Hough Transform to Detect Circles in Images

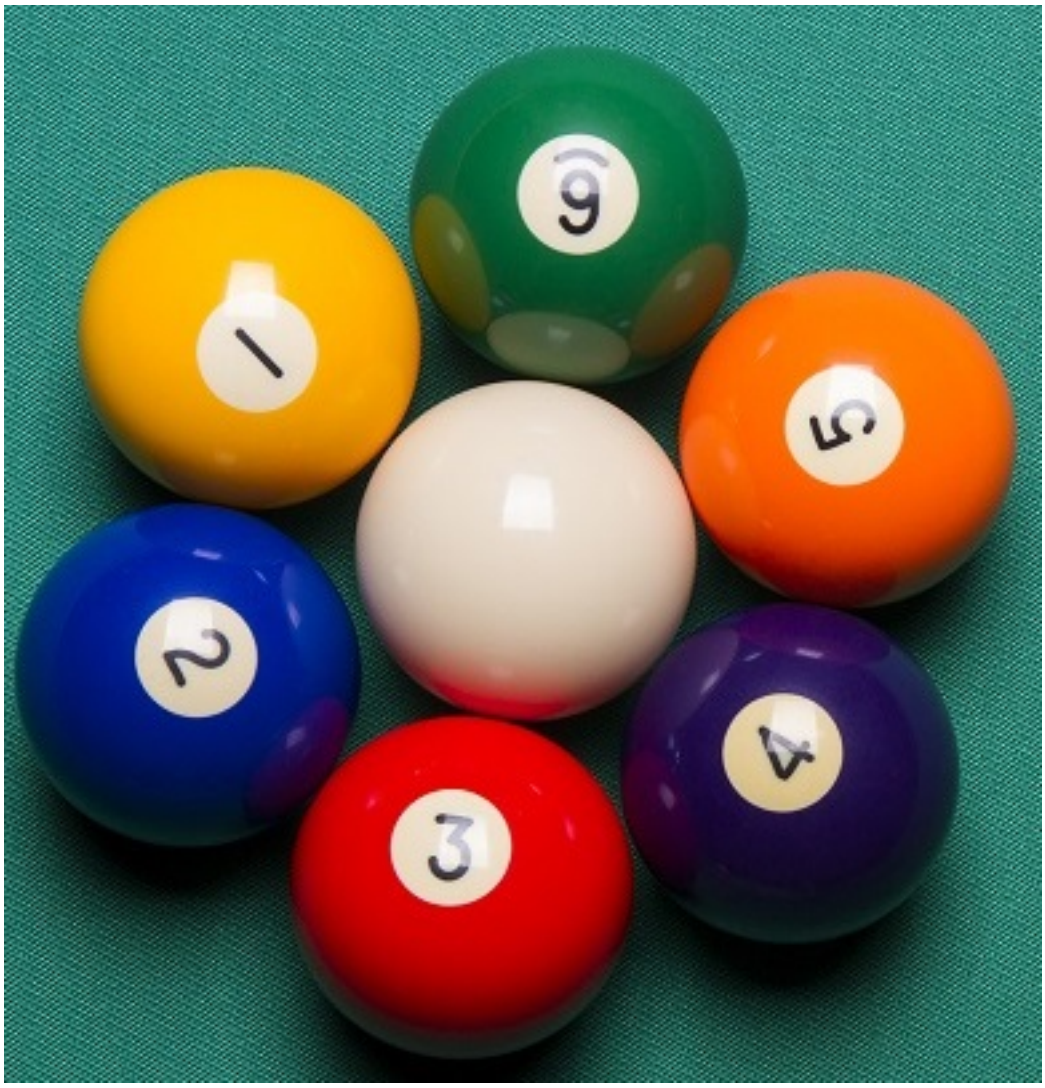
Introduction

Circular Hough Transform is a variation of the classical Hough Transform that detects circles instead of straight lines in an image. The theory behind the two algorithms are the same: they let all edge pixels "vote" and select the candidates with the most votes to be the detected shapes. The difference is in how we construct an accumulator to record all votes. A cool application of circle detection is the head counter. For example, a security camera that monitors the passenger traffic at a subway station by counting people's heads in a real-time image.

Preprocessing

Same as line detection, we apply an edge detector to the grayscale image and strengthened the contour with a threshold() command.

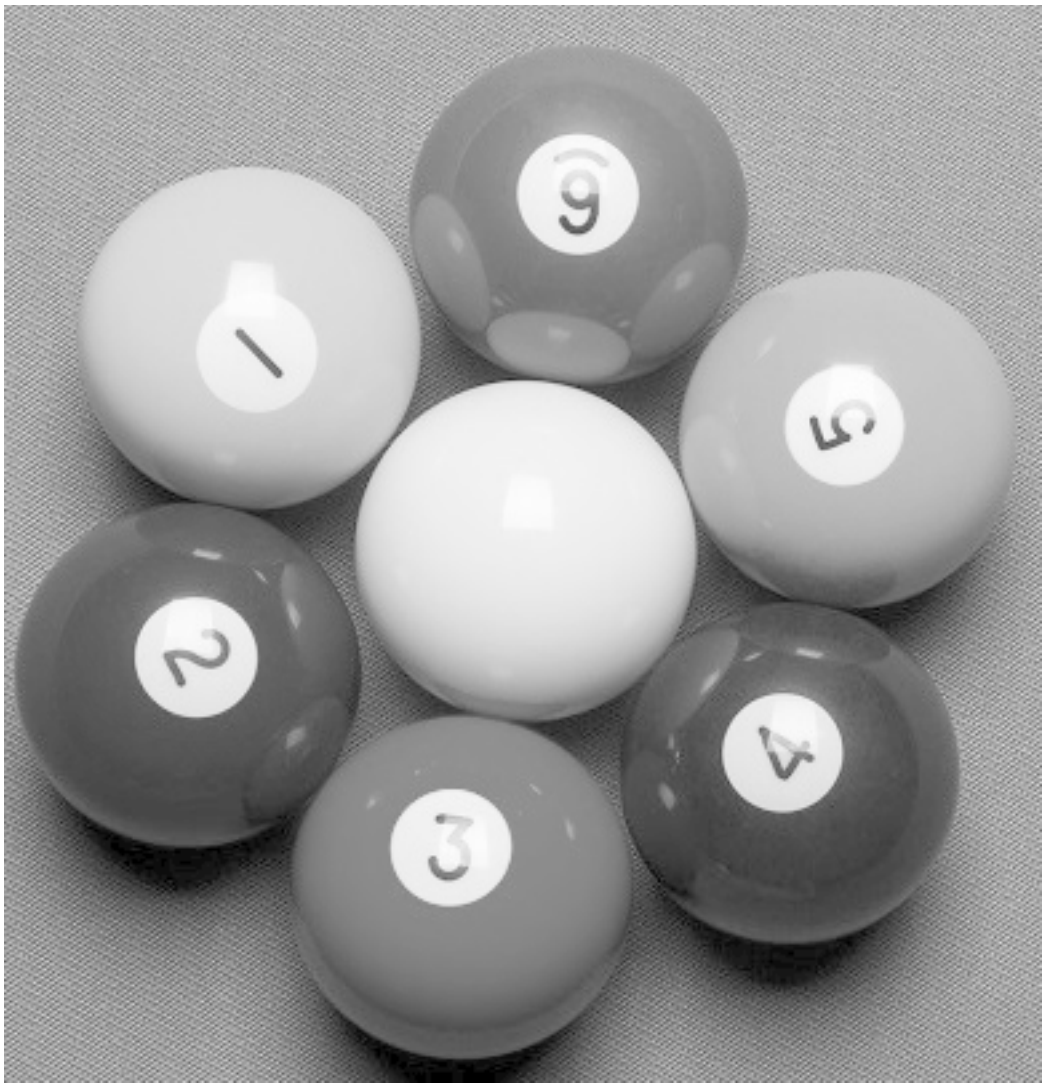
```
> with(ImageTools):
> original := Import("Billiard.jpg"):
> Embed(original);
```



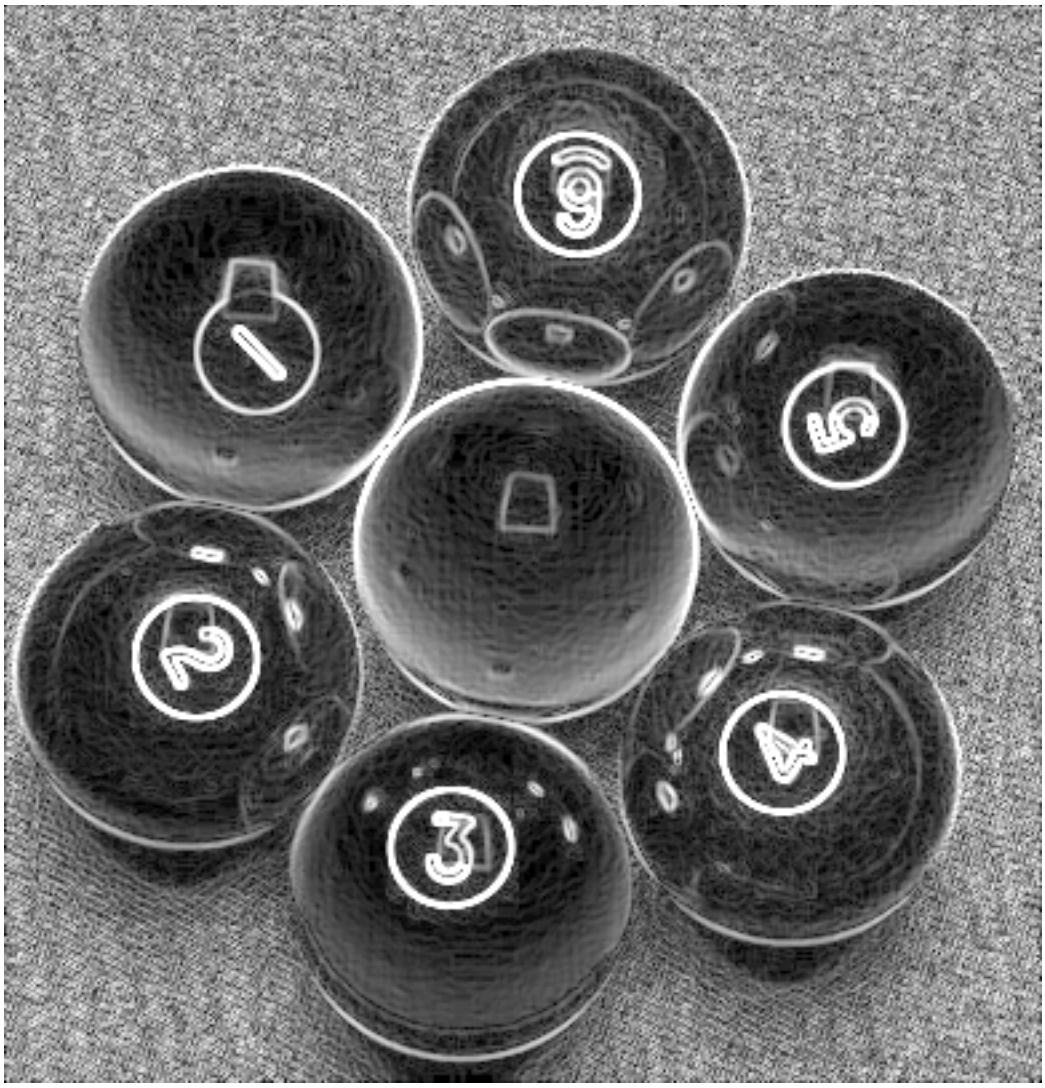

```
> img := Matrix(ToGrayscale(Import("Billiard.jpg"))):
> nRows, nCols := LinearAlgebra:-Dimension(img);
```

$$nRows, nCols := 404, 389 \tag{2.1}$$

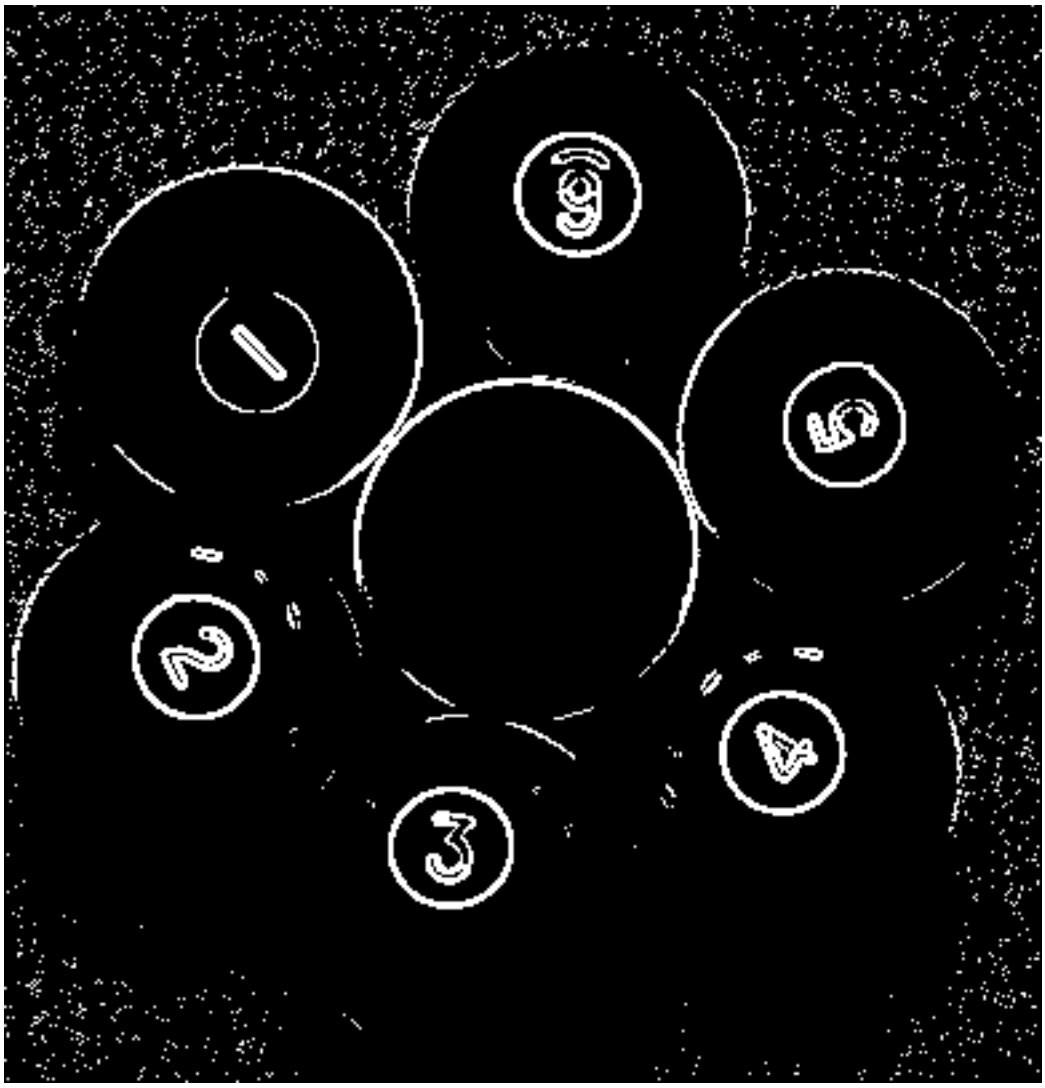
```
> Embed(img);
```




```
> Gx_sobel := Matrix ([[1,2,1], [0,0,0],[-1,-2,-1]]):
  Gy_sobel := Matrix ([[-1,0,1],[-2,0,2],[-1,0,1]]):
> img_x := Convolution (img, Gx_sobel):
  img_y := Convolution (img, Gy_sobel):
  img := Array (abs (img_x) + abs (img_y), datatype=float[8]):
> Embed(img);
```



```
> img := Threshold(img,0.7,low=0,high=1,method=both):
> Embed(img);
```



Notice the white spots spreading throughout the background of the image, they are the patterns on the pool table. These noises will also be considered a valid edge pixel to vote in the voting procedure despite they are not part of any circle objects. They slow down the voting procedure by increasing the data size and add potential source of errors to the result. Possible solutions include applying noise-removal filters like Gaussian filter. In this application, we will proceed with the current image to reserve the contours of the balls.

Hough transform

We start by locating all the edge pixels (i.e. non-zero pixels after the edge detection) in the image. Recall that we strengthened the edge pixels with a Threshold() command, so we will look for pixels with values of 1.

```
> countPixels := proc(M)
            local r,c,i,j,row,col:
            row := Array([],datatype=integer[4]);
            col := Array([],datatype=integer[4]);
            r,c := LinearAlgebra:-Dimensions(M);
            for i from 1 to r do
```

```
                    for j from 1 to c do
                        if M[i,j] = 1 then
                            ArrayTools:-Append(row,
  i, inplace=true):
                            ArrayTools:-Append(col,
  j, inplace=true):
                        end if:
                    end do:
            end do:
            return row,col:
  end proc:
> row,col := countPixels(img);
```

$$row, col := \left[\begin{array}{c} 1 .. 7971\ Array \\ Data\ Type: integer_4 \\ Storage: rectangular \\ Order: Fortran\_order \end{array}\right], \left[\begin{array}{c} 1 .. 7971\ Array \\ Data\ Type: integer_4 \\ Storage: rectangular \\ Order: Fortran\_order \end{array}\right] \quad \textbf{(3.1)}$$

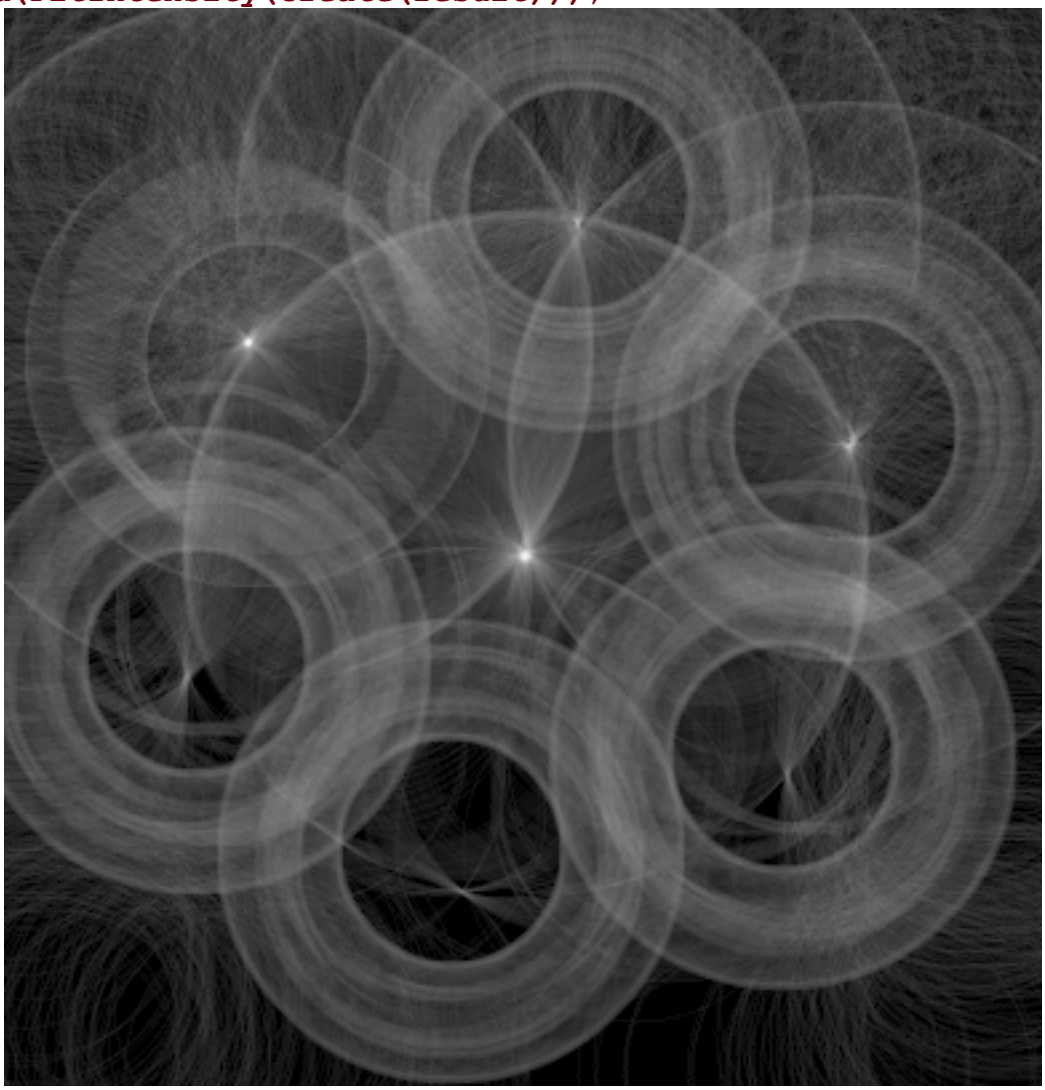
We want to iterate through each edge pixel and record their votes. In order to do so, we need to understand the parameterization of circles and build a corresponding accumulator. Notice that a circle is characterized by three pieces of information, the x, y coordinates of its centre and the radius, i.e. x = c_x + r*cos(θ), y=c_y + r*sin(θ). If we know or can estimate the radius of the circles existing in the image, we can carry out voting procedure in c_x-c_y space, where a point (x_1, y_1) is denoted by a circle in the c_x-c_y space centered at (x_1, y_1) with known radius r. An intersection of two circles in $c_x - c_y$ space will then represent a circle located at the intersection that passes through two points. In this way, a grid with 300 intersections is a circle that goes through 300 edge pixels, meaning that it is likely a solid contour circle in the original image.We use an accumulator that shares the width and height of the original image to record the intersections.

```
> circle := proc(r,c,len,col::Array(datatype=integer[4]),
  row::Array(datatype=integer[4]),rad,res::Array(datatype=integer
  [4]))
            local i,j,k:
            for k from 1 to len do
                    for i from 1 to r do
                        for j from 1 to c do
                            if ceil(sqrt((i-row[k])
  ^2+(j-col[k])^2))=rad then
                            res[i,j] := res[i,j]+1:
                            end if:
                        end do:
                    end do:
            end do:
  end proc:
> Ccircle := Compiler:-Compile(circle):
> circularHough := proc(M,len,r,c,rad,row,col)
            local i,theta,x,acc,circ:
```

```
            acc := Array(1..r,1..c,datatype=integer[4]):
            Ccircle(r,c,len,col,row,rad,acc);
            return acc;
  end proc:
> result := circularHough(img,numelems(row),nRows,nCols,65,row,
  col):
```

The command above is initiating a voting procedure with a predicted radius of 65. The voting process will take about a minute to finish. So while you are waiting, let's look at the case when the radius is unknown. A straightforward solution is to test a range of possible radii with a 3-dimensional $c_x - c_y - r$ parameter space. This results in a 3-dimensional accumulator, where each layer of the accumulator is essentially $c_x - c_y$ space accumulator. However, this might be very expensive due to the amount of computation needed. The number 65 used above comes from a lucky guess and three following trial runs, it should produce a good estimation of the six balls' radius.

```
> Embed(FitIntensity(Create(result)));
```



We can see that there are a couple of bright spots, those are intersections of most votes and thus the detected centres. Notice that the three balls in the lower half of the image

are not "detected" compare to the upper part. This is because after we removed most of their contours after the threshold() command in the preprocessing section.

Result

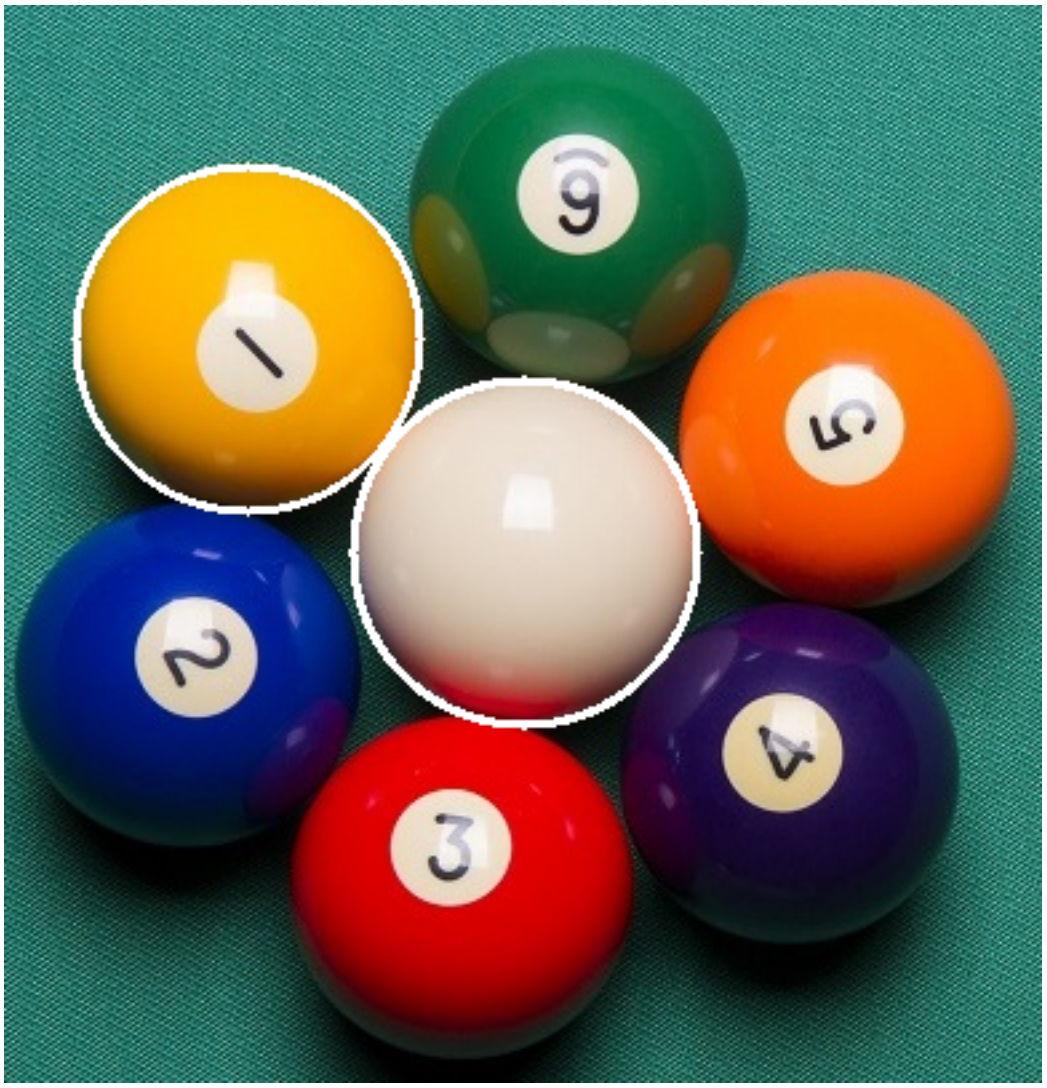
We will extract the detected centres by filtering the voting result with a threshold value of 200, which gives 12 candidates with most votes.

```
> filtered := map(x->piecewise(x<200,0,1), result):
> row1,col1 := countPixels(filtered);
```

$$row1, col1 := \begin{bmatrix} 1 \,..\, 12\ Array \\ Data\ Type: integer_4 \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix}, \begin{bmatrix} 1 \,..\, 12\ Array \\ Data\ Type: integer_4 \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \tag{4.1}$$

Let's project the detected circles to the image.

```
> Cpred := proc(M::Array(datatype=float[8],order=C_order),x,y,rad,
  r,c)
            local i,j,k:
            for i from 1 to r do
                      for j from 1 to c do
                                if ceil(sqrt((i-y)^2+(j-x)^2))=
  rad then
                                          M[i,j,1] := 255:
                                          M[i,j,2] := 255:
                                          M[i,j,3] := 255:
                                end if:
                      end do:
            end do:
  end proc:
> pred := Compiler:-Compile(Cpred):
> for i from 1 to numelems(col1) do
            pred(original,col1[i],row1[i], 65, nRows, nCols):
  end do:
> Embed(original);
```

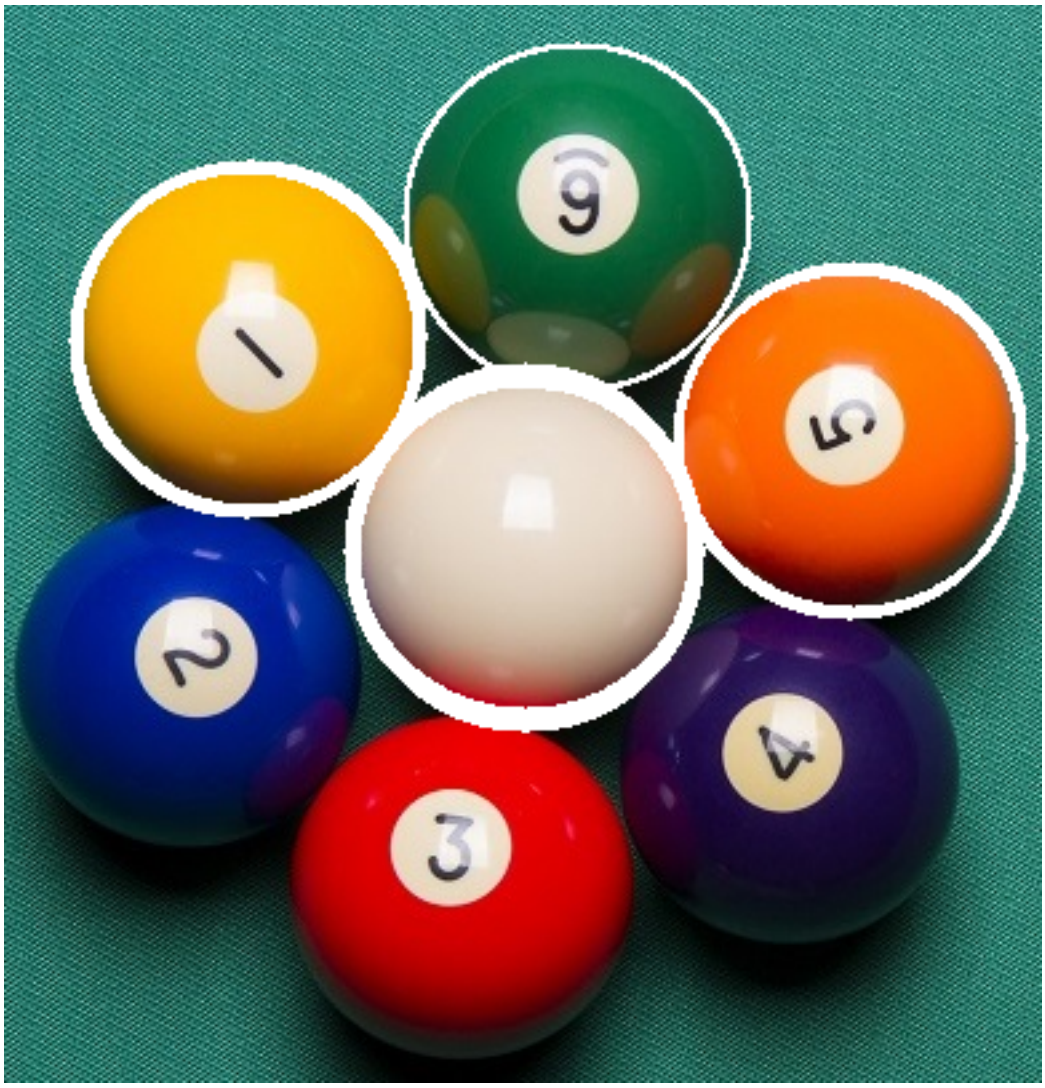


We can see that only two of the six balls are detected, what if we lower the threshold value and allow for more candidates?

```
> row2,col2 := countPixels(map(x->piecewise(x<100,0,1), result));
```

$$row2, col2 := \left[\begin{array}{c} 1 \,..\, 65\ Array \\ Data\ Type: integer_4 \\ Storage: rectangular \\ Order: Fortran\_order \end{array}\right], \left[\begin{array}{c} 1 \,..\, 65\ Array \\ Data\ Type: integer_4 \\ Storage: rectangular \\ Order: Fortran\_order \end{array}\right] \quad \textbf{(4.2)}$$

```
> for i from 1 to numelems(row2) do
            pred(original,col2[i],row2[i], 65, nRows, nCols):
  end do:
> Embed(original);
```



Now the green ball and the orange ball are also included, but the growing white bands around the white and yellow balls are evidences that a lot of these candidates are repetitive "close guesses". This is because simply filtering the accumulator is not an optimal way of extracting the result. The ultimate goal of the task is to extract the local maxima points so there are no repetitions.

Extra Fun

Try the above procedure to identify the smaller circles from the image.
The author's BEST record so far:


9
1
5
2
3
4