Machine Vision

Using *Open Computer Vision* to detect apples in images

282962 Robotics and Automation

Assessment 3: “Machine vision assignment”

Jamie Churchouse - 20007137

Contents

1 Introduction **2**
2 Methodology **3**
2.1 Open Computer Vision 3
2.2 Approaches 3
2.3 Foreground & Background 4
2.4 Hough Circles 4
2.5 Post Processing 5
3 Results & Discussion **6**
3.1 Efficacy 6
3.2 Strengths & Weaknesses 6
3.3 Improvements 6
4 Conclusion **7**
5 Appendix **8**
5.1 Code 8
5.2 Efficacy Table 9
5.3 Sample of Output Images 10

Note that underlined text denotes a hyperlink.

Save the trees - don't print this, please!

1 Introduction

Production lines are intricate systems that need to operate like clockwork processing thousands of items per hour. Pack houses for produce such as apples need to ensure that their lines are operating efficiently, and ensure errors are caught and rectified. One of these checks is to ensure that each box contains the right number of apples, and the system only has a short period of time to calculate whether this is true; if it is not, there may be financial or reputational penalties for the company.

There are many types of sensors available on the market; some are simple, cheap, and provide minimal information, others are the exact opposite. In the case of detecting apples in boxes, there are a number of feasible options: weighing the boxes to see if they are the expected weight, probing points in the box to estimate volume, or even simply paying a human to count them, but these have major drawbacks. *Machine Vision* is the term used to describe the technique of a computer system processing images to extract useful data. In the case of counting apples, a single photo can be taken of the target, and then an algorithm can process it and return the number of apples to great accuracy in milliseconds - all without expensive or complicated sensor arrays, or holding up the production line.

The objective of this project is to develop an algorithm to locate and highlight apples in provided images, and display the number of them. Additional constraints include that the programme should process images from a directory, rather than explicitly from the code, and that the programme should be fully autonomous. Furthermore, I have defined explicit performance constraints: no false positives, and less than five percent false negatives. False positives could result in overcounting, whereas false negatives would result in undercounting. A shipment of produce that has more than expected would not negatively affect the reputation of the company, whereas under delivering could earn the company a poor reputation, hence the first constraint. The second constraint is the 95% efficacy. 95% is an arbitrary value due to the lack of any requirements in the brief, but I have set it to this value because it is a reasonable target to reach given the resources for this project.

2 Methodology

2.1 Open Computer Vision

Open Computer Vision (OCV) is an open source (hence *Open*) library for Python, C++, Java, and MATLAB, for Computer Vision (hence *Computer Vision*) processing. For its simplicity, and because it is what we've been taught in class, I will be using Python 3 and the corresponding OCV library for this project.

OCV's main features include the ability to read in images from files as matrices representing the colours of individual pixels, running operations on those matrices such as colour space conversions, blurs, and thresholds, and functions to add shapes and text onto an image. My solution will use a combination of these operations and techniques to count the number of apples in the images.

2.2 Approaches

The initial brainstorm for how to approach this problem involves four possibilities: machine learning, counting apple coloured pixels then dividing that by the average number of pixels in an apple, object detection, and 'looking for round things'

Machine learning totally bypasses OCV in favour of something I'm not so familiar with, and I feel it wouldn't be in the spirit of the exercise - but it would have been highly effective. The pixel counting system theoretically would give me a good number plus or minus an apple or two - which seems like too much of a disparity to the true number of apples to be of much use. Initial research into object counting looked promising, but the colour of the box in some areas made it difficult to separate it from the apples, and the time required to optimise the programme would have been more than I could afford, leaving us with 'looking for round things' approach.

This technique uses the *HoughCircles* (HC) function, which, on every point on a contour, 'draws' a circle centred on the point. These drawn circles overlap with creating a great magnitude at the centre of the object circle, which can be tested against a threshold and determined to be the centre of a circle in the input image. This video by Thales Sehn Körting shows a very intuitive animation as to how this operation works.

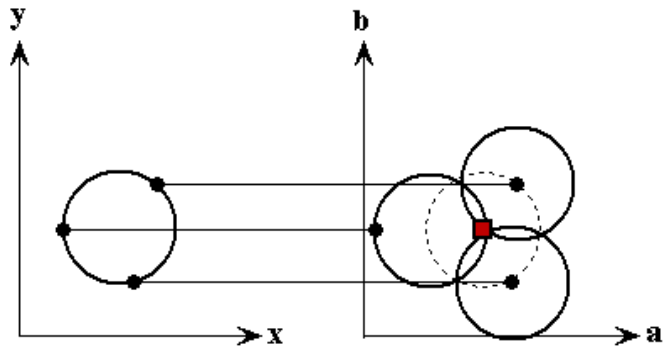
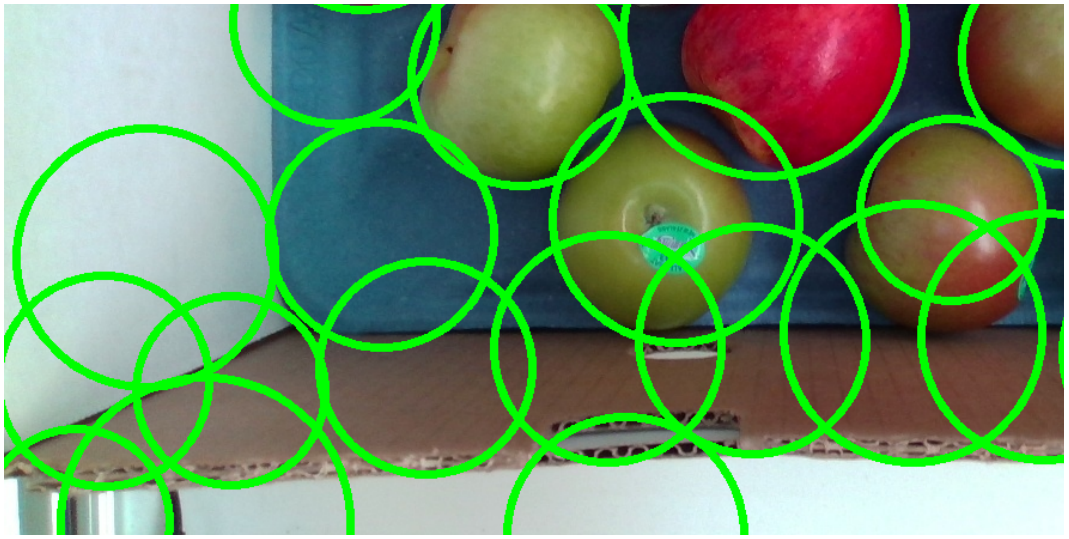


Diagram illustrating how Hough's Circle detection works (click image for link)

2.3 Foreground & Background

HC requires a greyscale image to work on, along with a few other parameters that will be discussed later. Just converting the sample images to greyscale then processing them with HC produced poor results, even after tweaking all the parameters. There was too much box and background being detected, and random ‘apples’ floating in free space.



Supernumerary detections

The main two reasons for these discrepancies are noise, and the input image having the wrong information. I then decided to explore blurring techniques to cut down on noise, and on removing as much “non apple colour” from the image as possible to reduce the amount of visible background and box.

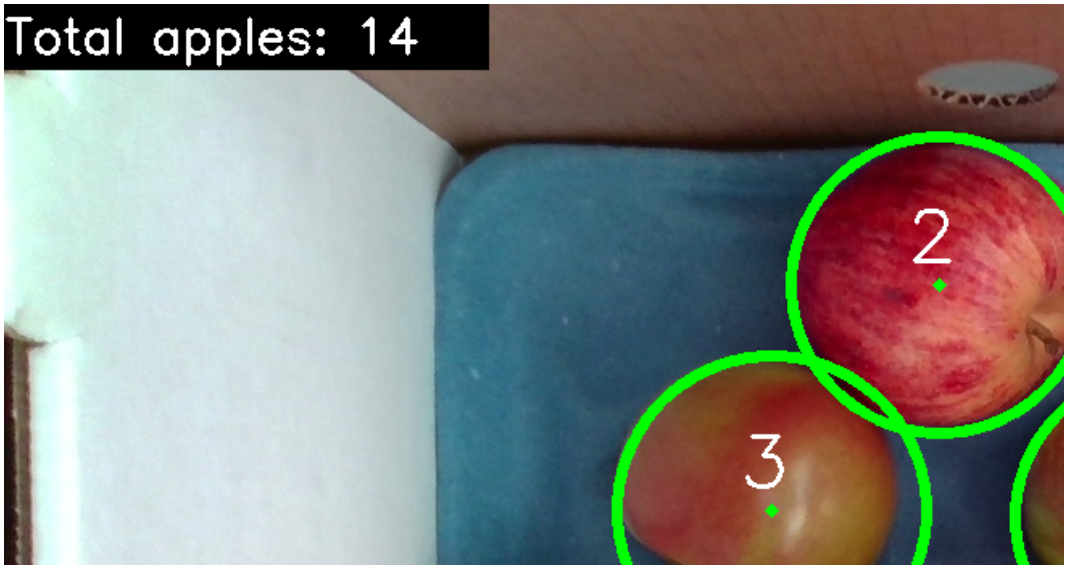
Converting the image to HSV colour space and running thresholding operations allowed me to generate a ‘mask’ with the entirety of the box floor and white side panels removed, but I was unable to completely isolate the apples. Dilation and erosion techniques helped reduce the presence of the box walls, but the HC function needed to carry the rest of the slack - thankfully there weren't many circular shaped sections in the mask over the box areas.

2.4 Hough Circles

HC takes in a few parameters that were very useful in eliminating illegitimate detections. The minimum and maximum radius parameters do exactly what you expect, they tell the function what the minimum and maximum sizes of the target circles can be. Because there was a small range of possible apple sizes, this pair of parameters cut down the list of valid detections very nicely. The minimum distance parameter defines the minimum distance from the centre of one circle to the centre of every other circle. As there were no overlapping apples in the sample images, I was able to use this parameter to remove more incorrect detections. The other two parameters, “param1” and “param2”, as far as I could tell, just affected the sensitivity of the function.

2.5 Post Processing

Now that the apples were being reliably and accurately detected, it was just a matter of tidying up the output images. I decided to create another directory to store the output images. I had already implemented a process to outline the detected apples with a circle, but I also wanted to add the ID number of the apple on it too for the viewer's benefit. Furthermore, I included a total in the top left corner of the image to indicate the number of detected apples in an easy to see and standard location.



Apple identifiers, total apple count

On that note, when I was writing the function to print the total number of apples on the image, I started by writing the comment "*# Print a total on the image*", and then the AI included in VS Code suggested the following command that worked exactly how I wanted it to - first try. This was a very cool thing to experience.

```
# Print a total on the image
cv2.putText(img_out,f"Total apples: {len(detections[0,:])}",(20,20),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255),2)
```

AI generated function

The full code, and output from V1.0, can be viewed on my GitHub linked in section 5.1.

3 Results & Discussion

3.1 Efficacy

As stated at the start of this report, my target efficacy is to correctly identify 95% of apples that are fully visible in the image without any false positives. To verify whether my programme had reached these targets, I manually counted the number of apples visible in each photo into a spreadsheet and compared them with what the programme attained. This table is available in section 5.2. During this process, I also verified that there were no false positives - that criteria was met.

Not only does my programme detect 95% of the full apples, it detects 99.1% of them! Detection of partial apples is far less impressive at less than 50%, but that was not the objective of this project, and it could easily be rectified by nudging the camera to a slightly better angle. A sample of output images can be viewed in section 5.3.

3.2 Strengths & Weaknesses

I'm very happy with the success of this exercise. The programme takes images from a predefined directory and iterates over each one without issue. The programme has been designed for robustness in that any exceptions will be handled within the programme itself rather than crashing. Lots of tweaking was required to a variety of parameters to get detection to work to this level, and the positivity of the results shows that it has paid off.

Because the algorithm detects circular shapes, any apple that is partially in frame has a realistically less than 50% chance of being detected. When I decided on this approach, I knew this would be a flaw, so ensured that every other element of the programme was robust to compensate, but it is still an area I would've liked to improve on should the resources have been available.

3.3 Improvements

This programme works by detecting circular shapes of red or green colour. While I haven't tested, I would believe that this system would be easily fooled by oranges, lemons, and other fruit that may have been misplaced from nearby equipment. Even the lighting may affect efficacy. Object detection may be more robust, but not perfect, however, I believe some form of machine learning algorithm would be the most reliable and cost effective solution in the long run. Realistically, a 5% inaccuracy, or even my 1%, would still amount to thousands of missed items over a shift. The efficacy of a real world system would need to be 99.99% at worst, hence a more robust system would be required.

4 Conclusion

The objective of this project has been to design a programme to detect and highlight the apples in a set of sample images. My solution exceeded my self defined target of 95% efficacy by detecting 99.1% of all fully visible apples without any false positives. This solution would not, however, stand up to a real world industrial environment due to the resilience that a system like that would require.

The code for this project, along with the output images from Version 1.0, can be found on my GitHub linked in section 5.1.

5 Appendix

5.1 Code

The repository for this project is available on my GitHub.

5.2 Efficacy Table

	ACTUAL			V 1.0	
ID	FULL	PART		full	part
0	10	4		10	0
1	12	2		12	0
2	8	4		8	4
3	11	3		11	3
4	12	2		12	1
5	9	3		9	2
6	9	3		9	2
7	9	3		9	1
8	8	4		8	2
9	10	4		10	4
10	9	4		8	2
11	9	4		8	2
12	13	2		13	0
13	10	2		10	0
14	13	2		13	0
15	10	2		10	0
16	13	2		13	0
17	10	2		10	2
18	9	6		9	2
19	9	4		8	2
20	14	3		14	1
21	13	3		13	1
22	10	6		10	4
23	16	2		16	1
24	14	4		14	1
25	10	6		10	4
26	12	4		12	3
27	13	3		13	0
28	16	2		16	1
29	12	6		12	2
	ACTUAL			V 1.0	
	FULL	PART		full	part
SUM	333	101		330	47
EFFICACY				99.10%	46.53%
TOTAL					86.87%

5.3 Sample of Output Images

Image ID: 01

Image ID: 16

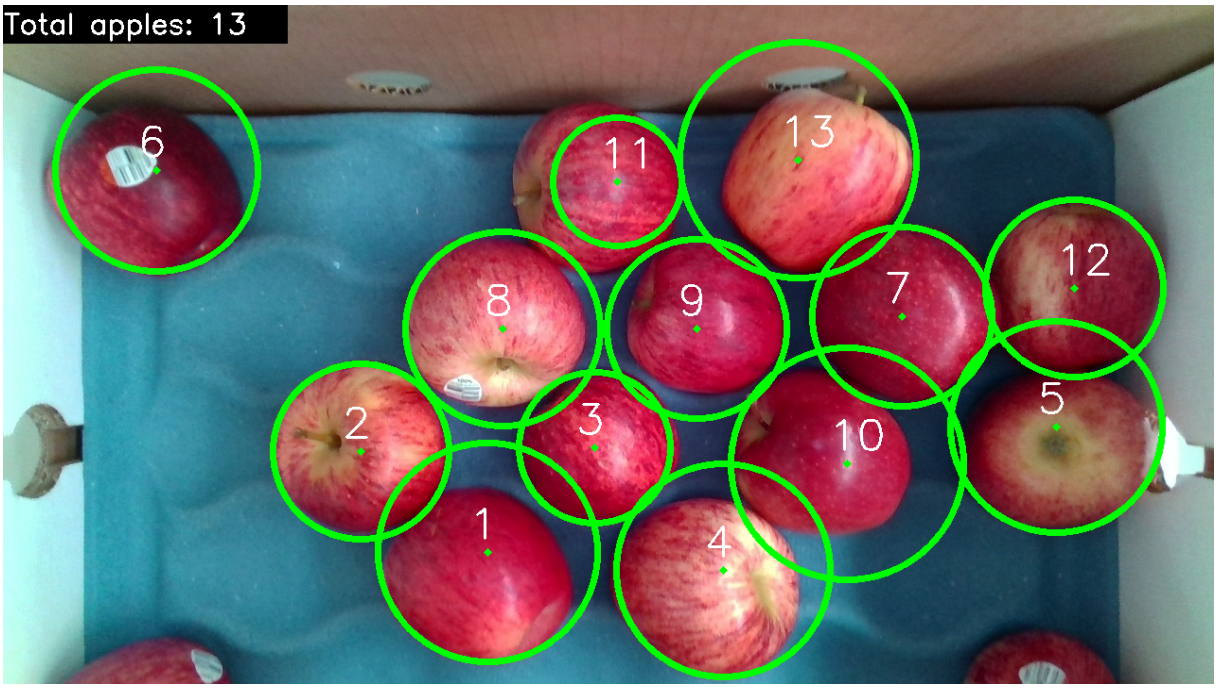


Image ID: 23