
cv::findingCoins();
Ian Dudder & Karran Singh

Project Goals

Project Idea: Develop Coin Recognition Software

Objectives:

1. Recognize elliptical objects in a test image.
2. Determine which elliptical objects are coins and which are not.
3. Determine if the coin is heads up or tails up.
4. Determine the type of each coin (i.e. penny, nickel, dime, etc).
5. Report the value of the coins in the image as a collection.

Initial Methodology

Objective 1: Recognizing Ellipses

- Perform edge detection on the image.
- Use Hough's Circle to identify circles.

Objective 2: Identify Coins in the image

- Perform edge detection on each detected circle.
- Count the number of edges in each circle. More edges indicates we have found a coin.

Objective 3: Determine if a coin is heads up or tails up.

- Finding more edges indicates tails, fewer edges indicates heads.

Initial Methodology

Objective 4: Determine the type of coin

- Compute a color histogram of each coin.
- Based on color, determine if the coin is copper-colored or silver-colored.
- Compare the coins to a template to distinguish between the silver coins.

Objective 5: Determine value of the coins.

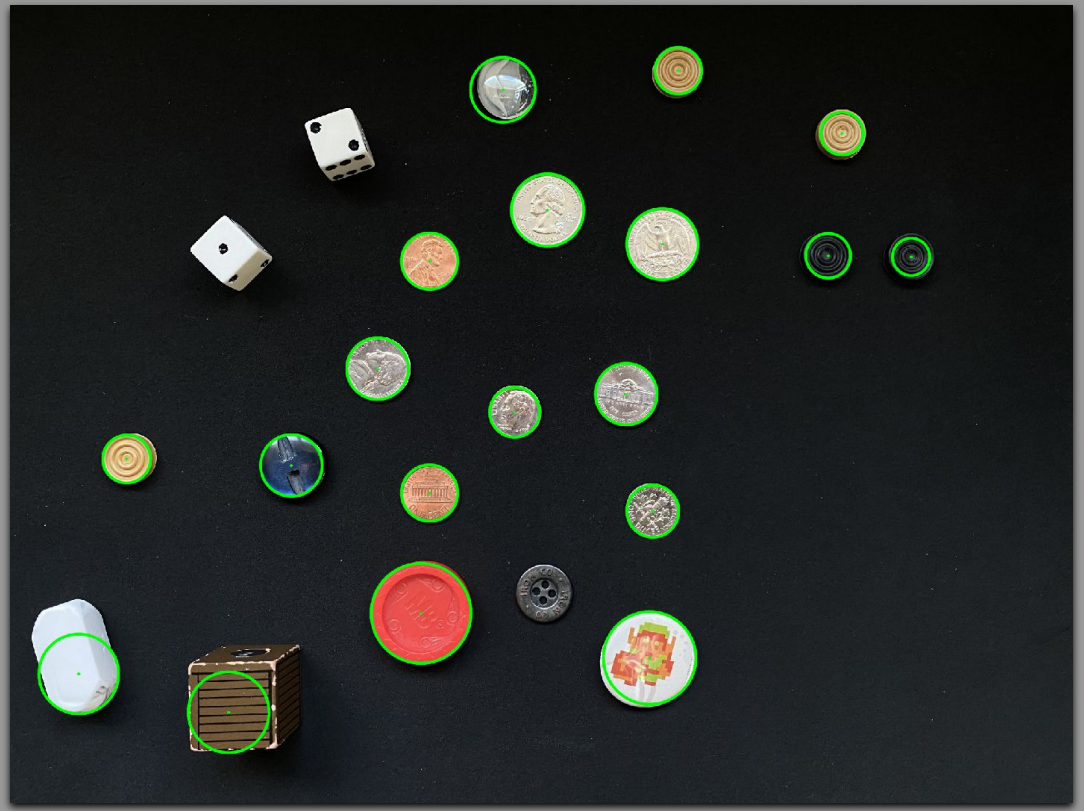
- Count up the number of each coin and compute the value of the collection as a whole.

Obstacle 1

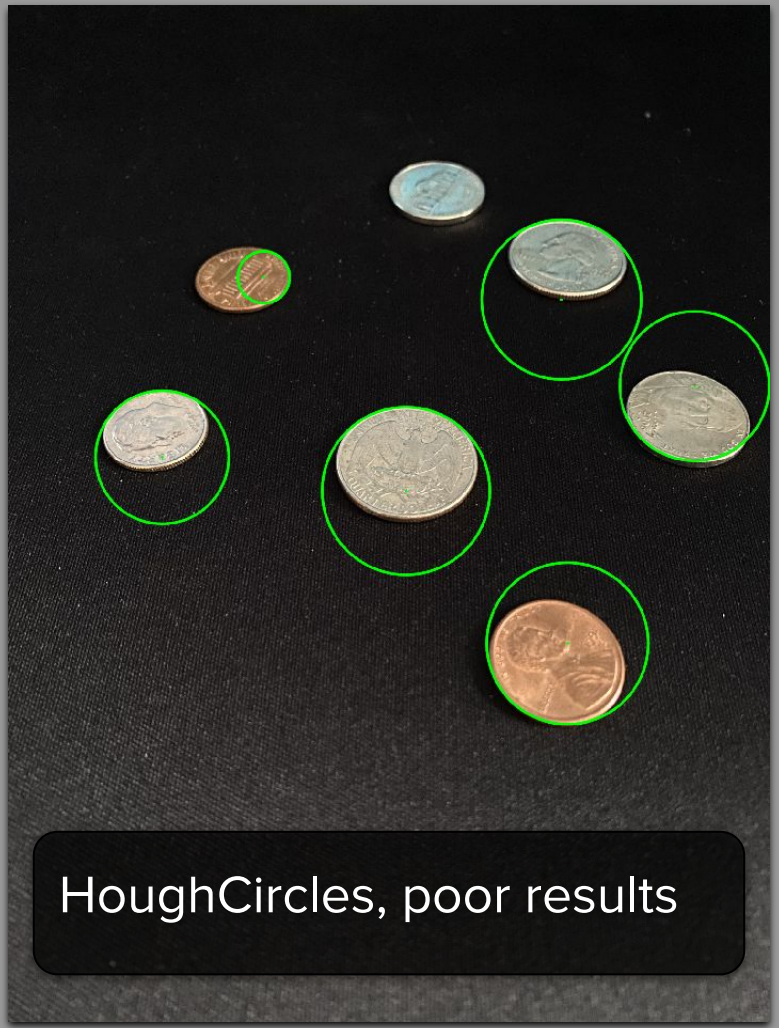
In reality, our initial methodology fell through.

Obstacle 1: Hough Circles was not precise enough

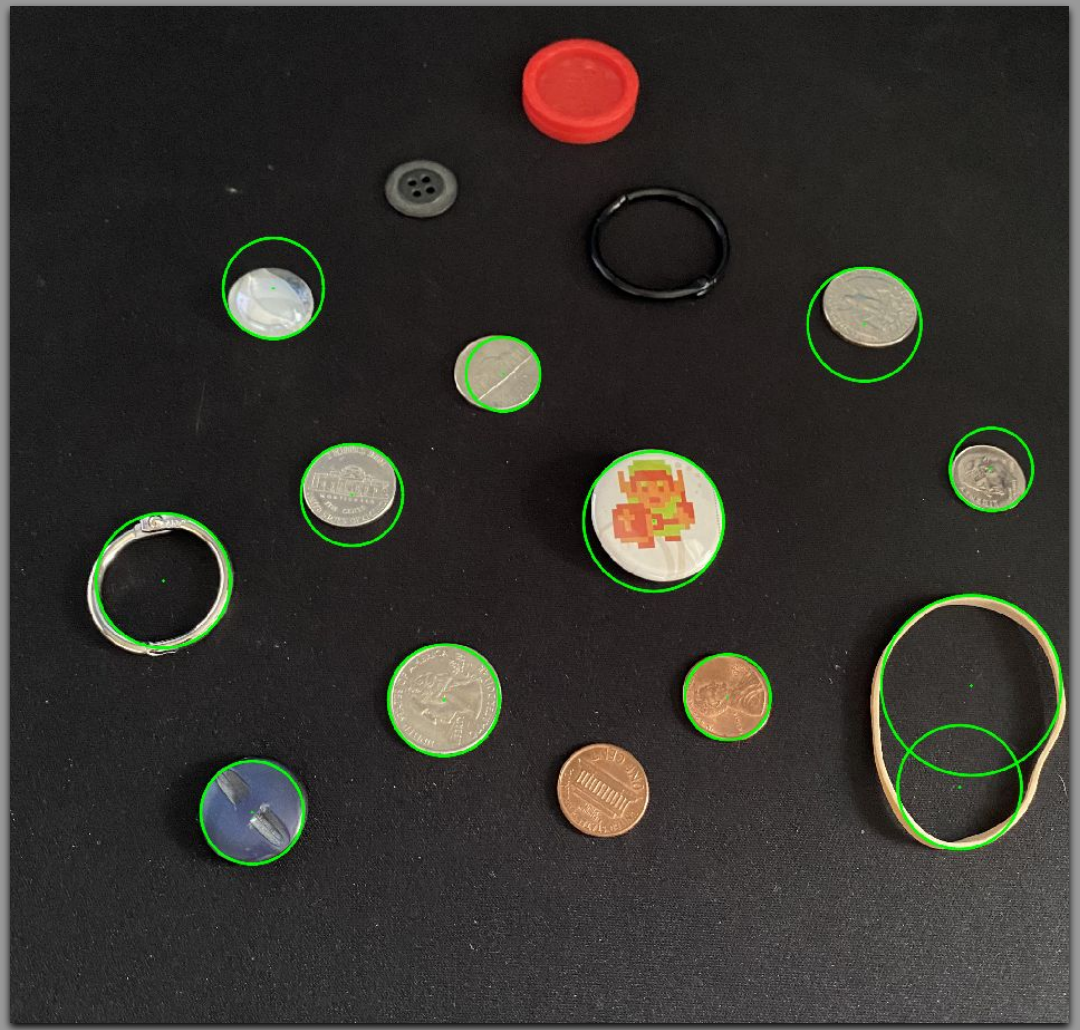
- If the coins were at an angle, the circle was larger than the coin, leaving some of the background along with the coin.
- It was difficult to compute a bounding rectangle from which to extract a sub-image from.



HoughCircles, mixed results



HoughCircles, poor results



Revised Process

Use cv::findContours() for shape recognition rather than Hough Circles.

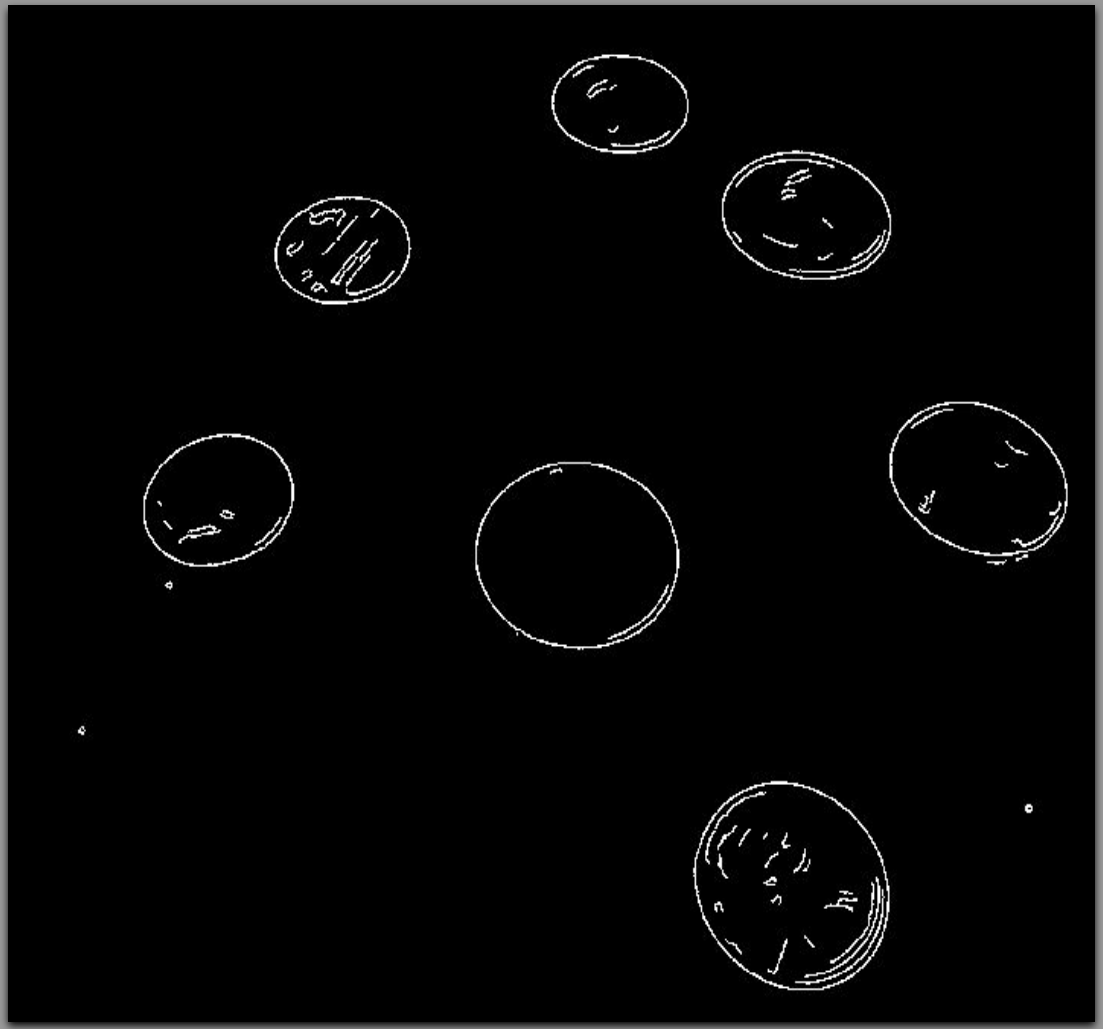
- Provided us with exact locations and allowed us to create a bounding rectangle to extract a sub-image from.
- Allowed us to create a best-fit ellipse that eliminated non-elliptical objects.

Challenges

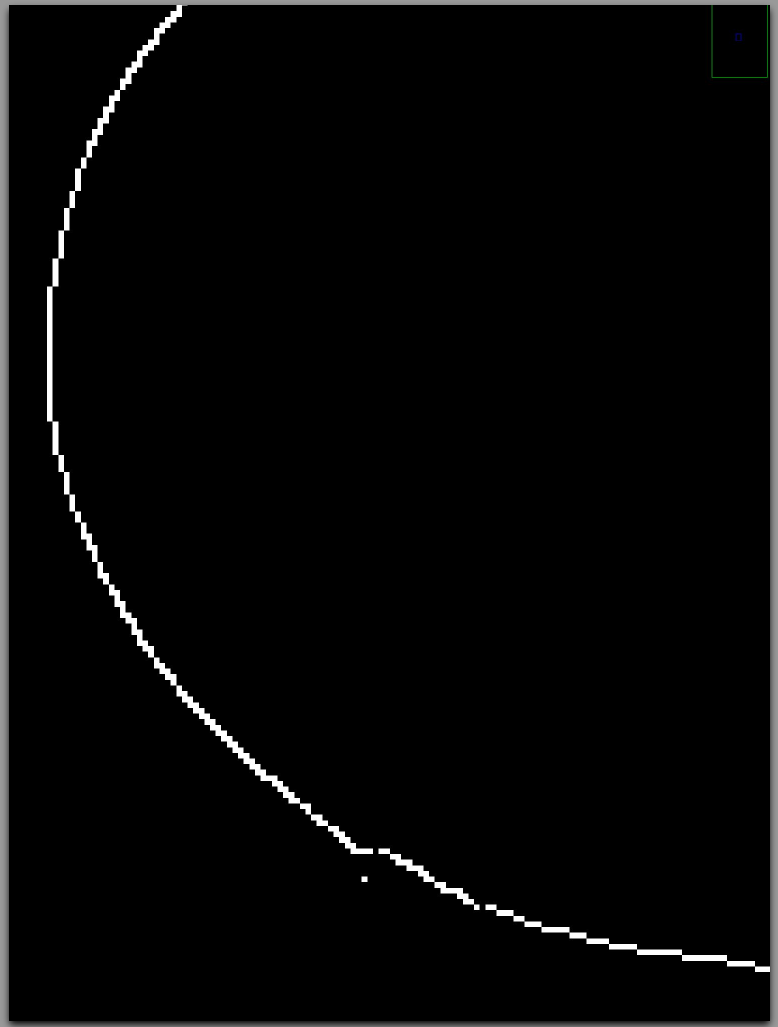
Contours requires edges to connect in order to count as a shape.

In order to remove noise we needed to blur the image.

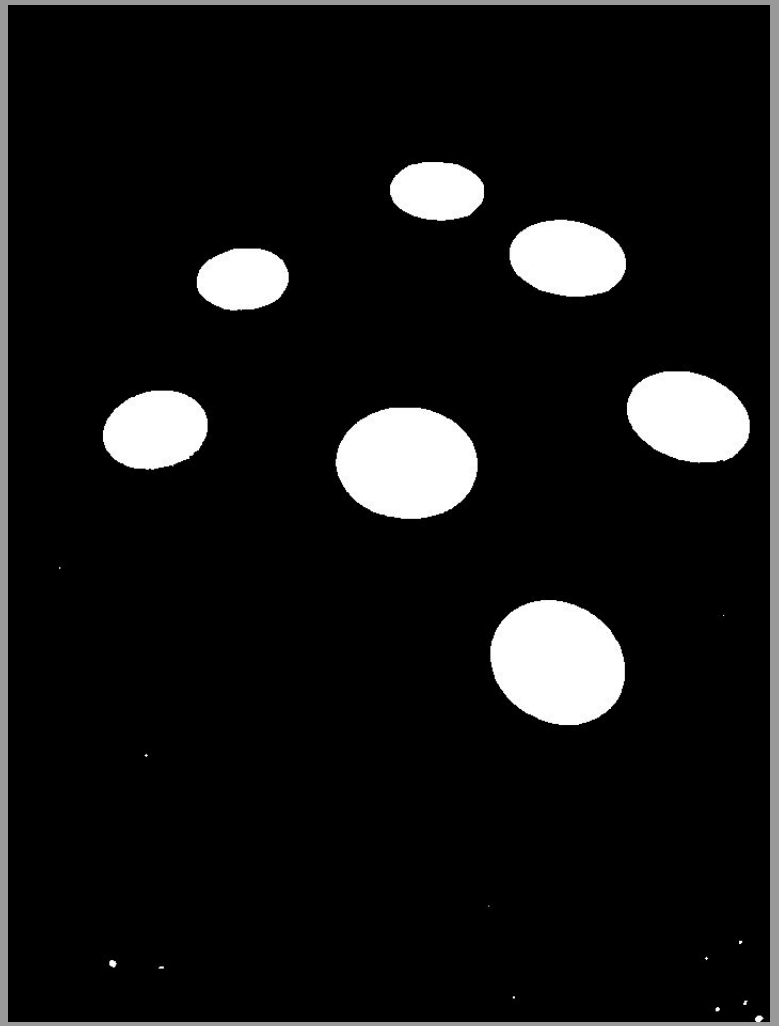
It quickly became a challenge to strike the right balance between eliminating noise while retaining enough edges.



Finding contours with edge images using canny

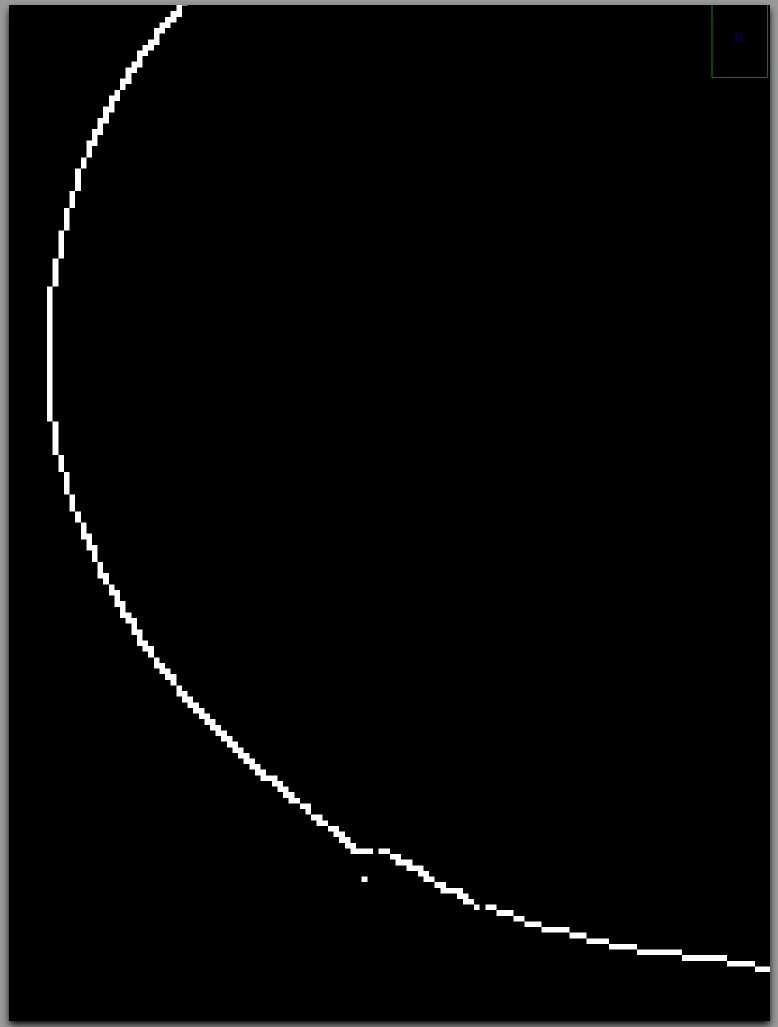


Broken edges

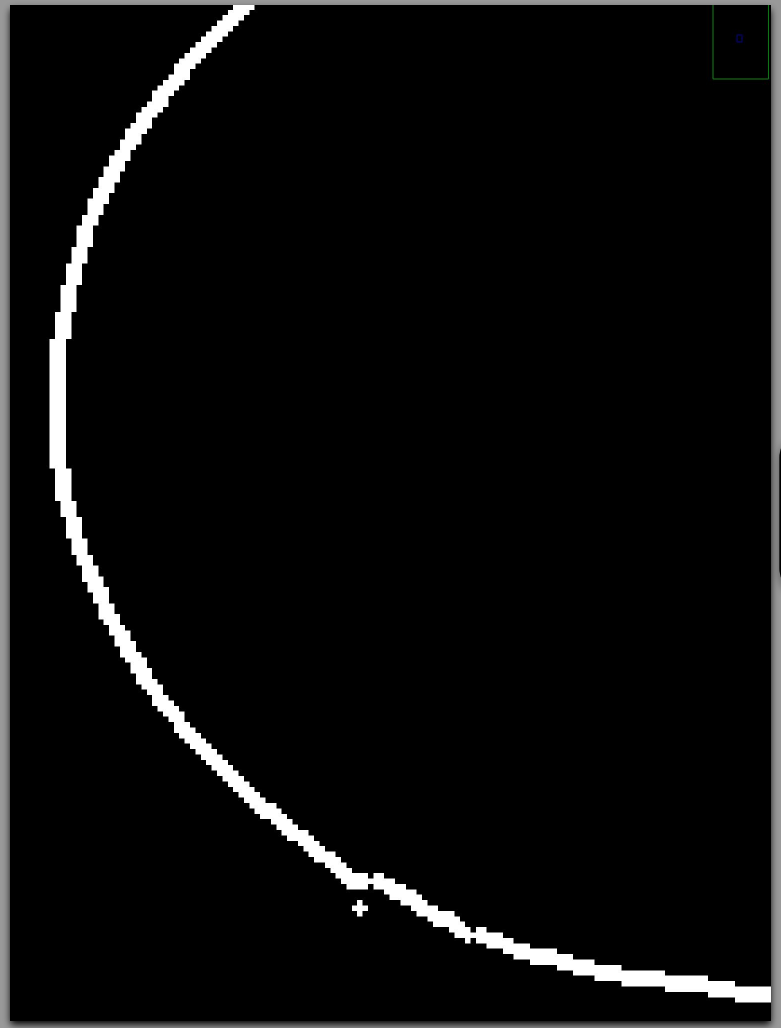


Finding contours with threshold images

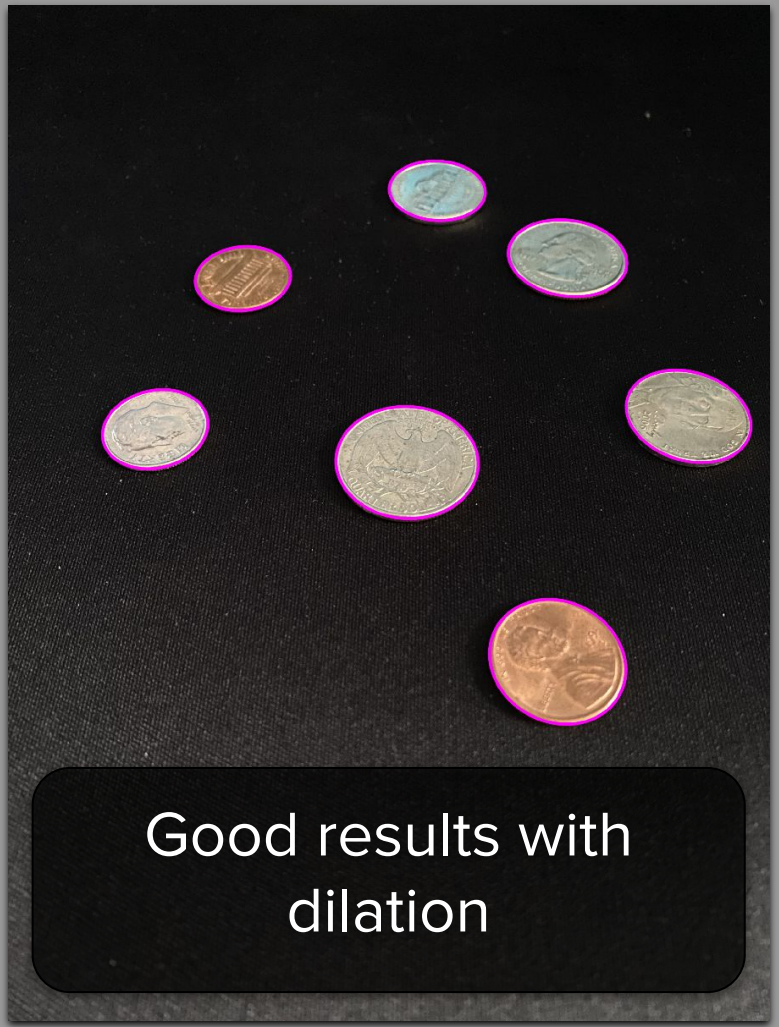
Threshold with some shadows



Back to canny



cv::dilate


Good results with
dilation

Obstacle 2

Obstacle 2: Sheer count of edges was not enough to determine if a circle was a coin or not, nor was it able to determine if a coin was heads up or tails up.

More information was needed.

Template Matching

Use template matching to determine if an ellipse is a coin.

- Scale down template to be the same size as the patch.
- Rotate the image until a best match is found.
- Compare matches to one-another for best match.
- Compute percentage of matched edges and threshold it to eliminate objects that are not coins.

Template Matching

Patch

Edge Image

Steps

1. Create a sub-image (patch) for each contour found.

2. Compute the edge image of the patch.

Template Matching

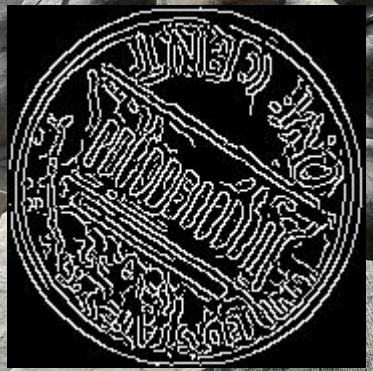
3. Compare the patch's edges to a template. Scale down the template to be the same size. Try every rotation of the template and keep track of matched edges.

Patch

Rotating Template

Template Matching

4. Remember which rotation and template coin gave us the best match.



=

Patch

Rotating Template

Results

By remembering the number of edges that match, we can determine if the ellipse is a coin.

Coins will match ~50% of their edges with the templates. Objects that are not coins will have less than 50% of their edges match.

Another useful tactic: compute the ratio of edges to non-edges for an object. Coins have ~14% of their total patch-space as edges. Smoother non-coin objects have less.

Despite keeping track of the best matches for every rotation of every coin, template matching cannot tell us what type of coin it is.

Obstacle 3

Color is unreliable.

The color comparison used to distinguish between copper and silver-colored coins is not fine-tuned yet.

This gave us significant challenges due to the heavy variation of color between images.

Computing color histograms and finding the most common colors in the coins was ineffective even with large ranges of buckets allowed.

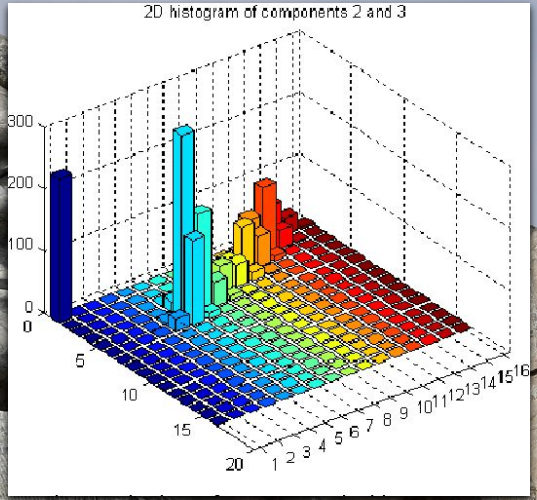
Obstacle 3

BucketSize = 4;

- Penny BGR = 160, 160, 224
- Nickel BGR = 224, 224, 224
- Quarter BGR = 160, 160, 160

BucketSize = 8;

- Penny BGR = 176, 208, 240
- Nickel BGR = 240, 240, 240
- Quarter BGR = 140, 140, 140



Color Histogram Example

Obstacle 3

BucketSize = 16;

Penny BGR = 168, 200, 232

Nickel BGR = 232, 232, 232

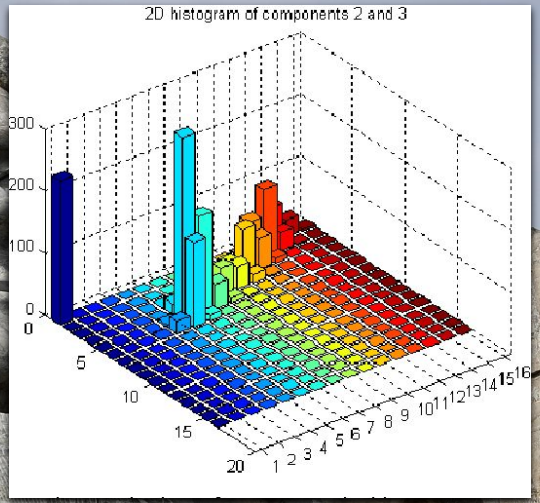
Quarter BGR = 152, 152, 152

BucketSize = 32;

Penny BGR = 164, 188, 228

Nickel BGR = 228, 228, 228

Quarter BGR = 140, 148, 148



Color Histogram Example

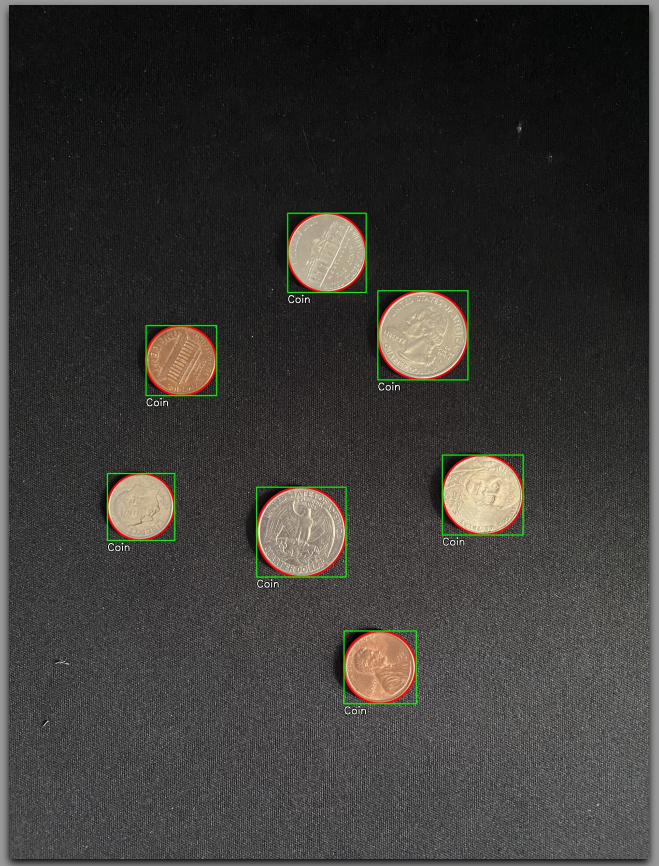
Obstacle 3

Project Results

Completed Objectives 1 and 2.

Made progress on objectives 3 and 4, but neither is fine-tuned enough to be accurate yet.

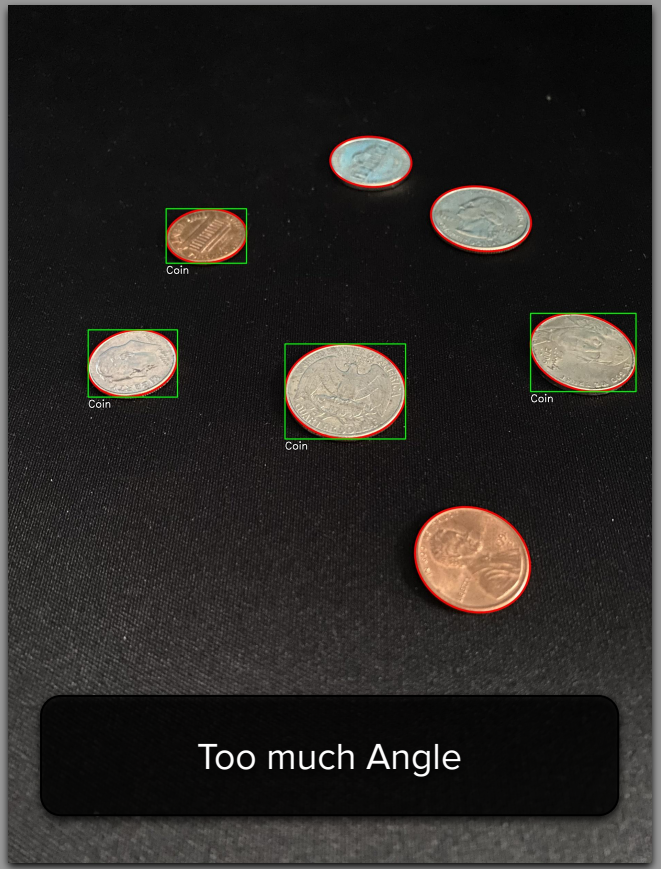
Success Images

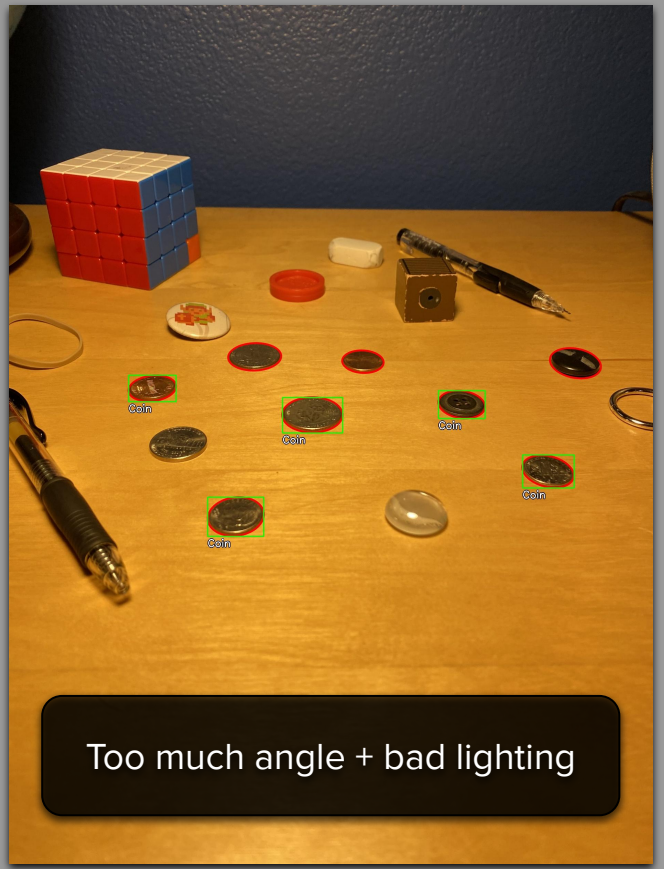



Coin
Coin
Coin
Coin
Coin
Coin
Coin
Coin

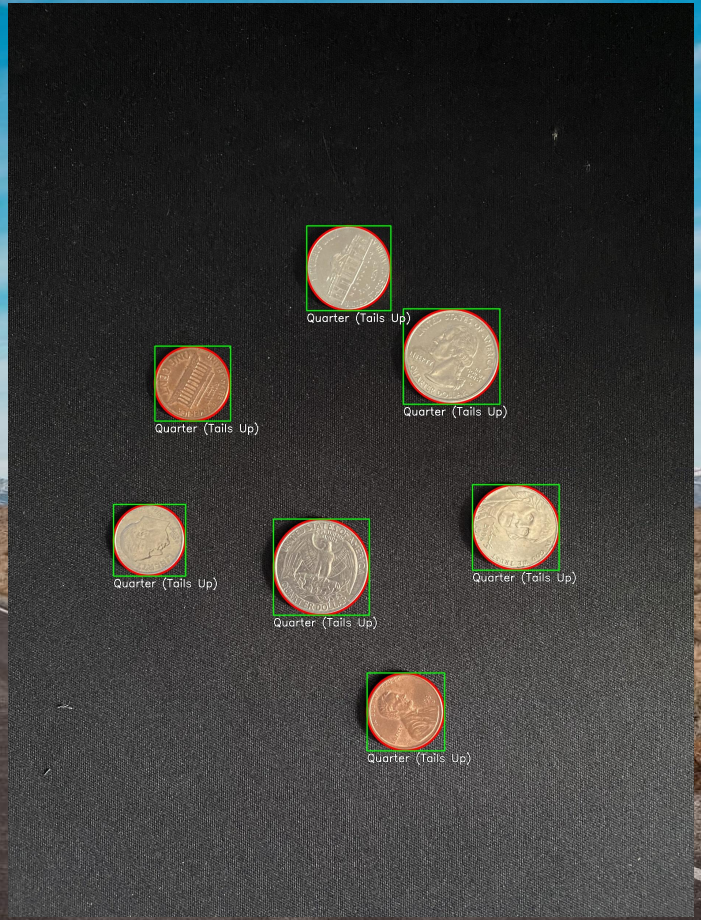

Coin
Coin
Coin
Coin
Coin
Coin
Coin
Coin
Coin

Challenges


Coin
Coin
Coin
Coin
Too much Angle


Coin
Coin
Coin
Coin
Coin
Too much angle + bad lighting

Room for Improvement



Moving Forwards

Type-determination is not very smart...

Neither is detection of heads/tails orientation.

Moving Forward

Using this next week ahead of us, we will...

- Improve our template matching to make better guesses at coin-type.
- In 90-degree shots, compare the relative size of detected coins to one another to determine coin type.
- Compare number of edges between detected coins to determine heads-up or tails-up

Moving Forward

- Fine-tune color matching to allow for better recognition of coin-type.
- Improve upon our ability to recognize coins in a tilted image.


That’s all! Thanks!