Arrowsmith: Automatic Archery Scorer

Chanh Nguyen and Irving Lin
Department of Computer Science, Stanford University

ABSTRACT

We present a method for automatically determining the score of a round of arrows lodged in an archery target face. That is, given an image consisting of a complete target face, and given a set of arrows that have struck within the target face, we generate a score for each arrow with regard to the two circles it is between. To do this, we present a multi-step process to determine the location and shape of the set of concentric and evenly distributed circles (that may be distorted by perspective) representing the target face, locate and orient the each arrow, and find the pinpoint location of where the arrowheads pierce the target face. We are able to handle images of a target face at arbitrary angles by automatically rectifying the image so that the target becomes circular. We test the performance of our system on a set of real images taken from mobile phones and find that it performs reasonably well.

Categories and Subject Descriptors
I.4.8 [**Image Processing**]: Scene Analysis – *Object Recognition*

General Terms
Planar Rectification, Object Recognition, Detection

Keywords
Archery, Scoring, Arrows, Targets, Arrowhead

1. INTRODUCTION

Archery is a growing sport around the world, with competitors from the junior level to the collegiate and Olympic levels. During a competition, archers line up at a measured distance from a target and attempt to fire arrows into the target center. The target consists of 10 nested concentric circles, with the innermost one worth 10 points and the outermost one worth 1 point. One of the most time consuming tasks of archery training and competition is manually determining a score for each arrow on a target. In a typical competition, archers shoot for only 4 minutes before having to walk to the target and determine the score - a process that takes up to 5 minutes, meaning the scoring time can consume more than half the competition! Since competitions usually last 2 to 4 days, a computer-assisted scoring mechanism can save quite a significant amount of time. Finally, many archers don't keep score during training because it is too much of a hassle, although keeping score is one of the best ways to track progress.

2. DATA AND MODEL

Our project uses 147 images taken of standard 10 concentric circle archery targets without a restriction of the number of protruding arrows, and taken without concern for scale, perspective, or rotation.

The algorithm should be relatively noise, exposure, and perspective invariant. In particular, the method should handle cases where the image of the target is captured from any reasonable angle. However, we began developing our system under some simplifying assumptions: 1) the target circle scoring zones are perfect circles without distortion, perspective shifts, and occlusions (other than the arrows), and are fully included in the image; 2) the arrows are well-defined and have no shadows, and 3) the image does not contain motion blur or camera shake (that is, it's sharp).

In the early stages, we did most of our testing on the top image in Figure 1. We fabricated this image in order to provide a controlled and simple test bed for our implementation ideas and to determine which were most

effective. After gaining domain specific knowledge with the simple case, we proceeded to further enhance and develop our algorithm on the real world images (Figure 2).

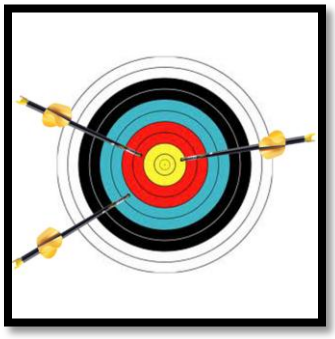


Figure 1: Contrived Image

Figure 2: Real-world image

3. PROCESS

Our current algorithm has two distinct stages, the second of which relies heavily upon the first. The first task is rectifying the target face and finding the circles. We then analyze the arrows afterwards. The entire process is shown in Figure 3.

3.1 Image Processing

The images we analyze will most likely be taken from mobile devices, and thus, the quality is always a concern. Images will be of varying exposure, contain noise, have distortion and perspective shifts, and contain differing sized target faces that are not necessarily centered. Also, one of the first things we noticed was that for our purposes, pixel color values and textures were more likely to be detrimental than useful to the detection problem, and that all we really needed were the edges. Thus, we tried several variations of the Canny edge detector method seen in Figure 4 a, b, and c, and found that Canny with a simple Gaussian blur worked best, as we'll show later on in Section 4.3 on Arrowhead detection.

3.2 Rectification

Rather than detecting ellipses for planar rectification, we take advantage of the fact that target faces are printed on a square sheet of paper. We detect the four corners of the sheet of paper to calculate a homography matrix which is used to rectify the image.

We use a Harris corner detector, which is able to detect the corners of the paper given a low threshold, but that also means it detects many other unrelated corners in the image (Figure 5b). In order to single-out the corners of the paper, we rely on medium-to-long lines in the image, found using a Hough transform. Since there are many lines present in any image, we pick out a set of 4 lines that roughly intersect at 4 or more Harris corners.

However, since the arrows themselves create long lines, they

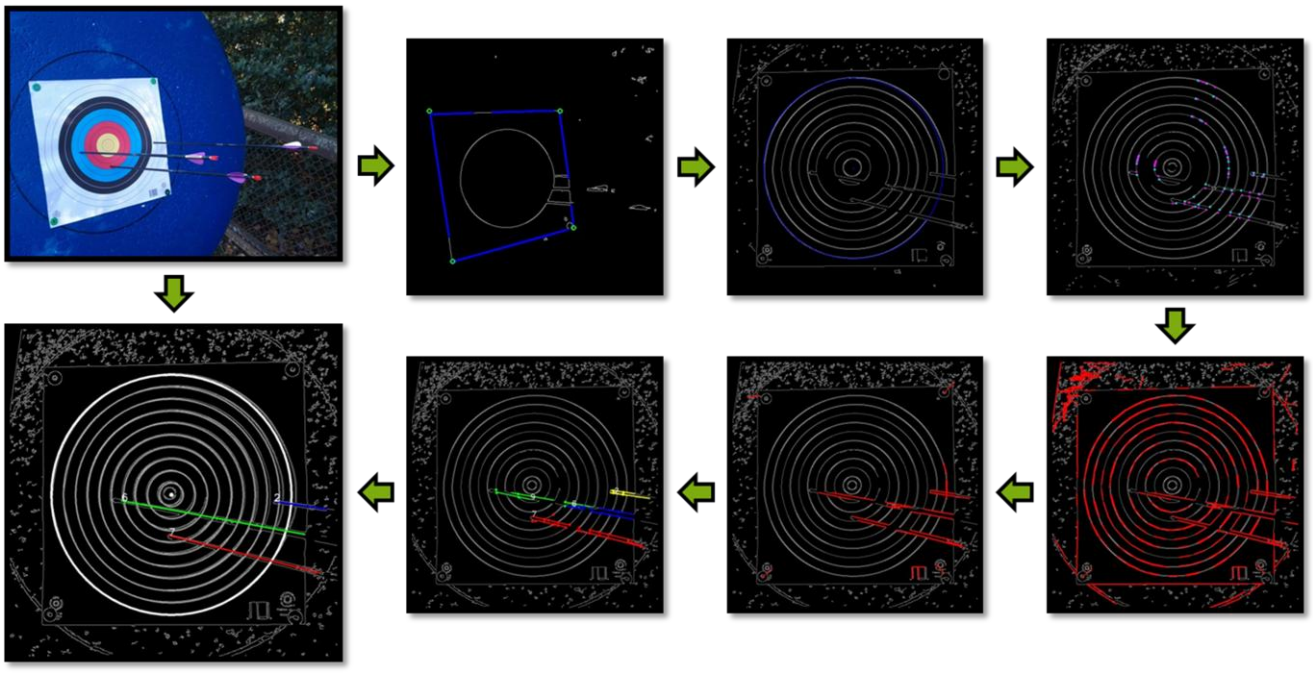


Figure 3: The entire process of scoring arrows, described in Section 3.

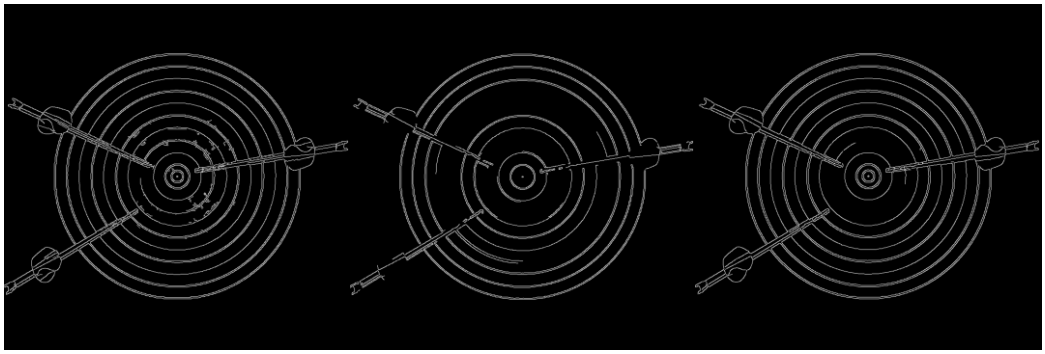


Figure 4: (a) Canny without blurring, (b) Canny with DOG, and (c) Canny with Gaussian

could be misconstrued as a paper border. Thus, we restrict the 4 lines such that the longest line is not much larger than the shortest line, forcing the shape to be somewhat of a skewed square. We also require that none of the line segments are part of the same line.

Our final tuning has optimized the threshold for line detection and Harris corners such that we are able to detect the piece of paper on 22/23 Medium difficulty samples, and 24/50 Hard samples. We did not use rectification for the Easy set because those images were already shot at near-direct angles, meaning rectification would not greatly improve the accuracy and incurs a risk of mis-transforming the whole image.

The main challenge presented by the Hard samples was the extreme angle of the paper and the fact that one or more corners of the paper were sometimes outside of the image. This is one of the limitations of our algorithm. The algorithm was also confused when the edges of the paper were not completely flat, causing the edges of the paper to not intersect at its corners.

Although using 4 points is enough to calculate a usable perspective transformation, it can sometimes result in the center of the scoring rings to be transformed to an incorrect location. To fix this, we would need to detect a 5th point for the homography equations which would ideally be the center of the rings.

3.3 Circle Detection

We started by detecting circles using a basic Hough transform. However, OpenCV's HoughCircles library discards concentric circles, perhaps to avoid false positives on the same object, and gives us the strongest circle it can detect (Figure 6). Rather than finding ways to detect all the circles, we exploit the fact that the circles on a target are evenly spaced and instead try to detect the outermost circle, from which all the inner circles can be more precisely calculated. Since OpenCV allows a radius range to be

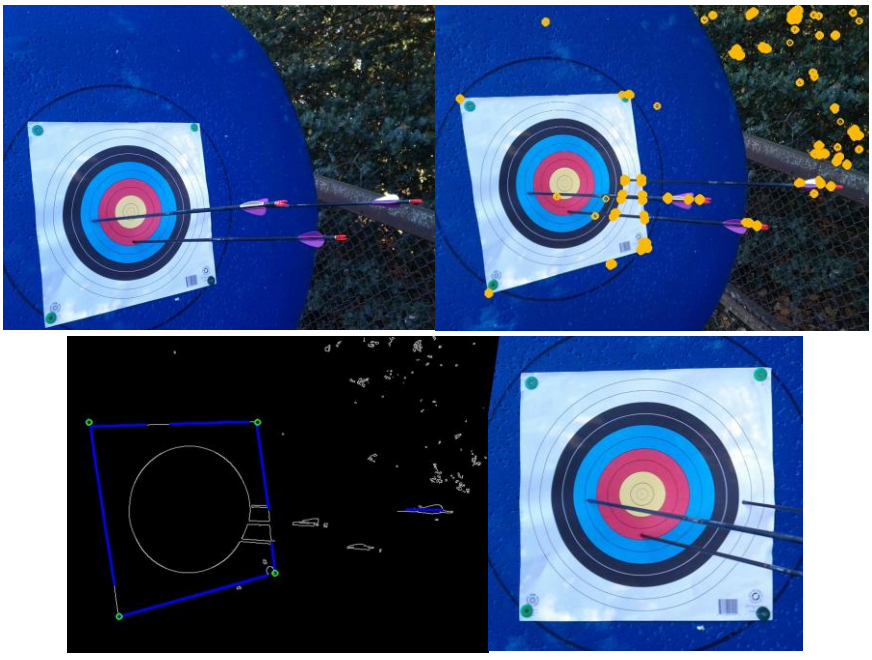


Figure 5: (a) Original image, (b) Harris corners, (c) Detected corners of target (d) Rectified image

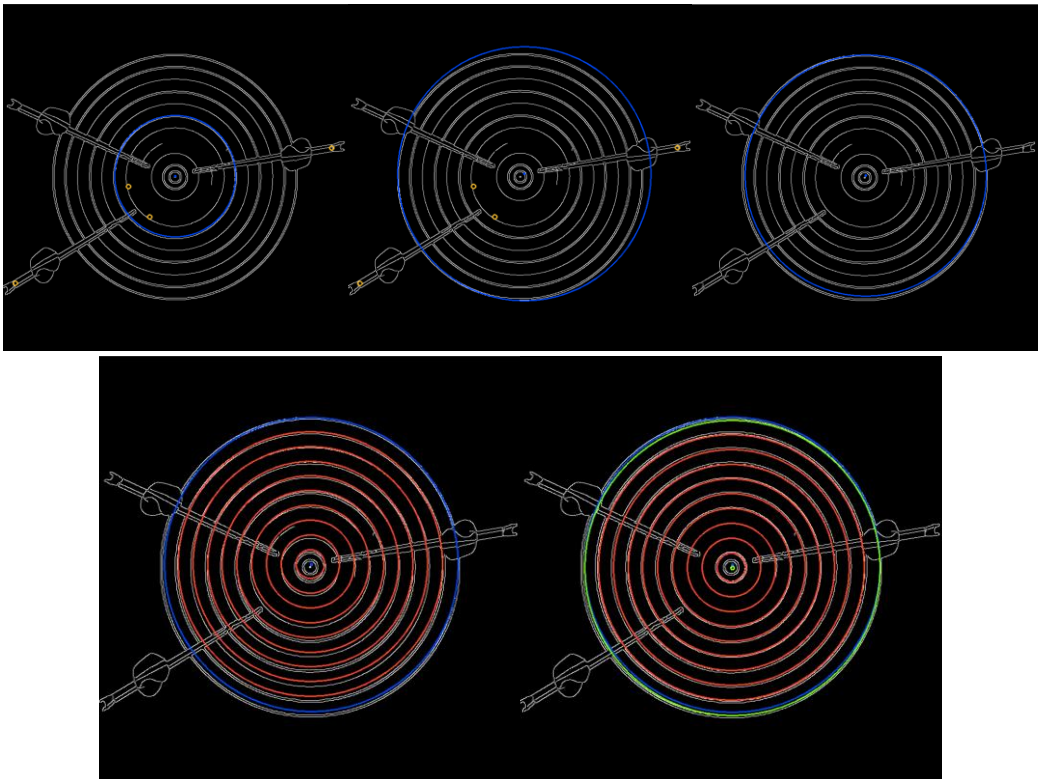


Figure 6: (a) Hough Circles (b) Hough Circles with binary search for largest circle (c) With Gaussian preprocessing (d) Rings estimated using Hough Circles (e) Rings estimated using template matching

specified, we performed binary search on the range of radii in order to arrive at the largest circle (Figure 6d). As a final step, we applied a Gaussian blur to the image before applying the Hough transform for additional precision.

When we calculated the inner rings, we immediately saw how the error of the outermost circle had a much more noticeable effect on the smaller circles (Figure 6d). In order to calculate a more precise location for our outermost circle (green in Figure 6e), we used the Hough circle (blue in both figures) as an initial approximation for a more precise template matching process, in which we tested at scales of .7 to 1.3 in a 50 by 50 range (our image is 800x800) using a pyramid sliding window approach. Since the Hough circle helps us narrow down the search domain of the template matching, we can be very precise without being too expensive. This will be critical since the algorithm is intended to be used on mobile phones.

3.4 Gap Detection

One of the problems with detecting arrows comes from a lack of contrast between the arrow and its background. Since part of the target consists of two black rings and the arrows are usually black, they can sometimes blend in with the black rings and cause a large gap in the edge-detected form of the arrow. However, human eyes can easily perceive the arrow because we can see where it occludes the lines of the target. We can give a computer the same information: instead of looking for where there is an arrow, we look for where part of the target is missing.

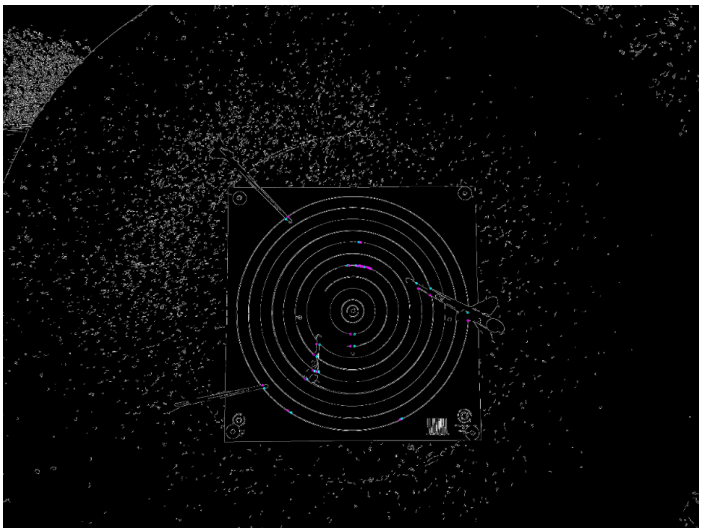


Figure 7: Gap Detection

We rely on the image of the target being correctly rectified and the rings of the target precisely detected. In order to detect gaps in each ring, we walk along the ring in a Canny edge detected image and mark areas where the edge is missing. We can ignore very small gaps and very large gaps,

which are a result of blurring in the pre-processing step, and focus on gaps that are likely to be caused by arrows.

Since arrows can cause our circular walks to deviate, we force the walks to stay within some distance of where we think the ring is actually located. Thus if the location of our rings are not precise, we will have many false-positive gaps (Figure 7).

3.5 Arrow Detection

Arrowhead intersection was a particularly difficult task, and one we had to make a lot of assumptions for initially. We tried many solutions in our attempt to solve the problem, but in the end, the simplest turned out to be the most elegant.

Our first attempt was to use template matching to find sections of the photograph that were similar to the what an intersection of an arrow with a target face looked like. However, this was not successful as efficient template matching is not rotationally invariant, in addition to the complexity of varying backgrounds that take up most of the matching template (the arrowhead is really small and narrow).

We also tried Harris corner detection, with the hopes that the intersection would appear as a corner that we could distinguish. However, as you can see in Figures 8a and 8b, Harris corner detection could not separate the intersection point from any of the other corners in the image, and because of the aliased edges of the circle that no amount of blurring could solve (Figure 8b uses a Difference of Gaussian with Canny and with a particularly low threshold), it ended up returning more circle edges than anything else.

In the end, while the simplest method of using a Hough transform to find the lines with the right amount of parameter adjustment resulted in the best possible outcome for our contrived image (Figure 8f), we ended up finding out that correctly parameterizing a Difference of Gaussian for the real world images resulted in better arrow line detection. One of the issues we faced was finding a fine balance of the arrow shaft between hyperextension as a result of too much noise and hyperflexion as a result of too much blur non-definition.

In addition, when running Hough transform on the real world images, we end up with a significantly worse signal to noise ratio for lines, which means we had to do a lot of tweaking of parameters followed by post-processing and filtering to get it down to what we wanted (Figure 9).

When examining the resulting over-detection of lines in our images (such as Figure 9), we noticed a two critical features we could take advantage of. The first was that there was a lot of noise outside of the square target paper. Since we effectively knew where the location of the center of the circle and its radius after rectification, we roughly could estimate the size of the bounding target paper. We then removed any line that didn't at least one point inside the target bounding box. We realized that arrows may protrude outside of the

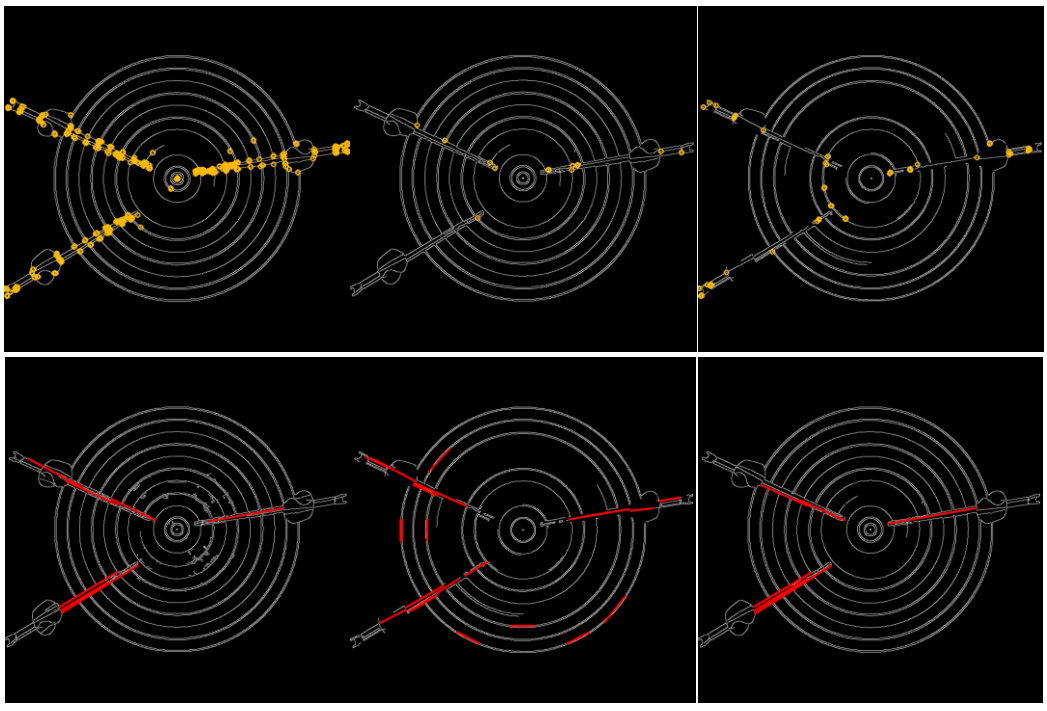


Figure 8: (a) Harris Corner with Gaussian+Canny, (b) Harris Corner with Gaussian+Canny, (c) Harris Corner with DOG+Canny, (d) Hough Lines with no blurring+Canny, (e) Hough Lines with DOG+Canny, and (f) Hough Lines with Gaussian+Canny

bounding box, and thus felt it would be useful to keep as much of that information as possible.

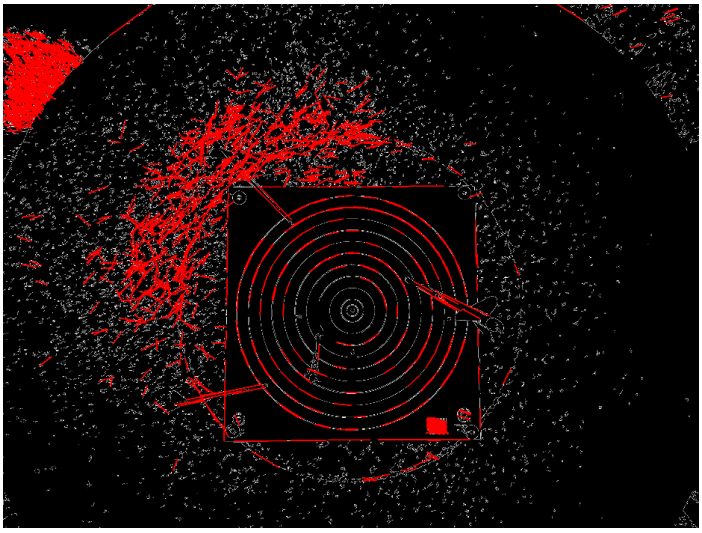


Figure 9: Noisy Hough Lines

The second feature we had to get rid of was a result of the resolution of the circles not being sufficient enough to prevent tangentially lines from appearing on the Hough transform results. Clearly, these lines would affect the overall score, and we needed to find of way of removing them. Again, we were able to take advantage of finding and knowing the circles. Any line less than a certain percentage of the radius that was tangential to a point on one of the circles and was close enough to that point on the circle was removed.

Incorporating these two line noise reduction features, we were able to get much more usable results (Figure 10).

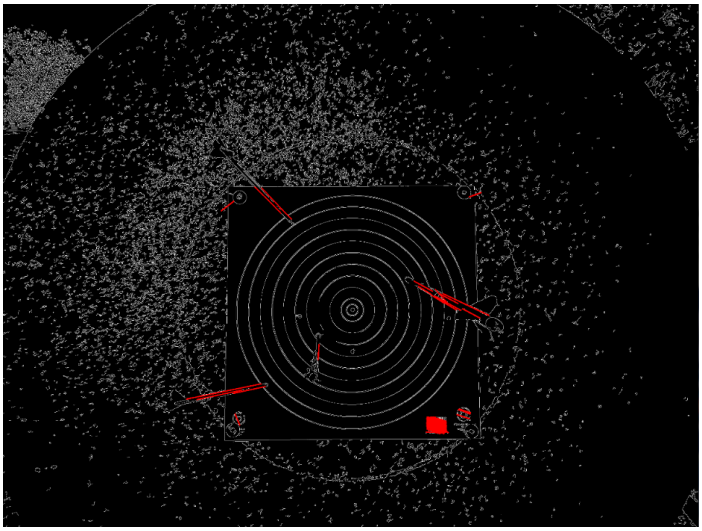


Figure 10: Hough Lines Post-Filtering

However, the main issue to the Hough transforms is that now, each arrow may have several lines of varying lengths and angles, not all of which are connected, and we'll need to determine the best one relative to each arrow.

3.6 Arrow Line and Gap Clustering

Finding which lines correspond to which arrows is no trivial task. First, the perspective of the arrow isn't straight on, the arrow can change width. Second, arrows may be parallel to each other. Third, arrows may be on the same line. Fourth, arrows may intersect. And finally, arrows may be extremely short, and be very difficult to distinguish from other random lines that may appear from arrow feathers.

We ended up determining that clustering the arrows would provide for the best results. To cluster the arrows, we iterate through the set of filtered lines, and if the given line is within a certain score of the current set of clusters we have, combine it with the cluster. Otherwise, create a new cluster with that arrow. The score is calculated based on a weighted parameterization of the angle, the distance from the midpoint of the line to the midpoint of the cluster, and the shortest distance from the midpoint of the line to the averaged lines of the cluster.

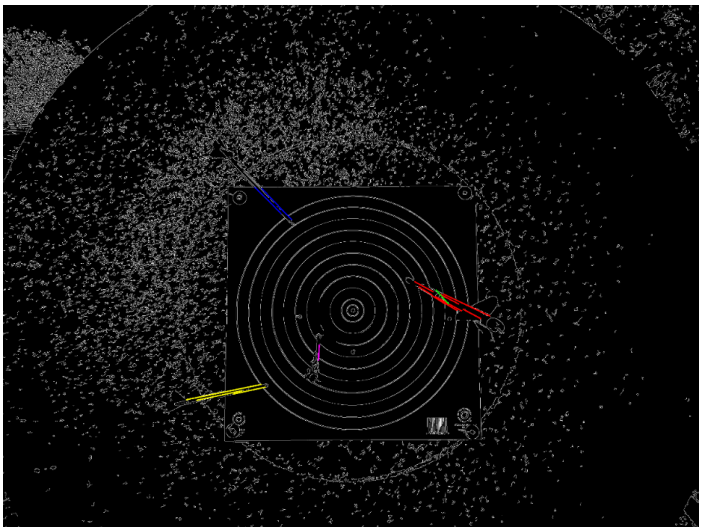


Figure 11: Line Clustered Arrows

The result of the clusters turned out to be quite good (Figure 11), although occasionally there would be an arrow feather line that was at a different enough angle to create a new cluster (green line in Figure 11), and occasionally an arrow that was overly long that was classified into two separate arrows (Figure 12). These are clearly problems, as having two separate clusters results in extra scored arrows.

To combat these two problems, we introduce two different methods. The first method is to use the gaps we calculated earlier to help add points to see which lines are actually

arrows, and which lines are more likely to be off-chance occurrences of feathers. For each pair of gaps, we calculate the score for the closest line (without considering clusters), and then assign those pair of gaps to the cluster that contains the line. The intuition is that you tend to have two lines for each edge of the arrow, but even if you don't if you have a long enough line, you should go through multiple gaps. Finally, since gaps most likely occur along the main arrow, they are much more likely to be assigned to the main arrow cluster than the feather.

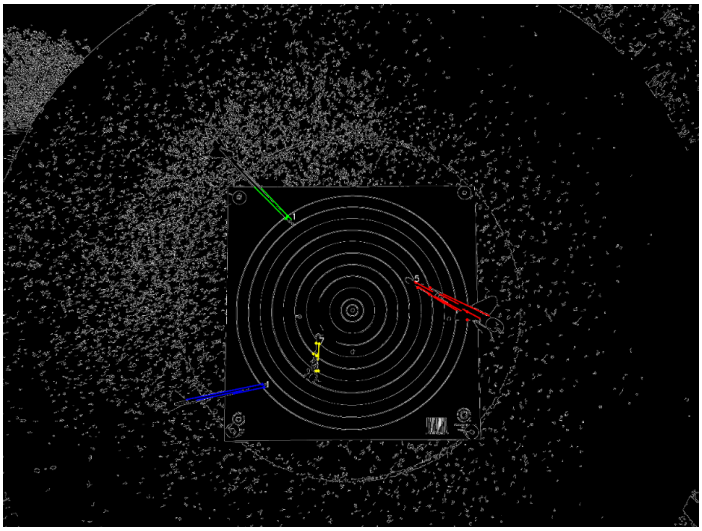


Figure 12: Merged Gap and Line Clusters

To score the gap to a line, we use weighted parameterization of the closest distance to the line segment (represents overall separation) and closest distance to the line (represents angle). Afterwards, we weigh each line in the cluster as 3 points to 2 points for each gap pair, and if the score for the cluster does not exceed a threshold, we remove it (Figure 12).

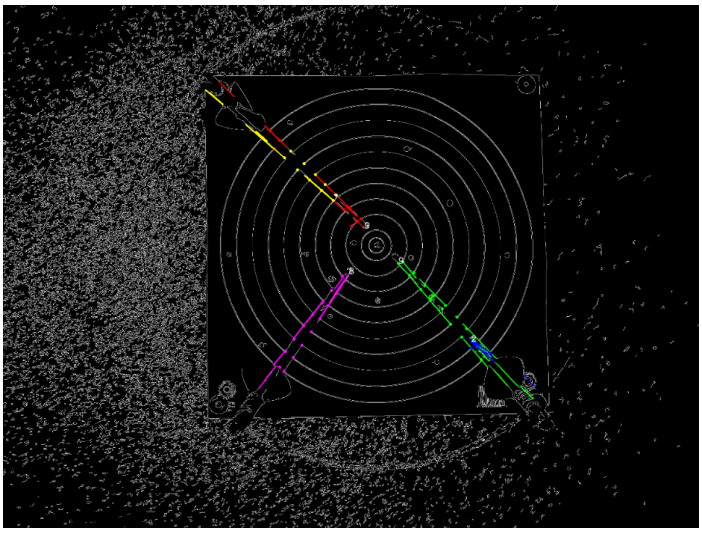


Figure 13: Separated Line and Gap Clusters

However, this doesn't fully solve the problem where two clusters are formed from one arrow (Figure 13). To fix this problem, we noticed that the only case where having two clusters form is bad is if they are indeed the same line. Thus, we calculate the best-fit line for each cluster to get the representative line for the cluster, loop through all of the points in the cluster to find the maximum closest ends of the new representative line, and rerun the original clustering algorithm until convergence (Figure 14).

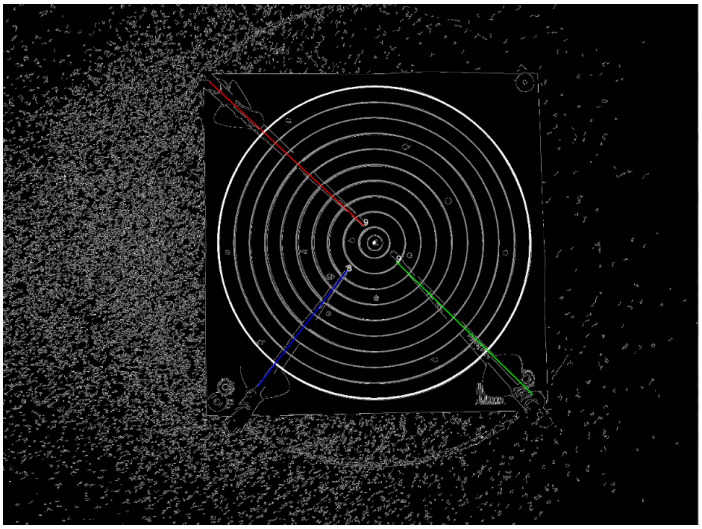


Figure 14: Reclustered Line and Gap Representation

The introduction of these two solutions helped drastically improve detection on most of the problems we encountered when scoring. In addition, the added benefit of reducing each cluster down to one line allows us to have a nice way of representing the clustered arrow.

3.7 Score

Finally, once we have the lines, we can score it. Since we don't have an accurate way of finding the arrow feathers, we naively assume that, given a line segment, the closer of the two points that define the segment is the intersection of the arrowhead (Figure 14). This isn't necessarily the best solution, but it is a rare case when this does not hold.

4. EVALUATION

To test the performance of our system, we have collected images that simulate the typical use case of an archer using a mobile phone camera to take pictures at arbitrary angles. We have a collection of 95 images to evaluate our system. Each image contains 1 to 6 arrows on a target. We have divided them into 3 sets: Easy, Medium, and Hard (Figure ?). Easy images are taken of the target at a direct angle, while Hard images are taken from the side of the target at a more

extreme angle. Because our algorithm relies on correct rectification, the angle of the image can make it very challenging for our system. When viewed at an angle, the arrows also do not seem to converge upon the center of the target, which makes it difficult to determine the orientation of an arrow.

Since real life situations can be arbitrarily challenging, we have limited to scope of our problem to deal with cases where the arrows are not too close together. All of our training samples also use the same background and lighting conditions.

Figure ?: Easy (top), Medium, and Hard (bottom) test examples.

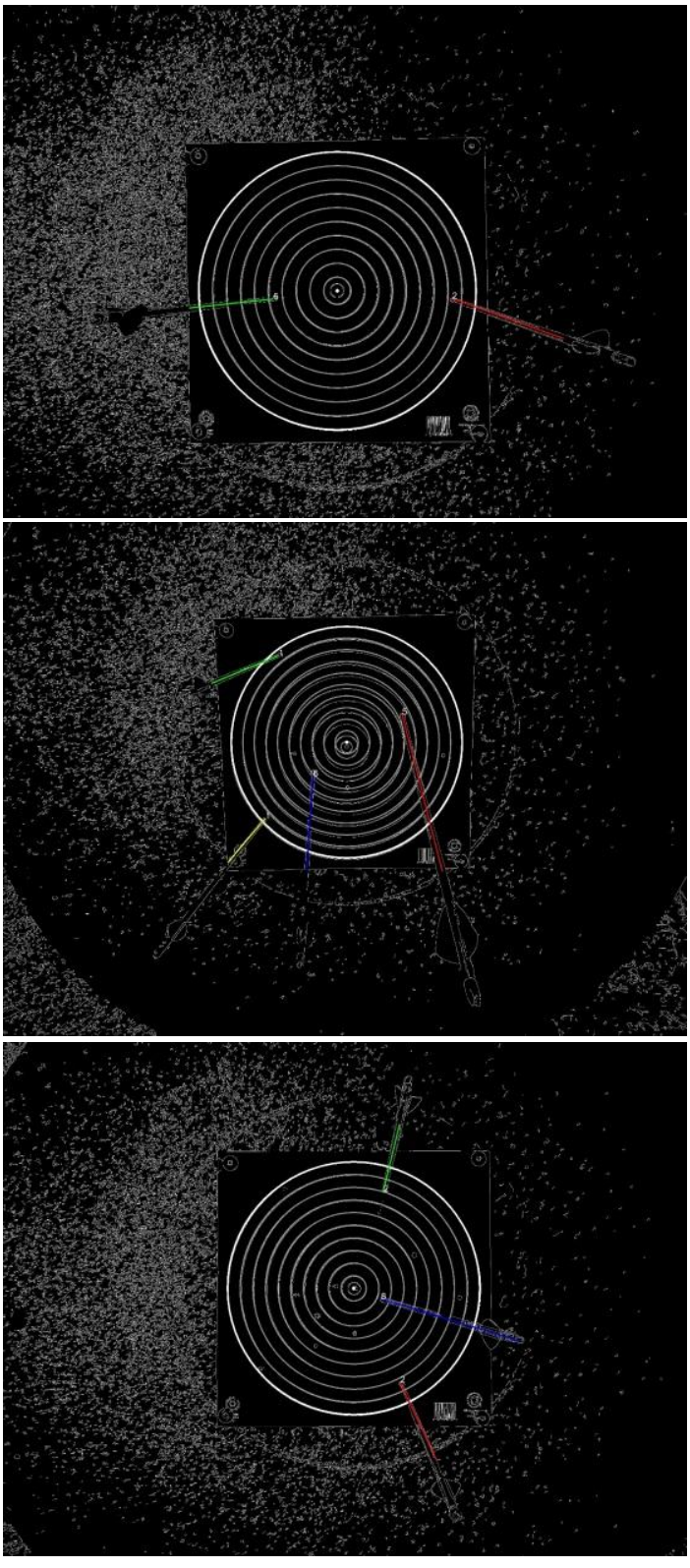
The performance of our system can be measured in two ways. First, we can see how many arrows the system was able to score correctly. We were able to detect a majority of arrows in Easy situations, but the more difficult situations threw us below a majority.

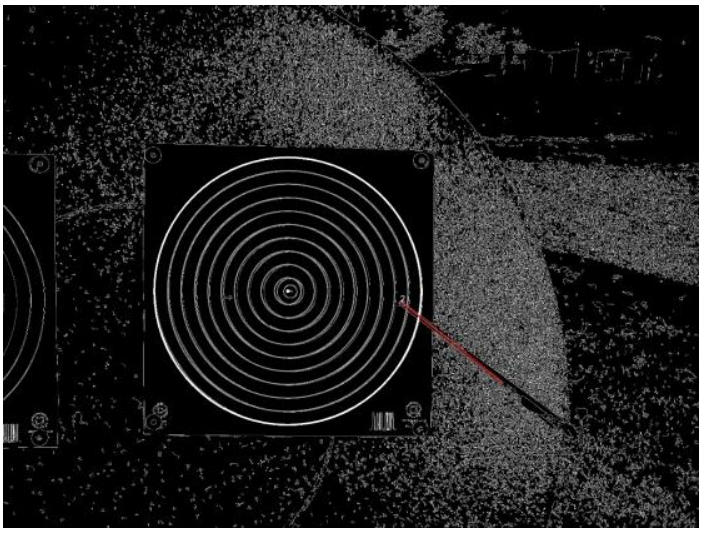
	Number of Arrows Correct	Total	Percent
Easy	53	62	85.5
Medium	47	63	74.6
Hard	47	119	39.4

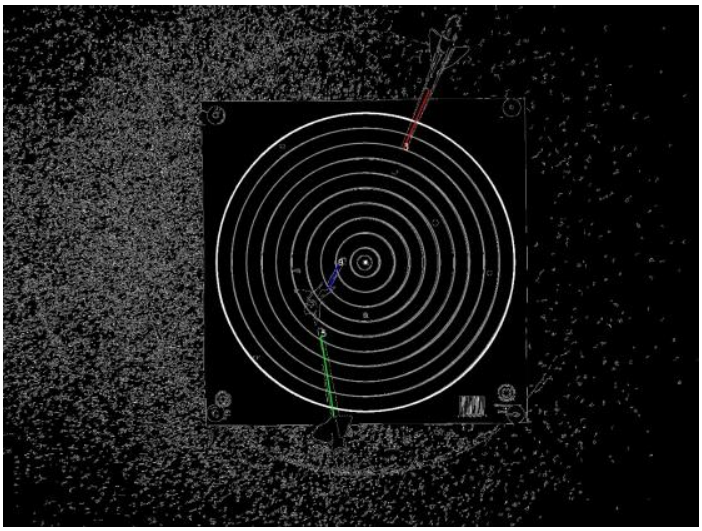
We then examined how many images out of the set our system was able to score correctly, meaning that the score was correct for all arrows on the image and did not include false positives.

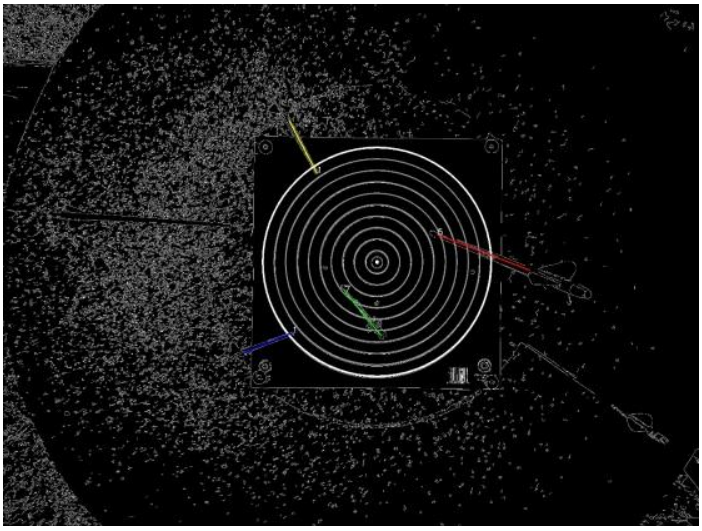
	Number of Images Correct	Total	Percent
Easy	16	22	72.7
Medium	12	23	52.1
Hard	6	50	12.0

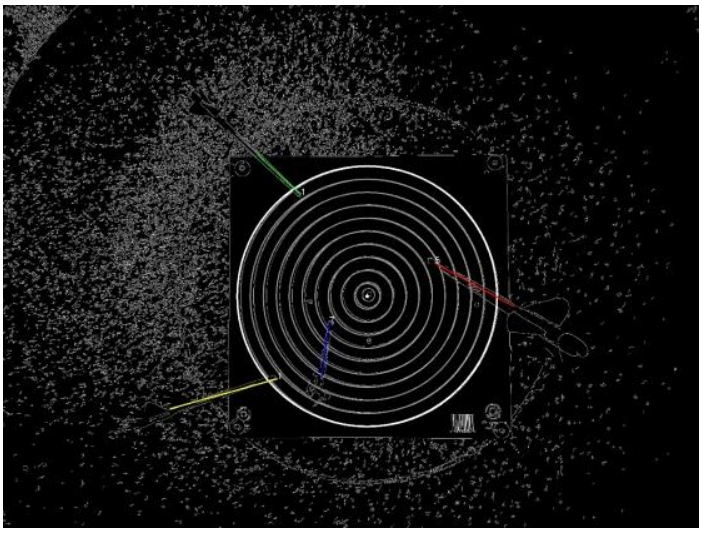
Here, we show some results:











5. CONCLUSION

Our system can successfully detect and score arrows given an image taken from a standard mobile phone. The chance of success is heavily influenced by the angle at which the image is taken. When the angle sharpens, many factors come into play that challenge our system, such as rectification accuracy and arrow occlusion.

While we initially relied on SIFT features for detection of arrows, we found better performance by using line and gap information to accurately locate arrows and eliminate false positives.

Our system can further be improved by detecting more points for rectification and finding ways to separate clusters of arrows while joining together fragmented or occluded arrows. Moreover, for the rare instance where the actual intersection is further from the center than the closer of the two segment points, it would be useful to identify the ends of the arrows properly.

Finally, we'd love to port this to a mobile device and that would be able to calculate the arrows in real time.

6. REFERENCES

[1] Dubrofsky, Elan. *Homography Estimation*. Master's Essay: The University of British Coumbia, 2009.

[2] Open CV Documentation. opencv.itseez.com. 2011.