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 which two circles it's 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. Our partially implemented method currently produces results that demonstrate its capability to detect circles and accurately find points of arrowhead intersection.

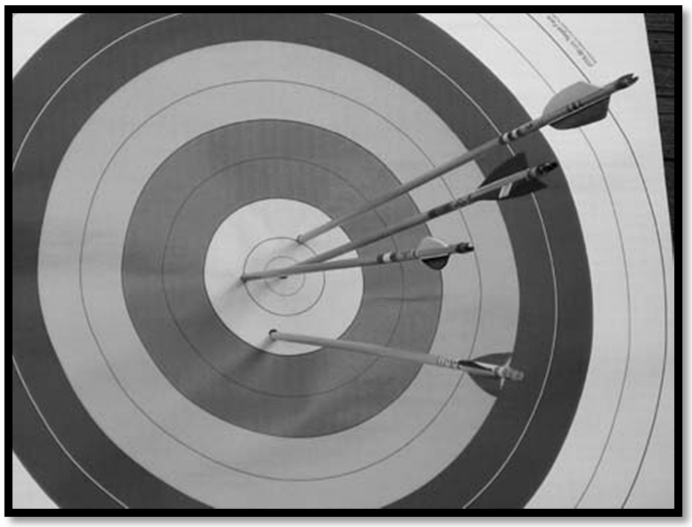
Categories and Subject Descriptors
I.4.8 [**Image Processing**]: Scene Analysis – *Object Recognition*

General Terms
Mobile Computer Vision, 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. PRIOR WORK

[N/A as of yet]

3. DATA AND MODEL

Our project uses images taken of standard 10 concentric circle archery targets without a restriction of the number of protruding arrows. 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, for our milestone, we make a few critical assumptions about the image: 1) the target circle scoring zones are perfect circles without distortion, perspective shifts, occlusions other than the arrows, and fully included in the image 2) the arrows are well-defined, spread out, and do not obscure or interfere with one another in any way, 3) the arrows have no shadows, and 4) the image does not contain motion blur or camera shake (that is, it's sharp). In particular, we did most of our testing on the image in Figure 1. We fabricated this image in order to test our implementation on a simple case. In order to handle cases where the target is not conveniently head-on, we will detect the target's ellipse, apply a rotation to make the major axis vertical, and apply then a horizontal scale to complete the conversion into a circle.

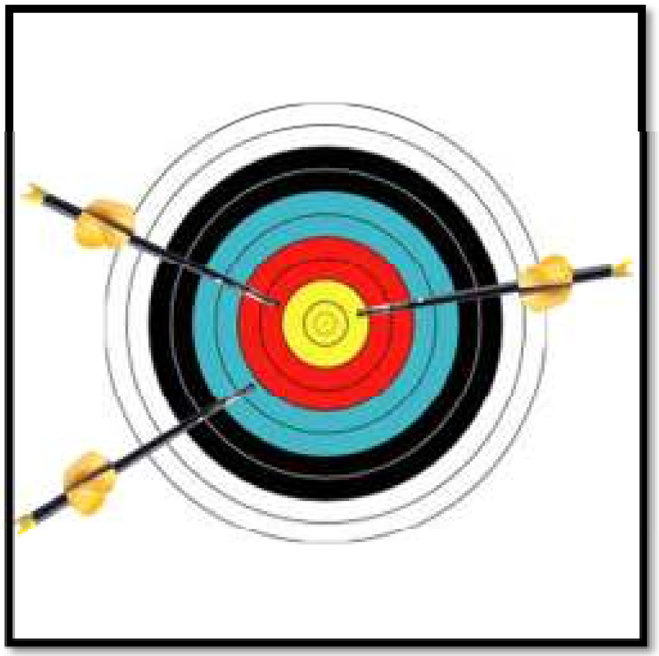


Figure 1: Initial Test Image

4. PROCESS

Our current algorithm utilizes a multi-step process to determine an accurate scoring mechanism. We split up our task into processing the image and then tackling two separate components (detecting circles and arrowhead intersection). We then combine the results to obtain our score.

4.1 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 Figure1 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.

4.2 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 3a). 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 specified, we performed binary search on the range of radii in order to arrive at the largest circle (Figure 3b). As a final step, we applied a Gaussian blur to the image before applying the Hough transform for additional precision (Figure 3c).

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 3d). In order to calculate a more precise location for our outermost circle (green in Figure 3e), 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.

4.3 Arrowhead Intersection

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.

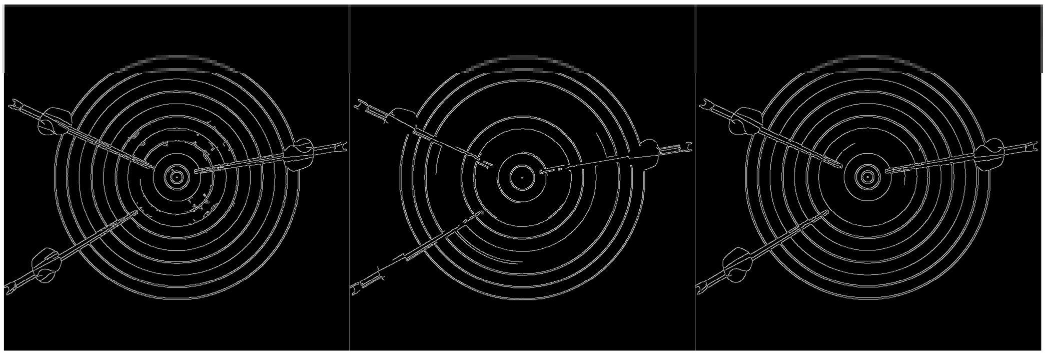


Figure 2: (a) Canny without blurring, (b) Canny with DOG, and (c) Canny with Gaussian

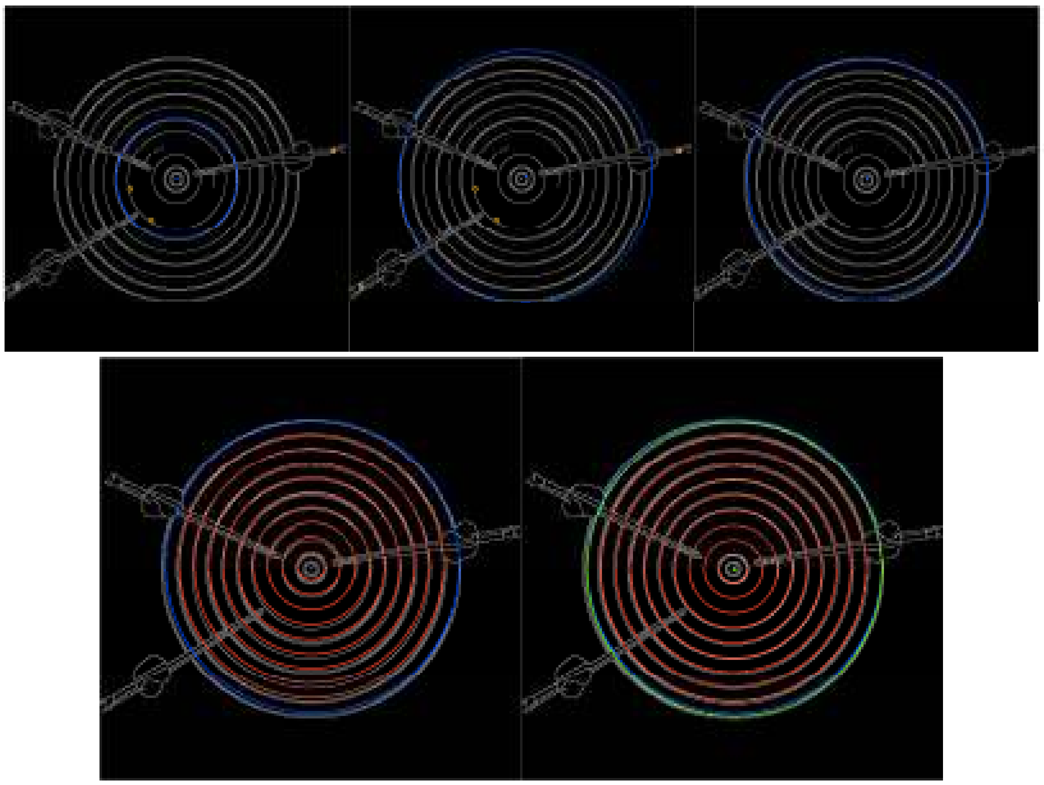


Figure 3 (a) Hough Circles (b) Hough Circles with binary search for largest circle (c) With Gaussian preprocessing
(d) Rings estimated using Hough Circles (e) Rings estimated using template matching

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 a massive failure as template matching is 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 4a and 4b, Harris corner detection fell way short of its potential, and because of the aliased edges of the circle that no amount of blurring could solve (Figure 4b 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, the simplest method of using a Hough transform

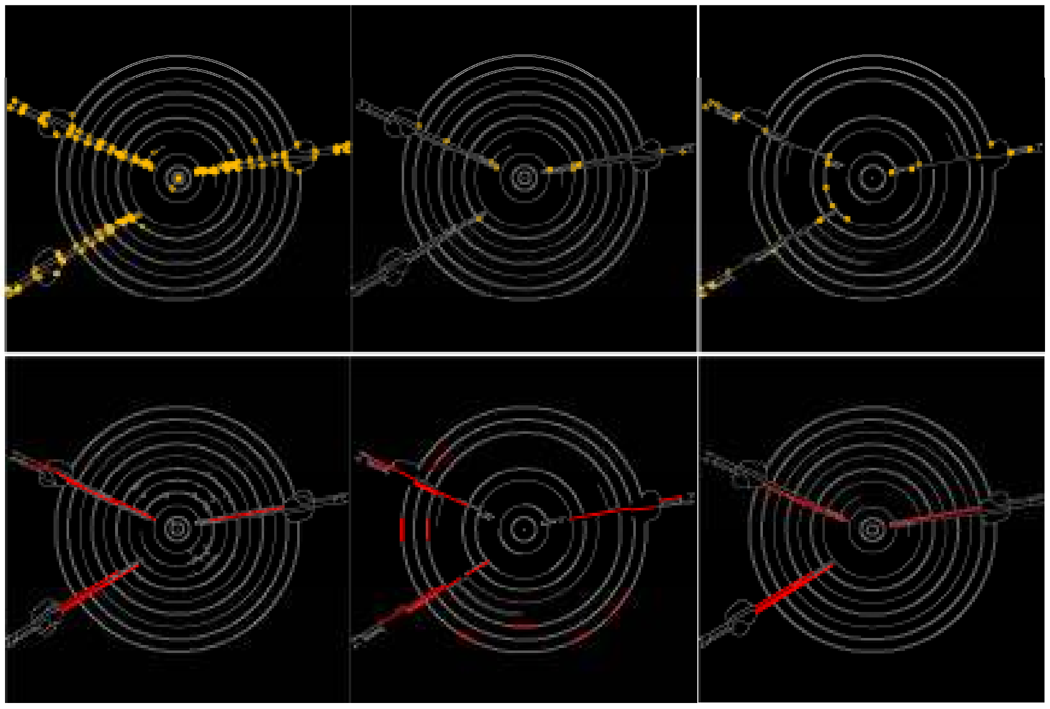


Figure 4: (a) Harris Corner with Gaussian+Canny, (b) Harris Corner with Gaussian+Canny, (c) Harris Corner with COG+Canny,
(d) Hough Lines with no blurring+Canny, (e) Hough Lines with COG+Canny, and (f) Hough Lines with Gaussian+Canny

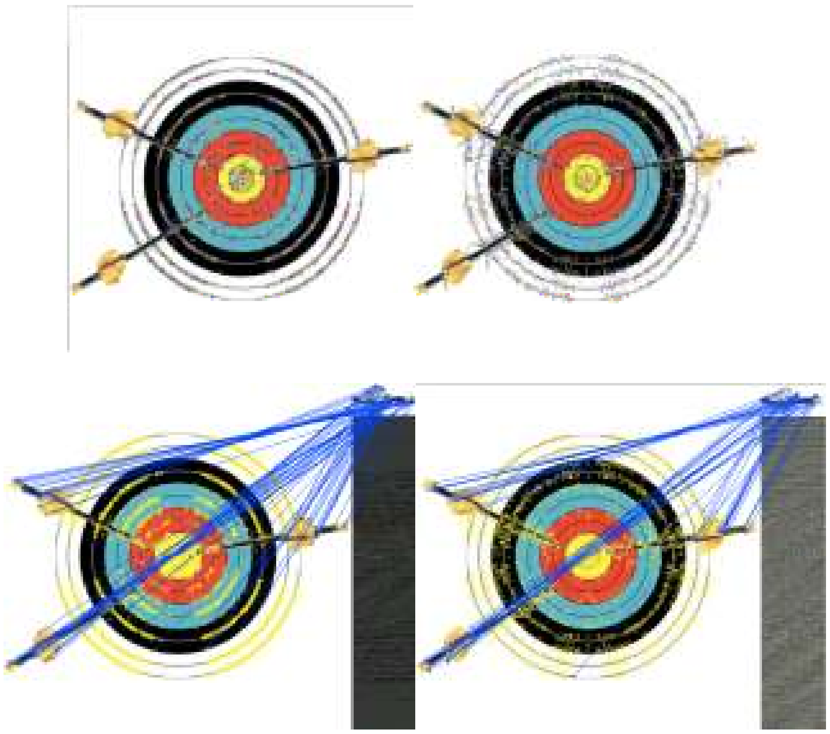


Figure 5: (a) SIFT Keypoints (b) SURF Keypoints (c) SIFT matches (d) SURF matches

to find the lines with the right amount of parameter adjustment resulted in the best possible outcome (Figure 4c). In addition, we chose to use the simple Canny with Gaussian blur for the base image to run the Hough transform on as that allowed for the most arrow definition relative to noise. 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.

The main issue to the Hough transforms is that now, each arrow may have multiple lines, and we'll need to determine the best one relative to each arrow, described in the next section.

4.4 Arrow Fletching Detection

In order to reduce the chances of double-counting an arrow or leaving out an arrow, we want to detect the number of arrows based on the most distinguishable part of the arrow: the fletching (vanes). In the sport of archery, there are several varieties of fletching, from sleek shaped to shield shaped and even curvy ones as shown in Figure 5. Aside from variety, fletching can also be of any color combination and can be any rotation around the arrow. The fletching will also be skewed depending on the angle of the arrow with respect to the camera. Thus, we need to perform object detection that can pick out arrows with any type of fletching, entering the target at any location and angle, and spun in every way possible.

To start, we have detected keypoints and extracted features using SIFT and SURF methods. We are currently in the process of building a training set of fletchings, but Figure 5c and 5d demonstrate how SIFT and SURF compare when it comes to matching with one training example.

4.5 Putting It Together

Once we have have an idea of how many arrows are on the target and where they may be, as well as where the rings are, we can calculate the score by taking the point of the arrow closest to the target.

5. CONCLUSION

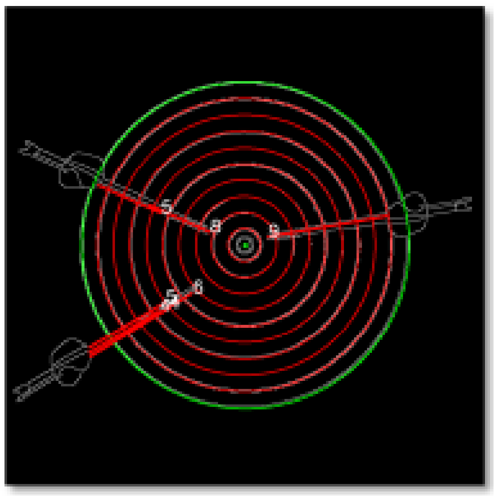


Figure 6: Combined efforts

Our current milestone update has provided promising results, and by trying out and physically seeing the results of many of the algorithms we learned in class, we have gained a lot of intuition on what might work and what likely won't for our project. We hope to build on the success by making the overall solution more robust and accurate. In particular, we plan on having the final project implement these goals:

- Instead of detecting circles, we should assume that the target will be captured at an angle. Thus, we should instead detect for ellipses and transform the ellipses back into the original intended concentric circles.
- Determining which of the multiple detected lines per arrow correspond to which arrow. This is a simple problem if you can guarantee that the arrows are a set distance apart, but depending on the perspective of the camera as well as the accuracy of the shots, this does not have to be the case, and thus, becomes a nontrivial problem.
- Distinguishing and handling arrow shadows.
- Implement on mobile device (considering there's already a ported OpenCV library for Android) for real time point calculation.
- Dealing with lots of random holes from previous arrowhead piercings.
- Testing on real and more challenging scenarios.

6. REFERENCES

[N/A as of yet]