Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

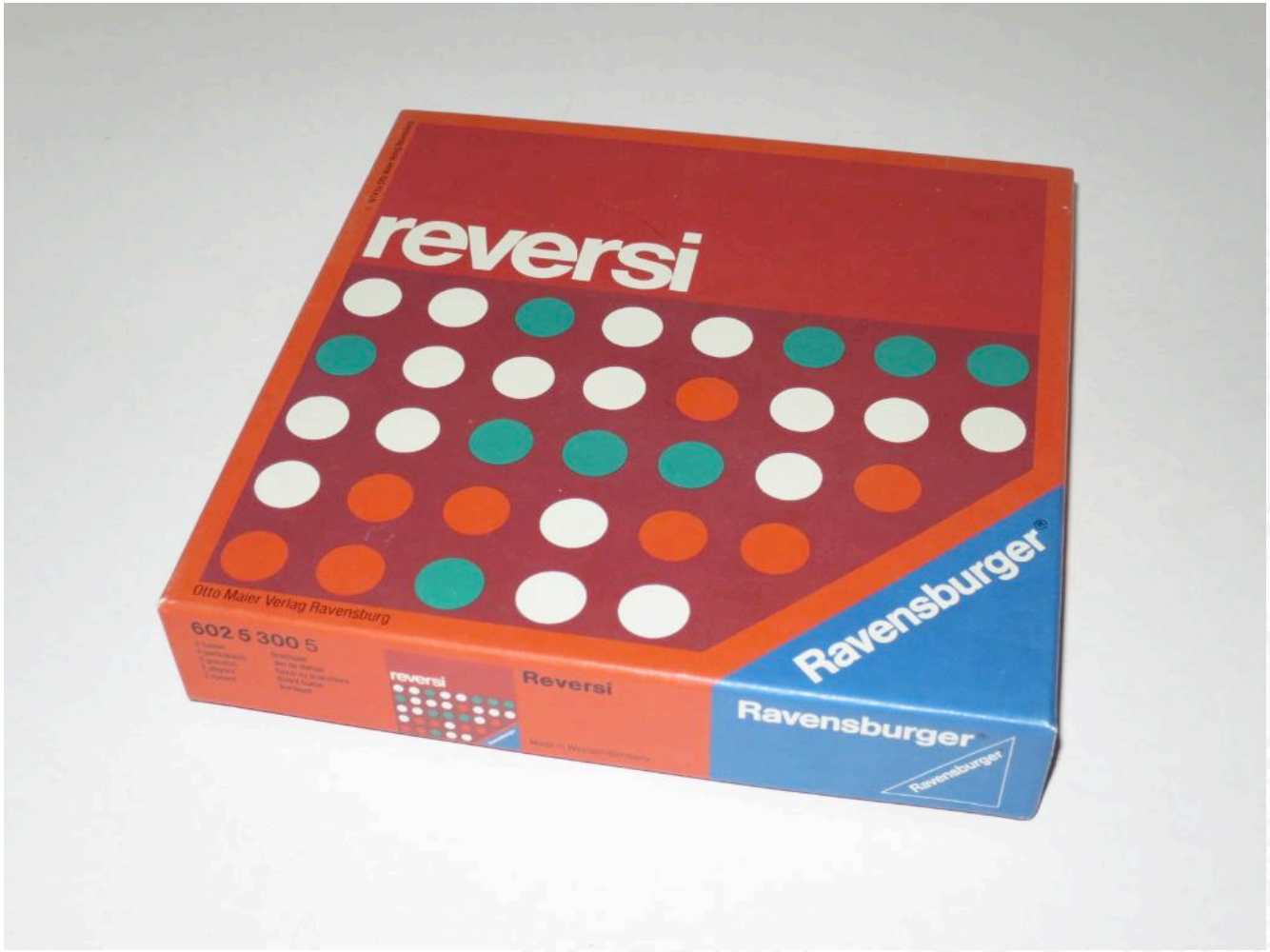
MIN-Faculty
Department Informatics
Scene Analysis and Visualisation (SAV)

Efficient Applicative Programming Environments for Computer Vision Applications

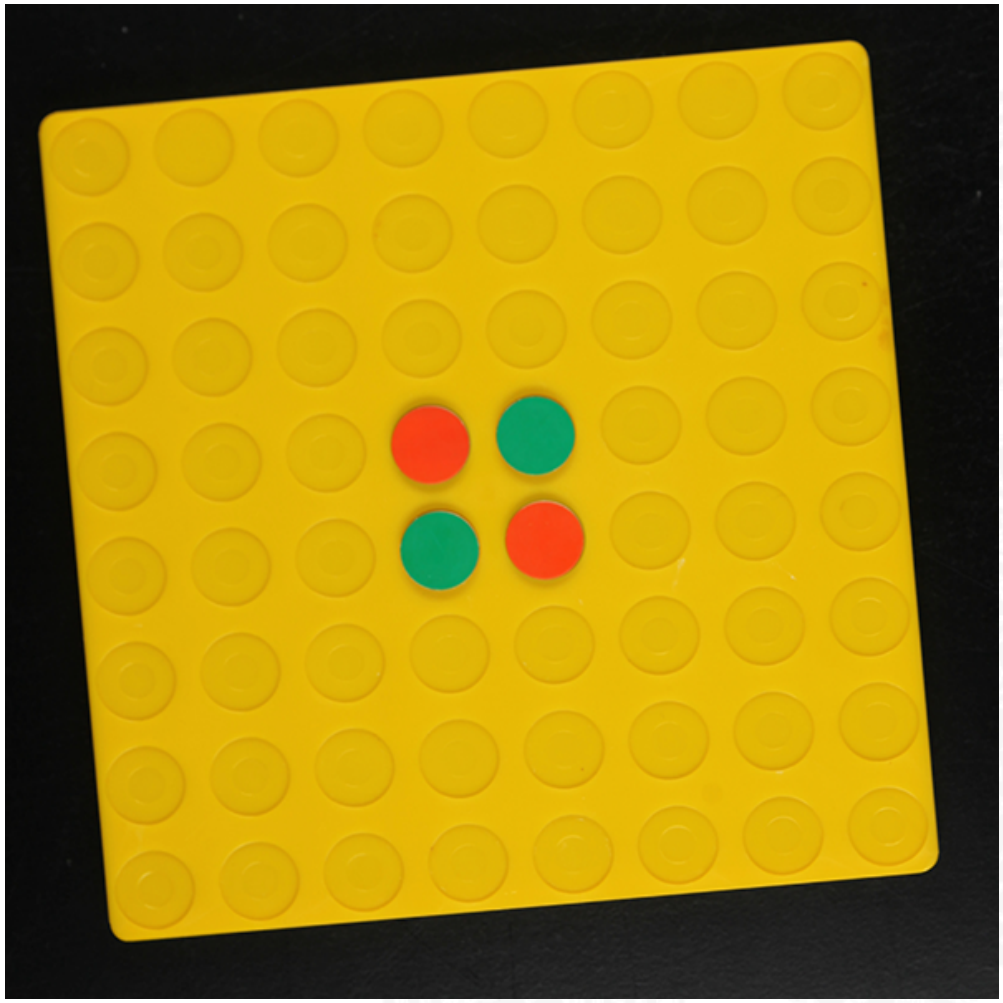
Integration and Use of the VIGRA Library in Racket

Dr. Benjamin Seppke,
Prof. Dr. Leonie Dreschler-Fischer

Task 1: Chose a board game



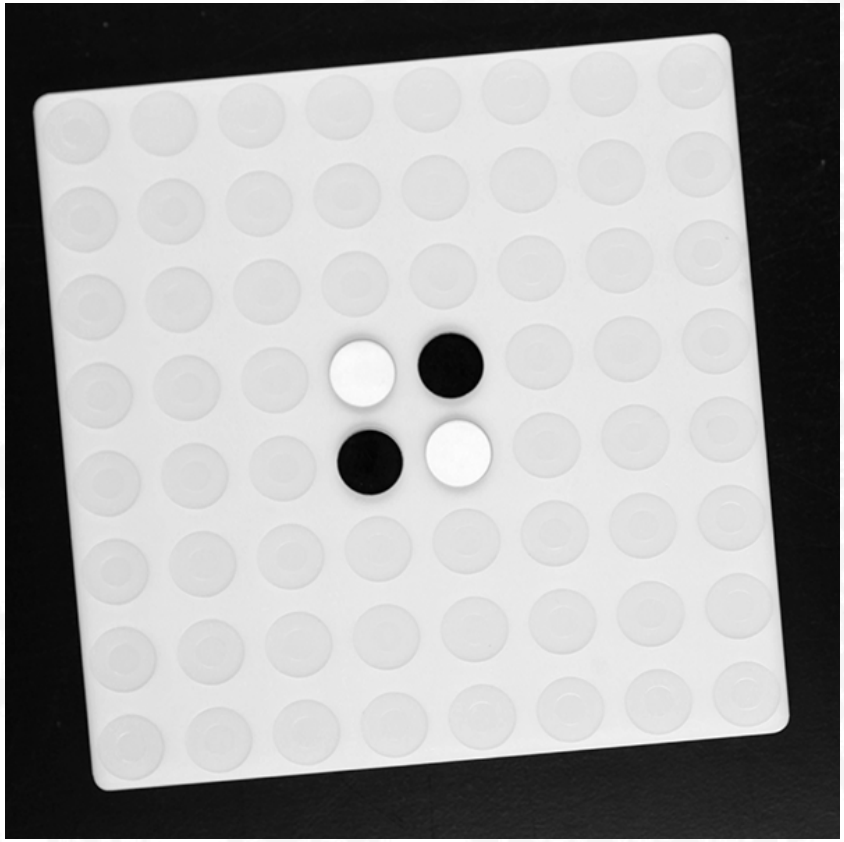
Task 2: Take images



Task 3: Detect the game board inside the image

Transform to greyvalue image

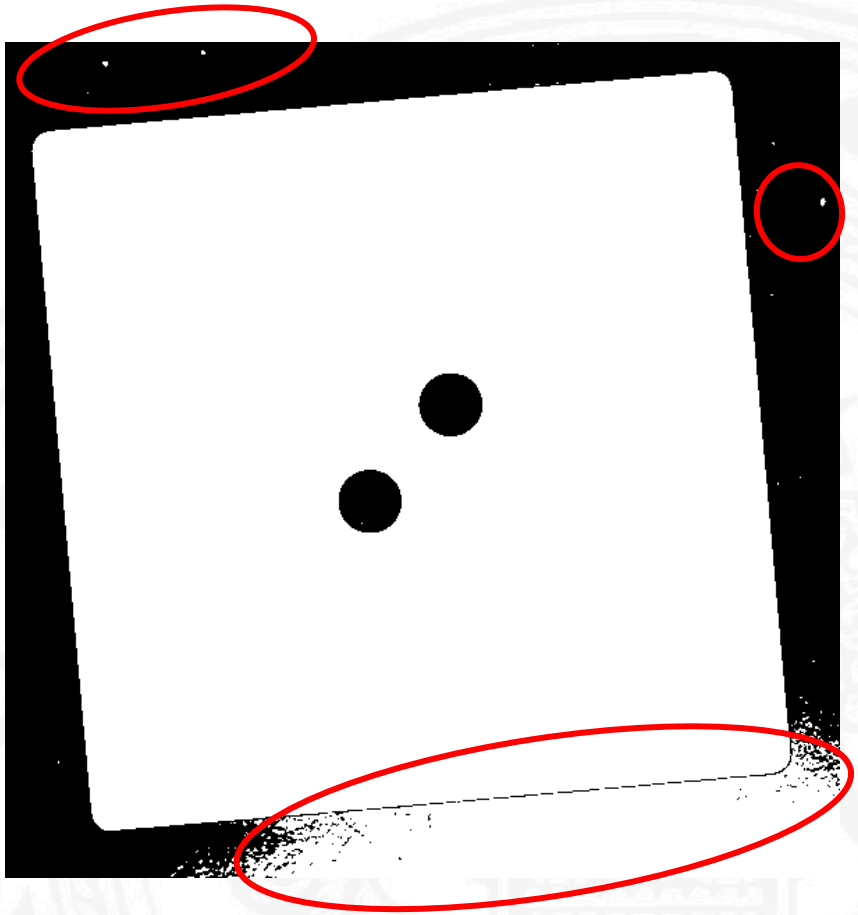
```
(define img_gray
   (image->red img))
```



Task 3: Detect the game board inside the image

Using a threshold:

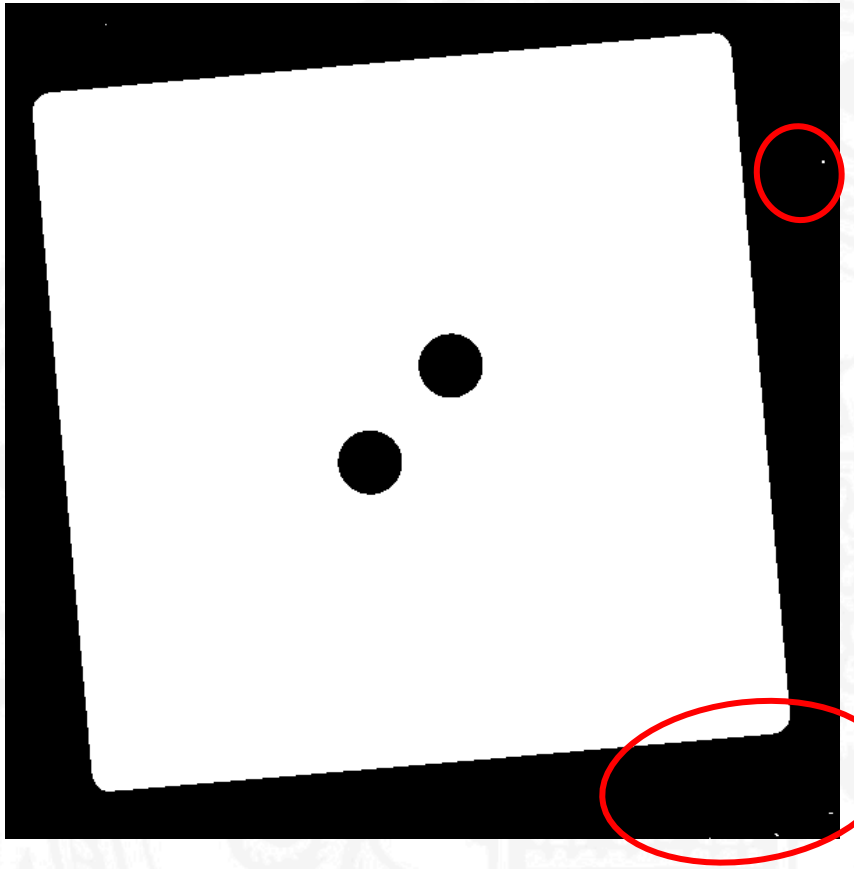
```
(define (threshold v t)
   (if (< v t)
       0.0
       255.0))

(define thresh_img25
   (image-map
      (curryr threshold 25)
      img_gray))
```



Task 3: Detect the game board inside the image

Using a threshold:

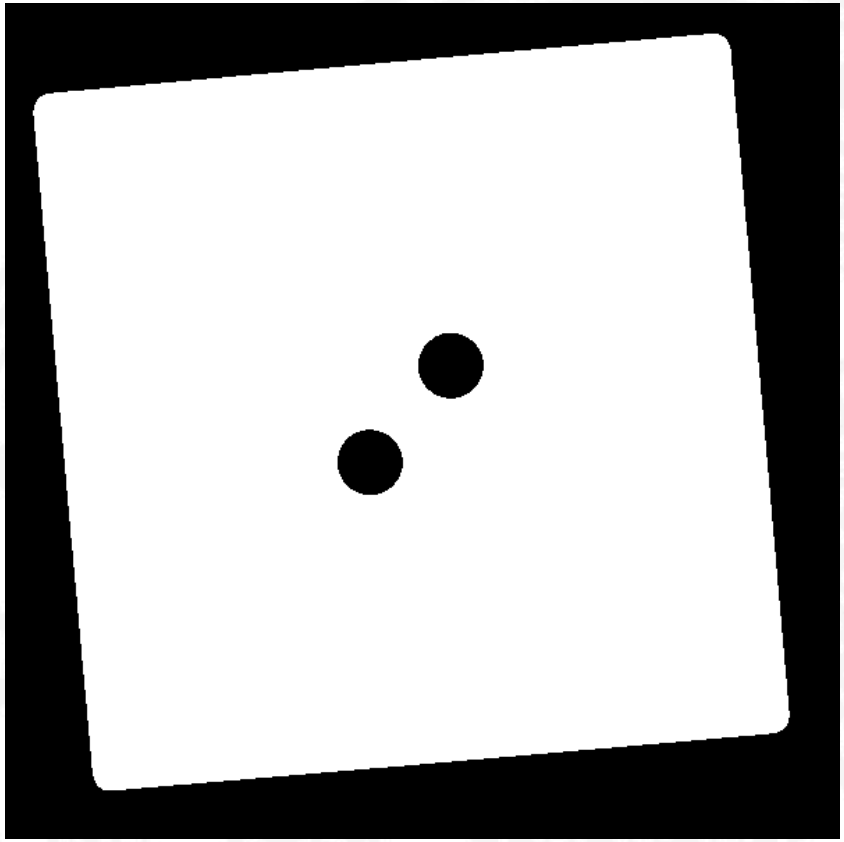
```
(define (threshold v t)
   (if (< v t)
       0.0
       255.0))

(define thresh_img50
   (image-map
      (curryr threshold 50)
      img_gray))
```



3. Detect the game board inside the image

Using a threshold:

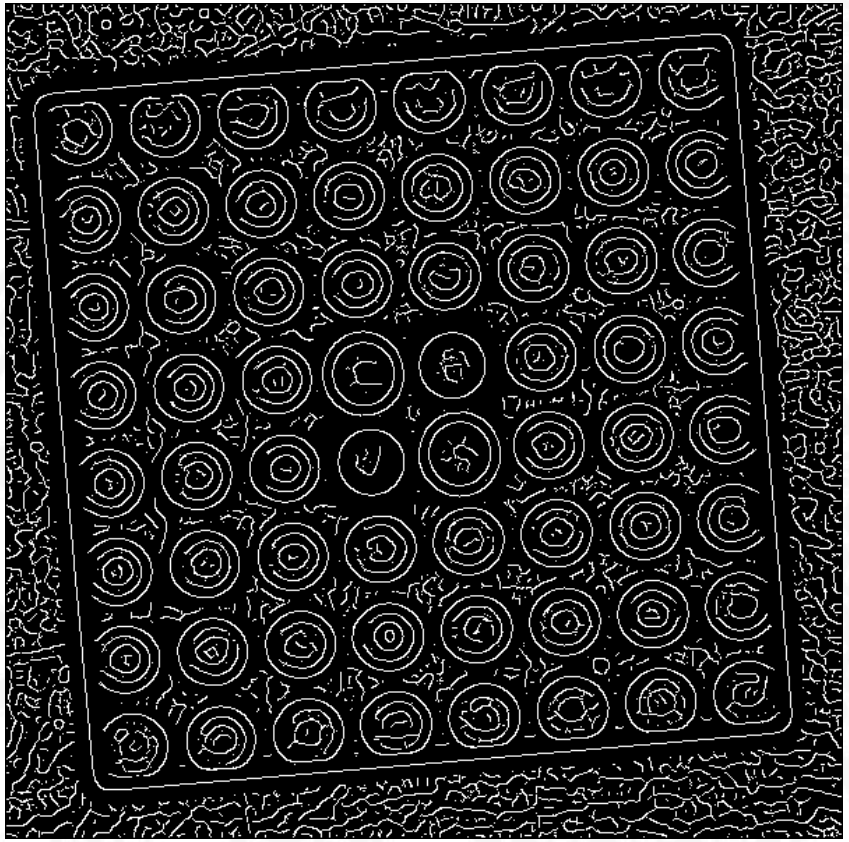
```
(define (threshold v t)
   (if (< v t)
       0.0
       255.0))

(define thresh_img100
   (image-map
      (curryr threshold 100)
      img_gray))
```



Task 3: Detect the game board inside the image

Using the Canny edge detector:

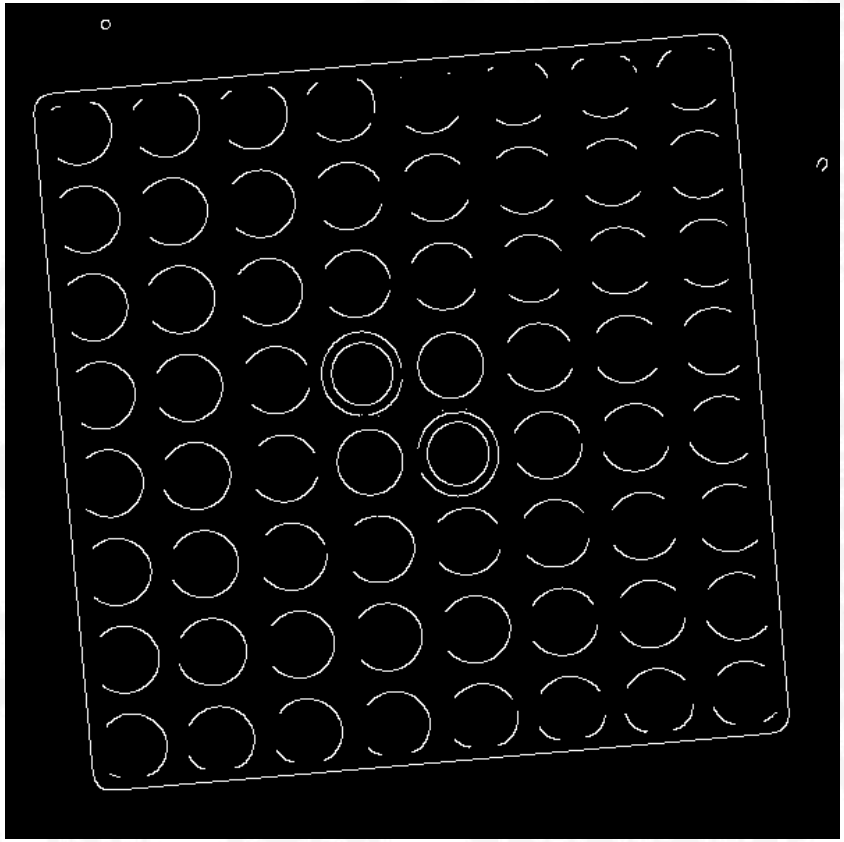
```
(define canny_img2
   (cannyedgeimage
      img_gray
      3.0          ;; scale
      0.0          ;; threshold
      255.0))      ;; marker
```



Task 3: Detect the game board inside the image

Using the Canny edge detector:

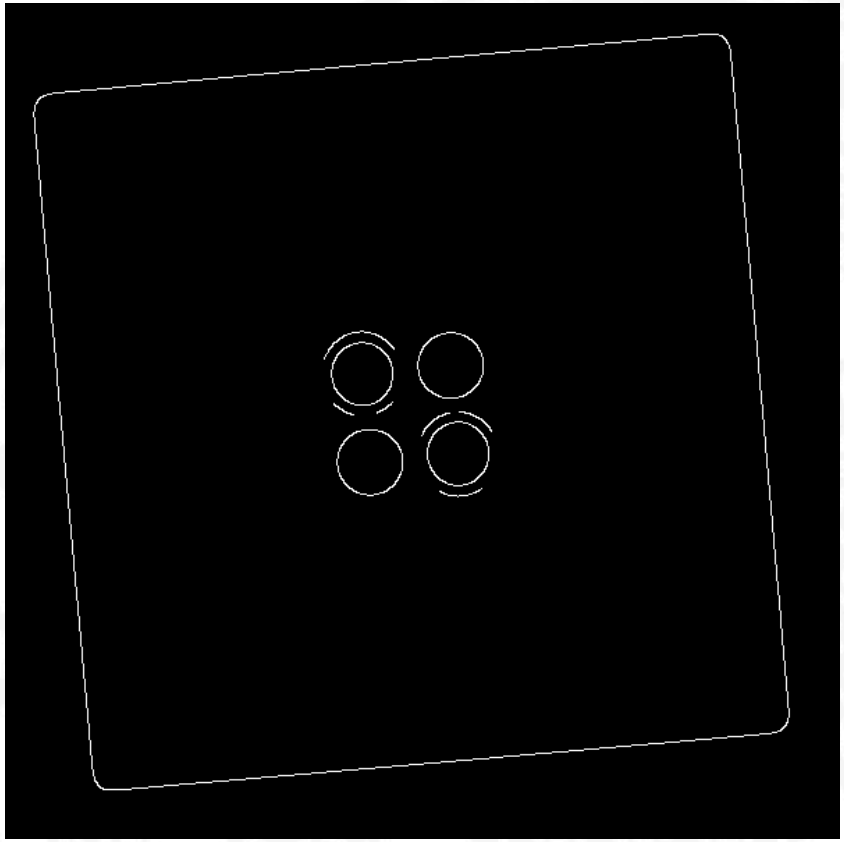
```
(define canny_img2
   (cannyedgeimage
      img_gray
      3.0          ;; scale
      1.0          ;; threshold
      255.0))      ;; marker
```



Task 3: Detect the game board inside the image

Using the Canny edge detector:

```
(define canny_img2
   (cannyedgeimage
      img_gray
      3.0          ;; scale
      2.0          ;; threshold
      255.0))      ;; marker
```



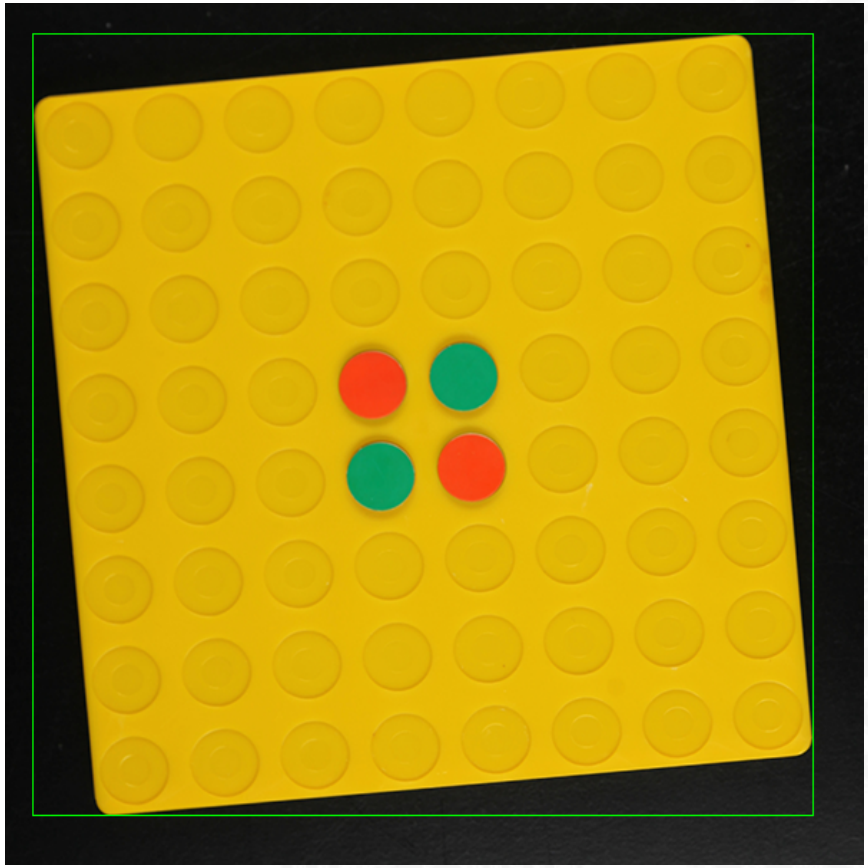
Task 4: Separate the game board from the background

Based on either result, find the bounding box:

```
(define (findBBox x y col bbox)
   (when (> (car col) 0.0)
      (begin  (when (> x (vector-ref bbox 2))
                    (vector-set! bbox 2 x))
              (when (< x (vector-ref bbox 0))
                    (vector-set! bbox 0 x))
              (when (> y (vector-ref bbox 3))
                    (vector-set! bbox 3 y))
              (when (< y (vector-ref bbox 1))
                     (vector-set! bbox 1 y)))))
(define canny_bbox
   (vector (image-width img) (image-height img) 0 0))
;; idx:            0:left, 1:top, 2:right, 3:bottom

(image-for-each-pixel (curryr findBBox bbox) canny_img2)
```

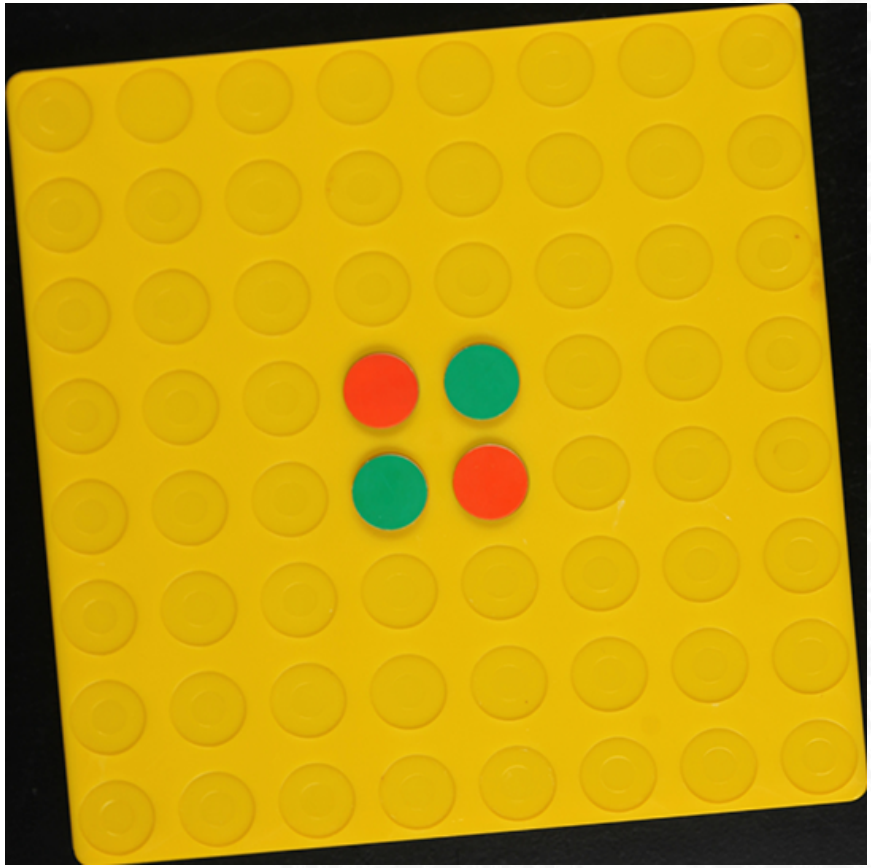
Task 4: Separate the game board from the background

Based on either result, find the bounding box:



Task 4: Separate the game board from the background

Crop according to the bounding box:



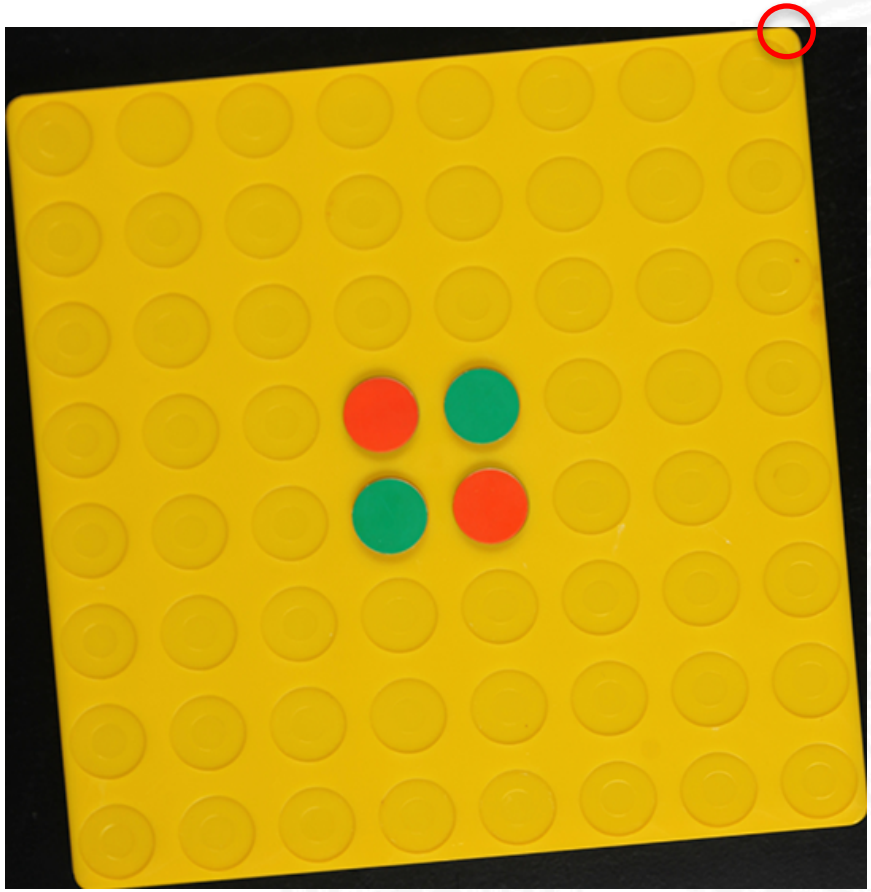
Finding the rotation

- Find first non-black pixel in row

```
(define (findFirstPixelInRow img x1 x2 row)
  (let ((not_found
                  (= (apply max (image-ref img x1 row))
                     0)))
    (if not_found
       (if (= x1 x2)
           #f
            (findFirstPixelInRow
               img
               (+ x1 (sgn (- x2 x1))) x2 row))
        x1)))
(define canny_left
        (- (findFirstPixelInRow canny_img
                (vector-ref canny_bbox 0)
                (vector-ref canny_bbox 2)
                (vector-ref canny_bbox 1))
                (vector-ref canny_bbox 0)))
```

Task 4: Separate the game board from the background

Example of finding first non-black pixel in row:



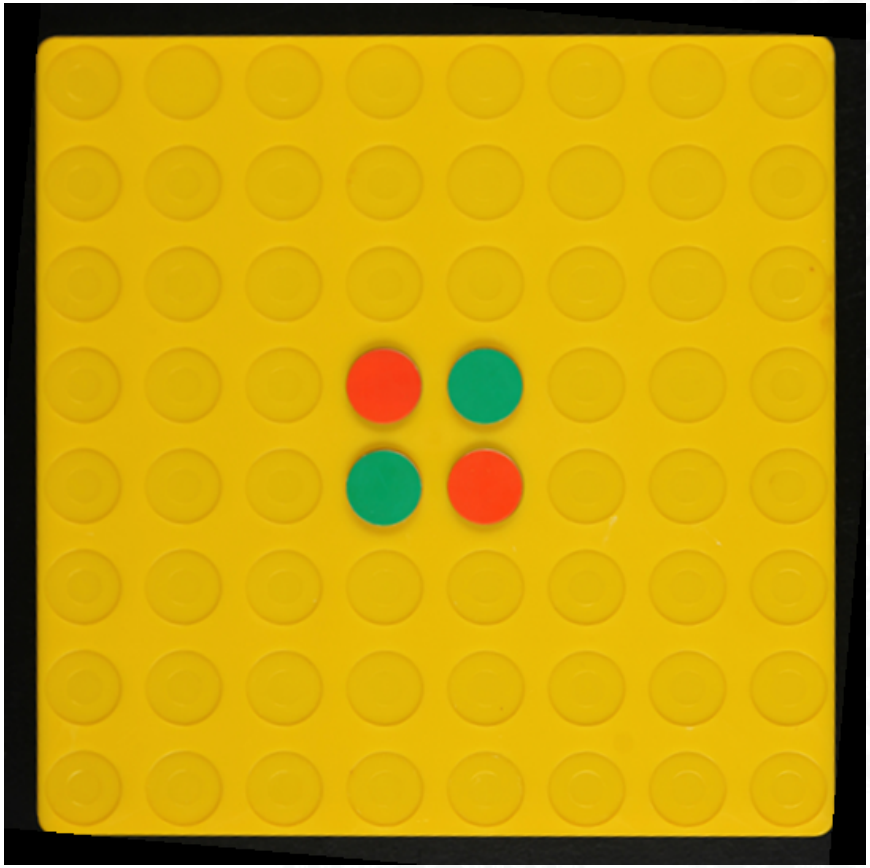
Detect Rotation for round corners

- Use maths to approx. round corners

```
(define canny_bbox-width
        (-   (vector-ref canny_bbox 2)
                (vector-ref canny_bbox 0)))
(define canny_corner-size
        (round (* canny_bbox-width 0.025)))
(define canny_right
        (- (findFirstPixelInRow canny_img
                (vector-ref canny_bbox 2)
                (vector-ref canny_bbox 0)
                (vector-ref canny_bbox 1))
         (vector-ref canny_bbox 0)))
(define canny_pos (+ (/ (+ canny_left canny_right) 2)
                 canny_corner-size))
(define canny_angle
        (/ (* (atan  (- canny_bbox-width canny_pos)
       canny_pos ) -180) pi))
```

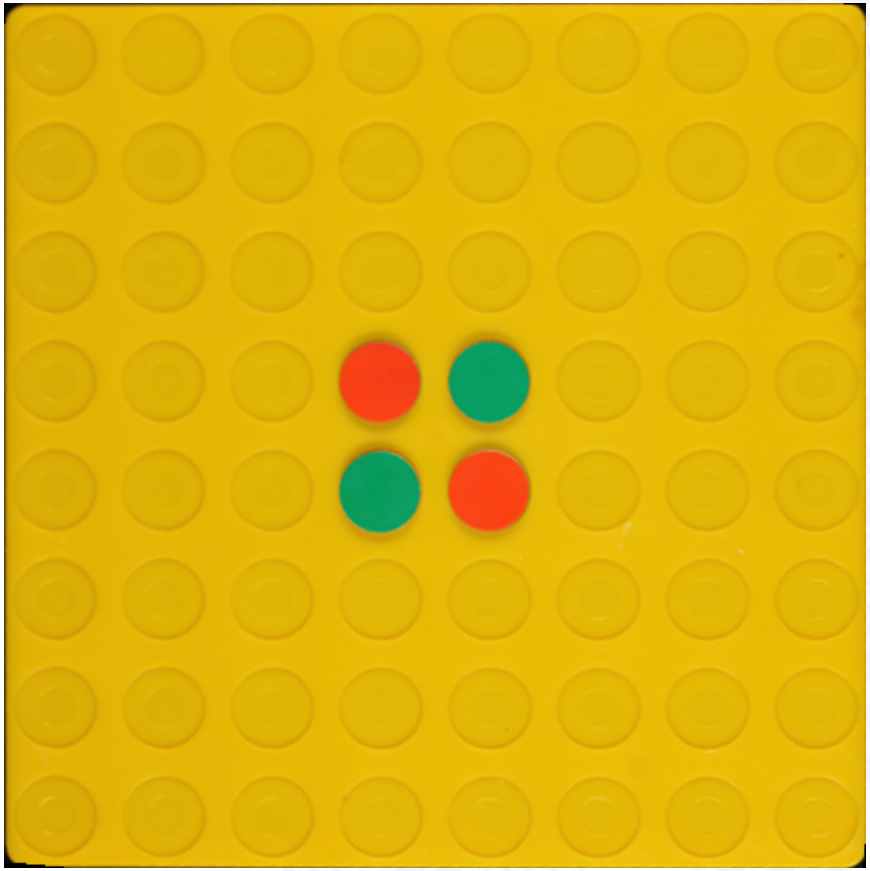
Task 4: Separate the game board from the background

Detect rotation and correct/rotate the cropped image:



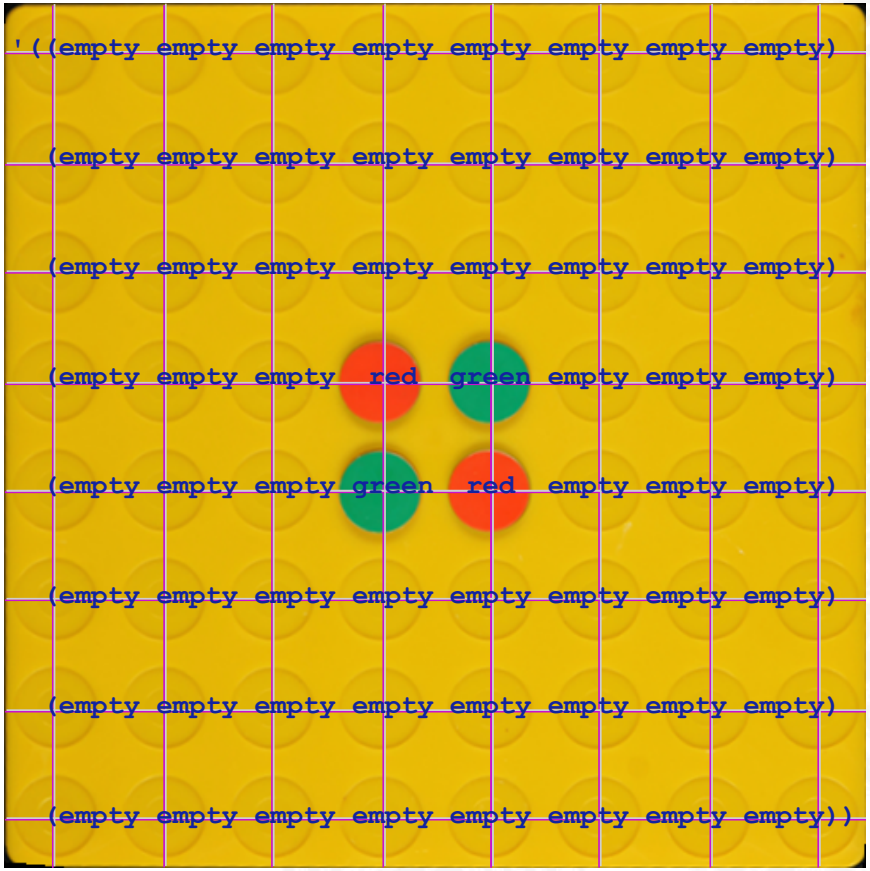
Task 4: Separate the game board from the background

Find bounding box and crop (again):



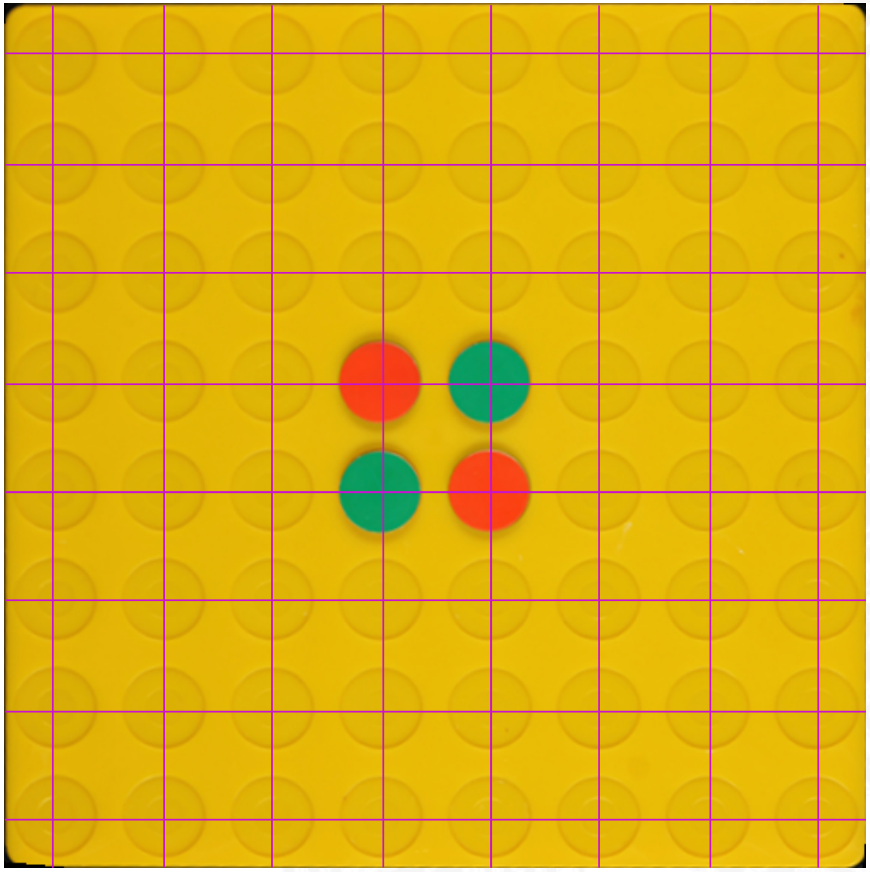
Task 5: Derive the Game State

Transfer from image colour values to game state:



Task 5: Derive the game state

Define a “game grid” and sample positions:



Task 5: Derive the Game State

Transfer from image values to game state:

```
'((empty empty empty empty empty empty empty empty)
  (empty empty empty empty empty empty empty empty)
  (empty empty empty empty empty empty empty empty)
  (empty empty empty  red  green empty empty empty)
  (empty empty empty green  red  empty empty empty)
  (empty empty empty empty empty empty empty empty)
  (empty empty empty empty empty empty empty empty)
  (empty empty empty empty empty empty empty empty))
```