Dice Vision Project Report

CSE 5524
Autumn 2021

Ian Blake
Hunter Figgs
Robert Wetzler

Mr. Barker TH 11:10

Date of Submission: 12/02/2021

1. Introduction

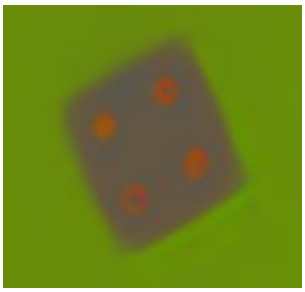
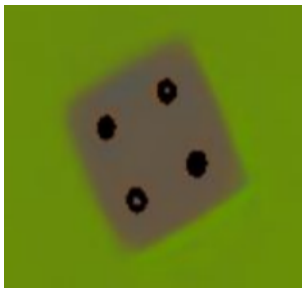
The goal of our project was to use computer vision techniques from the class to detect and read the dot-value of dice throughout a given video sequence. This project consisted of three main pieces: (1) data collection and pre-processing, (2) dot and motion detection, and (3) dot clustering via k-means. Each step was performed by a single group member and will be described in their respective section below.

2. Data Collection and Pre-processing

The initial data collection was critical for the pre-processing step to successfully produce image sequences that would be conducive to reading dice and their corresponding dot values. Since this project was open ended, the environment in which the dice existed and were manipulated could be controlled to allow for confident processing. The dice were placed in a box whose background was a mostly solid green texture, with some slight variations due to the construction paper used, which would require some attention. There was an overhead lamp, to provide consistent and even lighting to the scene. A camera was set up to record the dice and their green background roughly normal to the bottom of the box and the face of the dice when still. The box was shaken in order to agitate the dice and get them to tumble and land on new faces throughout the image sequence. The resulting image sequence was scaled down to 480p in order to reduce computation time while still retaining an adequate amount of quality for reading the dice.

Conventional background subtraction was of no use in this scenario because the variation in the green background translated and scaled as the box was shaken to move the dice. The chroma-key technique was therefore employed in order to isolate the pixels containing the dice. The first attempts at chroma-key were unsuccessful for two main reasons: (1) some of the variations in the paper, such as reflective spots, appeared extremely bright or reflected slightly different colors, which put them outside the range of the chrom-key and (2) the dice sometimes cast shadows, depending on their position relative to the overhead lamp, which again fell outside the range of the chroma-key. These two issues are displayed in the following figures.

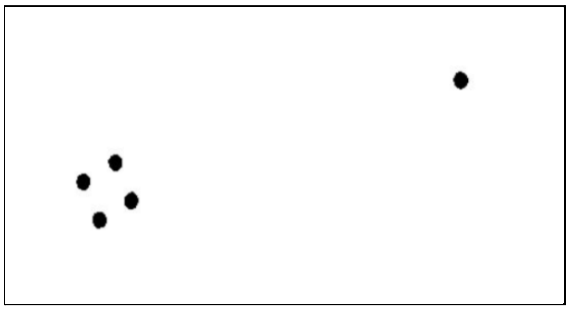
To solve this issue, two computer vision techniques were employed. The first was a mild Gaussian blur in an attempt to smooth over some of the harsh highlights. This technique did help some, by reducing the occurrence of highlight zones being missed by the chroma-key, although it did nothing to address the shadows. The second technique used was normalization of the luminance in each image in the image sequence, which successfully made the background appear a quite uniform shade of green, throughout the image sequence. One issue arose however, due to this normalization process: the darkest regions, such as the dots on the die would tend to be skewed toward one color, sometimes a shade of green – although often red, as seen below – which would cause them to be removed in the chroma key. This was fixed by forcing all pixels with a luminance under a certain threshold to stay completely black, through the chroma-key. The following images are the normalized luminance with and without this modification, as well as the result of the background subtraction.


The next technique employed was thresholding, in order to force values under a certain threshold value to be black (hopefully the dice dots) and force values over that threshold value to be white (hopefully the faces of the dice). This worked very well, however some artifacts were leftover in this process, such as the highlights in the middle of the dots reflecting the lamp light and bits and occasional disconnected pieces of the dice, especially when they were not perfectly normal to the camera (and thus revealing their sides to the camera). To combat this issue, small objects were removed under a certain threshold area (in number of pixels). This solved all but the last issue, which was when dot highlights or irregular pieces of the die were connected to the main face of the dice. This final issue was resolved with running a closing algorithm on the images, which is a combination of erosion and dilation. The results of thresholding, small-object-removal, and closing can be seen below.



The pre-processing was presumed to be completed at this point, since algorithms such as region segmentation, template-matching, and/or mean-shift tracking were planned on being utilized for dice and dot detection. However, it was realized that simple dot detection and k-means could be used to provide reasonable results. This method would be most effective if only the dots were present in the image, so with manipulation of the chroma-key process, the background was set to be the same white color as the dice face. This final alteration of the pre-processing can be seen below.



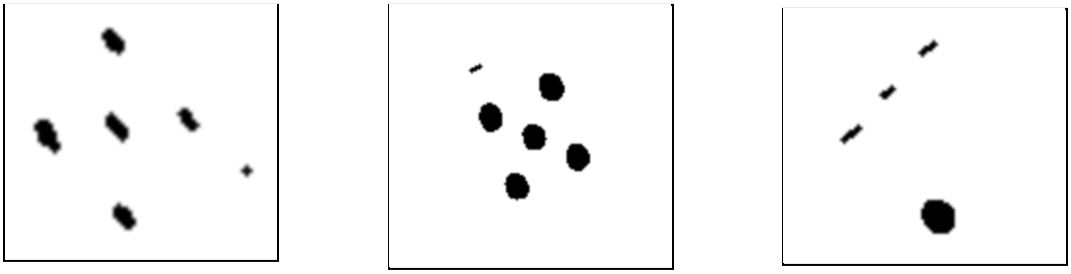
3. Dot and Motion Detection

Before attempting to count the number of dots on screen, the dice first must be determined to be still. Otherwise, if there is detected movement, the dice are likely rolling, causing an inaccurate reading as multiple faces may be visible at once. This motion detection was implemented using image differencing between the current and previous processed binary frames. The absolute value of the sum of the pixels of the difference image is then taken. This magnitude is then compared to a threshold and, if below that threshold, the dice are determined to be still enough to capture a reading. The threshold used was 150 which, given that each frame is binary and each dot center contains around 100 pixels, allowed for some small dice movement between frames. This threshold allowed for dice to be read as they are sliding and not completely still, but would ignore them upon any drastic movement caused by rolling. Note that this threshold is dependent upon dot size and would have to be adjusted if the camera's resolution or distance to the die is changed.

After the dice are determined to be still, the number of dots in the entire image are counted. The recursive connected components algorithm was used for this step. This algorithm functions by first iterating through each pixel in the binary frame, as provided by the previous processing step. If a zero (black) pixel is found, the region count is incremented and depth-first search is employed starting at that pixel. The depth-first search function sets the pixel to a label and appends its x and y coordinates to a provided list for that region, then recursively calls itself for any neighboring (top, bottom, left, right) pixels that are zero. After depth-first search is complete, every coordinate of the pixel from the region is included in the list that was passed to

the function. The average of these coordinates is taken to determine the center coordinate of the region. After the recursive connected components function completes, the center coordinate of every region is returned by the function.

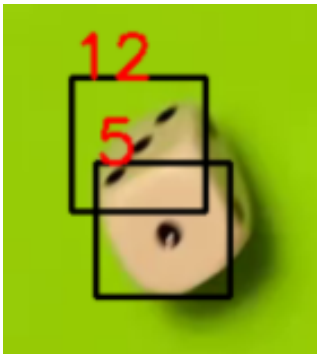
An issue that occurred using the connected components algorithm was that sometimes stray regions of black pixels would be included in the input image which weren't intended to be counted as dots, as seen in the example images below. These erroneous regions were typically caused by another face of a die being slightly visible. To avoid counting these regions, a minimum region size threshold was utilized. To count the number of pixels in each region, a variable was simply incremented during the depth-first search processes. After depth-first search completes, the pixel count is compared to the minimum threshold and, if it is smaller, the region is ignored. This threshold, similar to that of the image differencing, is dependent on expected dot size and would have to be adjusted for other input sources.



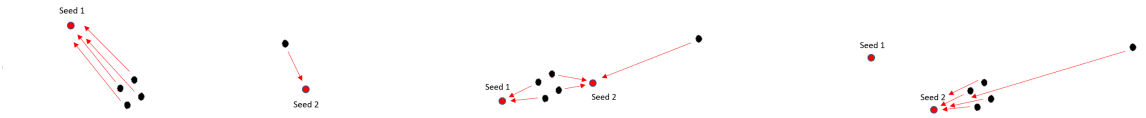
4. Dot Clustering via K-Means

Given the success of identifying the dots on the entire image, it was decided to use a K-means function to identify the two dice faces. A modified K-means function was developed to better fit our circumstances. First, two random seeds were generated on the image. The function added each dot to the corresponding seed group, before recalculating the mean, or "center" of the die faces. Performing this three times was sufficient to establish the die centers. These two die centers were then stored in a List, granted they met two viability checks. This entire process was repeated one-hundred times and then the most frequent centers were taken from the List using the mode() function. It was necessary to perform the calculation so many times because of the errors caused by random seeding. K-means functions are worthwhile, but oftentimes do not return results that match what humans would determine is correct.

The first viability check was to ensure that all dots on a given face were within fifty pixels of the die's center. This ensured that dots on completely opposite sides of the image weren't determined to be on the same die face. The second viability check was to ensure that the two centers were not within 100 pixels of each other. Otherwise there were instances where the two die faces were determined to be right on top of each other, on only one die. As seen below.



Three instances of random seeding can be seen below. The first, an ideal scenario where each batch of dots is added to their correct group on the first try. The second image represents a scenario where recalculating the mean, or centroid, of the group multiple times leads to the correct outcome. Finally, the third image would be caught by the first of the two viability checkers, for dots being too far from the center of the die face.



This modified K-means function only worked correctly when the dot detection algorithm counted the right number of dots on the entire image. As seen in the final video, frames where the die are still slightly in motion and showing several faces, or just blurry causing no dots to be seen, would cause errors in detecting the die face.

5. Summary and Contributions

Our group was able to successfully detect and read the dot-value of dice throughout the given video sequence. It is without a doubt that with additional time and effort, this project could very well have reasonable applications when it comes to casino games like “Shoot To Win Craps”, where a physical aspect of dice rolls and chance can be seen even through a computer interface. Hunter Figgs collected the necessary data and worked on pre-processing the image frames, Robert Wetzler worked on the dot and motion detection algorithms, and Ian Blake worked primarily on the dot clustering via K-means. There was partial overlap on several sections of the project, and everyone contributed to both the powerpoint and report.