Digital Image Processing in MATLAB

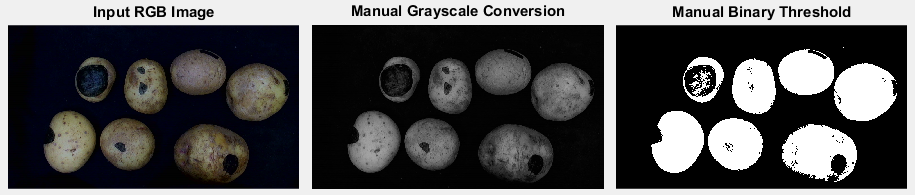
Task 1: Digital image analysis for static objects

MATLAB can be used to analyse stationary images in a myriad of ways. Even basic knowledge of image processing is enough to be able to recognise objects by their shape, size, colour, or a number of other attributes. The solution to this task utilises all three of these at various stages of the computer vision system model.

**Methods**

Given the input of an RGB colour image of potatoes on a mainly black background, the goal of the pre-processing stage was to enhance important features and suppress any noise. The image was manually converted to grayscale format by applying the standard NTSC weighted averages of the red, green, and blue channel values, giving the overall “Luminosity” of each pixel (Gonzalez & Woods, 2002; Cook, 2009). It was then converted to binary using a threshold transformation obtained through repeated testing, minimising data loss. All grayscale pixels above this threshold were set to one, and all others set to zero (see figure 1 below). This contributed to both the pre-processing and segmentation stages by starting foreground regions separation.

Figure 1



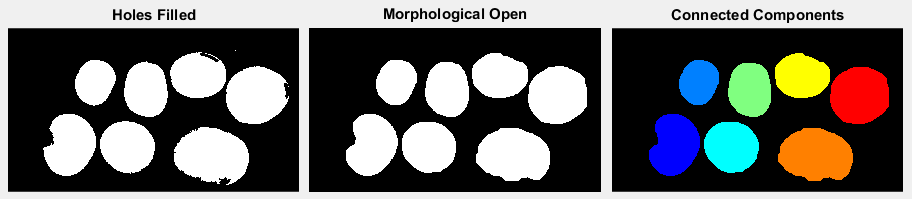
Filtering was then applied to deal with noise, utilising region filling and the MATLAB functions ‘imopen’ and ‘imclose’. Region filling was done in the custom ‘fillHoles’ function, after performing a ‘close’ operation to bridge gaps such as those on the edge of the top left potato in figure 1. The binary image was linearly inverted and MATLAB’s ‘bwconncomp’ function used to identify all connected components, now consisting of the background and all enclosed holes. All components but the background were set to have zero pixel values. The image was then inverted again with all enclosed holes filled (see figure 2).

A morphological ‘open’ was performed to ensure separate objects are not joined as a result of the initial ‘close’ operation. This eroded then dilated all components, first shrinking the objects before growing them, contributing further to the segmentation of the foreground image regions. Doing so smoothed the objects to remove thin boundary pixels, while also eliminating small outlier pixels during the erosion. It used a disk structuring element of radius 18, as this yielded the best results for potato-shaped objects. Differently shaped or sized elements tended to join separate potato objects together, producing false positive data.

The connected components of the final binary mask are then found by using ‘bwconncomp’ on the final binary mask, showing that the pre-processing steps up to this point have segmented the correctnumber of potatoes. This concluded pre-processing and segmentation, after which work moved onto the description & representation stage to calculate the summary statistics.

The primary method of obtaining statistical information was through MATLAB’s ‘regionprops’ function, which uses the returned component structure as an input. While the function can obtain up to 27 different features for a region, including area, eccentricity, and pixel list among others, certain properties were specified to increase performance for this task. These were the 'image', 'boundingbox', 'centroid', 'majoraxis' and 'minoraxis' properties, from which all required statistics could be elicited.

Figure 2



To obtain all potato descriptors for each object the following process was contained within a loop, iterating for the number of components found earlier. Firstly, the region descriptor ‘centroid’ is returned by regionprops as an X and Y co-ordinate. This is a measure of the average co-ordinate within the segmented potato, and is effectively its centre of mass. This can be visually overlaid onto the potatoes using MATLAB’s ‘plot’ function as in figure 3.

The major and minor axis length properties can be used to calculate the object’s eccentricity, a boundary descriptor indicating ‘roundness’ (where the less circular an object is the more eccentric it is). The formula for this is the “ratio of the distance between the foci of the ellipse and its major axis length” (Mathworks, 2014). Distance between foci is calculated using the following equation:

, where j and n are the major and minor radii (Page, 2009).

With region and boundary descriptors computed, all that remained were texture descriptors. A statistical approach was taken as spectral quantification of texture assumes a periodic underlying frequency, which is not the case due to the non-uniformity of potato markings.

The bounding box property was then broken into its 4 values: the X and Y co-ordinates of the top left corner and the height and width of the box. These were passed into a custom ‘getTextureDescriptors’ function, along with the RGB input image and ‘image’ property of the current component returned by ‘regionprops’ as arguments. This function manually calculates the mean and variance of each RGB channel and the grayscale image.

‘getTextureDescriptors’ loops through each non-zero pixel, accumulating their values and dividing by the total pixel count to get the mean. Squared differences from the mean can then be accumulated in a separate loop, and divided by the total pixel count to get the variance. Square rooting the RGB variance values finds the standard deviation for each channel, as well as using the grayscale variance to calculate the component’s smoothness.

Finally, the entropy or ‘statistical measure of randomness’ is calculated using the following mathematic formula: (Gonzalez & Woods, 2002). Doing so required the removal of ‘zero’ values for the histogram counts, as well as normalisation so all counts add up to one.

If this task required further work the next step would be pattern classification, using these statistic descriptors to help it ‘learn’ a model of what a potato is. With this it could make meaningful decisions using a robust dataset to classify regions and objects as ‘potato’ or ‘not potato’.

**Results**

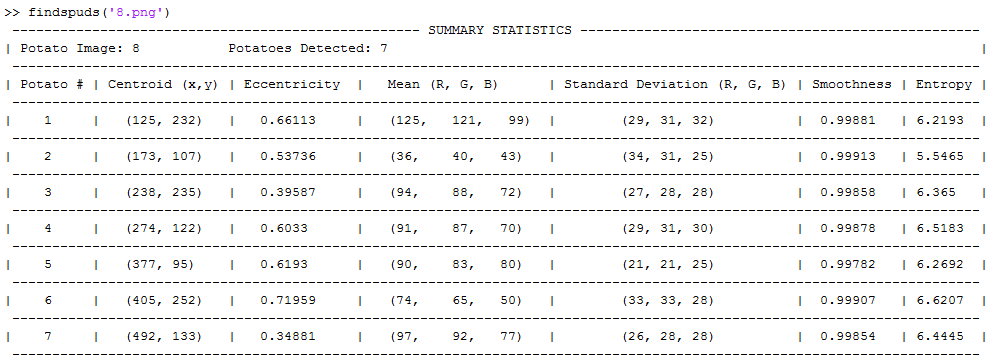
Testing revealed that the code solution to this problem had successfully detected the correct number of potatoes for all available input images, avoiding false positives and minimising data loss. The regions corresponding to potatoes are largely enclosed, with some extremities outside of the bounding box due to the opening required by region filling (see below).

Figure 3



MATLAB’s ‘disp’ command outputs the collected statistics, set up in a table structure. The descriptor data for the example image used throughout this report is below.

Figure 4



**Conclusion & Evaluation**

The solution yields results with above 95% accuracy and is therefore considered a success.

Looking at the performance of some of the above methods, there were some significant observations. MATLAB’s ‘imfill’ function serves the same purpose as ‘fillHoles’ however it ultimately performs better. This is because it views two single diagonally adjacent pixels as joined, unlike the manual method. This means that no extra work is needed to enclose what the manual method sees as gaps. The ‘close’ beforehand and large radius ‘open’ afterward were necessary to circumvent this issue. This does result in slightly increased data loss, which is noticeable on some final potatoes, particularly on the two far-right potatoes in figure 3.

Calculating most of the descriptors manually to demonstrate understanding of the theory does sacrifice function performance in some places, as well as output quality. Most noticeable are the mean and standard deviation calculations, because they both use nested for loops. In Big-O notation terms, this means a high time complexity of ‘’ for **both** the mean and standard deviation calculations. MATLAB’s ‘mean’ and ‘std’ functions are leaner and produce faster results, and do not use as much processor time or require as much variable memory. These would be used to further develop this task.

Task 2: Object segmentation with a changing background

As this task required the function to ‘learn’ a background model, the challenge was finding a method capable of retaining as much useful data as possible while filtering out unimportant background data. The solution uses mean, standard deviation and k-means clustering to model the background, and segments the potatoes through the use of colour slicing, custom functions, median filtering and morphological operators.

**Methods**

Like the first task, the goal is to produce a binary mask to apply to the ‘beltpotato’ images, setting everything that isn’t a potato to black. To do this the potatoes had to be discerned from the background with a background model made from the ‘emptybelt’ images.

Given an RGB input image of potatoes on a belt, the function first adaptively calculates a ‘mean image’ of however many background images are available (six in this instance). This is done manually in ‘getBackgroundMean’, looping through all images and converting them to the HSV colour space. François & Medioni showed that HSV handles noise and shadow better than RGB, and uses the colour information better (1999).

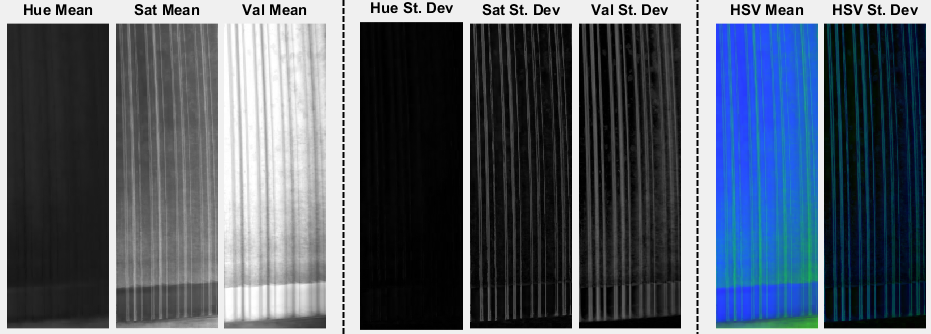
The function adds each pixel value in each channel across all six images together, dividing by the number of backgrounds to find the mean of each pixel. It returns the HSV mean image which is then used in ‘getBackgroundStd’ to calculate a standard deviation background image in HSV. Again, this is done manually using squared differences and variances for each channel to produce an image for each channel. These are concatenated and returned to the ‘segmentspuds’ function for further operation.

Figure 5

As evidenced by figure 5, this resulted in a more evenly spread average image of the background. However, the dark lines between rollers remained prominent and caused problems with the proposed methods if left in. To circumvent this the spatial filtering pre-processing step was applied, using a motion filter of length 22. This kernel convolved over the HSV mean and standard deviation images and effectively ‘blurred’ the lines to make them less prominent (see figure 6).

The background model at this point produced acceptable results, however further research found that K-means clustering could be applied to produce an even better model. K-means clustering is a traditional image processing technique, and has been previously used in segmentation of background and foreground regions (Indupalli, et al., 2006). A similar potato segmentation exercise applied the same principle to video frames, and the results were very promising (jordanblake7, 2014). For this, however it uses MATLAB’s ‘kmeans’ function to cluster all pixels in both images into just five colours – a number found through testing to be the best – normalising the background. With the background modelling complete, the input image was converted to HSV and contrast stretched to further segregate the foreground. This used MATLAB’s ‘stretchlim’ function to find both of the limits.

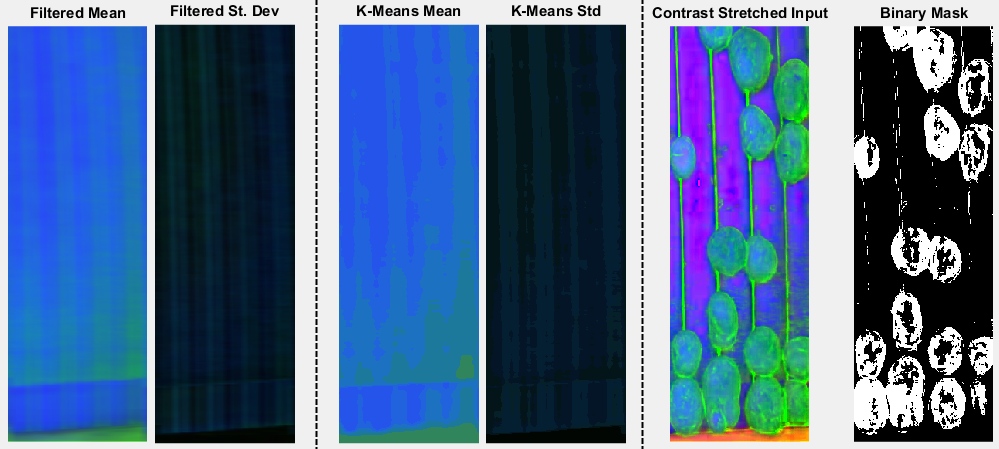
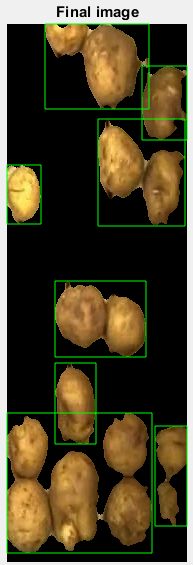
With final pre-processed input and background images, the segmentation could take place. This adopted the colour slicing technique, by specifying a range of colours that were of interest. This method’s performance was tested using a prototypical intensity threshold, which was fine-tuned until the best possible output was achieved. This threshold was found using MATLAB’s ‘roipoly’ and ‘imhist’ functions to select multiple potato objects and find a comfortable minimum value, one that would produce as much true positive and as little false positive data as possible.

Figure 6

If the HSV pixel’s value channel passed this intensity threshold, it would then be directly compared to the mean and standard deviation images. If the pixel’s saturation value was greater than the background pixel’s mean saturation value plus its standard deviation value, then it is the desired colour. The binary mask in figure 6 is then constructed by setting only these pixels to one.

Pre-processing and segmentation operations are then applied to de-noise the mask. A morphological dilate is performed with a disk element of radius 4 as it best complements the ‘potato’ shape. This closes gaps so MATLAB’s ‘imfill’ can fill all enclosed holes to more fully form the objects. At this point a custom function ‘removeComponents’ loops through all components from ‘bwconncomp’, setting any with fewer pixels than the input argument to have zero values. This removes most of the false positive data along the lines between rollers. A morphological erosion then uses the same element as the dilate to counteract it as there is now more meaningful data. A 5\*5 median filter then convolves over the image to remove noise and smooth the image without blurring. Any smaller components remaining are then removed to produce the final binary mask, which is then applied via a loop to the input image (see figure 8).

**Applying task one’s function**

The part of the first task’s function used to calculate the summary statistics and display the bounding boxes was inserted into ‘segmentspuds’. As a large number of the potatoes were not separated from each other, the statistics were not all accurate. Texture descriptors like the entropy, mean, standard deviation values are still representative in most images, however boundary and region descriptors like centroid and eccentricity are inaccurate.

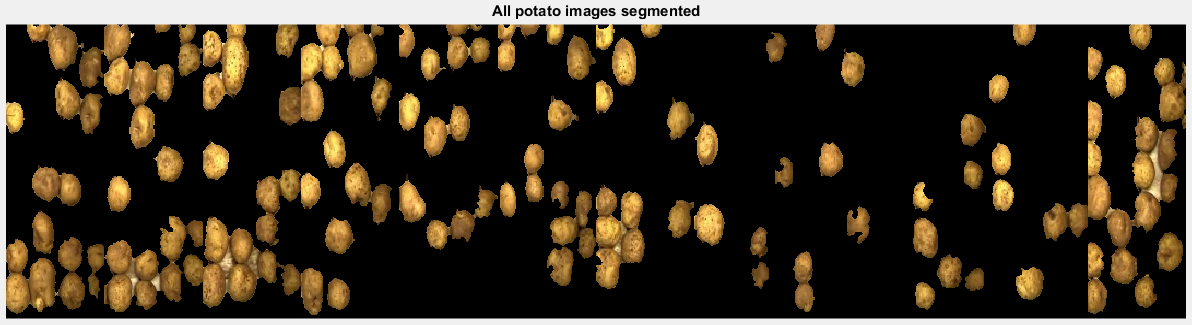
A more in-depth colour-based segmentation algorithm could be used in future work to separate potatoes from one another.

Figure 7

**Results**

Testing revealed that the code solution recognised every single potato to some degree, and only three significant instances of ‘potato roller’ appearance. Most potatoes were over 90% segmented, however there were a small number that weren’t properly detected.

Figure 8



**Conclusion & Evaluation**

The solution yielded very good results, segmenting the background with considerable efficiency while minimising data loss. It could be improved by fine-tuning the morphological operations or attempting another approach such as background subtraction. This would be looked at for future improvement.

The HSV colour space has proven to be invaluable in image processing, and facilitated the colour slicing operation with great effect. Using the mean and standard deviation to form a background model worked very well, judging by the initial binary mask in figure 6 and its subsequent alterations.

K-means clustering has proven it can produce good results, however it does take a noticeable length of time to complete. It is fairly computationally heavy and so it is questionable whether this method would be useful on a large scale. The method would definitely require optimisation in order to be viable for real-world application, as it can cause performance and memory issues if done large-scale. On top of this are the manual mean and standard deviation functions, which again take a decent amount of memory and time to complete for the volume of data available. If this were to be done on video frames it could take considerable length of time to calculate a background model this way.

References

Cook, J. D., 2009. *Three algorithms for converting color to grayscale.* [Online]   
Available at: http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/  
[Accessed 1 December 2014].

François, A. R. & Medioni, G. G., 1999. Adaptive color background modeling for real-time segmentation of video streams. *Proceedings of the International Conference on Imaging Science, Systems, and Technology,* 1(121), pp. 227-232.

Gonzalez, R. & Woods, R., 2002. *Digital Image Processing in MATLAB.* 2nd ed. New Jersey Prentice Hall

Indupalli, S., Ali, M. & Boufama, B., 2006. A Novel Clustering-Based Method for Adaptive Background Segmentation. *The 3rd Canadian Conference on Computer and Robot Vision,* p. 37.

jordanblake7, 2014. *MATLAB Potato Segmentation.* [Online] Available at: https://www.youtube.com/watch?v=NtWDeJ4\_3D4 [Accessed 1 December 2014].

Mathworks, T., 2014. *regionprops.* [Online] Available at: http://uk.mathworks.com/help/images/ref/regionprops.html [Accessed 10 December 2014].

Page, J., 2009. *Foci (focus points) of an ellipse.* [Online] Available at: http://www.mathopenref.com/ellipsefoci.html [Accessed 10 December 2014].