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. Algorithms

The algorithm that was developed is a variant of the one proposed by Payne for mango crop detection [1]. The algorithm consists of six steps, five of which apply masks to the image, and the sixth counts the total crop yield. The image is analyzed for orange coloured regions within the RGB and YCbCr colour spaces to create colour masks. This is then combined afterwards to produce a single orange level mask. Finally, circular regions are counted to determine how many oranges are seen within the image.

A. Values

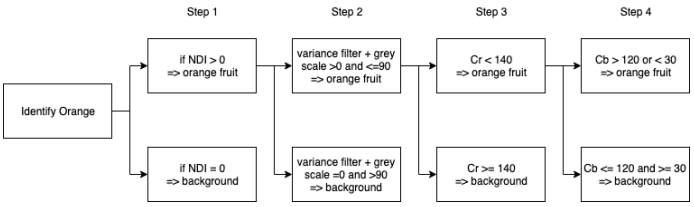


Fig. 1: Outline of the proccess of the stages of the algorithm

B. Stages

1) Step 1: The first step in this algorithm is to produce a normalised index between the red and green channels. The index is calculated such that $NDI = (g-r)/(g+r)$ for each pixel in both channels. The values are then thresholded where $NDI > 0$. This step helps provide emphasis on regions with higher redness levels, and less emphasis on foliage.

2) Step 2: In Payne's algorithm, a $3x3$ variance filter is used for each colour channel, then combined and converted to grayscale, and finally thresholded to meet a desired property. This process provided to be insufficient for oranges, and instead a $3x3$ adaptive mean threshold filter was used. This new filter was able to remove excess foliage from the image, as well as remove areas that are irrelevant to the oranges, such as the sky.

3) Step 3: The image is then converted into YCbCr colour space, and Cr and Cb channels extracted to threshold from. The Cr channel measures the red difference within the image, so any component which favours a medium to high red level will work. As such, the threshold value of $pixelvalue >= 140$ is used, as this was seen to be the average minimum Cr value found across multiple images containing oranges.

4) Step 4: The Cb channel is used to determine the blue difference within the image. This means that a lower to medium value is acceptable for oranges, as this would produce an orange colour when combined with the Cr channel. As such, a threshold condition of $30 <= pixelvalue <= 120$ is used.

5) Step 5: With all of the masks created, they are then combined together to form a final mask using equation 1. This mask will select only orange components of the image.

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

6) Step 6: The final step is to count the crop yield on the tree. The process is to first apply a Gaussian blur to the image and perform a Hough transform to detect circles. These circles are then analyzed to determine how much of the region contains the selected pixel value, and is of reasonable size. Once each circle is checked, the final result will be an image with orange regions are circled.

C. Circle Filtering

Once the mask is generated after step 5, that mask will be used to count the number of circles, each circle representing an orange fruit. Pseudo code for this is shown below in Algorithms 1, 2, and 3. The first step in the process is to apply a Gaussian blur to the mask to soften the edges for the edge detection algorithm. Since the Hough transformation will generate more circles then there actually are oranges, we filtered the number of circles down based on calculating a rolling average of the radius, how many pixels are within the circle, how much of the detected circle is within the frame and

how much overlap there is. This is shown below in algorithm 1 and discussed below.

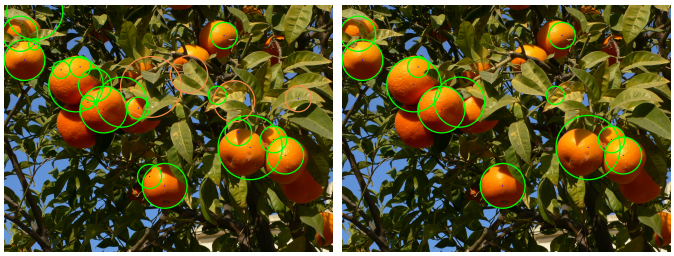
Algorithm 1: Orange counting algorithms

```
1  function CountCircles (img);
   Input  : Binarized image of the oranges img
   Output: NumOranges
2  img = medianBlur(img, 3) circles = HoughCircles(img,
     2.5, 150, 100, 100, 0, 3000)
3  found = fillZero(img.size) averageR = 0
4  for circle ∈ circles do
5      ratio = countWhitePixels(img, circle)
6      notTakenRatio = countNotTaken(found, circle)
7      if circle.radius > img.size/4 ∨ ratio < 20 then
8          continue
9      /* make sure that the radius does not differ too
         much */
10     if (averageR = 0) ∨ (averageR · 0.5 ≤
         circle.radius ≤ averageR · 1.5 then
11         /* very confident this is an orange*/
12         if ratio ≥ 50 then
13             averageR = (averageR + circle.radius) / 2
14             markFound(found, circle)
15             markCircle(img, circle, color.GREEN)
16         end
17         else if ratio ≥ 30 then
18             if averageR · 0.8 ≤ circle.radius ≤
                 averageR · 1.2 then
19                 markCircle(img, circle, color.ORANGE)
20             else if circle.radius ≤ averageR · 1.4
                 then
21                 markCircle(img, circle, color.PURPLE)
22             else
23                 markCircle(img, circle, color.RED)
24         end
25         else
26             /* not filled enough, but close size*/
27             if averageR · 0.8 ≤ circle.radius ≤
                 averageR · 1.2 then
28                 markCircle(img, circle,
                     color.MAROON)
29         end
30     end
31 end
```

The values that were selected for the Hough transform were chosen as; $dp = 2.5$, $minDist = 150px$, $thresholds = 100$, $minimumradius = 0$, and $maximumradius = 3000$. The dp is an inverse ratio used in the accumulator which is used to detect circles, we chose this value based on running the algorithm with multiple images and this yielded the best result. The value of 150px was chosen to be the minimum distance because most of the images were large enough that this value did not cause a significant number of the oranges to be missed. This is discussed in more detail in section IV-A. The 2 Hough transform parameters that are used for Canny Edge detection were set to 100, an arbetrary value between 0 and 255, since the base image that was being used had been previously binarized to contain only values of 0 and 255 (not including the Gaussian blur on the edges). [2].

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 2 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



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

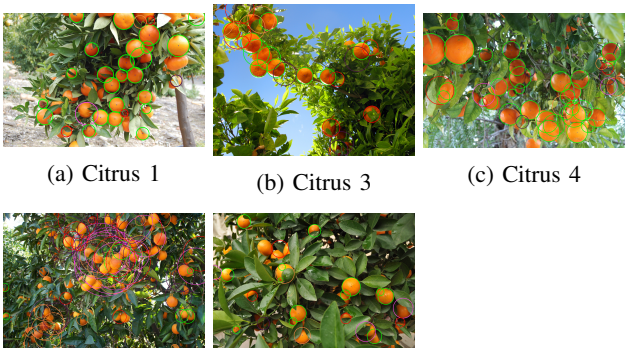
Fig. 2: An example image going through each step to detect the oranges within the image, (Note: only the green, confident detections are shown for the sake of simplicity)

The circle counting algorithm produces various different circles for the oranges as can be seen in fig. 3. Each circle colour represents how confident that the algorithm is that that circle is an orange. They are listed below in decreasing order of confidence.

- Green - over 50% filled, within 50% of the other green circles
- Orange - over 30% filled, within 20% of the green circle sizes
- Red - over 30% filled, smaller than 40% larger than the green circle sizes
- Maroon - under 30% filled, within 20% of the green circle sizes
- Purple - over 30% filled, larger than 40% bigger than the green circle sizes

III. Results

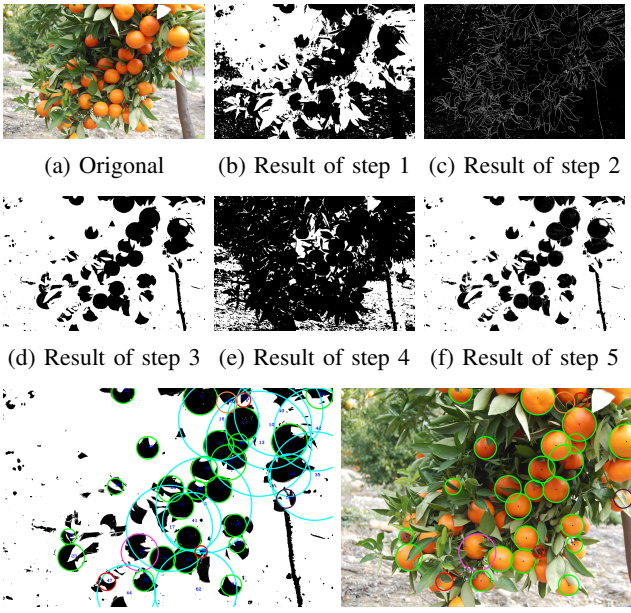
The algorithm was tested against a set of 8 images, each with varying brightness and foliage levels. Among all of



(a) Citrus 1 (b) Citrus 3 (c) Citrus 4

(d) Citrus 6 (e) Citrus 8

Fig. 3: The results of the circle counting 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. 4: An example image going through each step to detect the oranges within the image

the images, there is approximately a 73% success rate for detecting the oranges with high confidence. There are also results for each image where the algorithm is only moderately confident in the result, with each image having around 1-2. In each resulting image, oranges with high confidence are highlighted in a green circle, and those with moderate confidence are circled in orange. For example, the image ”citrus_1” in fig. 3a detects 24 out of the 32 possible oranges within the image, missing those that are mostly covered in foliage. Same thing can be seen for ”citrus_3” in fig. 3b , with 10 oranges detected out of the possible 14. It can be seen that the images with high levels of brightness cause the algorithm to become less accurate, with images ”citrus_6” in fig.3d and ”citrus_8” in fig. 3e having approximately 0.1% accuracy each. Finally, there are cases where an orange is counted twice, which can be seen in the images ”circles_3” in fig. 3b and ”circles_4” in fig. 3c .

IV. 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) This is a direct result of the value that was chosen for the minimum distance between circles in algorithm 1. 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.

V. Psudo code

A. Circle Filtering

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.

[2] V. K. Yadav, S. Batham, A. K. Acharya, and R. Paul, “Approach to accurate circle detection: Circular Hough Transform and Local Maxima concept,” in *2014 International Conference on Electronics and Communication Systems (ICECS)*, (Coimbatore), pp. 1–5, IEEE, Feb. 2014.

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 $whitePixels \leq 50$ **then**
  return True
circleMask = fillMask(img.size, circle)
areaNotTaken = 0
areaNewOrangeInFrame = 0
areaFilledNotTaken = 0
for $i \in img.height$ **do**
  for $j \in img.width$ **do**
    if $\neg foundMask[i,j] \wedge circleMask$ **then** areaNotTaken++ ;
    if $circleMask$ **then** areaNewOrangeInFrame++ ;
    if $\neg foundMask[i,j] \wedge circleMask \wedge img[i,j]$ **then** areaFilledNotTaken++ ;
  end
end
/* check that orange is distinct */
if $areaNotTaken/areaNewOrangeInFrame \cdot 100 < 10$ **then** return False ;
/* check that non overlaping section is partly filled */
if $areaFilledNotTaken/areaNewOrangeInFrame \cdot 100 < 10$ **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)