Insight into Computer Vision

Segmentation, Detection, Feature Calculation & Tracking

1 Plant Segmentation & Leaf Detection

1.1 Plant Segmentation

Plant segmentation is a technique to classify an image into plant and background pixels, and is useful for further analysis, such as plant classification and in turn precision application of pesticides (Hamuda, Glavin and Jones, 2016). Detecting leaves can further act as a feature for such classification and may help localise plant disease (Singh and Misra, 2017). In this assessment we are first tasked with highlighting plant foreground in a 16-image dataset.

To begin, the green channel is extracted from the RGB-image to highlight the 'green' region of interest (Sharma, Mishra and Shrivastava, 2012). A median filter is then passed over the image to smooth the noise that comes with the "salt and pepper" (Ivković et al., 2015) gritty texture of the soil. The contrast is then adjusted, within a range that increases the intensity of the plant pixels of concern (Schaefer et al., 2011). The image is then binarized using an adaptive threshold, which helps capture objects in complex lighting conditions (Liao et al., 2020).

Some noise has still made its way into this binarized image, in the form of scattered small blobs and border-connected objects. As the plant is centred in every image, this border noise can be removed, using morphological reconstruction, with a preferred pixel connectivity of '8' (Soille, 1999). The small blobs are removed by isolating connected components of a connectivity of 150 pixels or less.

This created mask is used to segment the plant, however, there exists some small noise which connects to the leaf in some images and cannot be segmented out in the RGB colour space. Some of these connections can be removed by converting the image to the LAB colour space. Unlike the 'red', 'green', 'blue' channels of RGB, LAB contains a channel for 'lightness', 'green-red', and 'blue-yellow', and approximates human vision uniformity (Connolly and Fleiss, 1997) – which would explain why the human eye can spot the brown-green connected noise, but computation cannot in the RGB colour space. The region of interest can now be extracted by binarizing the inverse of concatenation of the lightness and green-red channels, and then removing any small noise items that remain using the previous method.

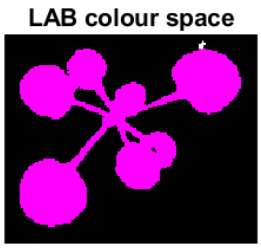


Figure 1 LAB colour space conversion successfully isolates plant from noise in 'plant011'.

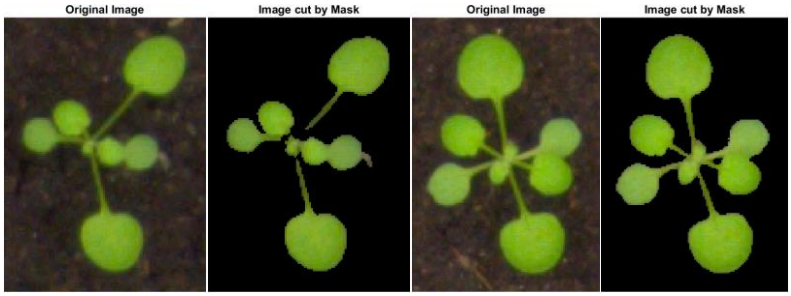


Figure 2 Left: Segmented plant 002 (DICE=0.952).

Right: Segmented plant 005 (DICE=0.974).

The Sørensen-Dice (1948) similarity coefficient was calculated on comparing the created masks with the ground truth segmentations, to assess quality of overlap. The mean DICE score was found at a respectable 0.9597, with individual image scores visualised in *Figure 3*.

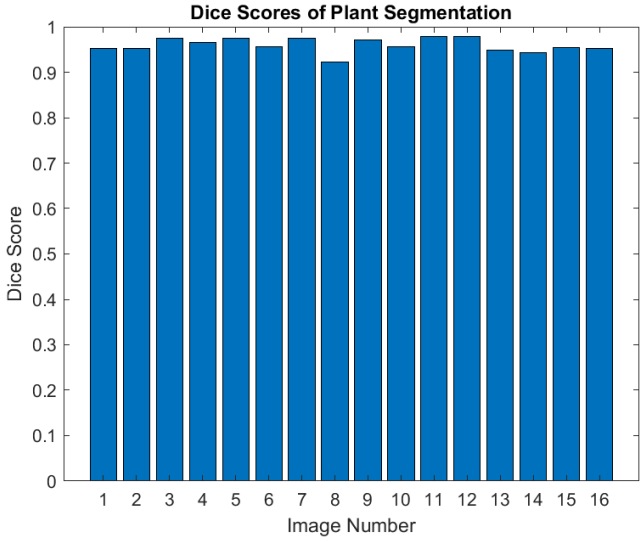


Figure 3 DICE Scores for all plant image segmentations. (Mean = 0.9597).

1.2 Leaf Detection/Count

Due to the basis of leaf detection being on segmentation, the high plant detection DICE score was a necessity. Leaf detection was a less trivial task and involved exploring and comparing various techniques to achieve a near ground-truth leaf count.

Some techniques for detecting leaves were firstly ruled-out. K-means clustering required a value of K to separate regions (Zheng et al., 2018), which is unknown, as would depend on how many leaves were per-image- the task at hand. Similarly, Markov Random Fields, a probabilistic segmentation technique, requires some class estimate, which is unfortunate as it is known to detect weak edges (Wu et al., 2011), which would exist between leaves.

Sift features (Lowe, 1999) were calculated, in the hopes it would capture the scale invariance of leaves. The features appeared to pinpoint some leaves, but also pinpointed stalks and between-leaves as features, only mimicking a leaf count by chance. This may be of further use for classifying between types of plants.

An active contour model (Kass, Witkin and Terzopoulos, 1988) was applied to the segmented plant image in the hope to localise leaves, but no matter the number of iterations, the snake's edges did not bypass the local minima (strong edges of plant) to split leaves. Similarly, Harris (1988) edge detection could not pinpoint between-leaf weak edges to enact a split.

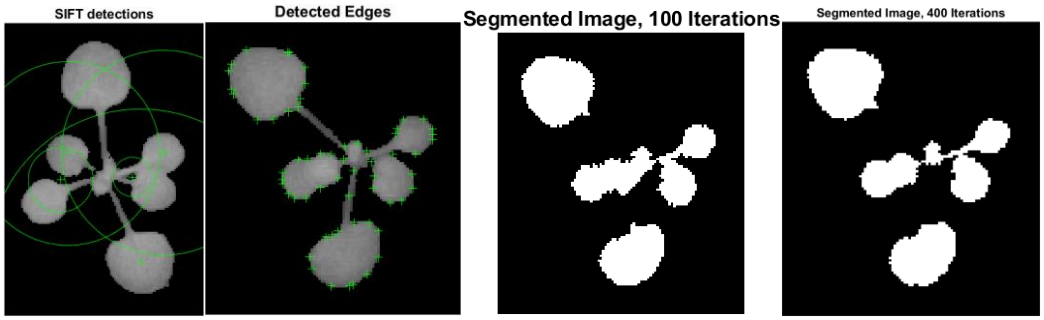


Figure 4 From left to right: Sift detections ; Harris edges ; active contours at 100 and 400 iterations

Mean shift (Comaniciu and Meer, 2002) was performed to try to isolate clusters of contrast-enhanced plant images. However, as the clusters in the image were diverse, certain cluster labels varied per algorithmic run, both giving different results each time and still struggling to separate visually joined leaves.

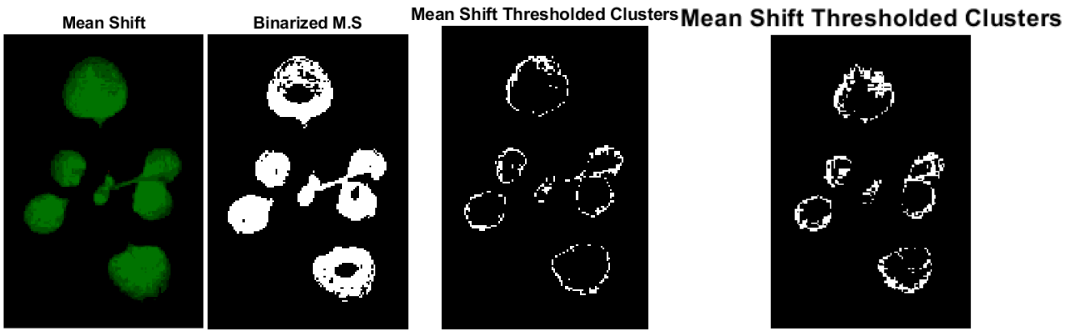


Figure 5 Mean shift, binarized mean shift, and the same cluster on two separate model runs.

The following methods had more success. A simple method was to open the image with a disk morphological element of pixel size ‘4’ and count the larger components that remained. This gave a mean count error of 2.2. As leaves are circular in structure, a circular Hough transform (Rizon et al., 2005) was performed to detect them after transforming the image into a parameter space. This achieved a 0.93 mean error after an optimum sensitivity parameter was set, and it was found that the model performed worse after image opening, as that performed too many component splits, increasing the count.

The best solution was created via a watershed transform (J. Roerdink and Meijster, 2017), which separated visually joined leaves based on their intensities. To avoid its sensitivity for leaf ridge split, it was found that opening and dilating the image helped with accuracy, which reduced mean error to 0.81. Unfortunately, watershed still is sensitive to splitting leaves when not required. All watershed leaf detections are displayed in *Appendix 5.1*.

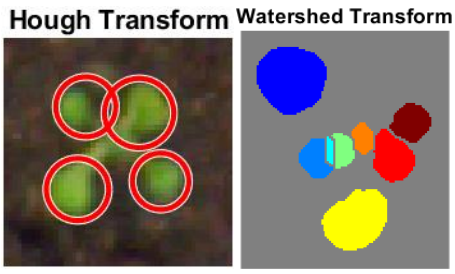


Figure 6 Circular Hough transform, and Watershed transform.

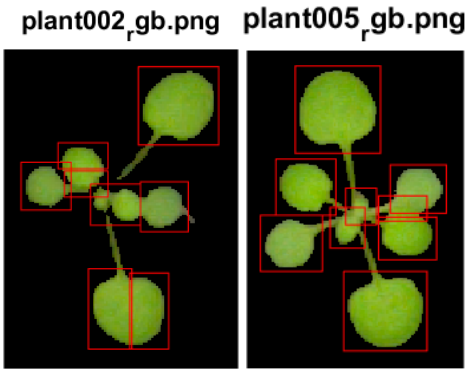


Figure 7 Final 'Watershed' leaf detection results for 'plant002' and 'plant005'.

fnames	GTCounts	Hough1counts	Hough2counts	CCcounts	sifts	wss
"plant001_rgb.png"	7	8	9	5	7	7
"plant002_rgb.png"	7	7	7	5	5	6
"plant003_rgb.png"	8	7	7	4	8	7
"plant004_rgb.png"	8	10	10	6	5	9
"plant005_rgb.png"	8	8	9	6	9	9
"plant006_rgb.png"	7	7	8	6	7	7
"plant007_rgb.png"	8	9	10	6	7	8
"plant008_rgb.png"	6	3	3	5	4	2
"plant009_rgb.png"	6	6	6	4	4	6
"plant010_rgb.png"	7	8	9	4	5	8
"plant011_rgb.png"	8	7	10	5	13	7
"plant012_rgb.png"	9	8	9	5	13	9
"plant013_rgb.png"	8	10	14	6	11	9
"plant014_rgb.png"	8	9	10	7	10	9
"plant015_rgb.png"	6	6	6	4	4	6
"plant016_rgb.png"	7	6	6	5	5	6

method	error
"Hough"	0.9375
"HoughOpened"	1.5625
"ConnectedComp"	2.1875
"SIFT"	1.9375
"Watershed"	0.8125

Figure 8 Ground truth leaf counts versus method counts and calculated mean error.

1.3 Conclusion & Expansion

Further extensions to this work could improve leaf detection accuracy. It may be possible to increase the sensitivity of Hough transform and then reduce detections using a form of non-maximum suppression (Hosang, Benenson and Schiele, 2017). Perhaps after getting a rough leaf estimate using the successful methods, KNN – clustering or a Markov Random Fields model could be used based on this seed. Another method could be extending watershed transform into a deep-learning solution which adds convolution (Bai and Urtasun, 2017). It may also prove useful to examine fully convolutional networks for segmentation, such as U-NET (Ronneberger, Fischer and Brox, 2015), or perhaps a modern object detection method for leaf counting, such as the one-shot YOLO (Terven and Cordova-Esparza, 2023). However, these would require more images to train, and require possibly extensive annotation.

2 Onion/Weed Feature Calculation

Feature extraction is a method of reducing the size of image data by obtaining derived information from the segmented image, to perform the likes of object classification (P.S and V.S, 2016). This part of the assessment involves extracting shape and texture features from images of onions and weeds taken in four channels: red, green, blue, and near infrared, performing importance analysis, and training a support vector machine.

2.1 Feature Extraction

2.1.1 Shape Features

Using the provided segmentation masks, shape features were extracted from every weed and onion, returning circularity, eccentricity, solidity, and non-compactness scores. Non-compactness was computed using the inverse of Schwartzberg (1962) compactness.

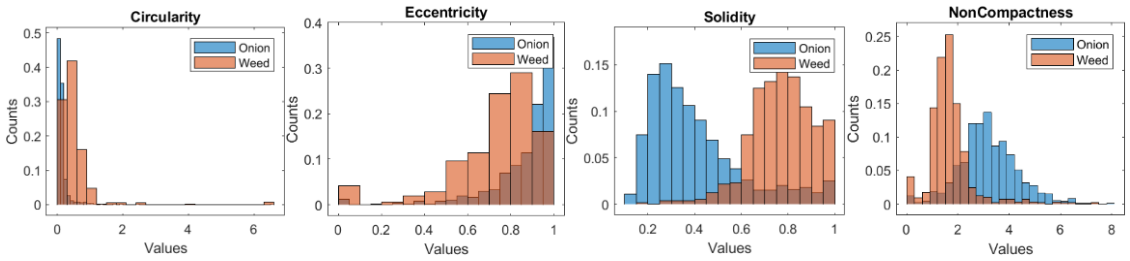


Figure 9 Shape feature distributions from onion and weed patches.

On viewing shape distributions in *Figure 9*, both non-compactness and solidity are visually separable for the weed and onion classes, which hints at them both being a useful feature for class differentiation. This is comprehensible, as weeds form more dense "solid" masses than the elongated non-compact onions in the images.

2.1.2 Texture Features

Haralick et al. (1973) texture features were extracted by firstly computing the grey covariance matrix for every weed and onion in 0, 45, 90 and 135 degree orientations and returning the mean and range for angular second moment, contrast and correlation over the orientations. This was repeated for each colour channel r,g,b and near-infrared (from the depth image). This required a considerably high amount of computation than calculating shape features, and so results were saved to text files for visualisation/ classification. These texture feature distributions did not appear as class separable as the shape features in general, however, the range of values over the orientations appear to be more descriptive than the means. Between the colour channels, although feature values differed, the distribution mostly remained uniform, with a strong skew. The exception was the slight difference in contrast distribution in the depth channel compared to the RGB ones (*Appendix 5.2.*).

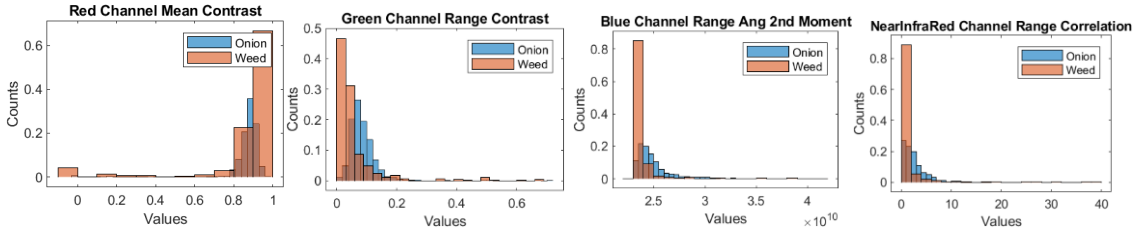


Figure 10 Texture feature distributions from onion and weed patches, sampled from each colour channel.

2.2 Classification

Support vector machines with linear kernels were trained using various batches of features for comparison of class separability. The training set consisted of all weeds and onions in image samples 1-18, with patches in 19-20 reserved for testing. Training the SVM with only all shape features gave reasonable test results, with an average F1-score of 95%, with onions having a higher precision value, and weeds having a higher recall value. 1 onion was misclassified, while 9 weeds were misclassified.

On training using all texture features (from all channels), classification of the test set completely failed. Every object was classified as an onion. Interestingly, when the SVM was trained with both shape and texture features, the same problem arose, indicating that the inclusion of texture features tainted the descriptive power of the shape features.

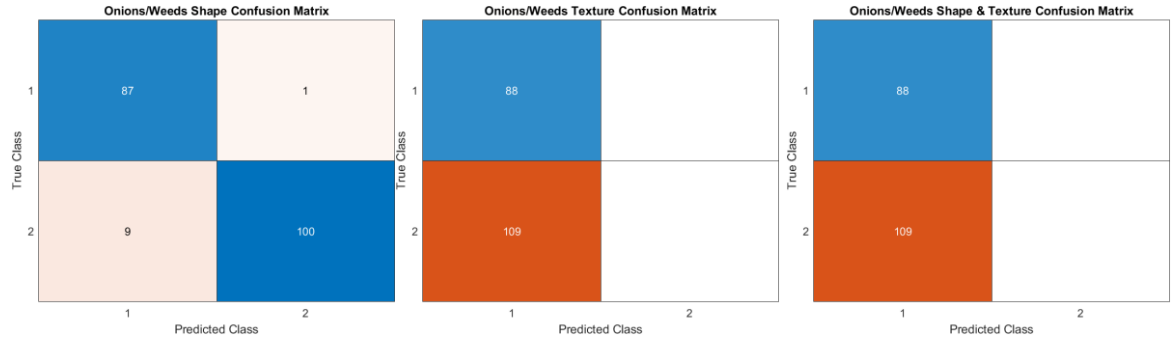


Figure 11 Confusion matrices for initial SVM classifications (1 = Onion, 2=Weed)

2.3 Importance Sampling

Importance sampling is a method of choosing the most descriptive features for classification and has been proven useful for the likes of splitting tree nodes for a random forest classifier (Cao et al., 2011). Features were assessed based on their importance score computed using chi-squared tests (Huan Liu and Setiono, 1995). It was found that the ten most descriptive features consisted of every shape feature except eccentricity, the range of contrasts in every colour channel, green mean contrast, and blue range correlation. This coincides with the findings from examining the distributions, with most shape features and range texture values being the most descriptive.

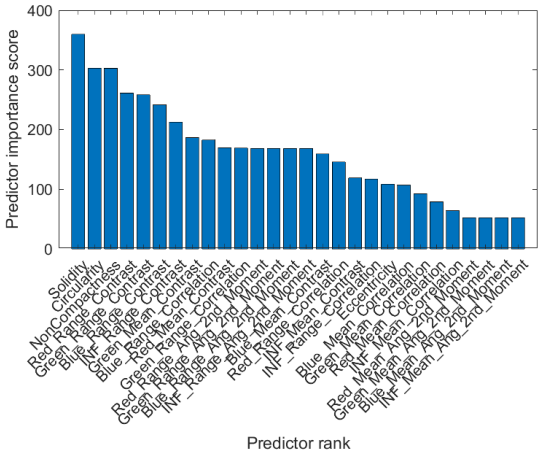


Figure 12 Results from chi-squared feature importance score calculation.

The SVM classifier was retrained using the top ten most descriptive features. Curiously, the performance metrics slightly dropped. The classifier was then trained with the top 4 features, which consisted of solidity, circularity, noncompactness and red range contrast. This has a negligible negative effect on the F1 score compared to the shape feature classifier, however, the precision and

recall values are more balanced between the classes, with weed precision and onion recall slightly higher. The classifier only predicted 2 onions incorrectly, and 7 weeds incorrectly.

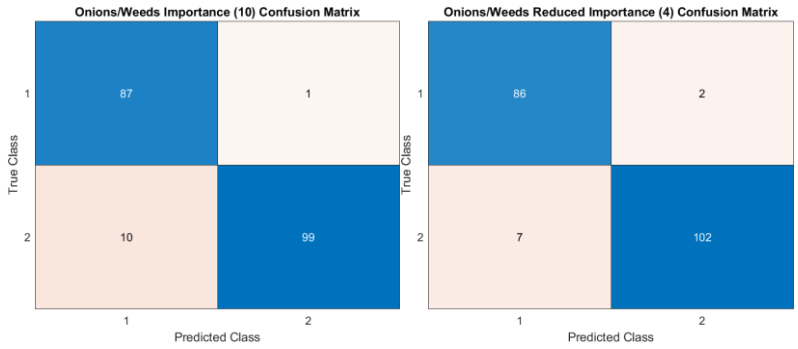


Figure 13 SVM Classifier Confusion matrices for importance sampling 10 and 4 features respectively.

SVM_Features	Onion_Precision	Weed_Precision	Onion_Recall	Weed_Recall	F1
"All_Shape"	0.98864	0.91743	0.90625	0.9901	0.9506
"All_Texture"	1	0	0.4467	NaN	0.65474
"Shape_Texture"	1	0	0.4467	NaN	NaN
"10_Important"	0.98864	0.90826	0.89691	0.99	0.94594
"4_Important"	0.97727	0.93578	0.92473	0.98077	0.94994

Figure 14 Classification Performance Metrics.

2.4 Visual Classification Output

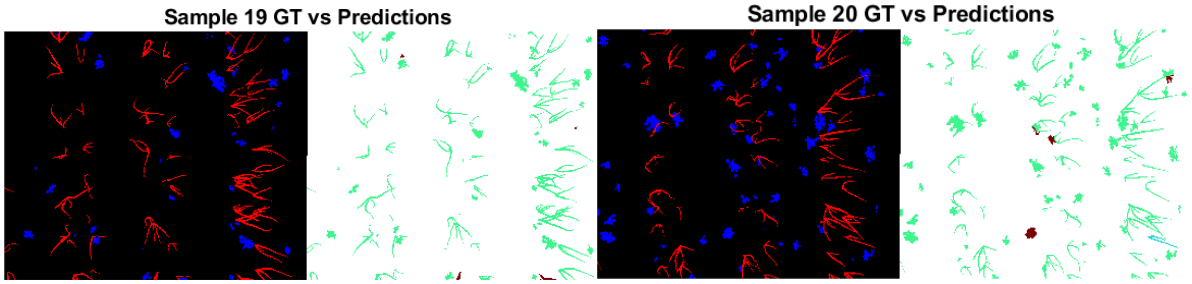


Figure 15 Visualising Classification Results of Sample 19 & 20.

On viewing a labelled representation of predictions, onions that are misclassified are partially obscured at the image borders. In a developed solution, perhaps this could be remedied by only classifying vegetation within a certain field of view. Except for one large weed, the misclassified weeds are often very small. This could mean that there are not enough distinguishable features extracted using the shape features and the grey covariance matrix.

2.5 Conclusion & Expansion

A classifier has been trained efficiently using importance-sampled extracted features, with those related to shape having more predictive power in this scenario. Other classifiers could be compared, such as decision trees and k-nearest neighbour, and perhaps the addition of deep learning convolutions could extract useful features by kernel learning (Tang et al., 2017). Fourier transform may also be a useful technique to extract texture-based features (Zhou, Feng and Shi, 2001), or the selection of sift features (Dandois et al., 2017) if a conventional approach is preferred. The desired solution should be tuned to the right performance metric, as a system to detect weeds for defoliant-spraying would be focussed on weed precision, while one for onion fertiliser-spraying would require a high onion recall.

3 Object Tracking

Object tracking is a computer vision technique which assigns a label to a certain instance over many frames of a video, which is made difficult due to the noisy readings taken from a sensing device. To follow the target, there is a requirement to predict its trajectory over many frames, and the difficulty comes with detecting an object from a three-dimensional space in a two-dimensional one. It is used for surveillance (Ojha and Sakhare, 2015), vehicle safety (Dewangan and Sahu, 2020), critical healthcare procedures (V. Sa-Ing et al., 2017), and counting animal migrations in their habitat for conservation (Kay et al., 2022). This assessment requires the creation of a Kalman (1960) filter to track an object using noisy detected coordinates from the cartesian observation model.

3.1 (1) Kalman Filter

Firstly, it is necessary to define the Kalman model, which consists of a two-stage cycle, the prediction of the object, and update of its coordinates. This Kalman filter is configured with a constant velocity model, which assumes that the object's speed and direction remain unchanged throughout the track.

The prediction stage is defined as:

$$\hat{\boldsymbol{x}}_k = \boldsymbol{F}\boldsymbol{x}_{k-1}$$
$$\hat{\boldsymbol{P}}_k = \boldsymbol{F}\boldsymbol{P}_{k-1}\boldsymbol{F}^T + \boldsymbol{Q}$$

Our state vector *x* begins at our initial state (0,0), or in matrix format, the transposed vector [0 0 0 0]'. *F* becomes our constant velocity model, which with a constant time interval *dt* = 0.2, becomes [1 *dt* 0 0; 0 1 0 0; 0 0 1 *dt*; 0 0 0 1]. *Q* is the given defined motion noise, which also exists as the state covariance *P* for the initial coordinate. As such, the initial prediction equation becomes:

```
xp = F * x; % predict state
Pp = F * P * F' + Q; % predict state covariance
```

This returns our predicted coordinates [xp, Pp]. Following this is the update stage of the Kalman filter,

$$\boldsymbol{x}_k = \hat{\boldsymbol{x}}_k + \boldsymbol{K}(\boldsymbol{z}_k - \hat{\boldsymbol{z}}_k)$$
$$\boldsymbol{P}_k = \hat{\boldsymbol{P}}_k - \boldsymbol{K}_k\boldsymbol{S}_k\boldsymbol{K}_k^T$$

$\hat{X}$ and $\hat{p}$ are the predicted state and covariance from the predict stage. S is derived as the innovation covariance, i.e., the uncertainty in the difference between the observed measurement and the predicted measurement:

$$\boldsymbol{S}_k = \boldsymbol{H}\hat{\boldsymbol{P}}_k\boldsymbol{H}^T + \boldsymbol{R}$$

```
S = H * P * H' + R; % innovation covariance
```

Where 'H' is the cartesian observation model, 'P' is the predicted state covariance and 'R' is the observation noise.

K is derived as the Kalman gain, which determines the weighting of the state estimate:

$$\boldsymbol{K}_k = \hat{\boldsymbol{P}}_k\boldsymbol{H}^T\boldsymbol{S}_k^{-1}$$

```
K = P * H' * inv(S); % Kalman gain
```

The predicted observation is then:

$$\hat{\boldsymbol{z}}_k = \boldsymbol{H}\hat{\boldsymbol{x}}_k$$

```
zp = H * x; % predicted observation
```

, which along with the original observation z, becomes the update stage to return the updated state and covariance:

```
xe = x + K * (z - zp); % estimated state
Pe = P - K * S * K'; % estimated covariance
```

This process is then repeated for every observed state, to achieve the following trajectory:

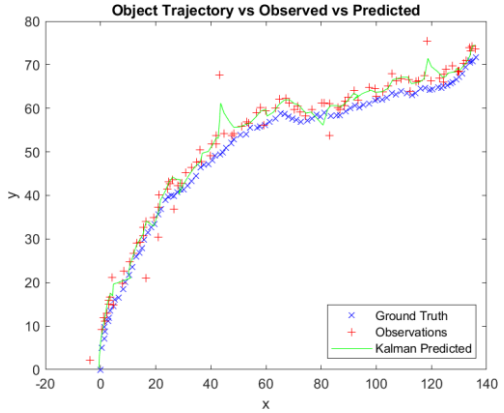


Figure 16 Plotted ground truth trajectory, observations, and predicted Kalman coordinates.

3.2 (2) Results

Error in both the observations and Kalman filter predictions are computed using Euclidean distance:

$d(\mathbf{p},\mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$, which in a 2-D plane equates to $e_i = \sqrt{(x_i - px_i)^2 + (y_i - py_i)^2}$.

Three metrics are calculated:

$$Mean\ error = \frac{\sum_{i=1}^{n} x_i}{n} \quad Standard\ Deviation = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}} \quad RMS = \sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_N^2}{N}}$$

Where the root mean squared (RMS) value penalises larger coordinate error (outliers).

As per the MATLAB table produced in *Figure 17*, the Kalman filter improves on estimating the real trajectory hinted from the observations, which in themselves are a poorer prediction of the true coordinate values.

err_type	obs_err	kal_err
"Mean Error"	3.1637	2.8646
"Standard dev."	2.2069	1.5001
"RMSE"	3.8511	3.2301

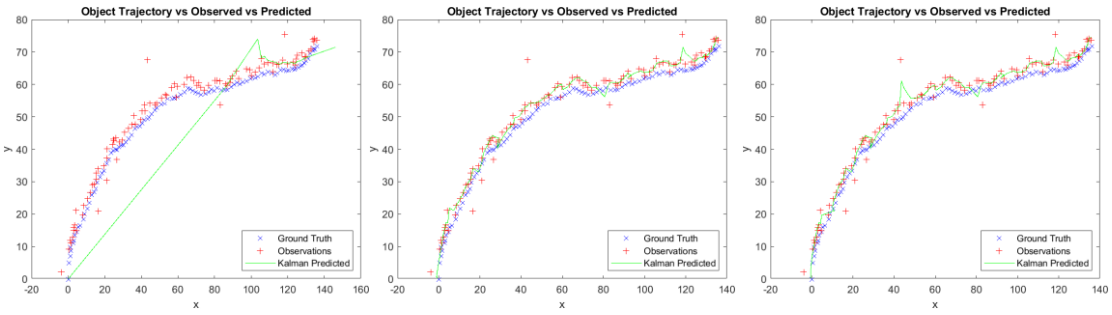
Figure 17 Error metrics for observation and Kalman filter error.

3.3 Validation Gating

To further improve the result of the Kalman filter, validation gating is a method which removes observations that are unlikely to be generated by the target, which can be seen as outliers in the trajectory graph. This can be done by not considering observed values in the Kalman update step that have a Mahalanobis distance greater than a threshold (Cox, 1993):

$$\left(\mathbf{z}_i - \hat{\mathbf{z}}_j\right)^T \mathbf{S}_{ij}^{-1} \left(\mathbf{z}_i - \hat{\mathbf{z}}_j\right) \leqslant \lambda$$

```
gate = (z - zp)' * inv(S) * (z - zp);
```



err_type	obs_err	kal_err
"Mean Error"	3.1637	37.287
"Standard dev."	2.2069	33.839
"RMSE"	3.8511	50.239

err_type	obs_err	kal_err
"Mean Error"	3.1637	2.8181
"Standard dev."	2.2069	1.2861
"RMSE"	3.8511	3.095

err_type	obs_err	kal_err
"Mean Error"	3.1637	2.8646
"Standard dev."	2.2069	1.5001
"RMSE"	3.8511	3.2301

Figure 18 Validation gating at thresholds 8, 150, 1000, respectively.

As seen in *Figure 18*, a threshold too low gives rise to a underestimated, almost linear trajectory (disregarding too many points), while a threshold too high has no impact on Kalman predictions. The threshold of '150' improves the kalman performance metrics by removing unlikely observations. This may be specific to the tracking task, as for instance, error in a tracked robots trajectory may be different from that of a tracked human.

3.4 Conclusion & Multiple object tracking

A Kalman Filter has been successfully implemented to plot a more realistic trajectory based on noisy observation points from the cartesian observation model. This could be extended to counting an object as it passes a certain point (Li et al., 2023). Certain difficulties arise in the real world, such as tracking an object that has been occluded in a certain number of video frames (Gabriel et al., 2003). Further complexities arise when tracking extends to multiple objects, such as associating the right object with the correct 'track' instance, new objects appearing in a scene being appropriately handled, and a phenomenon known as "switching" occurring when two objects cross one another in the observed view of a merged detection (Gad et al., 2022). These could be solved using optical flow methods (Ge, Chang and Liu, 2017) which take the motion vector into account, and multimodal sensing (Huiyu Zhou, Taj and Cavallaro, 2008), which can give better observed readings by pooling data from multiple devices.

4 References

Bai, M. and Urtasun, R. (2017). Deep Watershed Transform for Instance Segmentation. *arXiv:1611.08303 [cs]*. [online] Available at: https://arxiv.org/abs/1611.08303 [Accessed 8 May 2023].

Cao, D.-S., Liang, Y.-Z., Xu, Q.-S., Zhang, L.-X., Hu, Q.-N. and Li, H.-D. (2011). Feature importance sampling-based adaptive random forest as a useful tool to screen underlying lead compounds. *Journal of Chemometrics*, 25(4), pp.201–207. doi:https://doi.org/10.1002/cem.1375.

Comaniciu, D. and Meer, P. (2002). Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5), pp.603–619. doi:https://doi.org/10.1109/34.1000236.

Connolly, C. and Fleiss, T. (1997). A study of efficiency and accuracy in the transformation from RGB to CIELAB color space. *IEEE Transactions on Image Processing*, 6(7), pp.1046–1048. doi:https://doi.org/10.1109/83.597279.

Cox, I.J. (1993). A review of statistical data association techniques for motion correspondence. *International Journal of Computer Vision*, 10(1), pp.53–66. doi:https://doi.org/10.1007/bf01440847.

Dandois, J., Baker, M., Olano, M., Parker, G. and Ellis, E. (2017). What is the Point? Evaluating the Structure, Color, and Semantic Traits of Computer Vision Point Clouds of Vegetation. *Remote Sensing*, 9(4), p.355. doi:https://doi.org/10.3390/rs9040355.

Dewangan, D.K. and Sahu, S.P. (2020). Real Time Object Tracking for Intelligent Vehicle. *2020 First International Conference on Power, Control and Computing Technologies (ICPC2T)*. doi:https://doi.org/10.1109/icpc2t48082.2020.9071478.

Gabriel, P., Verly, J., Piater, J. and Genon, A. (2003). *The State of the Art in Multiple Object Tracking Under Occlusion in Video Sequences*. [online] www.semanticscholar.org. Available at: https://www.semanticscholar.org/paper/The-State-of-the-Art-in-Multiple-Object-Tracking-in-Gabriel-Verly/e010f44b4b59889a7264b3feb6f34be13e7f0243 [Accessed 8 May 2023].

Gad, A., Basmaji, T., Yaghi, M., Alheeh, H., Alkhedher, M. and Ghazal, M. (2022). Multiple Object Tracking in Robotic Applications: Trends and Challenges. *Applied Sciences*, 12(19), p.9408. doi:https://doi.org/10.3390/app12199408.

Ge, Z., Chang, F. and Liu, H. (2017). *Multi-target tracking based on Kalman filtering and optical flow histogram*. [online] IEEE Xplore. doi:https://doi.org/10.1109/CAC.2017.8243203.

Hamuda, E., Glavin, M. and Jones, E. (2016). A survey of image processing techniques for plant extraction and segmentation in the field. *Computers and Electronics in Agriculture*, 125, pp.184–199. doi:https://doi.org/10.1016/j.compag.2016.04.024.

Haralick, R.M., Shanmugam, K. and Dinstein, I. (1973). Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, [online] SMC-3(6), pp.610–621. doi:https://doi.org/10.1109/tsmc.1973.4309314.

Harris, C. and Stephens, M. (1988). A Combined Corner and Edge Detector. *Procedings of the Alvey Vision Conference 1988*. [online] doi:https://doi.org/10.5244/c.2.23.

Hosang, J., Benenson, R. and Schiele, B. (2017). Learning non-maximum suppression. *arXiv:1705.02950 [cs]*. [online] Available at: https://arxiv.org/abs/1705.02950.

Huan Liu and Setiono, R. (1995). Chi2: feature selection and discretization of numeric attributes. *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence*. doi:https://doi.org/10.1109/tai.1995.479783.

Huiyu Zhou, Taj, M. and Cavallaro, A. (2008). Target Detection and Tracking With Heterogeneous Sensors. *IEEE Journal of Selected Topics in Signal Processing*, 2(4), pp.503–513. doi:https://doi.org/10.1109/jstsp.2008.2001429.

Ivković, R., Milošević, I., Petrović, M. and Gvozdić, B. (2015). Timeline of Median filter. *Proceedings of the International Scientific Conference - Synthesis 2015*. doi:https://doi.org/10.15308/synthesis-2015-268-273.

J. Roerdink and Meijster, A. (2017). *The Watershed Transform: Definitions, Algorithms and Parallelization Strategies*. [online] Fundam. Informaticae. Available at: https://www.semanticscholar.org/paper/The-Watershed-Transform%3A-Definitions%2C-Algorithms-Roerdink-Meijster/f389e9f8d328443664f4792eb942e190a4d06f4d [Accessed 8 May 2023].

Kalman, R.E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, [online] 82(1), p.35. doi:https://doi.org/10.1115/1.3662552.

Kass, M., Witkin, A. and Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of Computer Vision*, 1(4), pp.321–331. doi:https://doi.org/10.1007/bf00133570.

Kay, J., Kulits, P., Stathatos, S., Deng, S., Young, E., Beery, S., Van Horn, G. and Perona, P. (2022). The Caltech Fish Counting Dataset: A Benchmark for Multiple-Object Tracking and Counting.

arXiv:2207.09295 [cs]. [online] Available at: https://arxiv.org/abs/2207.09295 [Accessed 8 May 2023].

Li, Y., Ma, R., Zhang, R., Cheng, Y. and dong, chunwang (2023). A tea buds counting method based on YOLOV5 and Kalman filter tracking algorithm. *Plant Phenomics*. doi:https://doi.org/10.34133/plantphenomics.0030.

Liao, J., Wang, Y., Zhu, D., Zou, Y., Zhang, S. and Zhou, H. (2020). Automatic Segmentation of Crop/Background Based on Luminance Partition Correction and Adaptive Threshold. *IEEE Access*, [online] 8, pp.202611–202622. doi:https://doi.org/10.1109/ACCESS.2020.3036278.

Lowe, D.G. (1999). *Object recognition from local scale-invariant features*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICCV.1999.790410.

Ojha, S. and Sakhare, S. (2015). Image processing techniques for object tracking in video surveillance- A survey. *2015 International Conference on Pervasive Computing (ICPC)*. doi:https://doi.org/10.1109/pervasive.2015.7087180.

P.S, S.K. and V.S, D. (2016). Extraction of Texture Features using GLCM and Shape Features using Connected Regions. *International Journal of Engineering and Technology*, 8(6), pp.2926–2930. doi:https://doi.org/10.21817/ijet/2016/v8i6/160806254.

Rizon, M., Yazid, H., Saad, P., Md Shakaff, A.Y., Saad, A.R., Sugisaka, M., Yaacob, S., Mamat, M.Rozailan. and Karthigaya, M. (2005). Object Detection using Circular Hough Transform. *American Journal of Applied Sciences*, 2(12), pp.1606–1609. doi:https://doi.org/10.3844/ajassp.2005.1606.1609.

Ronneberger, O., Fischer, P. and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *Lecture Notes in Computer Science*, 9351, pp.234–241. doi:https://doi.org/10.1007/978-3-319-24574-4_28.

Schaefer, G., Rajab, M.I., Emre Celebi, M. and Iyatomi, H. (2011). Colour and contrast enhancement for improved skin lesion segmentation. *Computerized Medical Imaging and Graphics*, 35(2), pp.99–104. doi:https://doi.org/10.1016/j.compmedimag.2010.08.004.

Schwartzberg, J.E. (1962). Three Approaches to the Mapping of Economic Development in India. *Annals of the Association of American Geographers*, 52(4), pp.455–468. doi:https://doi.org/10.1111/j.1467-8306.1962.tb00425.x.

Sharma, N., Mishra, M. and Shrivastava, M. (2012). COLOUR IMAGE SEGMENTATION TECHNIQUES AND ISSUES: AN APPROACH. *International Journal of Scientific & Technology Research*, [online] 1(4). Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=8379d497eb4ada7c69fad7e23e159719e5e1fc6f [Accessed 8 May 2023].

Singh, V. and Misra, A.K. (2017). Detection of plant leaf diseases using image segmentation and soft computing techniques. *Information Processing in Agriculture*, 4(1), pp.41–49. doi:https://doi.org/10.1016/j.inpa.2016.10.005.

Soille, P. (1999). *Morphological Image Analysis: Principles and Applications*. *Sensor Review*, Springer, pp.164–165. doi:https://doi.org/10.1108/sr.2000.08720cae.001.

Sørensen, T.J. (1948). *A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons*. I kommission hos E. Munksgaard.

Tang, J., Wang, D., Zhang, Z., He, L., Xin, J. and Xu, Y. (2017). Weed identification based on K-means feature learning combined with convolutional neural network. *Computers and Electronics in Agriculture*, 135, pp.63–70. doi:https://doi.org/10.1016/j.compag.2017.01.001.

Terven, J. and Cordova-Esparza, D. (2023). A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond. *arXiv:2304.00501 [cs]*. [online] Available at: https://arxiv.org/abs/2304.00501.

V. Sa-Ing, S. Thongvigitmanee, C. Wilasrusmee and J. Suthakorn (2017). *Adaptive Mean-Shift Kalman Tracking of Laparoscopic Instruments*. [online] International Journal of Computer Theory and Engineering. Available at: https://www.semanticscholar.org/paper/Adaptive-Mean-Shift-Kalman-Tracking-of-Laparoscopic-Sa-Ing-Thongvigitmanee/4b6a547b650e45facb627add0c0ca8e5fb6257e0 [Accessed 8 May 2023].

Wu, Y., Li, M., Zhang, P., Zong, H., Xiao, P. and Liu, C. (2011). Unsupervised multi-class segmentation of SAR images using triplet Markov fields models based on edge penalty. *Pattern Recognition Letters*, 32(11), pp.1532–1540. doi:https://doi.org/10.1016/j.patrec.2011.04.009.

Zheng, X., Lei, Q., Yao, R., Gong, Y. and Yin, Q. (2018). Image segmentation based on adaptive K-means algorithm. *EURASIP Journal on Image and Video Processing*, 2018(1). doi:https://doi.org/10.1186/s13640-018-0309-3.

Zhou, F., Feng, J.F. and Shi, Q.Y. (2001). *Texture feature based on local Fourier transform*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICIP.2001.958567.

5 Appendix

5.1 Leaf Detection Results

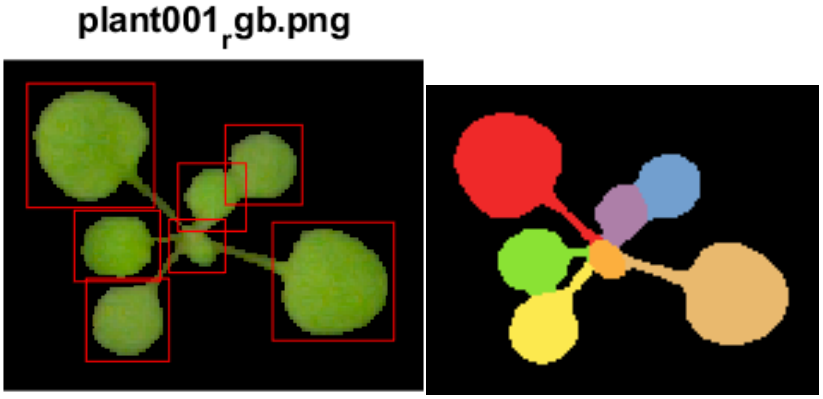


Figure 19 Plant 001 Watershed Leaf Detection Results.

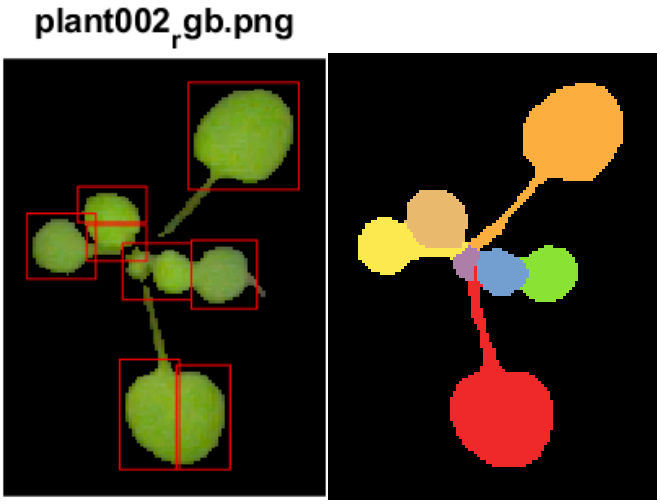


Figure 20 Plant 002 Watershed Leaf Detection Results.

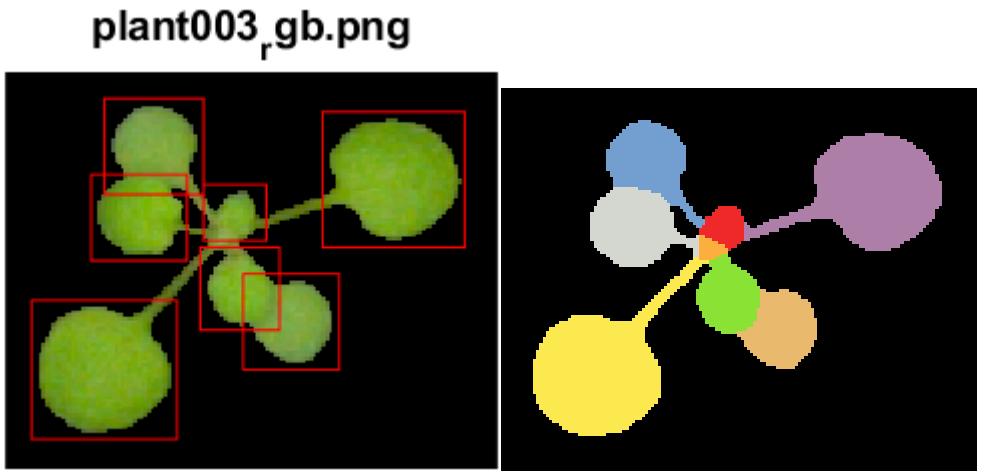


Figure 21 Plant 003 Watershed Leaf Detection Results.

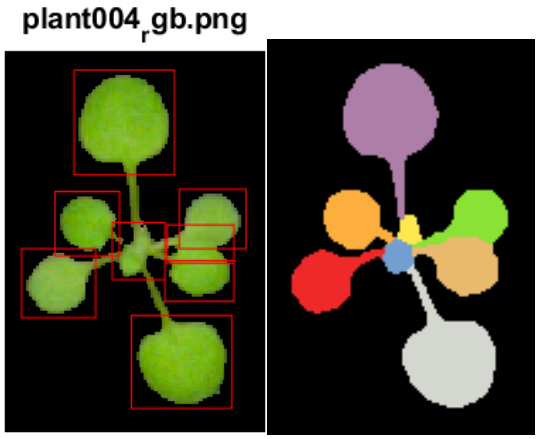


Figure 22 Plant 004 Watershed Leaf Detection Results.

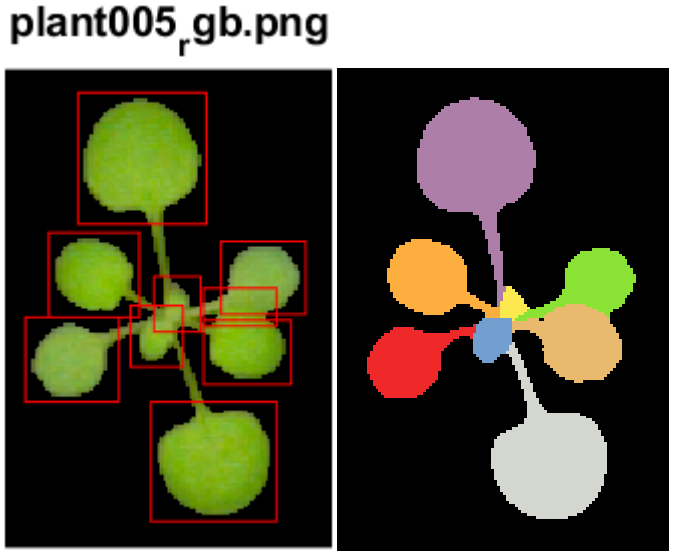


Figure 23 Plant 005 Watershed Leaf Detection Results.

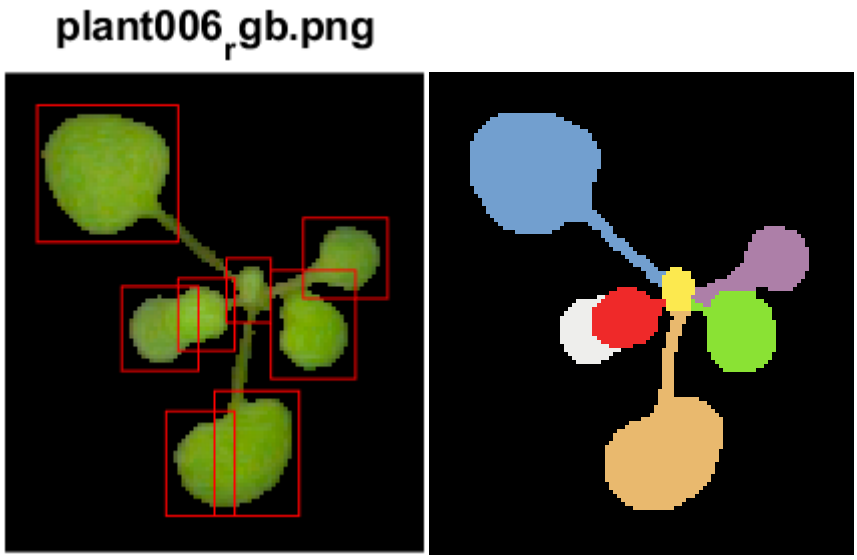


Figure 24 Plant 006 Watershed Leaf Detection Results.

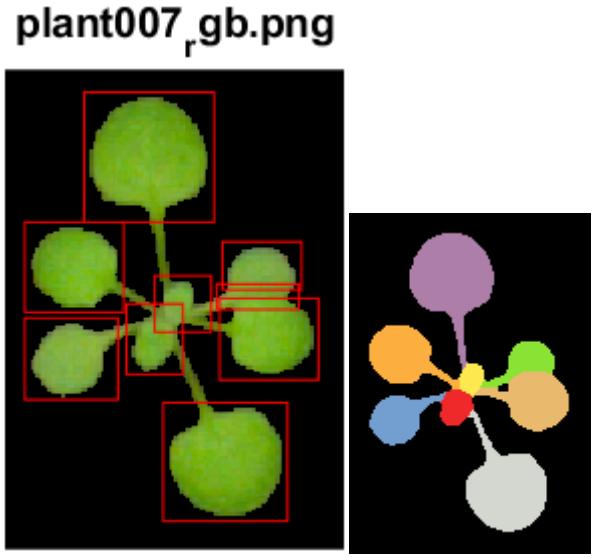


Figure 25 Plant 007 Watershed Leaf Detection Results.

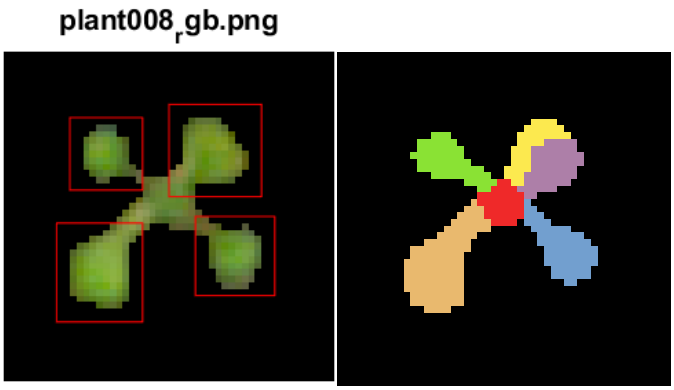


Figure 26 Plant 008 Watershed Leaf Detection Results.

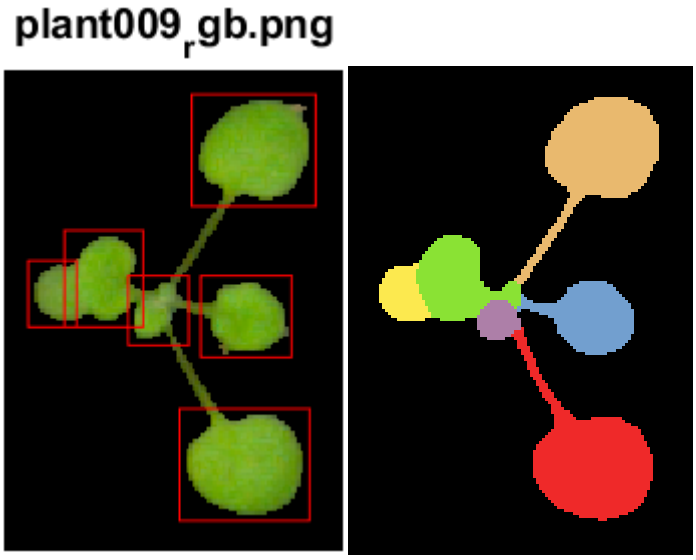


Figure 27 Plant 009 Watershed Leaf Detection Results.

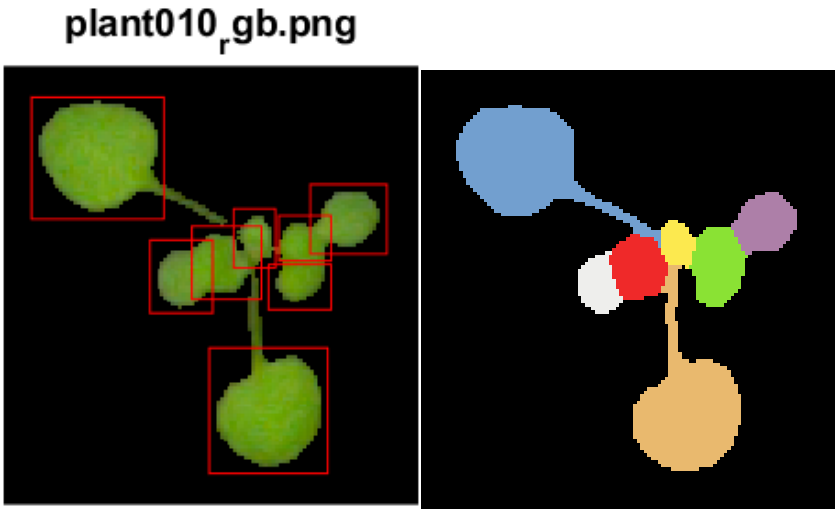


Figure 28 Plant 010 Watershed Leaf Detection Results.

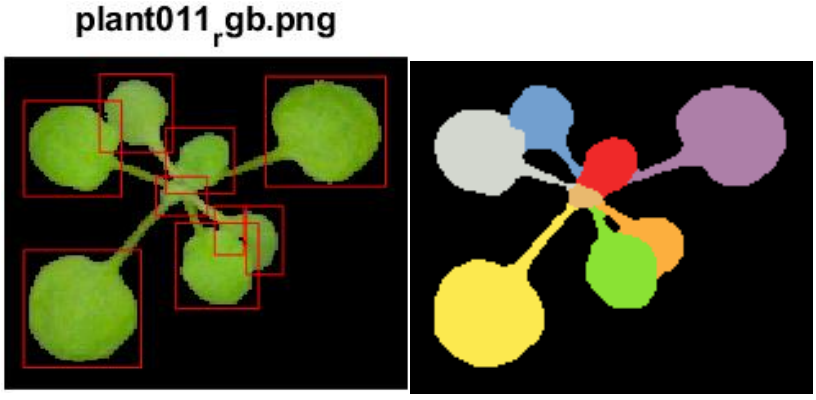


Figure 29 Plant 011 Watershed Leaf Detection Results.

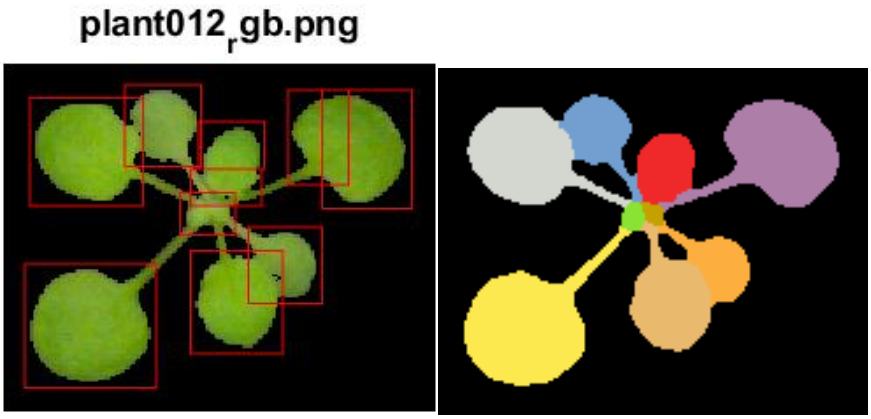


Figure 30 Plant 012 Watershed Leaf Detection Results.

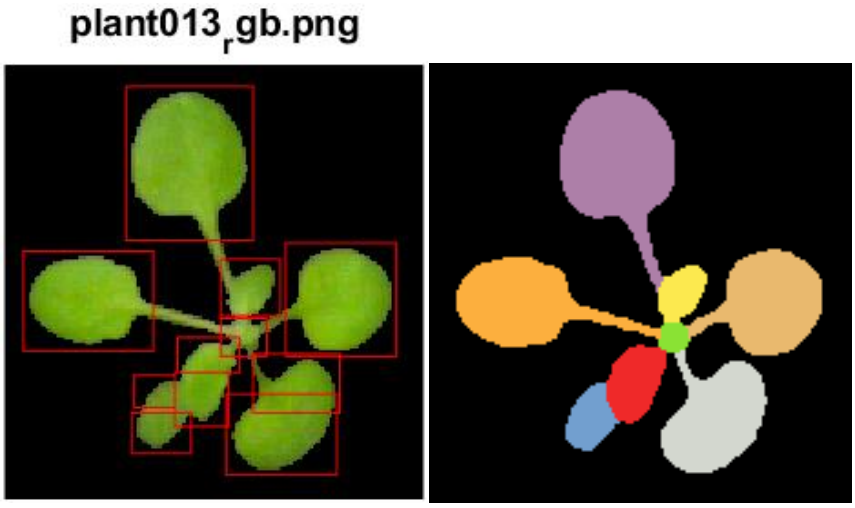


Figure 31 Plant 013 Watershed Leaf Detection Results.

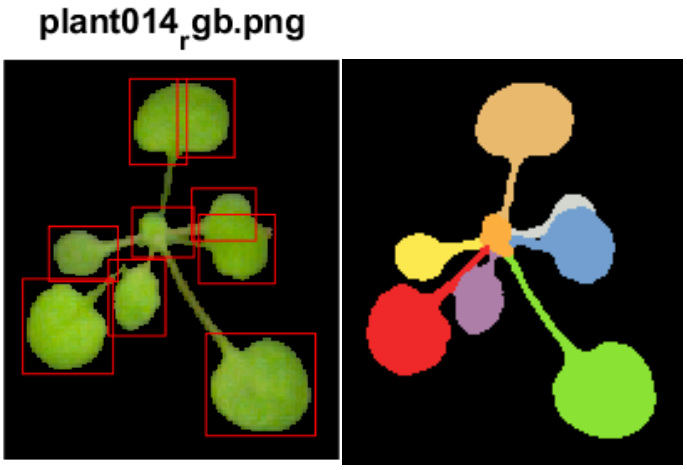


Figure 32 Plant 014 Watershed Leaf Detection Results.

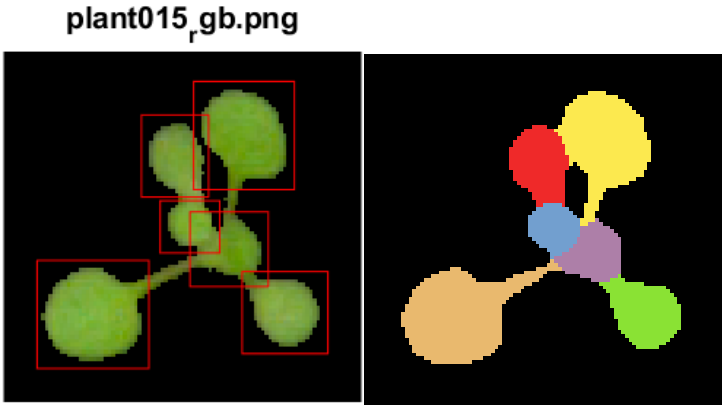


Figure 33 Plant 015 Watershed Leaf Detection Results.

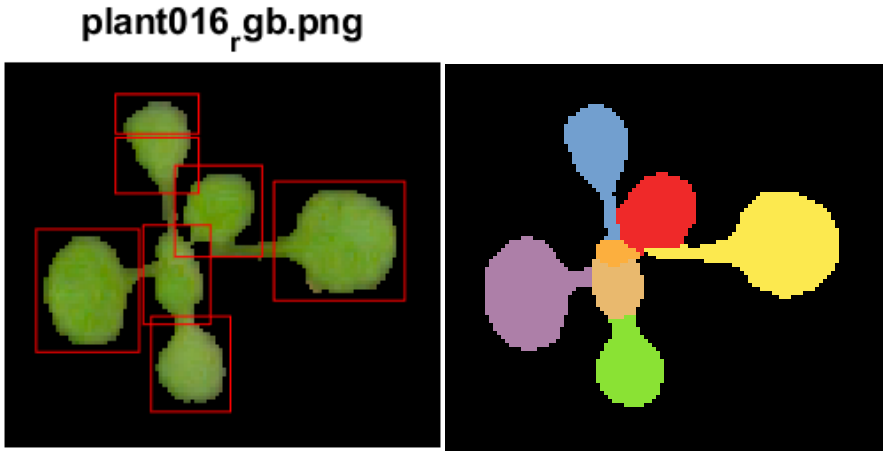


Figure 34 Plant 016 Watershed Leaf Detection Results.

5.2 Texture Feature Distributions per Channel

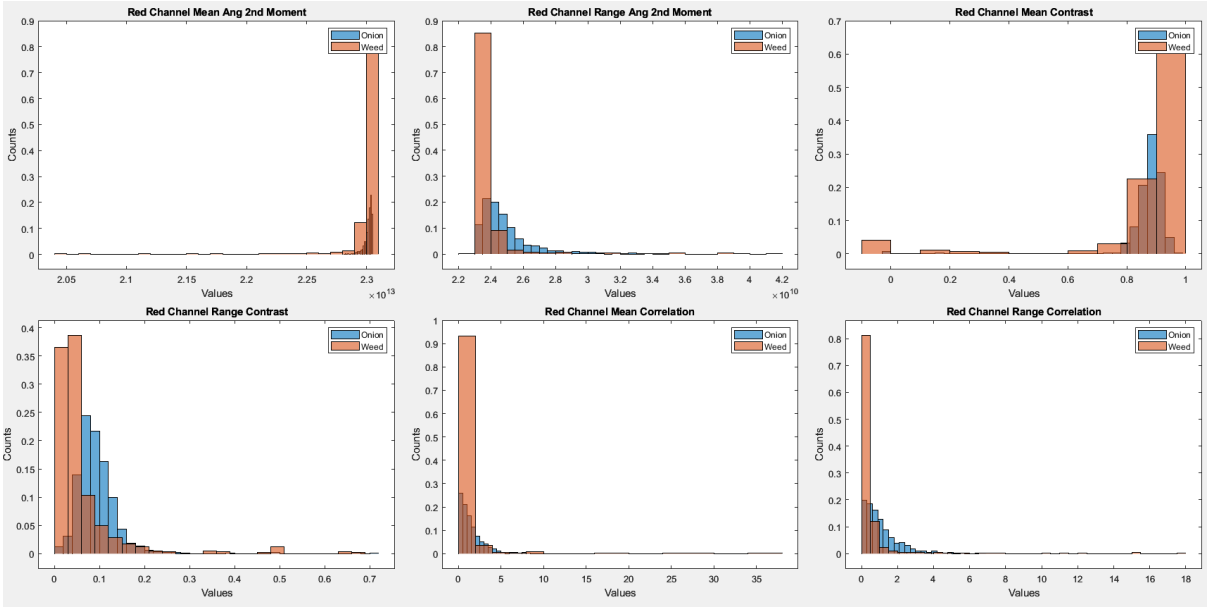


Figure 35 Red Channel Texture Features.

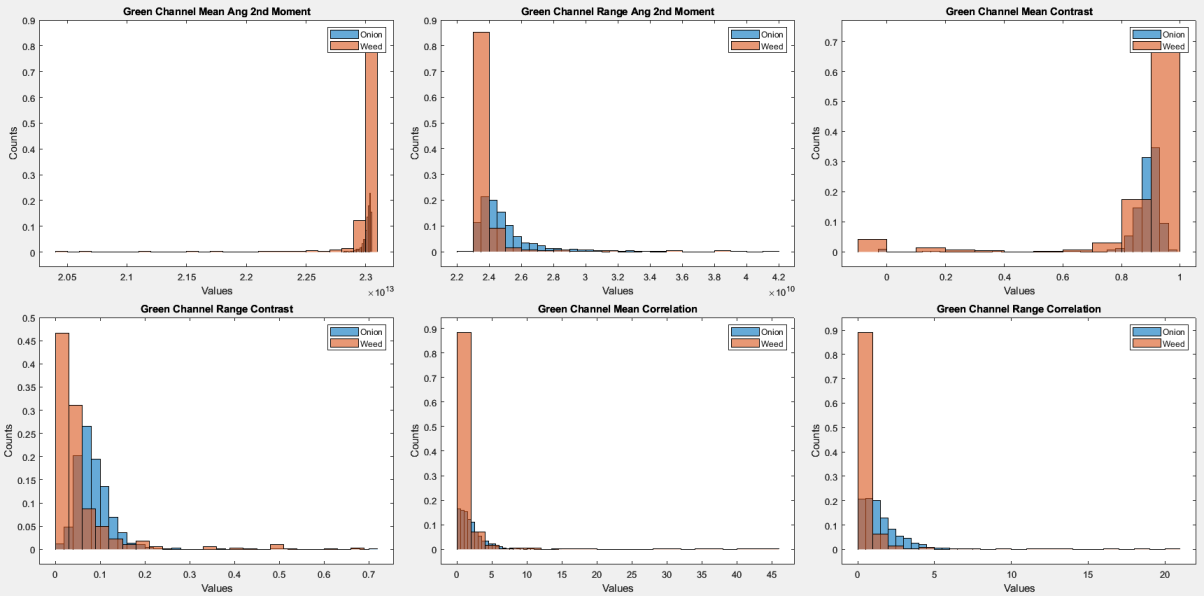


Figure 36 Green Channel Texture Features.

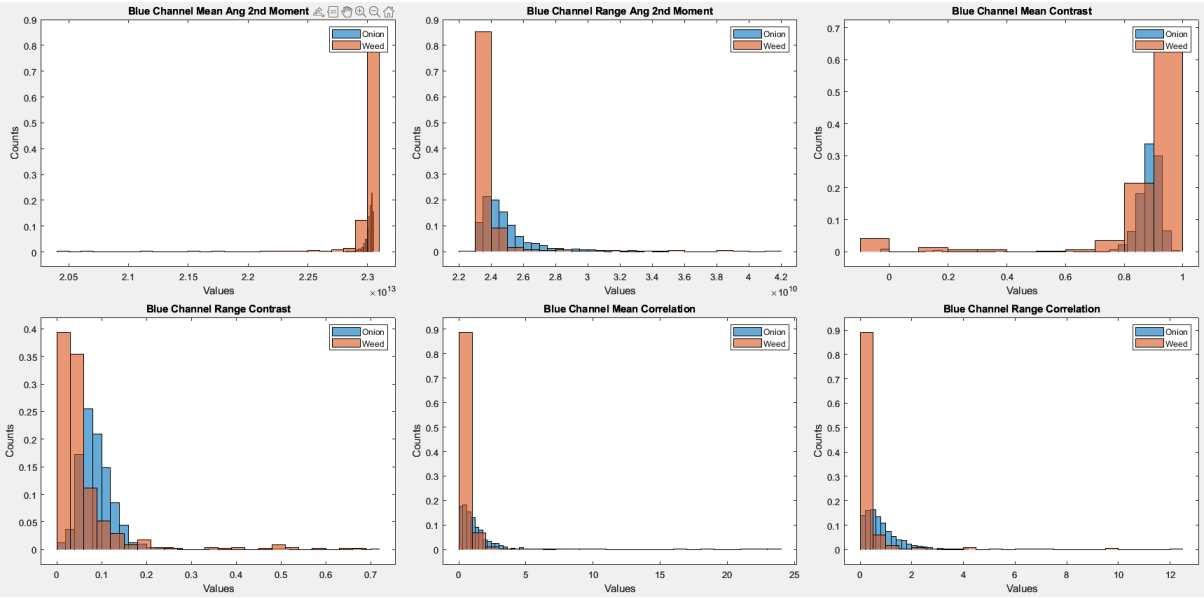


Figure 37 Blue Channel Texture Features.

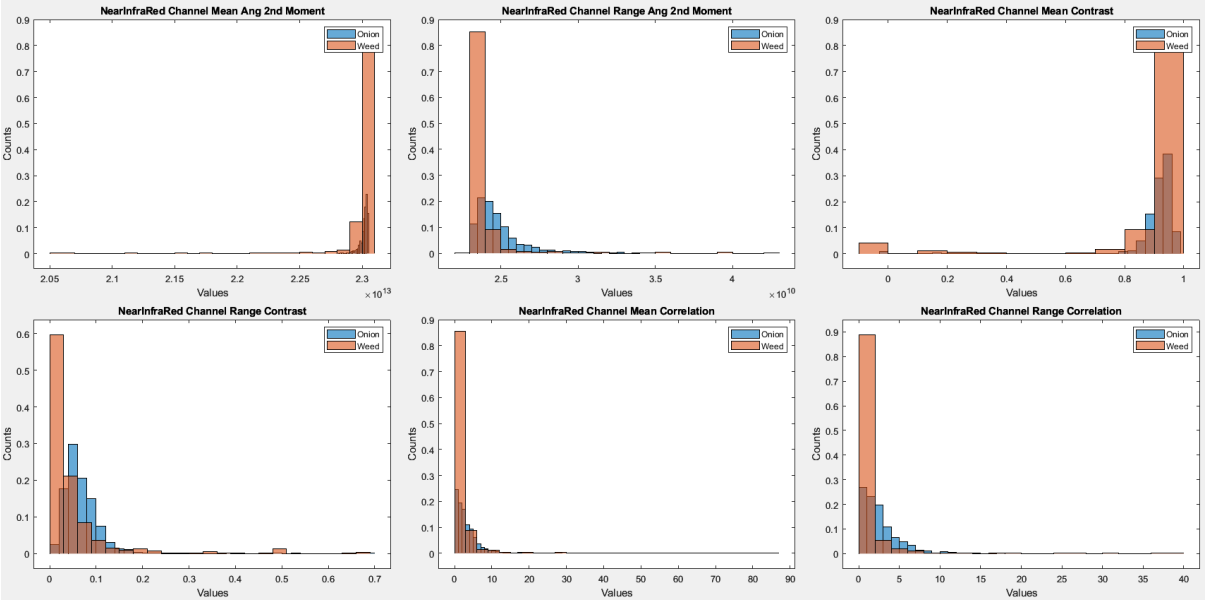


Figure 38 Near Infra-Red Channel Texture Features.

6 MATLAB scripts

6.1 Segmentation & Detection

6.1.1 Home-made Segmentation/Detection Functions File

```
%This file contains home-made functions for the segmentation/detection
%tasks
classdef segment_plant
    methods
      %get binary mask (uses morphology) other functions may use this mask
      %'segmentmask'
        function [mask] = getBinMask(~,im)
            %Keeps green channel and converts B and R to 'black' channel
            %https://uk.mathworks.com/help/images/display-separated-color-
channels-of-rgb-image.html
            [~,G,~] = imsplit(im);
            allBlack = zeros(size(im,1,2),class(im));
            justG = cat(3,allBlack,G,allBlack);

            %Convert to Greyscale
            grey = rgb2gray(justG);

            %Median Filter to remove noise
            Kmedian = medfilt2(grey);

            %Contrast Adjustment
            J = imadjust(Kmedian,[0.15 1],[]);

            %Binarize using custom threshold
            T = adaptthresh(J,0.73);
            BW = imbinarize(J,T);

            %Remove components connected to border
            CB = imclearborder(BW);
```

```
            %Remove Small Objects
            m = bwareaopen(CB, 150);

            %get original image segmented by mask
            maskedRgbImage = bsxfun(@times, im, cast(m, 'like', im));

            %now remove remaining 'brown' areas stuck to leaves by
            %converting to 'lab' colour space
            lab = rgb2lab(maskedRgbImage);

            labRemover = imbinarize(1-(~lab(:,:,1) + lab(:,:,2)));

            mask = bwareaopen(labRemover,5);

        end
        function [leafCountPreOpen,leafCountPostOpen] =
segmentHoughCount(~,segmentMask)
            %hough transform
            [c1, ~] = imfindcircles(segmentMask,[6,30], 'Sensitivity', 0.9);
            leafCountPreOpen = length(c1);

            %opening to segment leaves
            se = strel("disk", 4);
            op = imopen(segmentMask, se);

            %hough transform
            [c1, ~] = imfindcircles(op,[6,30], 'Sensitivity', 0.9);
            leafCountPostOpen = length(c1);

        end
        function [CCCount] = connectedCompCount(~,segmentMask)
            %opening and connected component count
            se = strel("disk", 4);
            op = imopen(segmentMask, se);
            CCCount = bwconncomp(op).NumObjects;

        end
        %get sift points to see if there is a way to match leaf count
        function [sift_points] = returnSIFTPoints(~,im, segmentmask) % returns
SIFT points from image

            %get original image segmented by mask
            maskedRgbImage = bsxfun(@times, im, cast(segmentmask, 'like', im));


            grey = rgb2gray(maskedRgbImage);


            sift_points = detectSIFTFeatures(grey, ContrastThreshold= 0.06);


        end
        %watershed transform to segment contiguous regions of interest
        %(i.e. split 'connected' leaves)
        %https://uk.mathworks.com/help/images/ref/watershed.html
        function [ws_count] = getWatershed(~,im, segmentmask) % returns watershed
count

            %get original image segmented by mask
```

```
            maskedRgbImage = bsxfun(@times, im, cast(segmentmask, 'like', im));

            grey = im2gray(maskedRgbImage);
            bw = imbinarize(grey);

            %open to remove stalks; dilate to remove unwanted leave
            %'ridges' (which watershed is sensitive to)
            se = strel('disk',4);
            se2 = strel('disk',2);
            op = imopen(bw,se); %open
            op = imdilate(op,se2); %dilate

            D = bwdist(~op);

            D= -D;
            L = watershed(D);
            L(~op) = 0;
            L = bwareaopen(L, 105); % remove small detections (not leaves)
            ws_count = length(regionprops(L));


        end
        function [mlce] = meanLeafCountError(~,GT,Pred)
            %gets mean leaf count error when given ground truth and
            %predicted value array.
            errors = [];
            for i = 1:length(GT)
                error = abs(GT(i)-Pred(i));
                errors = [errors error];
            end
            mlce = mean(errors);

        end
    end
end
```

6.1.2 Dice Score Calculation

```
%Gets DICE score of segmented whole-plant,  plots a histogram for each image,
displays scores as a table and displays mean score (command window output) (0.96)
clear; close all;

% p = pwd;
% d = pwd + "\plant image dataset";
p = "C:\Users\oakle\OneDrive - University of Lincoln\MSc Intelligent
Vision\CompVision\assessment\data\plant image dataset";

file_list = dir(p);

seg_plant = segment_plant; %get created functions

GT = {}; %ground truth
OG = {}; %original images

scores = []; %dice scores
fnames = []; %list of file names

%add to list of images and labels to be processed
```

```
for i = 1:numel(file_list)

    file = file_list(i);
    [filepath,name,ext] = fileparts(file.name);
    abs_path = fullfile(file.folder, file.name);

    %if file name contains label, add to GT (Ground Truth) list
    if regexp(file.name, "plant[0-9]+_label\.png")
        I = imread(abs_path); % load image
        I = I > 0; %binarize
        GT{end+1} = I; % append to image array
    end

    %if file name rgb, add to image list
    if regexp(file.name, "plant[0-9]+_rgb\.png")
        fn = string(file.name);
        fnames = [fnames fn];
        I = imread(abs_path); % load image
        OG{end+1} = I; % append to image array
    end

end

for l=1:length(GT)
    label = cell2mat(GT(l));
    og = cell2mat(OG(l));
    pred = seg_plant.getBinMask(og);
    score = dice(pred,label);
    scores = [scores score];

end

%reshape arrays for table
fnames = reshape(fnames,[length(fnames),1]);
scores = reshape(scores,[length(scores),1]);

%create & display table
metrics = table(fnames,scores);
display(metrics);

%bar chart of dice scores
bar(scores);
title("Dice Scores of Plant Segmentation");
xlabel("Image Number");
xticks(1:length(scores));
ylabel("Dice Score");
%display mean dice score
fprintf("Mean Dice:");
m = mean(scores);
fprintf(string(m));
fprintf("\n");
```

6.1.3 Counting Leaves

```
%Get's leaf counts for each method explored as a table, compares average error for
each method (command window output).
clear; close all;
```

```
p = "C:\Users\oakle\OneDrive - University of Lincoln\MSc Intelligent
Vision\CompVision\assessment\data\plant image dataset";

file_list = dir(p);%get directory content
GTCountsFile = readmatrix('Leaf_counts.csv');
GTCounts = GTCountsFile(:,2);% Ground Truth leaf counts

seg_plant = segment_plant; %get created functions

OG = {}; %original images

fnames = []; %list of file names
Hough1counts = [];%leaf counts (Hough, no opening)
Hough2counts = [];%leaf counts (Hough, opening)
CCcounts = []; %leaf counts (Connected Components)
sifts = []; %sift points
wss = []; %watershed count

%add to list of images and labels to be processed
for i = 1:numel(file_list)

    file = file_list(i);
    [filepath,name,ext] = fileparts(file.name);
    abs_path = fullfile(file.folder, file.name);

    %if file name rgb, add to image list
    if regexp(file.name, "plant[0-9]+_rgb\.png")
        fn = string(file.name);
        fnames = [fnames fn];
        I = imread(abs_path); % load image
        OG{end+1} = I; % append to image array
    end
end

for l=1:length(OG)
    og = cell2mat(OG(l));
    %get segmented mask
    segmented = seg_plant.getBinMask(og);
    %get leaf counts using various methods
    [Hough1c,Hough2c] = seg_plant.segmentHoughCount(segmented); %hough pre and
post opening
    ccC = seg_plant.connectedCompCount(segmented); %opening and counting connected
components

    %append to lists
    Hough1counts = [Hough1counts,Hough1c];
    Hough2counts = [Hough2counts,Hough2c];
    CCcounts = [CCcounts ccC];
    sifts = [sifts seg_plant.returnSIFTPoints(og, segmented).Count];
    wss = [wss seg_plant.getWatershed(og, segmented)];
end

%reshape arrays for table
fnames = reshape(fnames,[length(fnames),1]);
Hough1counts = reshape(Hough1counts,[length(Hough1counts),1]);
Hough2counts = reshape(Hough2counts,[length(Hough2counts),1]);
CCcounts = reshape(CCcounts,[length(CCcounts),1]);
sifts = reshape(sifts,[length(sifts),1]);
wss = reshape(wss,[length(wss),1]);
```

```
%create & display table
metrics = table(fnames,GTCounts,Hough1counts,Hough2counts, CCcounts,sifts,wss);
display(metrics);

%get errors
method = ["Hough" , "HoughOpened", "ConnectedComp","SIFT","Watershed"];
error = [seg_plant.meanLeafCountError(GTCounts,Hough1counts), ...
    seg_plant.meanLeafCountError(GTCounts,Hough2counts), ...
    seg_plant.meanLeafCountError(GTCounts,CCcounts), ...
    seg_plant.meanLeafCountError(GTCounts,sifts), ...
    seg_plant.meanLeafCountError(GTCounts,wss)];

method = reshape(method,[length(method),1]);
error = reshape(error,[length(error),1]);

errorMetrics = table(method,error);
display(errorMetrics);
```

6.1.4 Computing Bounding Boxes from Watershed Method

```
%Visualises bounding boxes using watershed method.
clc;clear;close all;

%read images and compare
clear; close all;

t = tiledlayout(4,4,'TileSpacing','Compact');

p = "C:\Users\oakle\OneDrive - University of Lincoln\MSc Intelligent
Vision\CompVision\assessment\data\plant image dataset";

file_list = dir(p);%get directory content
GTCountsFile = readmatrix('Leaf_counts.csv');
GTCounts = GTCountsFile(:,2);% Ground Truth leaf counts

seg_plant = segment_plant; %get created functions

OG = {}; %original images

fnames = []; %list of file names
wss = []; %watershed count

%add to list of images and labels to be processed
for i = 1:numel(file_list)

    file = file_list(i);
    [filepath,name,ext] = fileparts(file.name);
    abs_path = fullfile(file.folder, file.name);

    %if file name rgb, add to image list
    if regexp(file.name, "plant[0-9]+_rgb\.png")
        fn = string(file.name);
        fnames = [fnames fn];
        I = imread(abs_path); % load image
        OG{end+1} = I; % append to image array
    end
```

```
end

for l=1:length(OG)
    og = cell2mat(OG(l));
    %get segmented mask
    segmented = seg_plant.getBinMask(og);

    maskedRgbImage = bsxfun(@times, og, cast(segmented, 'like', og));

    grey = im2gray(maskedRgbImage);
    bw = imbinarize(grey);

    se = strel('disk',4);
    se2 = strel('disk',3);
    op = imopen(bw,se);
    op = imdilate(op,se2);



    D = bwdist(~op);
    D= -D;
    L = watershed(D);
    L(~op) = 0;
    L = bwareaopen(L, 105);
    bbox = regionprops(L,'BoundingBox');
    stats = length(regionprops(L));
    rgb = label2rgb(L,'jet',[.5 .5 .5]);


    nexttile;
    imshow(maskedRgbImage);
    hold on

    for k = 1:numel(bbox)
        rectangle('Position', bbox(k).BoundingBox, 'EdgeColor', 'r');
    end
    title(fnames(l))
end
```

****Note that further experimentation files are in supporting documentation. Included above are all the files that accompany this document's explanation.

6.2 Feature Extraction

6.2.1 Homemade functions file for shape and feature extraction.

```
%This file contains home-made functions for feature calculation
%tasks
classdef feature_calc
    methods
        %get shape feature calculations
        function [featNames, featCalcs] = getFeatures(self,binIm)
            %gets feature calculations
            %single out onions, compute features
            rp =
regionprops(binIm,"Circularity","Eccentricity","Solidity","Perimeter","Area");
```

```
            circs = [];
            eccs = [];
            ss = [];
            nComps = [];

            for i = 1: length(rp)
                circs = [circs; rp(i).Circularity];
                eccs = [eccs; rp(i).Eccentricity];
                ss = [ss ; rp(i).Solidity];
                p = rp(i).Perimeter;
                a = rp(i).Area;

                nComp = (p/(2*pi*(sqrt(a/pi))));     %inverse of Schwartzberg
                nComps = [nComps; nComp];

            end
            featNames =
["Circularity","Eccentricity","Solidity","NonCompactness"];
            featCalcs = [circs eccs ss nComps];
        end

        %get texture feature calculations
        function [featNames, featCalcs] = getTextureFeatures(self, patch_type,
patch_im, rgbIm, chan_type)

            %label individual connected components
            wCC = bwlabel(patch_im);
            wCC_count = bwconncomp(wCC).NumObjects;

            chann = self.getChannel(rgbIm,chan_type);

            %orientations for greycomatrix
            angles = [[0 1];[-1 1];[-1 0];[-1 -1]];


            %mean features
            Masms = zeros(wCC_count,1);
            Mconts = zeros(wCC_count,1);
            Mcorrs = zeros(wCC_count,1);

            %range features
            Rasms = zeros(wCC_count,1);
            Rconts = zeros(wCC_count,1);
            Rcorrs = zeros(wCC_count,1);

            %waitbar ('cos it takes a while!)
            waitbar_text = "Calculating " + chan_type +" channel texture features:
" + patch_type;
            f = waitbar(0,waitbar_text);

            %for each object
            for j = 1: wCC_count
                chan = chann;
                binaryImage = ismember(wCC, j);
                chan(~binaryImage) = NaN;

                asms = zeros(1,4);
                conts = zeros(1,4);
                corrs = zeros(1,4);
```

```
                %for each angle
                for i=1: length(angles)
                    %get graylevel covariance matrix
                    glcm = graycomatrix(chan, 'offset', angles(i,:),'NumLevels',
256, 'Symmetric', false);
                    asm = sum(glcm(:).^2);%get Mean angular 2nd moment
                    asms(i) = asm;
                    stats = graycoprops(glcm,["Contrast","Correlation"]);
                    conts(i) = stats.Contrast;
                    corrs(i) = stats.Correlation;
                end

                Masms(j,1) = mean(asms);
                Rasms(j,1) = range(asms);

                Mconts(j,1) = mean(conts);
                Rconts(j,1) = range(conts);

                Mcorrs(j,1) = mean(corrs);
                Rcorrs(j,1) = range(corrs);


                waitbar((j/wCC_count),f);
            end
            featNames = ["Mean Ang 2nd Moment", "Range Ang 2nd Moment", "Mean
Contrast", "Range Contrast", "Mean Correlation", "Range Correlation"];
            featCalcs = [Masms Rasms Mcorrs Rcorrs Mconts Rconts];
            close(f)
        end

        %gets image channel based on input
        function [chanIm] = getChannel(~, im, chanString)
            if chanString == "red"
                [R,G,B] = imsplit(im);
                chanIm = R;

            elseif chanString == "green"
                [R,G,B] = imsplit(im);
                chanIm = G;

            elseif chanString == "blue"
                [R,G,B] = imsplit(im);
                chanIm = B;

            elseif chanString == "inf"
                chanIm = im;

            end
        end
    end
end
```

6.2.2 Calculate Shape Features & Display distributions

```
%calculate shape features


clear;clc;close all;
```

```
p = "C:\Users\oakle\OneDrive - University of Lincoln\MSc Intelligent
Vision\CompVision\assessment\data\onions";

file_list = dir(p);

ims = {}; %images

featFuncs = feature_calc; %get created functions

%onion features
Ocircs = [];
Oeccs = [];
Oss = [];
OnComps = [];

%weed features
Wcircs = [];
Weccs = [];
Wss = [];
WnComps = [];



%add to list of images and labels to be processed
for i = 1:numel(file_list)

    file = file_list(i);
    [filepath,name,ext] = fileparts(file.name);
    abs_path = fullfile(file.folder, file.name);

    %if file name contains label, add to GT (Ground Truth) list
    if regexp(file.name, "\d\d_truth\.png")
        logIm = imread(abs_path); % load image
        greyIm = rgb2gray(logIm);%convert to greyscale
        ims{end+1} = greyIm; % append to image array
    end

end
f = waitbar(0,'Calculating Shape Features');

for l=1:length(ims)
    im = cell2mat(ims(l));
    %extract weeds and onions using label threshold
    oni = im == 76;

    weed = im == 29;

    %get features for each class in image
    [~, oniFeats] = featFuncs.getFeatures(oni);
    [~, weedFeats] = featFuncs.getFeatures(weed);

    %update onion features
    Ocircs = [Ocircs ;oniFeats(:,1)];
    Oeccs = [Oeccs ;oniFeats(:,2)];
    Oss = [Oss ;oniFeats(:,3)];
    OnComps = [OnComps ;oniFeats(:,4)];

    %update weed features
    Wcircs = [Wcircs ;weedFeats(:,1)];
```

```
    Weccs = [Weccs ;weedFeats(:,2)];
    Wss = [Wss ;weedFeats(:,3)];
    WnComps = [WnComps ;weedFeats(:,4)];
    waitbar((l/length(ims)),f,'Calculating Shape Features');

end

featureNs = ["Circularity","Eccentricity","Solidity","NonCompactness"];
oFeats = [Ocircs Oeccs Oss OnComps];
wFeats = [Wcircs Weccs Wss WnComps];

%write shape features to text file
writematrix(oFeats,"./savedShapeFeats/OnionShapeFeatures.txt");
writematrix(wFeats,"./savedShapeFeats/WeedShapeFeatures.txt");

close(f)


t = tiledlayout(2,2,"TileSpacing","compact");

%display shape feature distributions
for i = 1:length(featureNs)
    nexttile;

    histogram(oFeats(:,i),Normalization="probability");
    t_name = featureNs(i);
    title(t_name);

    hold;

    histogram(wFeats(:,i),Normalization="probability");
    xlabel("Values");
    ylabel("Counts");
    legend(["Onion","Weed"]);
    %legend("Location","northwest");
    axis padded;
end
```

6.2.3 Calculate Shape Features

```
clear;clc;close all;
%Calculate the normalised grey-level co-occurrence matrix in four orientations
(0°, 45°, 90°, 135°)
%for the patches from both classes
%, separately for each of the colour channels (red, green, blue, near infra-red)
% For each orientation, calculate the first three features proposed by Haralick et
al. (Angular Second Moment, Contrast, Correlation)
% and produce per-patch features by calculating the feature average and range
across the 4 orientations


p = "C:\Users\oakle\OneDrive - University of Lincoln\MSc Intelligent
Vision\CompVision\assessment\data\onions";

fc = feature_calc;


file_list = dir(p);
```

```
imsRGB = {}; %images
imsINF = {};

featFuncs = feature_calc; %get created functions

gt_ims = {}; %onion gt images

%add to list of images and labels to be processed
for i = 1:numel(file_list)

    file = file_list(i);
    [filepath,name,ext] = fileparts(file.name);
    abs_path = fullfile(file.folder, file.name);

    %if file name contains label, add to GT (Ground Truth) list
    if regexp(file.name, "\d\d_truth\.png")
        logIm = imread(abs_path); % load image
        greyIm = rgb2gray(logIm);%convert to greyscale
        gt_ims{end+1} = greyIm; % append to image array
    end

    if regexp(file.name, "\d\d_rgb\.png")
        rgb = imread(abs_path); % load image
        imsRGB{end+1} = rgb; % append to image array
    end

    if regexp(file.name, "\d\d_depth\.png")
        dep = imread(abs_path); % load image
        imsINF{end+1} = dep; % append to image array
    end

end

%per image waitbar
wb = waitbar(0,'Calculating texture features for image: 1');

WeedredFeats = zeros(1,6);
OnionredFeats = zeros(1,6);
WeedgreenFeats = zeros(1,6);
OniongreenFeats = zeros(1,6);
WeedblueFeats = zeros(1,6);
OnionblueFeats = zeros(1,6);
WeedinfFeats = zeros(1,6);
OnioninfFeats = zeros(1,6);

for l=1: length(gt_ims)
    im = cell2mat(gt_ims(l));
    oniM = im == 76;

    weedM = im == 29;

    %get texture features for each channel, and each class...
    [~, wFeatsR] = fc.getTextureFeatures("Weeds",weedM,cell2mat(imsRGB(l)),"red");
    [~, oFeatsR] = fc.getTextureFeatures("Onions",oniM,cell2mat(imsRGB(l)),"red");

    [~, wFeatsG] =
fc.getTextureFeatures("Weeds",weedM,cell2mat(imsRGB(l)),"green");
```

```
    [~, oFeatsG] =
fc.getTextureFeatures("Onions",oniM,cell2mat(imsRGB(l)),"green");

    [~, wFeatsB] =
fc.getTextureFeatures("Weeds",weedM,cell2mat(imsRGB(l)),"blue");
    [~, oFeatsB] =
fc.getTextureFeatures("Onions",oniM,cell2mat(imsRGB(l)),"blue");

    [~, wFeatsINF] =
fc.getTextureFeatures("Weeds",weedM,cell2mat(imsINF(l)),"inf");
    [~, oFeatsINF] =
fc.getTextureFeatures("Onions",oniM,cell2mat(imsINF(l)),"inf");

    WeedredFeats = [WeedredFeats; wFeatsR];
    OnionredFeats = [OnionredFeats; oFeatsR];

    WeedgreenFeats = [WeedgreenFeats; wFeatsG];
    OniongreenFeats = [OniongreenFeats; oFeatsG];

    WeedblueFeats = [WeedblueFeats; wFeatsB];
    OnionblueFeats = [OnionblueFeats; oFeatsB];

    WeedinfFeats = [WeedinfFeats; wFeatsINF];
    OnioninfFeats = [OnioninfFeats; oFeatsINF];

    image_no = int2str((l+1));
    waitbar_update = "Calculating texture features for image: " + image_no;
    waitbar((l/length(gt_ims)),wb,waitbar_update);

end

close(wb)



featureNs = ["Mean Ang 2nd Moment", "Range Ang 2nd Moment", "Mean Contrast",
"Range Contrast", "Mean Correlation", "Range Correlation"];


%removing first row from zeros instantiation
WeedredFeats(1,:) = [];
OnionredFeats(1,:) = [];
WeedgreenFeats(1,:)= [];
OniongreenFeats(1,:)= [];
WeedblueFeats(1,:)= [];
OnionblueFeats(1,:)= [];
WeedinfFeats(1,:)= [];
OnioninfFeats(1,:)= [];

%Save to text files... do not want to run again
writematrix(WeedredFeats,"./savedTextureFeats/redWeedFeatures.txt");
writematrix(OnionredFeats,"./savedTextureFeats/redOnionFeatures.txt");
writematrix(WeedgreenFeats,"./savedTextureFeats/greenWeedFeatures.txt");
writematrix(OniongreenFeats,"./savedTextureFeats/greenOnionFeatures.txt");
writematrix(WeedblueFeats,"./savedTextureFeats/blueWeedFeatures.txt");
writematrix(OnionblueFeats,"./savedTextureFeats/blueOnionFeatures.txt");
writematrix(WeedinfFeats,"./savedTextureFeats/INFWeedFeatures.txt");
writematrix(OnioninfFeats,"./savedTextureFeats/INFOnionFeatures.txt");
```

6.2.4 Plot Texture Features for Each Channel (reads txt file)

```
clear;clc;close all;
%Calculate the normalised grey-level co-occurrence matrix in four orientations
(0°, 45°, 90°, 135°)
%for the patches from both classes
%, separately for each of the colour channels (red, green, blue, near infra-red)
% For each orientation, calculate the first three features proposed by Haralick et
al. (Angular Second Moment, Contrast, Correlation)
% and produce per-patch features by calculating the feature average and range
across the 4 orientations
% Select one feature from each of the colour channels and plot the distribution.

redOnionFeat = readmatrix("./savedTextureFeats/redOnionFeatures.txt");
redWeedFeat = readmatrix("./savedTextureFeats/redWeedFeatures.txt");

greenOnionFeat = readmatrix("./savedTextureFeats/greenOnionFeatures.txt");
greenWeedFeat = readmatrix("./savedTextureFeats/greenWeedFeatures.txt");

blueOnionFeat = readmatrix("./savedTextureFeats/blueOnionFeatures.txt");
blueWeedFeat = readmatrix("./savedTextureFeats/blueWeedFeatures.txt");

infOnionFeat = readmatrix("./savedTextureFeats/INFOnionFeatures.txt");
infWeedFeat = readmatrix("./savedTextureFeats/INFWeedFeatures.txt");

featureNs = ["Mean Ang 2nd Moment", "Range Ang 2nd Moment", "Mean Contrast",
"Range Contrast", "Mean Correlation", "Range Correlation"];

%plotting hists

figure;
t = tiledlayout(2,3,"TileSpacing","compact");
for i = 1:length(featureNs)
    nexttile;

    histogram(redOnionFeat(:,i),Normalization="probability");
    t_name = "Red Channel " + featureNs(i);
    title(t_name);

    hold;
    histogram(redWeedFeat(:,i),Normalization="probability");
    xlabel("Values");
    ylabel("Counts");
    legend(["Onion","Weed"]);
    %legend("Location","northwest");
    axis padded;
end

figure;
t = tiledlayout(2,3,"TileSpacing","compact");
for i = 1:length(featureNs)
    nexttile;

    histogram(greenOnionFeat(:,i),Normalization="probability");

    t_name = "Green Channel " + featureNs(i);
    title(t_name);

    hold;
    histogram(greenWeedFeat(:,i),Normalization="probability");
```

```
    xlabel("Values");
    ylabel("Counts");
    legend(["Onion","Weed"]);
    %legend("Location","northwest");
    axis padded;
end

figure;
t = tiledlayout(2,3,"TileSpacing","compact");
for i = 1:length(featureNs)
    nexttile;

    histogram(blueOnionFeat(:,i),Normalization="probability");

    t_name = "Blue Channel " + featureNs(i);
    title(t_name);

    hold;
    histogram(blueWeedFeat(:,i),Normalization="probability");
    xlabel("Values");
    ylabel("Counts");
    legend(["Onion","Weed"]);
    %legend("Location","northwest");
    axis padded;
end

figure;
t = tiledlayout(2,3,"TileSpacing","compact");
for i = 1:length(featureNs)
    nexttile;

    histogram(infOnionFeat(:,i),Normalization="probability");
    t_name = "NearInfraRed Channel " + featureNs(i);
    title(t_name);

    hold;
    histogram(infWeedFeat(:,i),Normalization="probability");
    xlabel("Values");
    ylabel("Counts");
    legend(["Onion","Weed"]);
    %legend("Location","northwest");
    axis padded;
end
```

6.2.5 Count onions/weeds for train/test split

```
clear;clc;close all;

%counts objects for SVM train/test splitting

p = "C:\Users\oakle\OneDrive - University of Lincoln\MSc Intelligent
Vision\CompVision\assessment\data\onions";
ims = {};

%calculate how many weeds/onions are in images 1-18 for split
file_list = dir(p);

%add to list of images and labels to be processed
```

```
for i = 1:numel(file_list)

    file = file_list(i);
    [filepath,name,ext] = fileparts(file.name);
    abs_path = fullfile(file.folder, file.name);

    %if file name contains label, add to GT (Ground Truth) list
    if regexp(file.name, "\d\d_truth\.png")
        logIm = imread(abs_path); % load image
        greyIm = rgb2gray(logIm);%convert to greyscale
        ims{end+1} = greyIm; % append to image array
    end

end

trainWCount = 0;
trainOCount = 0;

testWcounts = [];
testOcounts = [];


for i=1:18
    im = cell2mat(ims(i));
    weed = im == 29;
    oni = im == 76;

    wcount = bwconncomp(weed).NumObjects;
    ocount = bwconncomp(oni).NumObjects;
    trainWCount = trainWCount + wcount;
    trainOCount = trainOCount + ocount;
end

% Train WeedCount 452 OnionCount 918


%%%%%%%
%Find weeds/ onions in test images for visualisation%
%%%%%%%
for i=19:20
    im = cell2mat(ims(i));
    weed = im == 29;
    oni = im == 76;

    wcount = bwconncomp(weed).NumObjects;
    ocount = bwconncomp(oni).NumObjects;
    testWcounts = [testWcounts wcount];
    testOcounts = [testOcounts ocount];
end

display(trainWCount);
display(trainOCount);
display(testWcounts);
display(testOcounts);
```

6.2.6 Classify different feature bunches with an SVM, perform importance analysis, reclassify x2, visualize classified test images.

```
clear;clc;close all;
```

```
%SVM comparison & Importance analysis


p = "C:\Users\oakle\OneDrive - University of Lincoln\MSc Intelligent
Vision\CompVision\assessment\data\onions";

%%%%%%%%%%%%%%%%%%%%
%Only Shape Features
%%%%%%%%%%%%%%%%%%%%

%train onions = 918 train weeds = 453
shapeFeatsO = readmatrix("./savedShapeFeats/OnionShapeFeatures.txt");
shapeFeatsW = readmatrix("./savedShapeFeats/WeedShapeFeatures.txt");


%onions = label 0; weeds = label 1;
%train
shapeTrainFeats = [shapeFeatsO(1:918,:) ; shapeFeatsW(1:453,:)];
trainLabels = [zeros(918,1); ones(453,1)];

%test
testFeats = [shapeFeatsO(919:end,:) ; shapeFeatsW(454:end,:)]; %198
testLabels = [zeros(88,1) ; ones(109,1)];

%fit SVM
SVMModel = fitcsvm(shapeTrainFeats,trainLabels);

%predict
[predictions,~] = predict(SVMModel,testFeats);

%confusion matrix
C = confusionmat(testLabels,predictions);

%get precision and recall for each class
shapePrecision = diag(C) ./ sum(C,2);
shapeRecall = diag(C) ./ sum(C,1)';

shapeF1 = 2 * (mean(shapePrecision) * mean(shapeRecall))/ (mean(shapePrecision) +
mean(shapeRecall));

figure;
shapeCc = confusionchart(C);
shapeCc.Title = "Onions/Weeds Shape Confusion Matrix";


%%%%%%%%%%%%%%%%%%%%
%Only Texture Features
%%%%%%%%%%%%%%%%%%%%
textFeatsRedO = readmatrix("./savedTextureFeats/redOnionFeatures.txt");
textFeatsRedW = readmatrix("./savedTextureFeats/redWeedFeatures.txt");

textFeatsGreenO = readmatrix("./savedTextureFeats/greenOnionFeatures.txt");
textFeatsGreenW = readmatrix("./savedTextureFeats/greenWeedFeatures.txt");

textFeatsBlueO = readmatrix("./savedTextureFeats/blueOnionFeatures.txt");
textFeatsBlueW = readmatrix("./savedTextureFeats/blueWeedFeatures.txt");

textFeatsInfO = readmatrix("./savedTextureFeats/INFOnionFeatures.txt");
```

```
textFeatsInfW = readmatrix("./savedTextureFeats/INFWeedFeatures.txt");

textFeatsO = [textFeatsRedO textFeatsGreenO textFeatsBlueO textFeatsInfO];
textFeatsW = [textFeatsRedW textFeatsGreenW textFeatsBlueW textFeatsInfW];

%train
textTrainFeats = [textFeatsO(1:918,:) ; textFeatsW(1:453,:)];
trainLabels = [zeros(918,1); ones(453,1)];

%test
testFeats = [textFeatsO(919:end,:) ; textFeatsW(454:end,:)];
testLabels = [zeros(88,1) ; ones(109,1)];

%fit SVM
SVMModel = fitcsvm(textTrainFeats,trainLabels);

%predict
[predictions,~] = predict(SVMModel,testFeats);

%confusion matrix
C = confusionmat(testLabels,predictions);

%get precision and recall for each class
textPrecision = diag(C) ./ sum(C,2);
textRecall = diag(C) ./ sum(C,1)';
textF1 = 2 * (mean(textPrecision) * mean(shapeRecall)) / (mean(textPrecision) +
mean(shapeRecall));

figure;
textCc = confusionchart(C);
textCc.Title = "Onions/Weeds Texture Confusion Matrix";

%%%%%%%%%%%%%%%%%%%%
%Shape & Texture Features
%%%%%%%%%%%%%%%%%%%%
shapeTextO = [shapeFeatsO textFeatsO];
shapeTextW = [shapeFeatsW textFeatsW];

%train
shapeTextTrainFeats = [shapeTextO(1:918,:) ; shapeTextW(1:453,:)];
trainLabels = [zeros(918,1); ones(453,1)];

%test
testFeats = [shapeTextO(919:end,:) ; shapeTextW(454:end,:)];
testLabels = [zeros(88,1) ; ones(109,1)];

%fit SVM
SVMModel = fitcsvm(shapeTextTrainFeats,trainLabels);

%predict
[predictions,~] = predict(SVMModel,testFeats);

%confusion matrix
C = confusionmat(testLabels,predictions);

%get precision and recall for each class
shapeTextPrecision = diag(C) ./ sum(C,2);
shapeTextRecall = diag(C) ./ sum(C,1)';
```

```
shapeTextF1 = 2 * (mean(shapeTextPrecision) * mean(shapeTextRecall)) /
(mean(shapeTextPrecision) + mean(shapeTextRecall));

figure;
shapeTextCc = confusionchart(C);
shapeTextCc.Title = "Onions/Weeds Shape & Texture Confusion Matrix";

%%%%%%%%%%%%%%%%%%%%
%Importance Scores
%%%%%%%%%%%%%%%%%%%%
full_data = [shapeTextTrainFeats trainLabels];
%Create table from data
full_data_T = array2table(full_data,'VariableNames', ...
    {'Circularity','Eccentricity','Solidity','NonCompactness'...
    'Red_Mean_Ang_2nd_Moment', 'Red_Range_Ang_2nd_Moment', 'Red_Mean_Contrast',
'Red_Range_Contrast', 'Red_Mean_Correlation', 'Red_Range_Correlation', ...
    'Green_Mean_Ang_2nd_Moment', 'Green_Range_Ang_2nd_Moment',
'Green_Mean_Contrast', 'Green_Range_Contrast', 'Green_Mean_Correlation',
'Green_Range_Correlation', ...
    'Blue_Mean_Ang_2nd_Moment', 'Blue_Range_Ang_2nd_Moment', 'Blue_Mean_Contrast',
'Blue_Range_Contrast', 'Blue_Mean_Correlation', 'Blue_Range_Correlation', ...
    'INF_Mean_Ang_2nd_Moment', 'INF_Range_Ang_2nd_Moment', 'INF_Mean_Contrast',
'INF_Range_Contrast', 'INF_Mean_Correlation', 'INF_Range_Correlation',...
    'patch_type'});

[idx,scores] = fscchi2(full_data_T,'patch_type');
figure;
bar(scores(idx));
set(gca,'xtick',1:28);
xticklabels(strrep(full_data_T.Properties.VariableNames(idx),'_','\_'));
xtickangle(45);
xlabel('Predictor rank')
ylabel('Predictor importance score')

%%%%%%%%%%%%%%%%%%%%
%Importance-Chosen (10) Features
%%%%%%%%%%%%%%%%%%%%
%10 most important are:
%Solidity; Circularity; NonCompactness; Red_Range_Contrast;
%Green_Range_Contrast; Blue_Range_Contrast; INF_Range_Contrast;
%Green_Mean_Contrast; Blue_Range_Correlation; Red_Mean_Contrast;

import_features =
full_data_T{:,["Solidity","Circularity","NonCompactness","Red_Range_Contrast",...

"Green_Range_Contrast","Blue_Range_Contrast","INF_Range_Contrast","Green_Mean_Cont
rast","Blue_Range_Correlation","Red_Mean_Contrast"]};


trainLabels = [zeros(918,1); ones(453,1)];



%test
testFeats = [shapeTextO(919:end,:) ; shapeTextW(454:end,:)];


%Extract important features from test too
full_test_T = array2table(testFeats,'VariableNames', ...
```

```
    {'Circularity','Eccentricity','Solidity','NonCompactness'...
    'Red_Mean_Ang_2nd_Moment', 'Red_Range_Ang_2nd_Moment', 'Red_Mean_Contrast',
'Red_Range_Contrast', 'Red_Mean_Correlation', 'Red_Range_Correlation', ...
    'Green_Mean_Ang_2nd_Moment', 'Green_Range_Ang_2nd_Moment',
'Green_Mean_Contrast', 'Green_Range_Contrast', 'Green_Mean_Correlation',
'Green_Range_Correlation', ...
    'Blue_Mean_Ang_2nd_Moment', 'Blue_Range_Ang_2nd_Moment', 'Blue_Mean_Contrast',
'Blue_Range_Contrast', 'Blue_Mean_Correlation', 'Blue_Range_Correlation', ...
    'INF_Mean_Ang_2nd_Moment', 'INF_Range_Ang_2nd_Moment', 'INF_Mean_Contrast',
'INF_Range_Contrast', 'INF_Mean_Correlation', 'INF_Range_Correlation'});

import_test =
full_test_T{:,["Solidity","Circularity","NonCompactness","Red_Range_Contrast",...

"Green_Range_Contrast","Blue_Range_Contrast","INF_Range_Contrast","Green_Mean_Cont
rast","Blue_Range_Correlation","Red_Mean_Contrast"]};


testLabels = [zeros(88,1) ; ones(109,1)];

%fit SVM
SVMModel = fitcsvm(import_features,trainLabels);

%predict
[predictions,~] = predict(SVMModel,import_test);

%confusion matrix
C = confusionmat(testLabels,predictions);

%get precision and recall for each class
ImportPrecision = diag(C) ./ sum(C,2);
ImportRecall = diag(C) ./ sum(C,1)';
ImportF1 = 2 * (mean(ImportPrecision) * mean(ImportRecall)) /
(mean(ImportPrecision) + mean(ImportRecall));

figure;
ImportCc = confusionchart(C);
ImportCc.Title = "Onions/Weeds Importance (10) Confusion Matrix";

%%%%%%%%%%%%%%%%%%%
%Less (4) Importance-Chosen Features
%%%%%%%%%%%%%%%%%%%
import_features =
full_data_T{:,["Solidity","Circularity","NonCompactness","Red_Range_Contrast"]};


trainLabels = [zeros(918,1); ones(453,1)];



%test
testFeats = [shapeTextO(919:end,:) ; shapeTextW(454:end,:)];


import_test =
full_test_T{:,["Solidity","Circularity","NonCompactness","Red_Range_Contrast"]};


testLabels = [zeros(88,1) ; ones(109,1)];
```

```
%fit SVM
SVMModel = fitcsvm(import_features,trainLabels);

%predict
[predictions,~] = predict(SVMModel,import_test);

%confusion matrix
C = confusionmat(testLabels,predictions);

%get precision and recall for each class
ReducedImpPrecision = diag(C) ./ sum(C,2);
ReducedImpRecall = diag(C) ./ sum(C,1)';
ReducedImpF1 = 2 * (mean(ReducedImpPrecision) * mean(ImportRecall)) /
(mean(ReducedImpPrecision) + mean(ImportRecall));

figure;
ReducedImpCc = confusionchart(C);
ReducedImpCc.Title = "Onions/Weeds Reduced Importance (4) Confusion Matrix";



%get table of metrics
SVM_Features = ["All_Shape", "All_Texture", "Shape_Texture", "10_Important",
"4_Important"]';

Onion_Precision =
[shapePrecision(1),textPrecision(1),shapeTextPrecision(1),ImportPrecision(1),Reduc
edImpPrecision(1)]';
Weed_Precision =
[shapePrecision(2),textPrecision(2),shapeTextPrecision(2),ImportPrecision(2),Reduc
edImpPrecision(2)]';

Onion_Recall =
[shapeRecall(1),textRecall(1),shapeTextRecall(1),ImportRecall(1),ReducedImpRecall(
1)]';
Weed_Recall =
[shapeRecall(2),textRecall(2),shapeTextRecall(2),ImportRecall(2),ReducedImpRecall(
2)]';

F1 = [shapeF1,textF1,shapeTextF1,ImportF1,ReducedImpF1]';

SVM_metrics =
table(SVM_Features,Onion_Precision,Weed_Precision,Onion_Recall,Weed_Recall,F1);
display(SVM_metrics);


%%%%%%%%%%%%%%%%%%%%%%%%%
%Output Image Predictions
%%%%%%%%%%%%%%%%%%%%%%%%%
figure;
test1 = imread("C:\Users\oakle\OneDrive - University of Lincoln\MSc Intelligent
Vision\CompVision\assessment\data\onions\19_truth.png");
test2 = imread("C:\Users\oakle\OneDrive - University of Lincoln\MSc Intelligent
Vision\CompVision\assessment\data\onions\20_truth.png");
test1O = predictions'; test1O= test1O(1:44);
test2O = predictions'; test2O= test2O(45:88);

test1W = predictions'; test1W= test1W(89:132);
```

```
test2W = predictions'; test2W= test2W(133:end);

t = tiledlayout(2,1,"TileSpacing","compact");
nexttile;
im = im2gray(test1);

%extract objects from ground truth
oni = im == 76;

weed = im == 29;

[lOnion,n_onion] = bwlabel(oni);
[lWeed,n_weed] = bwlabel(weed);

%go through each onion/weed and relabel(colour) with prediction
for i = 1:n_onion
    onionN = ismember(lOnion,i);

    if test1O(i) == 0
        labl = 100;
    else
        labl = 300;
    end

    im(onionN) = labl;
end

for i = 1:n_weed
    weedN = ismember(lWeed,i);

    if test1W(i) == 0
        labl = 300;
    else
        labl = 100;
    end

    im(weedN) = labl;
end

imshowpair(test1,label2rgb(im,"turbo"),"montage");
title("Sample 19 GT vs Predictions");
nexttile;

im = im2gray(test2);
oni = im == 76;

weed = im == 29;

[lOnion,n_onion] = bwlabel(oni);
[lWeed,n_weed] = bwlabel(weed);


for i = 1:n_onion-1
    onionN = ismember(lOnion,i);

    if test2O(i) == 0
        labl = 100;
    else
        labl = 300;
```

```
    end

    im(onionN) = labl;
end

for i = 1:n_weed
    weedN = ismember(lWeed,i);
    try %catches an error arose from txt file feature extraction indexing
        if test2W(i) == 0
            labl = 300;
        else
            labl = 100;
        end
    catch
        labl = 100;
    end
        im(weedN) = labl;
end

%show comparison.
imshowpair(test2,label2rgb(im,"turbo"),"montage");
title("Sample 20 GT vs Predictions");
```

6.3 Tracking

```
%Contains code for displaying ground truth trajectory, noisy observations
trajectory, and computed predicted trajectory from
%observations. Displays mean error, standard deviation and root mean squared error
(command window output).
%Code adapted from Computer Vision Workshop "Tracking 2", School of
%Computer Science, University of Lincoln (2023)
clear;close all;

%read ground truth coordinates
x = readmatrix("x.csv");
y = readmatrix("y.csv");

%read observations
a = readmatrix("a.csv");
b = readmatrix("b.csv");

%plot GT Coords
plot(x, y, 'xb');
hold on;

%plot observations
plot(a, b, '+r');

%concat observation coords
z = [a; b];

%get predictions using kalman filter
[px, py] = kalmanTracking(z);

%plot trajectories graph
plot(px,py,'g');
legend(["Ground Truth", "Observations", "Kalman
Predicted"],'Location','southeast');
```

```
title("Object Trajectory vs Observed vs Predicted");
xlabel('x');
ylabel('y');
hold off;

obs_errs = []; %observation errors
kal_errs = []; %kalman prediction errors

%%get errors between observations and ground truth coords
%%get errors between predicted and ground truth coords
for i=1:length(x)

    %get error (euclidean distance)
    ob_e = sqrt((x(i)-a(i))^2 + (y(i)-b(i))^2);
    k_e = sqrt((x(i)-px(i))^2 + (y(i)-py(i))^2);

    kal_errs = [kal_errs; k_e];
    obs_errs = [obs_errs; ob_e];
end

mean_ob = mean(obs_errs); %mean observation error
std_ob = std(obs_errs); %standard deviation obs error
rms_ob = rms(obs_errs); %root mean squared obs error

mean_ke = mean(kal_errs); %kalman mean error
std_ke = std(kal_errs); %standard deviation kal error
rms_ke = rms(kal_errs); %root mean squared kal error



err_type = reshape(["Mean Error", "Standard dev.", "RMSE"],[3,1]);
obs_err = reshape([mean_ob,std_ob,rms_ob],[3,1]);
kal_err = reshape([mean_ke,std_ke,rms_ke],[3,1]);

errorMetrics = table(err_type,obs_err,kal_err);
display(errorMetrics); %show table of errors


function [xp, Pp] = kalmanPredict(x, P, F, Q)
    % Prediction step of Kalman filter.
    % x: state vector
    % P: covariance matrix of x
    % F: matrix of motion model
    % Q: matrix of motion noise
    % Return predicted state vector xp and covariance Pp
    xp = F * x; % predict state
    Pp = F * P * F' + Q; % predict state covariance
end

function [xe, Pe] = kalmanUpdate(x, P, H, R, z)
    % Update step of Kalman filter.
    % x: state vector
    % P: covariance matrix of x
    % H: matrix of observation model
    % R: matrix of observation noise
    % z: observation vector
    % Return estimated state vector xe and covariance Pe
    S = H * P * H' + R; % innovation covariance
    K = P * H' * inv(S); % Kalman gain
```

```
    zp = H * x; % predicted observation
    %%%%%%%%% UNCOMMENT FOR VALIDATION GATING %%%%%%%%%
    gate = (z - zp)' * inv(S) * (z - zp);
    if gate > 1000
    %warning('Observation outside validation gate');
    xe = x;
    Pe = P;
    return
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    xe = x + K * (z - zp); % estimated state
    Pe = P - K * S * K'; % estimated covariance
end


function [px, py] = kalmanTracking(z)
    % Track a target with a Kalman filter
    % z: observation vector
    % Return the estimated state position coordinates (px,py)
    dt = 0.2; % time interval
    N = length(z); % number of samples
    F = [1 dt 0 0; 0 1 0 0; 0 0 1 dt; 0 0 0 1]; % CV motion model
    Q = [0.016 0 0 0; 0 0.36 0 0; 0 0 0.016 0; 0 0 0 0.36]; % motion noise
    H = [1 0 0 0; 0 0 1 0]; % Cartesian observation model
    R = [0.25 0; 0 0.25]; % observation noise
    x = [0 0 0 0]'; % initial state
    P = Q; % initial state covariance
    s = zeros(4,N);
        for i = 1 : N
         [xp, Pp] = kalmanPredict(x, P, F, Q);
         [x, P] = kalmanUpdate(xp, Pp, H, R, z(:,i));
         s(:,i) = x; % save current state
        end
    px = s(1,:); % NOTE: s(2, :) and s(4, :), not considered here,
    py = s(3,:); % contain the velocities on x and y respectively
end
```