Orange Detection Algorithm

Marshall Asch
University of Guelph
Guelph, Canada
masch@uoguelph.ca

Grant Douglas
University of Guelph
Guelph, Canada
gdouglas@uoguelph.ca

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—Image Processing, Fruit, Applied image colour segmentation, Automated counting**

I. Introduction

Counting and identifying the number of ripe fruits is important to automate harvesting fruits when they are ripe and to identify how many ripe fruits there are on a tree.

II. Ease of Use

A. How we Selected the threshhold vales

III. Algorithms

based off of this [1]

A. Values

B. Stages

1) Step 1: Run NDI

2) Step 2: In the RGB colour space run a mean filter.

3) Step 3: convert the image to the YCrCb colour space.

run a threshold on the Cr values. how the threshold is discusses in sec. III-A.

4) Step 4: run a threshold on the Cb values. how the threshold is discusses in sec. III-A.

5) Step 5: Then do a bit-wise and to create a mask

$$pixel_{final} = pixel_{NDI} \wedge pixel_{mean} \wedge pixel_{Cr} \wedge pixel_{Cb} \quad (1)$$

6) Step 6: Do a circle count on the mask image

IV. Results

V. Discussion

A. Limitations

When we were testing our algorithm we noticed that it worked significantly better on images larger than 1300x1300px. It also does not work on images where the diameter of the oranges is greater than half of the screen size, or if the oranges are very small (less then 150px). We determined that this short coming can be easily overcome by using image resolutions above that size. In the application where this algorithm would be used the image quality would be known and the zoom of the image would be consistent allowing the mentioned shortcomings to be overcame.

VI. Psudo code

A. Circle Filtering

The accumulator resolution was set to 2.5, experimentation showed this was the best value on out test image set. the minimum distance between circles was set to 150px. the canny edge detection parameters were both set to 100, inorder to use a single threshold, An arbitrary value was chosen between 0 and 255 because the image has been binarized to black in white in an earlier step. The minimum radious was set to 0 to allow small circles to still be detected, the max was set to 3000 so that all circle will be captured then filtered out lalter. Through experimentation we discovered that if we set the max radius to a smaller value (even tho all the actual circles in the tested images had a radius less than 500px) then valid circles would be missed.

On line 5 in algorithm 1, the percentage of pixels in the image that have been identified as part of the orange is taken to the number of pixels within the circle that are not part of the orange. Details of this are shown in algorithm 3. The White count algorithm takes into account part of the detected circle that is outside of the frame of the image, if less than 40% if the circle is outside the frame then that portion that is outside is assumed to be filled, if more then 40% is outside then it is assumed to not be filled.

on line 6 of algorithm 1, the percentage of pixels that are not already taken is counted as part of another orange. This reduces the number of overlapping circles that are detected. This can be seen in figure 3 where the image on the right does not filter the circles by how much they overlap and the one one on the right does. Details of this are shown in algorithm 2

References

[1] A. Payne, K. Walsh, P. Subedi, and D. Jarvis, "Estimation of mango crop yield using image analysis – Segmentation method," *Computers and Electronics in Agriculture*, vol. 91, pp. 57–64, Feb. 2013.

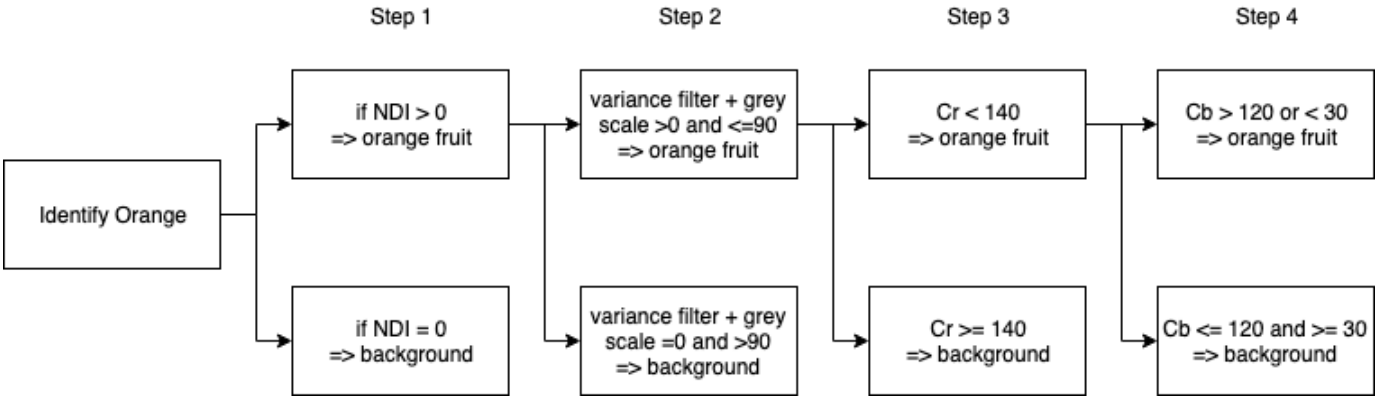
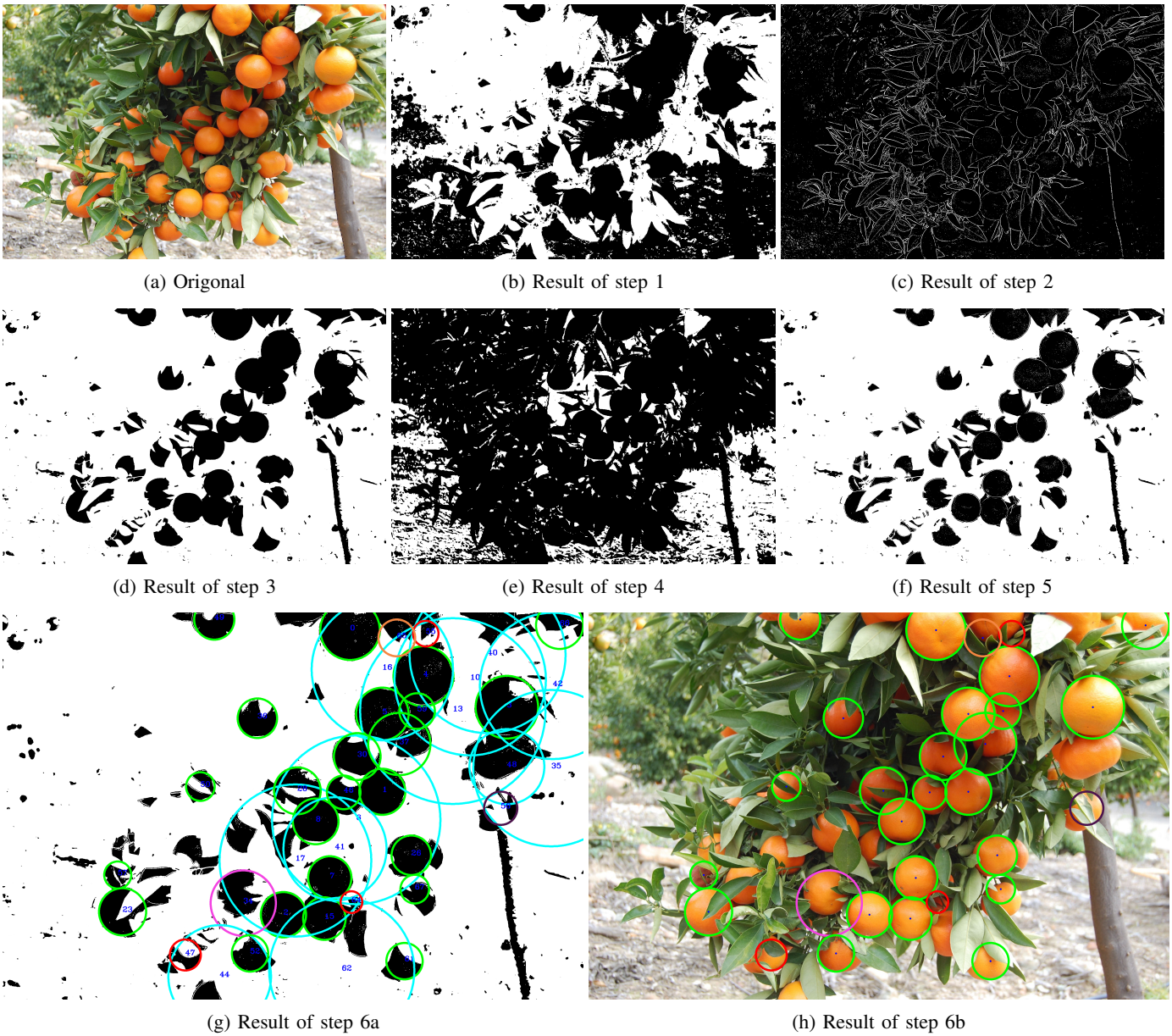


Fig. 1: Outline of the proccess of the stages of the algorithm



(a) Origonal (b) Result of step 1 (c) Result of step 2

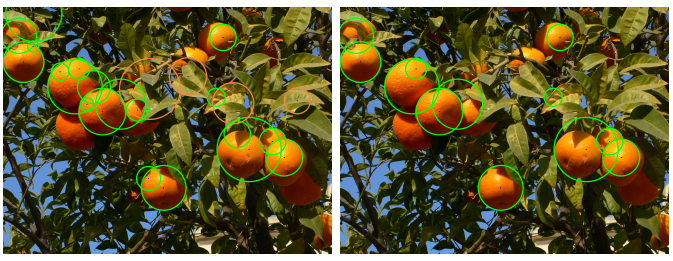
(d) Result of step 3 (e) Result of step 4 (f) Result of step 5

(g) Result of step 6a (h) Result of step 6b

Fig. 2: An example image going through each step to detect the oranges within the image

Algorithm 1: Orange counting algorithms

```
function CountCircles (img);
  Input  : Binarized image of the oranges img
  Output: NumOranges
img = medianBlur(img, 3) circles = HoughCircles(img,
  2.5, 150, 100, 100, 0, 3000)
found = fillZero(img.size) averageR = 0
for circle ∈ circles do
    ratio = countWhitePixels(img, circle)
    notTakenRatio = countNotTaken(found, circle)
    if circle.radius > img.size/4 ∨ ratio < 20 then
        continue
    /* make sure that the radius does not differ too
      much */
    if (averageR = 0) ∨ (averageR · 0.5 ≤
      circle.radius ≤ averageR · 1.5 then
        /* very confident this is an orange*/
        if ratio ≥ 50 then
            averageR = (averageR + circle.radius) / 2
            markFound(found, circle)
            markCircle(img, circle, color.GREEN)
        end
        else if ratio ≥ 30 then
            if averageR · 0.8 ≤ circle.radius ≤
              averageR · 1.2 then
                markCircle(img, circle, color.ORANGE)
            else if circle.radius ≤ averageR · 1.4
              then
                markCircle(img, circle, color.PURPLE)
            else
                markCircle(img, circle, color.RED)
        end
        else
            /* not filled enough, but close size*/
            if averageR · 0.8 ≤ circle.radius ≤
              averageR · 1.2 then
                markCircle(img, circle,
                  color.MAROON)
        end
    end
end
```



(a) Filtered out overlapping circles (b) Oranges with overlapping circles

Fig. 3: An example image going through each step to detect the oranges within the image

Algorithm 2: Determine if too much of the circle is already taken by another circle

```
function CheckOverlap
  (foundMask, circle, img, whitePixels);
  Input  : mask where the taken regions are set to 1 and
           untaken set to 0 foundMask
  Input  : the circle that is being checked circle
  Input  : the binarized image of oranges img
  Input  : percentage of pixels that are white
           whitePixels
  Output: ifthecircleshouldbeignored
if whitePixels ≤ 50 then
    return True
circleMask = fillMask(img.size, circle)
areaNotTaken = 0
areaNewOrangeInFrame = 0
areaFilledNotTaken = 0
for i ∈ img.height do
    for j ∈ img.width do
        if ¬foundMask[i, j] ∧ circleMask then
          areaNotTaken++ ;
        if circleMask then
          areaNewOrangeInFrame++ ;
        if ¬foundMask[i, j] ∧ circleMask ∧ img[i, j]
          then areaFilledNotTaken++ ;
    end
end
/* check that orange is distinct */
if
  areaNotTaken/areaNewOrangeInFrame · 100 <
  10 then return False ;
/* check that non overlaping section is partly filled */
if
  areaFilledNotTaken/areaNewOrangeInFrame ·
  100 < 10 then return False ;
return True
```

Algorithm 3: Count the number of pixels that are within the circle that have been marked as orange

function OrangeCount $(circle, img,)$;
Input : the circle that is being checked $circle$
Input : the binarized image of oranges img
Output: percentage of pixels that are white $whitePixels$
circleMask = fillMask(img.size, circle)
circleArea = $\cdot\pi \cdot circle.radius^2$
numOutOfFrame = circleArea
numOrange = 0
for $i \in img.height$ **do**
  for $j \in img.width$ **do**
    numOutOfFrame–
    if $img[i, j] \wedge circleMask$ **then** numOrange++ ;
  end
end
/* check that enough is in frame*/
if $numOutOfFrame/circleArea \cdot 100 > 40$ **then** numOrange += numOutOfFrame ;
percentFilledInFrame = numOrange / numTotal * 100
percentFilled = numOrange / (numTotal - numOutOfFrame) * 100
return min(percentFilledInFrame, percentFilled)