Testing Dice Fairness Using the OpenCV Machine Vision Library

Jesse Reichler, reichler@dcomail.org

1 Abstract

A fair die is one which is equally likely to land on any side on any given roll. Automating the process of evaluating dice fairness involves three processes: First, mechanically and repeatedly rolling a die; second, using machine vision to identify which side is showing face up on the die; third, collecting statistics to assess the "fairness" of the die across many die rolls. This paper discusses all three processes but focuses mainly on the second process – using the OpenCV machine vision library to identify die faces. In particular, we describe an end-to-end process geared towards robust performance on a wide variety of dice types, with minimal user guidance. Our source code is open source and is publicly available at https://github.com/dcmouser/dicer.

2 Introduction

Our goal is to build a standalone die fairness analyzing machine which is usable on a wide range of dice shapes (d4, d6, d8, d12, d20, etc.) and styles, by non-technical users. While there are several existing successful projects describing the analysis of die fairness [], most are fairly restrictive in the class of dice they can analyze, and they share in common the fact that they require careful supervision by a skilled technical operator who can visually validate results while adjusting parameters in order to get satisfactory results on a given die.

Perhaps the most significant challenge we have faced in this project is trying to get the dice face recognition robust and accurate enough that it does not requiring excessive hand-tuning parameters and human interaction for different dice. It is easy enough to hand-tune the system to work on a specific die – performing visual inspection of classification labels and tweaking settings to eliminate misclassifications, but it is quite another matter to find procedures that are reliable enough on a wide variety of dice to be usable with minimal user interaction, and by users who have no technical understanding of the process.

3 Hardware

Although this paper is primarily concerned with the software behind die face recognition and clustering, this section will discuss briefly how hardware design tradeoffs affect the task of testing dice fairness.

3.1 Should we use a top-down camera or a bottom-up camera?

A fundamental aspect of the hardware design is whether the camera should be looking down at the die resting on a surface, or whether the camera should be looking up through glass at the bottom face of a die. Both options have advantages and disadvantages.

A top-down camera has the advantage of a controlled background surface, which we can make a solid color. It also allows an observer to see the same "topmost" die face that the camera sees, making it easier for an observer to verify correct classification of images.

A bottom-up camera has the advantage of working well with a rotating tumbler container, described below.

3.2 Randomization method

How do we sufficiently randomize (roll) the physical die? We have a few options.

One would be to encase the die in a container and repeatedly (and at different locations) tap the bottom of a platform on which the die sits.

Another way would be to spin the bottom platform of the container sufficiently to cause the die to tumble like a ball bouncing on a roulette wheel. Ridges along the bottom or side may aid in randomization, though it may also make it more difficult to predict the background of the camera view.

A third way would be to use a cylindrical (or hourglass) like container with a rotating point in its center, spinning end over end to tumble the die from one half of the container to the other. We note that such a container would require a bottom-up camera.

Another complication that could occur when one is automating thousands of die rolls, is that the die rolling procedure could in fact be altering the geometry of the die, much like a rock tumbler smooths the corners of a rock as it tumbles. Looking at die rolling statistics over time for a single die might give some insight into whether this is in fact happening.

3.3 Centering die in camera view

Our visual processing routines require the die to be relatively near the center of the camera view. This is because the farther the die is from the center of the view, the more pronounced are the perspective effects on the image, and the more likely we are to have neighboring faces visible or invisible based on the die pose.

The basic approach to keeping the die centered under (or above) the camera is to use an inverted cone or funnel shape that forces the final die position to be within a target narrow area.

3.4 Lighting and elimination of shadows

Any shadows present in the image can cause difficulties for visual processing. We avoid shadows by using a ring of LED lights that ring the container and provide predictable lighting around the die and eliminate shadows.

4 Machine Vision for Die Face Recognition

Within the wide spectrum of machine vision tasks, recognizing die faces is a relatively simple one. We have the advantage of a highly controlled environment with good, controlled lighting, and a limited range of camera angles. However, the nature of the task does present us with a few unusual challenges, which we will discuss in this section.

4.1 Avoiding the need for training

One of the things we'd like from our die fairness analyzer is the ability to analyze dice which used custom graphics on the die faces that we have never seen before – without needing to be specifically trained on those new face designs.

4.2 What do we really need to be able to recognize?

One unusual aspect of our machine vision task is that we don't actually need to be able to *recognize* specific objects. Instead, we simply need to be able to divide die rolls into groups according to the unique die faces. Essentially this is an unsupervised clustering problem. If a die has six unique faces, then we want to label each die roll as falling into one of six groups – such that members of a group are more similar to each other than they are to any other group.

4.3 Minimizing the need for user input

Another goal we have is to minimize the amount of information we need to gather from the user prior to analyzing a die. Ideally we'd like to be able to analyze a new die without asking the user for any information at all – though we will see that this may not be feasible in all cases. Let's look at some of the information that is needed to analyze a die, and to what extent we can discover this information autonomously.

4.4 How many unique die faces?

One thing we need to be able to determine when analyzing die fairness is the number of unique die faces. Note that this is sometimes different from determining the total number of faces on a die, as some dice have the same symbol on different faces. Since we have no practical way to physically manipulate a die in order to explicitly count the number of die faces (unique or otherwise), we need another way of determining the number of unique die faces on a die. The easiest way to do this is to ask the user to explicitly specify how many unique faces a die has. In practice, this is the approach we have taken. However, there are alternatives. From visual inspection the software can try to identify the geometry of the die, and thus the total number of faces on it. The distance metric used for clustering may help us decide the most likely number of clusters, and if we assume that the die is relatively fair (that is that each die face is approximately equally likely to come up), we may be able to make an educated guess as to the number of unique faces on the die. For example, after visually identifying a six-sided square die, we know that there can be at most six unique faces, and possibly as few as two. After a large number of rolls, if the die is reasonably fair, we would expect to see certain frequency distributions based on the number of unique faces, and might choose the number of clusters that best conforms to one of these distributions. Additionally, the distance metric we use to cluster die faces may be able to help us figure out when we have found the right number of clusters (unique die faces).

4.5 Isolating the die from the background

We are fortunate to have a tightly controlled environment where isolating the die from the background of the camera image is relatively straightforward. We perform a calibration step of recording an image of the camera facing an empty background with no die present. This base background image is then *subtracted* (in LAB color space, using the OpenCv absdiff function [], which we experimentally found works best for background removal) from live images when a die is present.

Subtracting the background works quite in a controlled environment. It is susceptible to interference from shadows cast by the die, so we need to minimize the occurrence of shadows with good lighting surrounding the die enclosure, and it can be confused when the die is extremely close in color to the background color, so we try to use an unusual background enclosure color. A patterned background in and of itself is not a problem, however in practice it can be problematic if the camera position relative to it shifts – and for this reason we use only solid backgrounds.

We perform a few simple operations on the subtracted image to identify the contiguous foreground region of the die. We run a simple OpenCV noise reduction pass (parameters do not seem particularly important) on a grayscale version of the subtracted image to eliminate salt and pepper noise from the subtracted image, and then run the Otsu thresholding algorithm [] to generate a 0 or 1 label for each pixel reflecting our belief that it is part of the foreground image.

On our binary foreground image map we then identify all contours (OpenCV findContours function []), and select the largest such contour as the likely foreground image of the die. We then merge any nearby contours, and finally build a convex hull containing all such merged contours. This gives us a very clean convex hull mask identifying the foreground die face, and ignoring any spurious areas on the image. A simple sanity-check is performed to flag the case where an object is visible that is highly unlikely to be a die – this allows us to ignore frames where unexpected objects (such as a human hand) are briefly in the camera frame.

4.6 Isolating foreground graphic symbols from die background color

Another area where we occasionally need assistance from a user is in isolating the foreground symbols on die faces from the die background pattern. Note that we are not yet talking about isolating the *topmost* label, but simply separating all foreground labels from background pattern/color. In practice, most dice have a solid color background, and a solid contrasting color foreground graphic. For such dice, we can apply a simple thresholding operation to extract the foreground graphic. However, some dice have a multi-color complicated background, that can prove challenging to extract the foreground color from, and automatic thresholding will fail to isolate a foreground graphic.

Fortunately, because of the unique nature of the task, it is not always necessary for us to be able to isolate the foreground symbol. That's because our clustering algorithm is just searching for the closest binary image match, and can be used both on a complete image of the die or the isolated foreground symbol. If the background pattern of the die is unique enough on each die face, it may not be important for us to be able to isolate the foreground graphic.

In cases where we do want to be able to isolate the foreground graphic from a noisy background, there are a few options. Assuming that the foreground label is a single color on such dice (which has been our experience), the isolation task comes down to a matter of identifying the foreground graphic color and then isolating pixels near that color. We can then cluster the pixel colors into a small number of clusters (color quantization), and then either ask the user to click on a pixel that is part of the foreground graphic, or attempt to guess which color cluster corresponds to the foreground symbol by trying to identify which color cluster is most likely to be the foreground symbol. In practice we use the first approach, letting a user click on a foreground graphic pixel for such dice where automatic thresholding fails, and then performing color quantization (in LAB color space) to identify pixels with "nearby" colors. If one wanted to try to automatically choose the most likely color cluster belonging to a foreground label,

something like the Stroke Width Transform might be suitable for identifying which color clusters are most likely to represent a foreground symbol (typically a text label or engraved line drawing) – or perhaps an even simpler approach of looking for the color cluster with the smallest number of contours.

4.7 Isolating the top die face and ignoring neighboring faces

One challenge faced by the die analyzer is that we are interested in identifying the face of the die that is facing upward, while ignoring neighboring faces that may be visible. This problem is ameliorated when the die is centered directly below (or above) the camera. The further the die from the camera the harder it is to distinguish the front face from neighboring faces.

If we have a comprehensive model of the 3d die geometry, and could identify the 3d pose of the die, we could use this, and the position of the die relative to the camera, to very precisely isolate the image of the top die face from the neighboring faces. Fortunately, we have found that we do not need to perform such a complicated 3d pose recovery. Instead, if the hardware setup can keep the die reasonably centered under the camera, we can use a very basic understanding of the die geometry to isolate our foreground extraction to a region near the center of the die and use some fairly simple grouping contour grouping rules (described below) to exclude the foreground graphics from neighboring faces.

As in the case of isolating foreground graphics from background color, it may not always be strictly necessary to isolate the topmost die face from neighbors – we can, if needed, process the complete configuration of visible die face symbols (the topmost and any visible neighbors) as a single image and use that for clustering. This procedure may work if the die is reasonably centered in the camera view, so that different rotations of the die will leave the same neighboring faces in view and at roughly the same perspectives. In practice we have found that isolating the top die face symbol makes recognition and clustering significantly easier and less prone to mistakes, though it does depend on some heuristics to exclude neighboring face symbols. The problem with including neighboring face symbols is that they tend to vary more as the relative die pose and angle change, and can contribute significantly to inaccurate similarity measures. It may be that this could be ameliorated with a modified similarity metric that weighted center matches more highly and performed a more sophisticated feature matching or alignment procedure.

One strategy we think is worth pursuing in future work is using multiple similarity scoring functions and image clustering on different aspects of the images (e.g. simultaneously clustering based on full die images and on extracted foreground faces), and then using a weighted vote to classify images and/or flag images whose classification confidence is low (if votes disagree).

In practice we use a heuristic procedure to isolate the most-likely graphic symbol on the top face of the die. We start by finding the center of the convex hull of the die, and then find all contours in the black and white image of the die (we normalize the black and white image so that the background is always black). We then find the closest contour to the center point, and treat this as our starting top face graphic symbol. We then examine the remaining contours within the convex hull of the die, and merge those that are both "near enough" to our currently merged contour group, and which do not have points that fall outside of an exclusion mask determined by the shape of the die. We use the concept of a die "profile", which is largely based on the shape of the die, to determine the shape of the exclusion mask and the inter-contour distance threshold for merging (relative to die size). In practice this small set of parameters based on die shape have proven robust to a wide variety of dice.

Some dice, like standard D6 dice, are remarkably easy to identify, because neighboring die faces are barely visible as long as the die is relatively centered under the camera, and so can essentially consider all foreground symbols to be part of the top-most face.

We note that if one was only interested in testing standard d6 dice whose faces consist of from one to six circular "pips", one could use a dedicated (and much faster) function to count pips and bypass the normal process we discuss for clustering die images by similarity. Similarly, one could imagine using a dedicated, offline-trained number-recognition function to identify the numeric values found on many dice. The use of dice-type-specific recognition functions is an area that warrants future investigation, and it may be that such algorithms should be added for optimal performance (and/or for validating unsupervised clustering). However, in our current work we focus on the more generic and flexible process of clustering images based on a custom similarity metric.

Some dice, like d12 and d20 dice, are cleanly and reliably isolated with our heuristic, since their sphere-like shape makes it easy to mask out everything but the top-most face graphic.

The most difficult (and sensitive to camera angle) dice are those with irregular shapes (d4, d8, d10) where it can be difficult to identify the area circumscribing the top-most die face without a specific understanding of the 3d pose of the die. An area of future work would be trying to recover the 3d pose of the die using a model of its 3d shape, so that the topmost die face could be specifically isolated.

4.8 Group die face images with Agglomerative Clustering

There are many algorithms for clustering exemplars. For die fairness testing, we are interested in clustering two-dimensional images into a small, *fixed* number of clusters (the number of unique die faces), based on a *custom* similarity metric.

Given those requirements, the agglomerative clustering algorithm [] is a natural choice. Agglomerative clustering allows the use of an arbitrary distance metric, and incrementally merges closest exemplars into clusters until the desired cluster count is reached. To perform agglomerative clustering we use the scikit-learn machine learning library implementation [], with a similarity metric described later. Agglomerative clustering, unlike k-means or other similar clustering methods, does not take into consideration the density of exemplars in certain regions, and so is suitable for a situation like ours where we are simply interested in finding clusters that maximizes average intra-cluster similarity, without making any assumptions about the frequencies of the exemplar classes used during training.

Agglomerative clustering requires that we provide a set of exemplars, with a similarity score between every possible pair of exemplars. Computing these similarity scores between exemplars is cpu intensive because of the nature of the similarity score and the exponential nature of the pairwise combinations. Fortunately, the agglomerative cluster discovery algorithm does not have to be performed often nor with a large set of exemplars.

In ideal circumstances, we need only perform the cluster generation once, after we have collected **at least** one exemplar belonging to each die face. After the clusters have been formed, new images can be categorized by finding the closest match against one of the exemplars that have already been assigned to a cluster during training. This approach is feasible only because our highly controlled environment and our

similarity metric are such that we only need one exemplar image of each die face in order to recognize that die roll in the future.

There is a small risk in our case, in building clusters based on a small subset of exemplars (die face images). There is the possibility that the training exemplars might not include one image of each die face. The more die rolls we include in the training procedure, the less likely this is to happen, and straightforward statistics can tell us how confident we are that we will see every die face at least once, given the number of sides of the die and the number of rolls.

One way to side step this problem would be to require the user to physically manipulate a die (or select unique images from a large set) to show us each die face in turn – which would allow us to build the clusters most efficiently. However, this goes against our desire to minimize the need for user interaction and guidance.

Thankfully we can mitigate this risk of clustering with an incomplete set of exemplars by using an automated process that involves the similarity scores. The similarity metric tells us how similar any die face image is to another. The normal classification process works by computing the similarity between a new image and a candidate exemplar in each cluster, and classifying the new image as belonging to the cluster of the exemplar it is most similar to. But by looking at the actual similarity values, we can detect a case where a new image is unusually dissimilar (relative to other exemplars in the cluster) to the closest exemplar. Such a score is a warning flag that the new image may not be a good match to our known clusters.

When we encounter an image that is unusually dissimilar to our clusters, we can either flag it for manual human examination at a later time, or trigger a new round of agglomerative clustering including the new image. If a new round of agglomerative clustering produces different cluster results, we will need to re-classify any image we have previously classified based on previous clusters. Because of this need, we store all images of a die during testing, so that we can perform re-clustering and re-classification as needed.

In general, when we encounter a die image that is unusually dissimilar from existing clusters, we need to consider three possibilities: Firstly, that it is a match for the closest cluster, and the similarity score is simply an imperfect measure of similarity for whatever reason (rotation failed to align perfectly, etc.); Secondly, that the foreground image was imperfectly extracted, and includes extraneous pixels or incorrectly excludes pixels – thus resulting in a failure to assign the correct cluster; Thirdly, that the image represents a die face not used in the discovery of clusters. There is no exact solution to deciding between these possibilities, but we can view it as a meta-optimization procedure, and ask whether re-creating clusters using this outlier reduces or increases the average intra-cluster similarities (and maximizing inter-cluster similarities) for the images seen so far. Such an optimization would seem like a reasonable way to choose which images to use for the clustering process, absent human guidance. Regardless, we can still rank all images based on their similarity to a cluster prototype, and allow the user to review those that seem like outliers.

4.9 A similarity metric for die face images

At the highest level, our goal is to classify each image of a rolled die according to which symbol is on the top face of the die. If we were willing to go through a supervised training process, we could learn (using

neural networks, support vector machines, etc.) an arbitrary function mapping the die image with the correct class (die face symbol) as labeled by a teacher. However, since we wish to perform this task without the need for labeled training data, we instead use similarity-based clustering to assign each image to a cluster of similar images, with a similarity metric that we hope will place images of the same topmost die face symbol into the same cluster group. The key component in such a process is the similarity function.

The main challenge for recognizing die faces is rotation invariance – recognizing the same die face at any orientation.

A great deal of research has been done on efficient algorithms which can find the occurrence of a non-trivial object image inside a larger busy image despite changes in size and orientation, producing very successful algorithms like SURF and SIFT []. These algorithms typically work by detecting a list of features that are often preserved even when an object is scaled or rotated in an image. When larger image is found to contain a large number of features found in the object one is searching for, there is a candidate match. A simple count of matching features may be sufficient to indicate a match. If not, confirmation of a high-quality match may be attempted by trying to perform transformations of scale and rotation to align the matching feature sets; if a simple set of transformations can bring the candidate image features into register with features found on the larger image, we have a likely match.

Although a feature-detector based matching algorithm has been used in previous die-face clustering work [], we found such algorithms unreliable for the task. Our extracted foreground graphics were too sparse in features and too similar to one another to be reliably matched using SURF or SIFT.

As an example of the kind of problem faced by such feature detector matching algorithms on this task, consider that on some dice, the difference between a 6 and a 9 is the relative location of a tiny dot. In some cases the difference between two die faces is extremely subtle.

So the unusual characteristics of the visual recognition task we face is that image of a given die face are quite stable and regular, differing in little other respect than rotation and noise (which may include neighboring face symbols). Our similarity metric needs to be highly sensitive to even small variations (a dot differentiating between a 6 and a 9), while remaining rotation invariant.

The algorithm we use to measure similarity between images is a form of "template matching."[] Given two images we search for the rotation angle that brings the two images into best alignment. In order to make the comparison more robust, we perform a series of simple image manipulations, such as normalizing the size of the images, and dilating them [] and then calculate the average grayscale pixel value difference across the entire image as a measure of dissimilarity. The normalization and dilation reduce the impact of any small differences in alignment and size. To speed up the alignment process we step through angles at larger increments and the fine tune the best matching angle by searching for rotation angles at finer increments near the first-stage best angle.

We can combine our search for best rotation angle with a search for small translation adjustments, and even scale adjustments – though in practice we have found that the scale normalization makes translation and scale transformations superfluous. The general process is simple – we actively search for the best

transformation that brings two images into alignment, and take as our measure of similarity the bitwise similarity between our best alignment of the two images.

One potential area for future work is in the use of more sophisticated and efficient algorithms for searching for alignment transformation parameters [].

5 Statistical Analysis of Dice Roll Fairness

What can we say about the fairness of a die that has been analyzed? How many times do we have to roll a die in order to say something meaningful about its fairness? This section addresses these issues.

5.1 First things first: What is the source of the bias?

As a preliminary note, we should acknowledge that when we are testing a die, what we are actually evaluating is in fact the fairness of a die combined with the fairness of the rolling mechanisms we are use. When we talk about a fair die we are implicitly assuming a "fair" rolling procedure which lends no bias to the rolled die face. We can define a "fair" die rolling process as one which in which a fair rolled die is equally likely to land on each face.

Intuitively, we expect that if we assess a die as fair (unbiased), we can conclude that both the die and the die rolling process are fair. However, without some independent confirmation that the die rolling process is fair, it may be difficult for us to determine the source of bias (whether in the die or the die rolling process, or some combination) found when evaluating a die.

For the remainder of this discussion we will side-step this program by assuming that the die rolling process has been "validated" as fair – simply by establishing that some dice (of each shape) evaluate as fair. The alternative explanation would be that a biased dice rolling process is exactly canceling out a biased die, resulting in results that look unbiased. Ideally, however, one would validate the fairness of the dice rolling process by comparing die fairness results using multiple dice rolling processes, in order to assure ourselves that the die rolling process is not biasing die rolls.

5.2 How to test for fairness?

A fair die is one which is equally likely to land on any given face, for any given roll. On first intuition, a simple test for the fairness of a die would simply be to roll it many times and count how often it comes up on each side.

But is counting the frequency of landing on each face sufficient to assess fairness? No. In addition to expecting each face to come up an approximately equal number of times, we also expect there to be no influence of prior rolls on future rolls.

For dice rolling – especially using a dice rolling machine – the concern we might have is that the previous face will have an influence on the subsequent face. For example, imagine you rolled a standard D6 die and each time it landed on the face with one more pip than the previous roll, recycling back to the one-pip face after the 6-pip face. Looking only at a histogram of die faces would conclude that the die is fair – with each face coming up an equal number of times. So if we are concerned about the possibility of previous rolls influencing future rolls, we need to test for temporal correlations between rolls.

Questions we need to answer: How many times to do we have to roll a given die (with a specific number of faces) until we are "confident" that the die is fair? Is there a reasonable single statistic that can score the fairness of a die in a meaningful way (as opposed to a binary judgement of fair vs unfair)? What statistic can we use to assess fairness in terms of temporal correlations between rolls?

We separate these questions into two groups: First, statistical questions concerning the frequency of rolling each die face; second, statistical questions concerning temporal correlations between rolls.

5.3 Pearson's Chi-squared test

If we are interested in whether each face of a die is equally likely to be rolled, **Pearson's Chi-squared** test is a natural tool to use. The Chi-squared test is used to assess how confident we are that a set of observations did ***not*** come from a hypothetical frequency distribution we provide. More specifically, we are asking how confident we can be that the die roll observations were ***not*** generated by a die where each die face was equally likely to be rolled.

The result of a Chi-squared test of a set of die rolls will take the form "Given these N rolls, we can conclude that the die is unfair with X% confidence." Or put another way, (100-X)% of the time, a fair die would be expected to produce such these same results.

For example, if we do a test where we roll a D6 fifty times, the Chi-squared test might tell us that with 95% confidence die is ***unfair***, which is another way of saying that we would expect a ***fair*** die to produce such results 5% of the time. In the long run, rolling an unfair die more times will lead to higher confidence levels that the die is in fact unfair (or lower confidences if the die is in fact fair).

Unfortunately, while the Chi-squared test can tell us when we should be highly confident that a die is unfair – it does not provide a similar confidence statistic that the die is in fact fair. That is, if we roll a die a trillion times and each face comes up an identical number of times, the Chi-squared test simple tells us that we should have no confidence that the die is unfair. It would seem intuitive that in such a case, the more we roll such a die the more confident we should be that it is fair – but the Chi-squared test really only helps us positively identify a biased die.

This can present a problem, because if a Chi-squared test does not result in high-confidence of an unfair die, we do not have any systematic way of deciding whether we have rolled it a sufficient number of times to conclude that it is fair. A failure to conclude that the die is biased is not the same as concluding that the die is fair – it may also mean that we have not generated enough samples to detect the unfairness of the die.

5.4 Histogram Confidence Intervals

One way to assess the fairness of a die roll is look at confidence intervals for a histogram of die faces. For a fair die, we expect each face to come up approximately the same number of times. But how many times do we need to roll a die to be *confident* that we have an unfair (or fair) die?

Like the Chi-squared test, confidence intervals allow us to say how unlikely it is that the results of the die rolls would not have come from a fair die. So for any given set of die roll results, we can find the confidence level for which the results are outside the confidence intervals expected for a fair die.

5.5 A More Subtle Way to Talk about Die Fairness

We've described two ways to test for die fairness – though it would be more accurate to say that we've described two ways find unfair dice. While chi-squared nor histogram confidence intervals help us identify a clearly unfair die, they do not *directly* help us make an affirmative statement about a die being fair, or tell us if we have rolled sufficiently to be confident that a die is fair.

It may seem counter-intuitive that we can have a test to identify an unfair die, but that these same tests cannot tell us when a die is fair? Why can't we just say that if the test fails to show that a die is unfair, we are justified in saying that the die is fair?

A clue to understanding this asymmetry is to take a more nuanced view of what it means to be an *unfair* die. A fair die is one in which every face is equally likely to come up on any roll, and an unfair die is one where this is not the case – where there is *some* bias. But now consider two D6 dice. For the first die, one side is 10 times more likely to be rolled than any other face. For the second die, the faces are all equally likely except that one side is ever so slightly more likely to come up – perhaps one time out of a billion rolls. By our definition, *both* dice are *unfair*. But now we can see that talking about the *degree* of unfairness might be a useful thing.

With an appreciation that different dice may be more or less unfair (biased), we can better interpret the unfairness tests described above. The more unfair a die, the *faster* it will reveal itself to a chi-squared or histogram confidence interval test. So we should think of these tests as telling us: "This die is sufficiently biased that given this set of rolls, we can with a certain confidence that it is not fair." The smaller the bias, the more rolls we will need to tease out its bias.

If we try to use our tests to make an all-or-nothing judgement about die fairness, we never get to a point where we can be confident that a die is fair – all we can ever say is that we haven't collected enough data to confidently say that we have detected a bias.

5.6 Making Affirmative Statements about Die Fairness

So, the solution to making affirmative statements about die fairness – and indeed about making more practically useful statements about die unfairness, is to work with a more nuanced definition of die fairness.

Let's return to our two dice from the example above. Our first D6 die is ten times more likely to land on one side than another. Our second die has is fair except that it will land on one face ever so slightly more often – one time out of a billion rolls. Intuitively the first die is more biased than the second. And for practical purposes, the second die is as close to being "fair" as most humans could hope for.

The trick to operationalizing this concept of degree-of-bias and leveraging our statistical tests, is to ask about the *sensitivity* of the results. Intuitively, we will ask how many more unusual rolls would it take to turn a die from one in which we do not have enough evidence to declare it biased, to one in which we do have sufficient evidence to confidently declare it as biased.

Let's consider the *technically* "unfair" D6 die from our example above, where a single face is favored one time in a billion rolls. After a thousand rolls our standard application of chi-squared and histogram confidence intervals still have nothing to say other than that we cannot be confident that the die is unfair

(or put another way, any bias in the die is small enough that it has not yet been detected). But now let's ask: To what extent would these results have to be different in order for us to be confident that this is an unfair die? The extent to which the results would have to be *artificially* changed before we would confidently declare it biased seems a reasonable measure of our confidence in the die's fairness.

We might quantify that in a few different ways. One way might be to ask how many (or what percentage) of the die rolls would we have to change to favor a given die face before the results *would* lead to a confident conclusion that we have an unfair die? Another way might be to ask how many more rolls we would have to make – assuming the future rolls came up in the exact same proportions as our rolls so far – before we could be confident that die was biased?

Using such reasoning we can say something of practical value regarding the affirmative appearance of a fair die. We can make statements of the following kind: "After testing this die for a thousand rolls, if future die rolls matched the frequencies of past die rolls, it would require another hundred million die rolls before this die would appear as unfair with 95% confidence." Such statements can be of practical value in helping us understand when we have tested a die sufficiently, and can assume that, for all human intents and purposes, the die is *fair enough* for human use.

5.7 Assessing temporal correlations between rolls

While in general temporal correlations could stretch over many die rolls, the physical nature of the die rolling process gives us confidence that the only temporal correlations possible are between previous and subsequent rolls. The technical term for such processes is a Markov process. That is, we need look back no further than the previous die face when looking for correlations with the current die face.

With that assumption, it's straightforward (although possibly very time consuming) for us to check for the possibility of such temporal correlation bias, in exactly the same way we check for biases in single die rolls, by looking at the frequency of *pairs* of die rolls (e.g. How often does a roll of a 2 follow a roll of a 3?) If we are confident that all *pairs* of die rolls are equally likely, then we can be reasonably satisfied that the die is fair.

One difficulty with this test for temporal correlations is the large numbers of rolls that is be required to conclude with high confidence that a die is unbiased in terms of temporal correlations, especially for dice with many sides.

As a practical matter then, one might ask how likely it is that a die which evaluates as fair (with high statistical confidence), will nevertheless prove to have temporal correlations between die faces. Intuitively, it seems unlikely that physical imbalances in a die (or biases in the dice rolling process) that would lead to temporal correlations in die rolls would operate without also generating (even more) obvious biases in die face frequencies. One conceivable scenario might be a die (or a die rolling process) that had a tendency to always flip the die to the opposite (or same) side. If one restricted concerns to such biases, it should be possible to explicitly test for such biased patterns more efficiently, by explicitly looking for unlikely "streaks" in dice rolls (sequences of rolls), rather than looking at a histogram of large volume of die roll pairs.

6 Conclusion

In this paper we have described an end-to-end process for analyzing the fairness of a wide variety of dice, with minimal user interaction. The open source software we have written is based on the OpenCV machine vision library, and has been made publicly available here: [].

7 Bibliography

https://en.wikipedia.org/wiki/Otsu%27s_method

https://github.com/dcmouser/dicer

http://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

http://timothyweber.org/node/255

http://math.stackexchange.com/questions/57624/how-many-rolls-do-i-need-to-determine-if-my-dice-are-fair

http://stats.stackexchange.com/questions/14301/designing-a-test-for-a-psychic-who-says-he-can-influence-dice-rolls/14302#14302

http://forums.gamesquad.com/showthread.php?101532-How-to-Test-Your-Balanced-Dice-Are-they-Fair

http://science.slashdot.org/story/15/12/01/1715253/experimental-study-of-29-polyhedral-dice-using-rolling-machine-opencv-analysis

http://www.markfickett.com/stuff/artPage.php?id=389

https://www.dropbox.com/s/buyg5swn0o0und2/SnakeEyesFinalPres.pptx?dl=0

http://deltasdnd.blogspot.com/2011/10/testing-balanced-dice-power.html

https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test

https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

https://en.wikipedia.org/wiki/Speeded_up_robust_features

http://arstechnica.co.uk/the-multiverse/2016/08/how-fair-is-your-d20/