green and the encoded data is shown in blue.



Each pixel represents a small square subset of the image, and we need to determine whether the value is black, representing a 0, or white, representing a 1. A simple and reasonably accurate approach is to take the average grayscale value of the image as a threshold. Another approach I tried was first calculating the histogram of the image, looking for two peaks indicating the primary intensities of black and white pixels, and then computing the threshold as the median point between peaks. However, this approach didn't work well for markers in inconsistent lighting or shadows,

where there may be more than two dominant intensities. If a pixel's average value is above this threshold it is counted as a 1, otherwise it is considered a 0. The marker has now been converted from an image to a matrix of pixel values. Now the ID of the marker can be quickly pulled from a hashmap that has precomputed permutations of the code (rotations and reflections).



Seen above is an example of a detected marker and the perspective transformed detection view. The blue lines represent the pixel division of a 4 x 4 marker, and the green squares represent the segment of each pixel that is considered for calculating the average value. Average value is shown in each square, black font means the pixel value is above the threshold and a '1', while a white font means the pixel value is below the threshold and a '0'.

I used two datasets for training and validation. The first was collected by me and contains 503 images over a wide range of lighting conditions, perspectives, and backgrounds. This dataset is a fairly difficult challenge for these reasons. I recorded statistics for whether my program was able to detect at least one ArUCo marker per image. 76.0% of the training images were detected, and 68.5% of the validation images were detected. The other dataset is much larger but more uniform, containing images of markers on an old screen with interlacing issues. In this set, 58.9% of training images were detected and 58.1% of validation images were detected.

Because my approach didn't implement any form of machine learning, the training data acted as a test bed for me to develop an algorithm. Because of this, there isn't a drastic difference in the results between the testing and validation data. The images I collected were divided into training and validation based on my own intuition.

The major inaccuracies I noticed across the testing data were related to shadows occluding the markers, motion blur from camera movement, and non uniform backgrounds. Each of these problems would require a separate approach to resolve errors associated with it. Localized histogram equalization could improve recognition of markers with inconsistent lighting or shadows, a motion deblur filter could be used to alleviate motion blur, and better tuning of the preprocessing steps could reduce interference from background noise.

Overall, this model performs much better in reasonable real time applications. In my testing, simply using a webcam or phone camera, tracking was easily maintained across slow moving trackers of various sizes from different perspectives. I think the data I collected for training and validation was overly difficult to work with and didn’t accurately reflect the normal use cases for a classifier like this.